самоучитель

Анатолий Постолит

ОСНОВЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ПРИМЕРАХ НА Python

2-е издание

- Необходимые основы языка Python
- Элементы искусственного интеллекта
- Разработка приложений искусственного интеллекта
- Инструментальные средства и полезные библиотеки
- Новые версии ПО и библиотек
- Программная реализация нейронных сетей
- Библиотеки PyBrain, Scikit-learn, Keras, TensorFlow, ImageAI, OpenCV
- Наглядные примеры нейронных сетей, их обучения и использования

+ Материалы на www.bhv.ru

bhv®

Анатолий Постолит

ОСНОВЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ПРИМЕРАХ НА Python

самоучитель

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2024

УДК 004.8
ББК 32.813
П63

Постолит А. В.

П63 Основы искусственного интеллекта в примерах на Python. Самоучитель. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2024. — 448 с.: ил. — (Самоучитель)

ISBN 978-5-9775-1818-5

Описаны инструментальные средства для разработки приложений искусственного интеллекта. Даны основы языка программирования Python. Раскрыты основные понятия и определения искусственного интеллекта. Рассмотрены вопросы программной реализации элементов нейронной сети и построения многослойных нейронных сетей. Большое внимание уделено применению специализированных библиотек PyBrain, Scikit-learn, Keras, TensorFlow для формирования структуры нейронных сетей и их обучения, и библиотек ImageAI и OpenCV для обработки изображений. Материал иллюстрирован простыми и понятными примерами, демонстрирующими использование предварительно обученных нейронных сетей для распознавания объектов на изображениях, создания собственных наборов данных, формирования структуры сети, ее обучения и практического применения. Во 2-м издании обновлены программные коды и версии библиотек, улучшены рисунки, учтены пожелания читателей и исправлены ошибки.

Электронное приложение-архив, доступное на сайте издательства, содержит листинги описанных в книге примеров.

Для программистов

УДК 004.8
ББК 32.813

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Людмила Гауль*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн серии	*Марины Дамбиевой*
Оформление обложки	*Зои Канторович*

Подписано в печать 31.07.23
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 36,12.
Тираж 1500 экз. Заказ № 7366
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-1818-5

Оглавление

Предисловие

С момента изобретения компьютеров их способность выполнять различные задачи значительно расширилась. Их научили слушать и понимать речь, проговаривать тест, распознавать объекты на рисунках и в видеофайлах, управлять беспилотными автомобилями и летательными аппаратами, писать стихи, музыку, распознавать эмоции людей и т. п.

Искусственный интеллект дает возможность компьютеру или роботу, управляемому компьютером, мыслить и принимать решения разумно, подобно тому, как думают и действуют люди. Искусственный интеллект работает, как и мозг человека, он учится, набирается опыта, а затем на практике использует результаты своего обучения.

Люди давно стремились заставить вычислительные машины мыслить и вести себя так же, как человек, и таким образом научить их решать не свойственные компьютерам задачи — например, играть в шахматы, сочинять стихи, писать музыку. Машинное обучение и нейронные сети все шире распространяются в различных сферах деятельности, и все больше средств инвестируется в эти технологии. С ростом объемов и сложности данных повышается необходимость их обработки и анализа при помощи искусственного интеллекта. Ведь он дает гораздо более точные оценки и прогнозы, которые заметно повышают эффективность, увеличивают производительность и снижают расходы.

Согласно отчетам мировых аналитических агентств, наблюдается устойчивый дефицит специалистов по искусственному интеллекту и машинному обучению. Спрос на них растет на 12% в год, а предложение удовлетворяется лишь на 7%, в итоге в ближайшем будущем открытых вакансий будет на 250 тыс. больше, чем потенциальных претендентов. В России число вакансий для специалистов по машинному обучению и искусственному интеллекту с 2017 года выросло почти в 11 раз. Вокруг машинного обучения сформировался ореол очень большой сложности. Это действительно так, если вы хотите делать открытия, разрабатывать новые алгоритмы и войти в историю мировой науки. Но если просто применять уже известные решения на практике, то порог входа в мир искусственного интеллекта не такой уж и высокий. Практически все программисты обладают необходимой базой знаний для построения карьеры специалиста по искусственному интеллекту. Даже школьники

и студенты при желании и целеустремленности могут приобщиться к миру искусственного интеллекта и получить нужную, увлекательную и достаточно перспективную профессию.

Создание систем искусственного интеллекта отличается от разработки обычных информационных систем. При работе над ними используется другой технологический подход, нужны навыки формирования тренировочных наборов данных и машинного обучения. Для реализации систем искусственного интеллекта надо иметь простой и удобный язык программирования с большим количеством готовых библиотек. Python — как раз один из таких языков, и неудивительно, что на нем ведется большое количество проектов, связанных с созданием нейронных сетей и машинным обучением.

Проектирование нейронных сетей включает три этапа: формирование структуры сети, подготовку обучающих данных, тренировку сети. Обучение нейронной сети требует мощных вычислительных средств, и считалось, что программная реализация этого процесса нуждается в применении низкоуровневых языков программирования C или C++. Однако в последние годы для этих целей появились стандартные, хорошо спроектированные и эффективно работающие библиотеки — например: TensorFlow, Theano или Torch. Они написаны на языке, близком к «железу», и обеспечивают использование всех возможностей центрального и графического процессоров. При этом проектирование структуры нейронной сети и запуск процесса ее обучения можно выполнять на более удобном, хотя и менее эффективном языке, а для сложных вычислений будут использоваться модули из подключаемых библиотек. Python как раз и является таким удобным и простым в использовании языком.

Эта книга предназначена как для начинающих программистов (школьников и студентов), так и для специалистов с опытом, которые планируют заниматься или уже занимаются разработкой систем искусственного интеллекта с использованием Python.

В первых пяти главах книги говорится об инструментарии, который необходим для создания нейронных сетей, рассматриваются основные понятия и определения искусственного интеллекта, приводятся примеры реализации нейронных сетей на Python. В последующих главах делается обзор специализированных библиотек, предназначенных для реализации систем искусственного интеллекта, приводится много примеров.

В книге описаны практически все элементарные действия, которые выполняют программисты, работая над реализацией нейронных сетей, дано множество примеров и проверенных программных модулей. Рассмотрены базовые классы популярных библиотек, методы (функции) каждого из классов и примеры их использования.

Глава 1 книги посвящена формированию инструментальной среды пользователя для разработки приложений искусственного интеллекта (установка и настройка программных средств). Это в первую очередь интерпретатор Python, интерактивная

среда разработки программного кода PyCharm и среда разработки интерфейса PyQt. На момент подготовки этого издания книги вышел Python версии 3.11. Однако часть модулей специализированных библиотек при работе с последними версиями Python выдавали ошибки. Поэтому в текущем издании все примеры реализованы на Python версии 3.7, в среде которого все программы работали стабильно.

> ***Примечание***
>
> Сейчас самое время сделать одно важное замечание относительно версий библиотек Python. Ситуация на рынке инструментальных средств развивается настолько стремительно, что многие пользователи не успевают отслеживать эти изменения. Python является языком программирования с открытым исходным кодом, и в этой среде работает множество независимых разработчиков. При этом достаточно часто возникают ситуации, когда улучшение одной из библиотек приводит к разрушению работы целого ряда зависимых программных средств. Не зря в народе говорится, что «лучшее — враг хорошего», и эта поговорка очень актуальна применительно к языкам программирования. Учитывая это обстоятельство, на официальных сайтах независимых разработчиков хранятся практически все предыдущие версии программных модулей. В начале каждой главы этой книги указаны версии библиотек, которые были актуальными на момент ее подготовки. Если через какой-то промежуток времени появятся новые версии библиотек, которые окажутся несовместимыми с некоторыми программными модулями, то у вас всегда будет возможность вернуться к предыдущей стабильно работающей версии.

Глава 2 посвящена базовым элементам языка Python — переменным, функциям, массивам, условиям и циклам, классам и объектам, созданным на основе этих классов. Это самые простые команды, которых вполне достаточно для понимания примеров, приведенных в последующих главах. Однако приведенных в главе сведений не хватит для полноценной работы с Python, да и целью этого издания является не столько сам Python, сколько специализированные библиотеки для работы с нейронными сетями. И для более глубокого изучения этого языка программирования необходимо воспользоваться специальной литературой.

Глава 3 посвящена основным понятиям и определениям в области искусственного интеллекта. Рассмотрен искусственный нейрон как основа нейронных сетей, различные функции активации, которые выполняют важную роль при работе искусственных нейронов. Показана структура однослойных и многослойных нейронных сетей. Раскрыты такие понятия, как обучающая выборка и тестовая выборка данных, обучение нейронных сетей с учителем и без учителя.

В *главе 4* приведены простые примеры программной реализации элементов нейронной сети — в частности, персептрона. Рассмотрены принципы решения задач классификации объектов, программные модули обучения персептрона пониманию и распознаванию изображений, способности самостоятельно обучаться и подбирать веса связей между нейронами в простейшей нейронной сети. Даются примеры программного кода классификации объектов.

В *главе 5* обсуждены вопросы построения многослойных нейронных сетей. Все начинается с исследования структуры простейшего искусственного нейрона и про-

граммирования его функций на Python. Затем рассмотрены технология построения сети из нейронов, обучение нейронной сети. Представлена последовательность шагов при проектировании нейронных сетей.

В *главе 6* сделан обзор полезных библиотек для создания нейронных сетей на Python. Это такие известные и популярные библиотеки, как PyBrain, Scikit-learn, Keras и TensorFlow. Возможности каждой из них проиллюстрированы примерами с пояснениями назначения отдельных команд. Во всех примерах использованы уже готовые обучающие выборки данных, которые либо можно скачать из Интернета, либо они уже входят в состав этих библиотек. При этом не нужно терять время на формирование обучающих и тестовых наборов данных, а можно сосредоточиться на сущности алгоритмов, используемых в этих библиотеках.

Глава 7 посвящена изучению библиотеки ImageAI, предназначенной для создания нейронных сетей обработки изображений. Здесь подробно описаны классы распознавания и обнаружения объектов на изображениях, методы (функции) этих классов. ImageAI — мощная библиотека, имеющая целый набор уже обученных моделей нейронных сетей, которые пользователь может всего несколькими строками программного кода подключить к своим приложениям для распознавания порядка 80 различных объектов. На вход этим моделям можно подавать либо рисунки и фотографии в виде файлов, либо видеопоток с видеокамер.

В библиотеке ImageAI имеются модели нейронных сетей для обнаружения объектов в изображениях с архитектурой YOLOv3. Для таких моделей пользователь может сформировать свой обучающий набор данных и, по сути, научить модель распознавать любые объекты на фотографиях, в видеофайлах и потоковом видео.

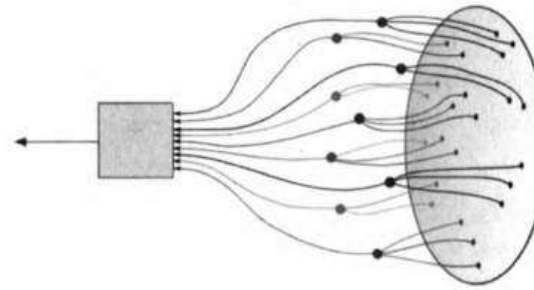
Заключительная, *восьмая*, *глава* посвящена использованию библиотеки OpenCV для создания приложений обработки изображений. В главе используются обученные классификаторы Хаара для распознавания объектов в любых изображениях. Это могут быть файлы с фотографиями, видеофайлы, потоковое видео. Интересная особенность этой библиотеки заключается в том, что модель нейронной сети можно научить распознавать на изображениях не только какой-то объект (автомобиль, пешехода, лицо человека), но и некоторые детали этого объекта — в частности, глаза, улыбку, определенную часть тела и даже эмоции. Также можно научить модель находить на изображениях конкретного человека.

На протяжении всей книги раскрываемые вопросы представлены достаточно упрощенными, но полностью законченными примерами. Ход решения той или иной задачи сопровождается большим количеством иллюстративного материала. Желательно изучение тех или иных разделов выполнять непосредственно за компьютером — тогда вы сможете последовательно повторять выполнение тех шагов, которые описаны в примерах, и сразу же видеть результаты своих действий. Это в значительной степени облегчает восприятие приведенных в книге материалов. Наилучший способ обучения — это практика. Все листинги программ приведены на языке Python. Это прекрасная возможность познакомиться с этим языком программирования и понять, насколько он прост и удобен в использовании.

Для поклонников Python есть еще одна хорошая новость: совсем недавно появился анонс о выпуске нового языка программирования для разработчиков искусственного интеллекта — Mojo. Mojo разработан как надмножество Python с компилятором, поэтому, если вы уже знаете Python, использование Mojo не должно вызвать затруднений. Язык Mojo сочетает в себе синтаксис Python с производительностью C. На некоторых задачах скорость работы скомпилированных программ в 35 000 раз выше, чем под интерпретатором Python.

Итак, если вас заинтересовали вопросы создания систем искусственного интеллекта с использованием Python и подключаемых библиотек, то самое время перейти к изучению материалов этой книги.

ГЛАВА 1



Инструментальные средства для разработки приложений искусственного интеллекта

На сегодняшний день существует множество языков программирования, каждый из которых имеет свои особенности. Но хочется выделить Python как популярную универсальную среду разработки программного кода с более чем тридцатилетней историей.

В конце 1989 года Гвидо Ван Россум создал Python — интерпретируемый язык программирования, который очень быстро стал популярен и востребован у программистов. В подтверждение этому можно привести компании-гиганты, которые используют Python для реализации глобальных проектов. Это Google, Microsoft, Facebook, Yandex и многие другие.

Область применения Python очень обширна. Его используют для решения самых различных задач: обработки научных данных, систем управления жизнеобеспечением, игр, веб-ресурсов, систем искусственного интеллекта.

За все время существования Python плодотворно использовался и динамично развивался. Создавались стандартные библиотеки для поддержки современных технологий — например, работы с базами данных, протоколами Интернета, электронной почтой, машинного обучения и многое другое.

Для ускорения процесса написания программного кода удобно использовать специализированную инструментальную среду IDE для Python (Integrated Development Environment — интегрированная среда разработки). Она имеет полный комплект средств, необходимых для эффективного программирования на Python. Обычно в состав IDE входят текстовый редактор, компилятор или интерпретатор, отладчик и другое программное обеспечение. Использование IDE позволяет увеличить скорость разработки программ (при условии предварительного обучения работе с этой инструментальной средой).

Еще одна замечательная особенность, которая вдохновляет разработчиков пользоваться Python, — это библиотека PyQt5Designer, благодаря которой можно достаточно быстро проектировать сложные оконные интерфейсы приложений. Пользователю достаточно просто перетаскивать различные готовые компоненты для создания собственных оконных форм.

Из материалов этой главы вы узнаете:

- что такое интерпретатор Python и как его установить;
- как начать использовать интерактивную среду разработки программного кода PyCharm;
- как можно установить различные дополнительные пакеты в Python с использованием менеджера пакетов pip;
- как можно использовать интерактивную среду разработки интерфейса PyQt5Designer.

1.1. Интерпретатор Python

Язык программирования Python является достаточно мощным инструментальным средством для разработки систем искусственного интеллекта (ИИ). Наибольшую ценность представляет даже не столько он сам, сколько набор подключаемых библиотек, на уровне которых уже реализованы все необходимые процедуры и функции. Разработчику достаточно написать несколько десятков строк программного кода, чтобы подключить необходимые библиотеки, создать набор нужных объектов, передать им исходные данные и отобразить итоговые результаты.

Для установки интерпретатора Python на компьютер первое, что нужно сделать, — это скачать дистрибутив. Загрузить его можно с официального сайта, перейдя по ссылке: **https://www.python.org/downloads/** (рис. 1.1).

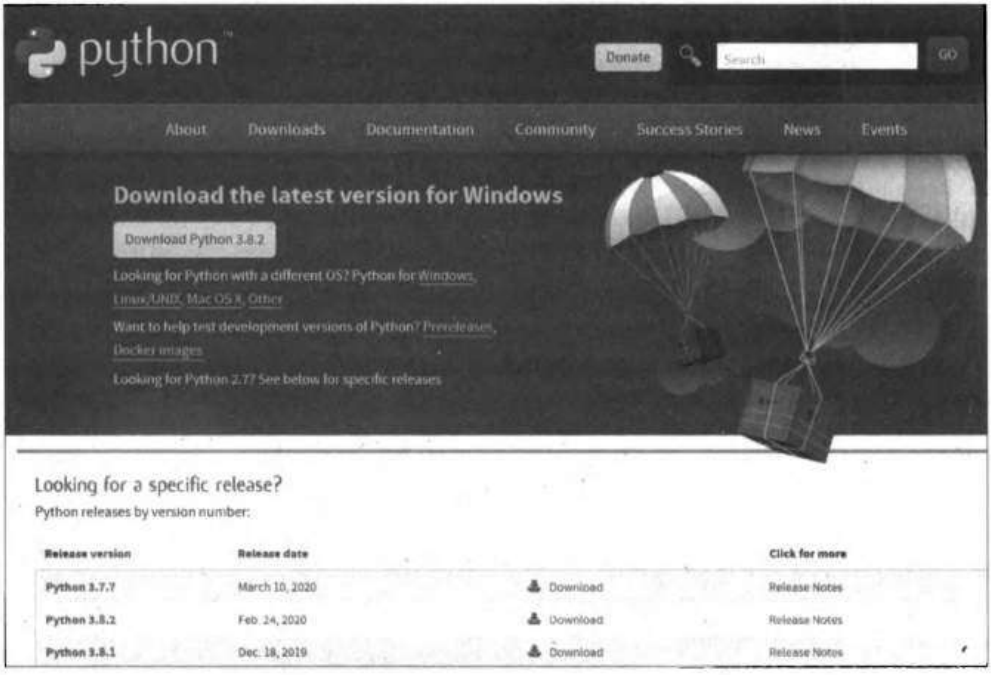


Рис. 1.1. Сайт для скачивания дистрибутива языка программирования Python

1.1.1. Установка Python в Windows

Для операционной системы Windows дистрибутив распространяется либо в виде исполняемого файла (с расширением exe), либо в виде архивного файла (с расширением zip).

Порядок установки:

1. Запустите скачанный установочный файл.
2. Выберите в открывшемся окне (рис. 1.2) способ установки.

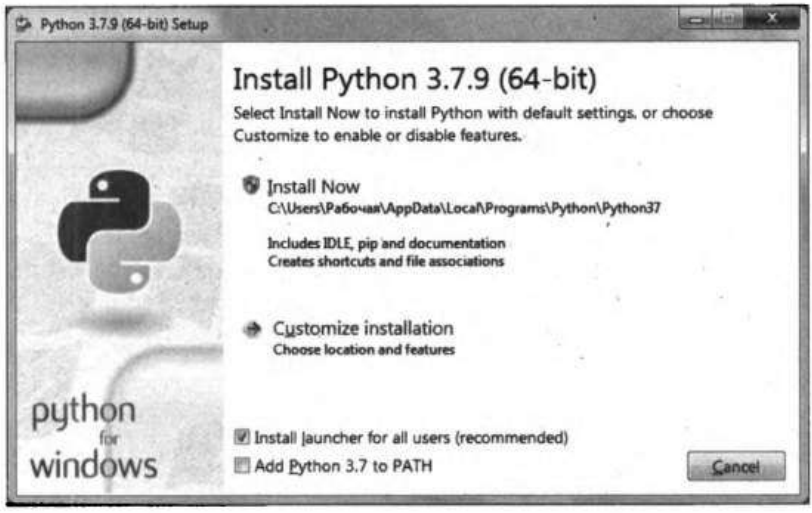


Рис. 1.2. Выбор способа установки Python

В этом окне предлагаются два варианта: **Install Now** и **Customize installation**. При выборе **Install Now** язык Python установится в папку по указанному пути. Помимо самого интерпретатора, будут установлены IDLE (интегрированная среда разработки), pip (пакетный менеджер) и документация, а также созданы соответствующие ярлыки и установлены связи файлов, имеющих расширение py, с интерпретатором Python. **Customize installation** — это вариант настраиваемой установки.

Опция **Add Python 3.7 to PATH** нужна для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке.

3. Отметьте необходимые опции установки — рис. 1.3 (окно открывается при выборе **Customize installation**).

На этом шаге нам предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python. Рекомендуется выбрать все опции:

- **Documentation** — установка документации;
- **pip** — установка пакетного менеджера pip;

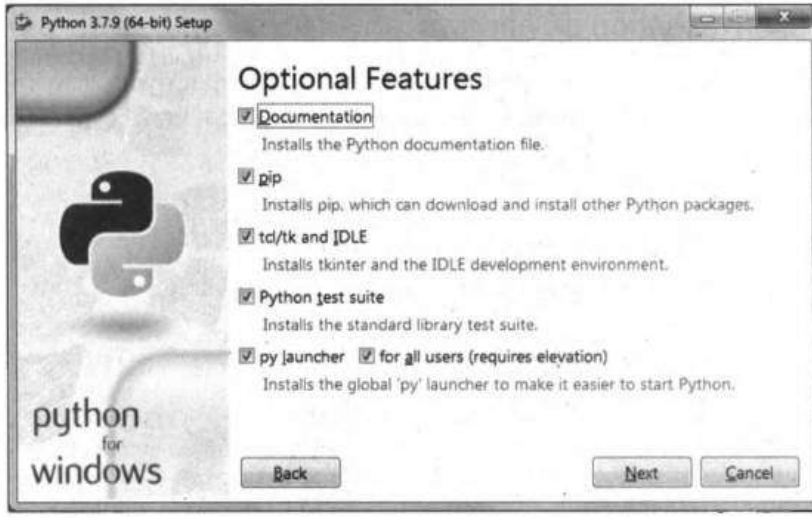


Рис. 1.3. Выбор опций установки Python

- **tcl/tk and IDLE** — установка интегрированной среды разработки (IDLE) и библиотеки для построения графического интерфейса (tkinter).

4. Выберите место установки — рис. 1.4 (окно открывается при выборе **Customize installation**).

Рис. 1.4. Выбор места установки Python

Помимо указания пути, это окно позволяет внести дополнительные изменения в процесс установки с помощью опций:

- **Install for all users** — установить для всех пользователей. Если не выбрать эту опцию, то будет предложен вариант инсталляции в папку пользователя, устанавливающего интерпретатор;

- **Associate files with Python** — связать файлы, имеющие расширение py, с Python. При выборе этой опции будут внесены изменения в Windows, позволяющие запускать Python-скрипты по двойному щелчку мыши;
- **Create shortcuts for installed applications** — создать ярлыки для запуска приложений;
- **Add Python to environment variables** — добавить пути до интерпретатора Python в переменную PATH;
- **Precompile standard library** — провести перекомпиляцию стандартной библиотеки.

Последние два пункта связаны с загрузкой компонентов для отладки, их мы устанавливать не будем.

5. После успешной установки вас ждет следующее сообщение (рис. 1.5).

Рис. 1.5. Финальное сообщение после установки Python

1.1.2. Установка Python в Linux

Чаще всего интерпретатор Python уже входит в состав дистрибутива Linux. Это можно проверить, набрав в окне терминала команду:

```
> python
```

или

```
> python3
```

В первом случае вы запустите Python 2, во втором — Python 3. В будущем, скорее всего, во все дистрибутивы Linux, включающие Python, будет входить только третья версия. Если у вас при попытке запустить Python выдается сообщение о том, что он не установлен или установлен, но не тот, что вы хотите, то у вас есть возможность взять его из репозитория.

Для установки из репозитория в Ubuntu воспользуйтесь командой:

```
> sudo apt-get install python3
```

1.1.3. Проверка интерпретатора Python

Для начала протестируем интерпретатор в командном режиме. Если вы работаете в Windows, то нажмите сочетание клавиш <Win>+<R> и в открывшемся окне введите `python`. В Linux откройте окно терминала и в нем введите `python3` (или `python`).

В результате Python запустится в командном режиме — выглядеть это будет примерно так, как показано на рис. 1.6 (картинка приведена для Windows, в Linux результат будет аналогичным).

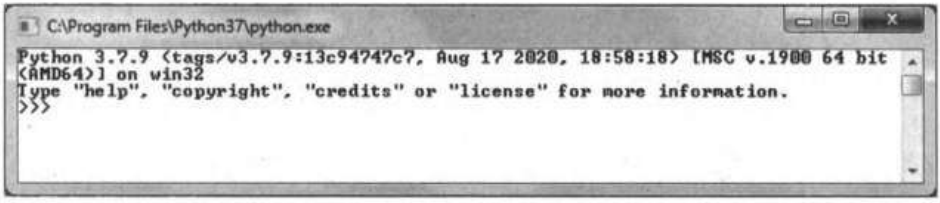


Рис. 1.6. Результат запуска интерпретатора Python в окне терминала

В этом окне введите программный код в виде одной строки:

```
print("Hello, World!")
```

Результат должен быть таким, как показано на рис. 1.7.

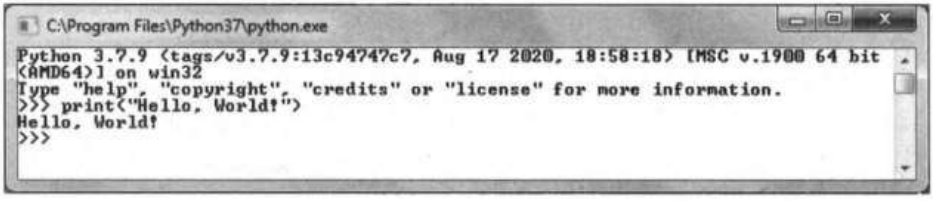


Рис. 1.7. Результат работы программы на Python в окне терминала

Если вы увидели аналогичный результат, значит, установка интерпретатора Python прошла без ошибок.

1.2. Интерактивная среда разработки программного кода PyCharm

В процессе разработки программных модулей удобнее работать в интерактивной среде разработки (IDE), а не в текстовом редакторе. Для Python одним из лучших вариантов считается IDE PyCharm от компании JetBrains. Для скачивания этого продукта нужно перейти по ссылке: **https://www.jetbrains.com/pycharm/download/** (рис. 1.8).

Среда разработки PyCharm доступна для Windows, Linux и macOS. Существуют два вида лицензии PyCharm: Professional и Community. Мы будем использовать версию Community, т. к. она бесплатна, и ее функциональности более чем достаточно для наших задач.

Рис. 1.8. Главная страница сайта для скачивания дистрибутива PyCharm

1.2.1. Установка PyCharm в Windows

1. Запустите скачанный дистрибутив PyCharm (рис. 1.9).
2. Выберите путь установки программы (рис. 1.10).

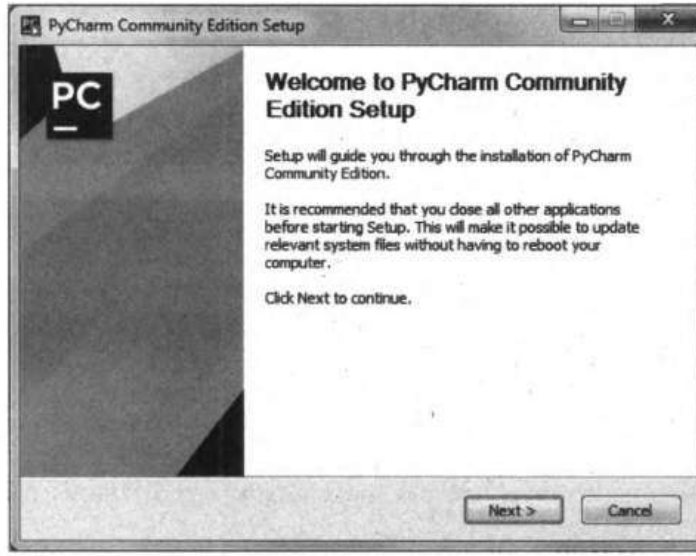


Рис. 1.9. Начальная заставка при инсталляции PyCharm

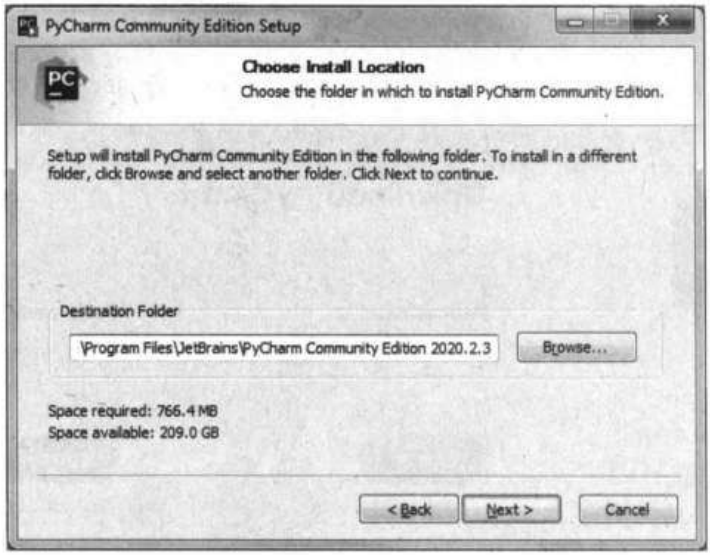


Рис. 1.10. Выбор пути установки PyCharm

3. Укажите ярлыки, которые нужно создать на рабочем столе (запуск 32- и 64-разрядной версии PyCharm), и сбросьте флажок в группе **Create associations**, если не требуется связывать с PyCharm файлы с расширением py (рис. 1.11).

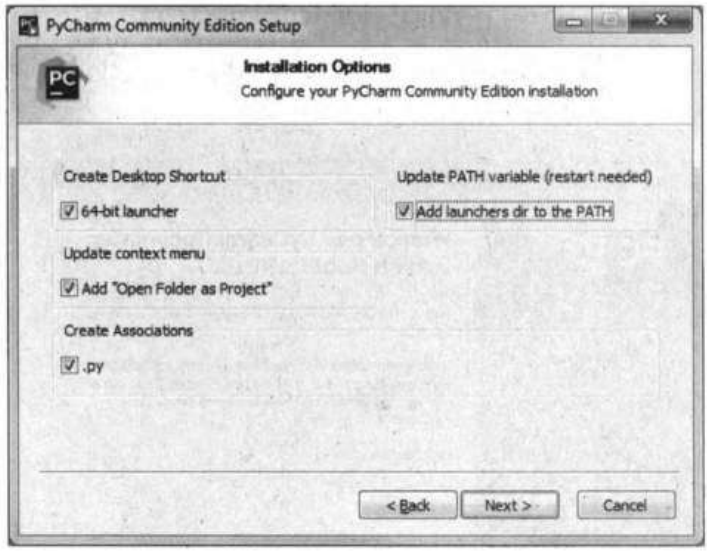


Рис. 1.11. Выбор разрядности устанавливаемой среды разработки PyCharm

4. Выберите имя для папки, которая появится в меню **Пуск** после установки PyCharm (рис. 1.12).
5. Далее PyCharm будет установлен на ваш компьютер (рис. 1.13).

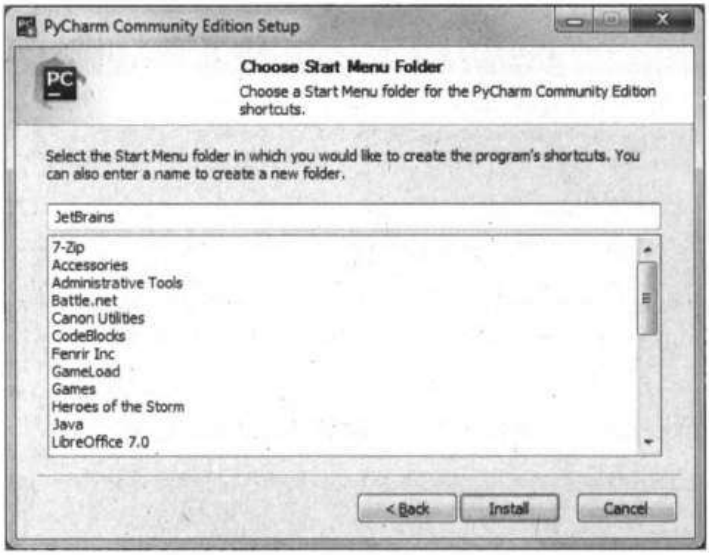


Рис. 1.12. Выбор имени папки PyCharm для меню **Пуск**

Рис. 1.13. Финальное окно установки пакета PyCharm

1.2.2. Установка PyCharm в Linux

1. Скачайте дистрибутив с сайта на компьютер.
2. Распакуйте архивный файл — для этого можно воспользоваться командой:

```
> tar xvf имя_архива.tar.gz
```

Результаты работы этой команды представлены на рис. 1.14.

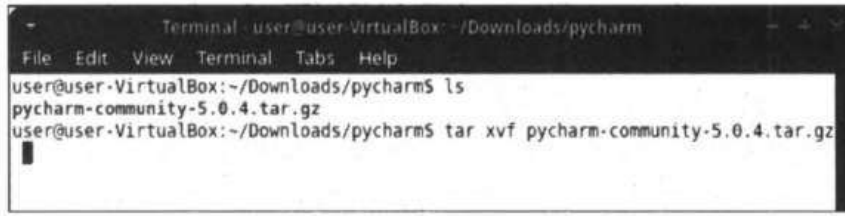


Рис. 1.14. Результаты работы команды распаковки архива PyCharm

3. Перейдите в каталог, который был создан после распаковки дистрибутива, найдите в нем подкаталог bin и зайдите в него. Запустите pycharm.sh командой:

```
> ./pycharm.sh
```

Результаты работы этой команды представлены на рис. 1.15.

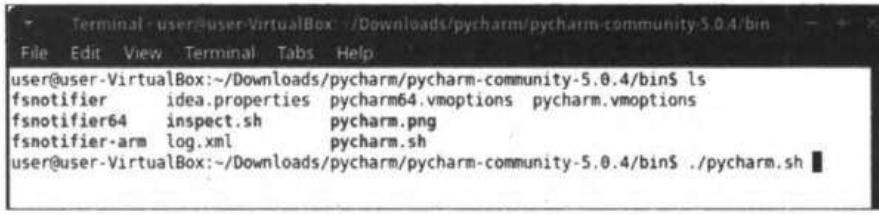


Рис. 1.15. Результаты работы команды инсталляции PyCharm

1.2.3. Проверка PyCharm

1. Запустите PyCharm и выберите опцию **New Project** в открывшемся окне (рис. 1.16).

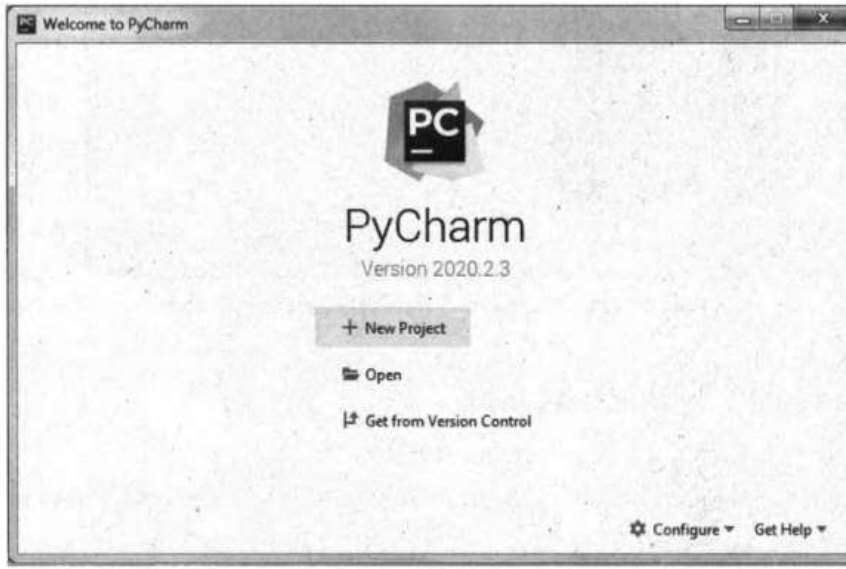


Рис. 1.16. Создание нового проекта в среде разработки PyCharm

2. Укажите путь до проекта Python и интерпретатор, который будет использоваться для запуска и отладки (рис. 1.17).

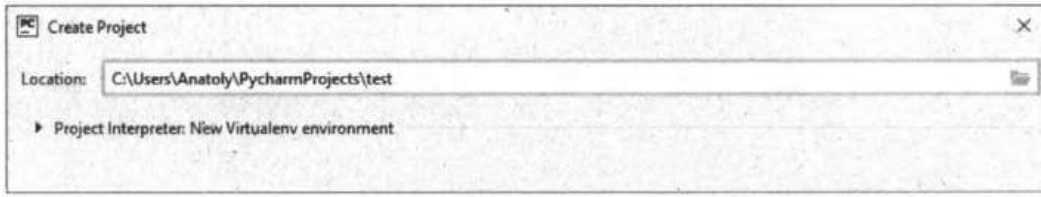


Рис. 1.17. Указание пути до проекта в среде разработки PyCharm

3. Добавьте в проект файл, в котором будет храниться программный код Python (рис. 1.18).

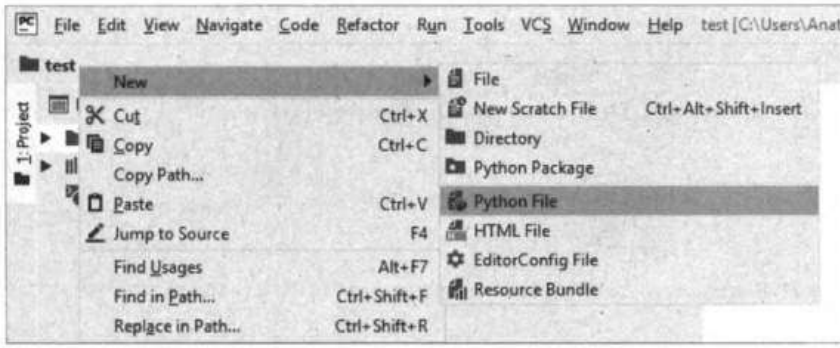


Рис. 1.18. Добавление в проект файла с программных кодом на Python

4. Введите одну строку кода программы (рис. 1.19).

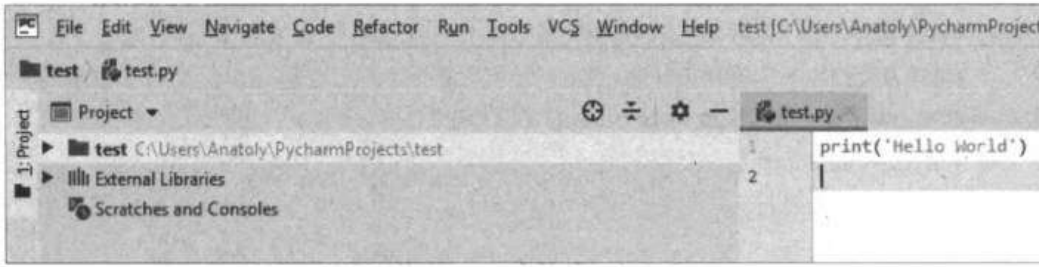


Рис. 1.19. Ввод одной строки программного кода на Python в среде разработки PyCharm

5. Запустите программу командой **Run** (рис. 1.20).

6. В результате должно открыться окно с выводом результатов работы программы (рис. 1.21).

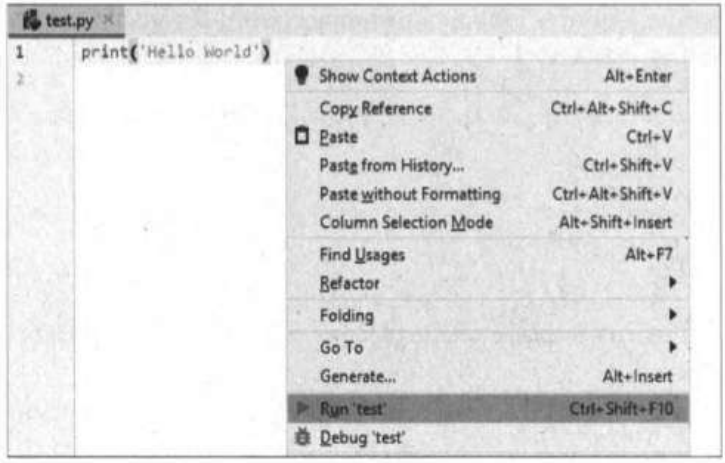


Рис. 1.20. Запуск программного кода на Python командой **Run** в среде разработки PyCharm

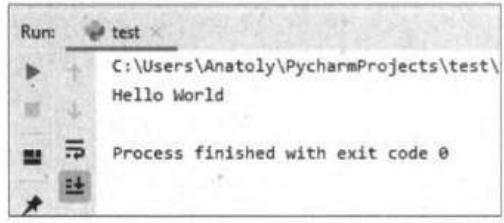


Рис. 1.21. Вывод результатов работы программы на Python в среде разработки PyCharm

1.3. Установка пакетов в Python с использованием менеджера пакетов pip

В этом разделе вы узнаете о том, откуда можно взять нужный вам дополнительный инструментарий для разработки своих программ. В частности:

- где взять отсутствующий пакет;
- как установить менеджер пакетов pip в Python;
- как использовать pip;
- как установить пакет;
- как удалить пакет;
- как обновить пакет;
- как получить список установленных пакетов;
- как выполнить поиск пакета в репозитории.

В процессе разработки программного обеспечения на Python часто возникает необходимость воспользоваться пакетом (библиотекой), который в текущий момент отсутствует на вашем компьютере.

1.3.1. Где взять отсутствующий пакет?

Необходимость в установке дополнительных пакетов возникает достаточно часто, поскольку решение практических задач обычно выходит за рамками базовой функциональности, которую предоставляет Python, — например, создание веб-приложений, обработка изображений, распознавание объектов, нейронные сети и другие элементы ИИ, геолокация и т. п. В этом случае необходимо узнать, какой пакет содержит функциональность, которая вам необходима, найти его, скачать, разместить в нужном каталоге и начать использовать. Все отмеченные действия можно сделать вручную, но этот процесс поддается автоматизации. К тому же скачивать пакеты с неизвестных сайтов может быть довольно опасно.

В рамках Python все эти задачи автоматизированы и решены. Существует так называемый Python Package Index (PyPI) — депозиторий, открытый для всех Python-разработчиков, в котором вы можете найти пакеты для решения практически любых задач. При этом у вас отпадает необходимость в разработке и отладке сложного программного кода — вы можете воспользоваться уже готовыми и проверенными решениями огромного сообщества программистов на Python: вам надо просто подключить нужный пакет или библиотеку к своему проекту и активировать уже реализованную в них функциональность. В этом и заключаются преимущества Python перед другими языками программирования, когда небольшим количеством программного кода можно реализовать решение достаточно сложных практических задач. В PyPI также реализована возможность выкладывать свои пакеты.

Для скачивания и установки нужных модулей в ваш проект используется специальная утилита, которая называется pip, — ее аббревиатура, которая на русском языке звучит как «пип», фактически означает «установщик пакетов» или «предпочитаемый установщик программ». Это утилита командной строки, которая позволяет устанавливать, переустанавливать и деинсталлировать PyPI-пакеты простой командой `pip`.

1.3.2. Менеджер пакетов pip в Python

Pip — это консольная утилита (без графического интерфейса). После того как вы ее скачаете и установите, она пропишется в `PATH` и будет доступна для использования. Эту утилиту можно запускать как самостоятельно — например, через терминал Windows или терминал пакета PyCharm:

```
> pip <аргументы>
```

так и через интерпретатор Python:

```
> python -m pip <аргументы>
```

Ключ `-m` означает, что мы хотим запустить модуль (в нашем случае pip).

При развертывании современной версии Python (начиная с Python 2.7.9 и более поздних версий) pip устанавливается автоматически. В PyCharm проверить наличие модуля pip достаточно просто. Для этого нужно войти в настройки проекта через

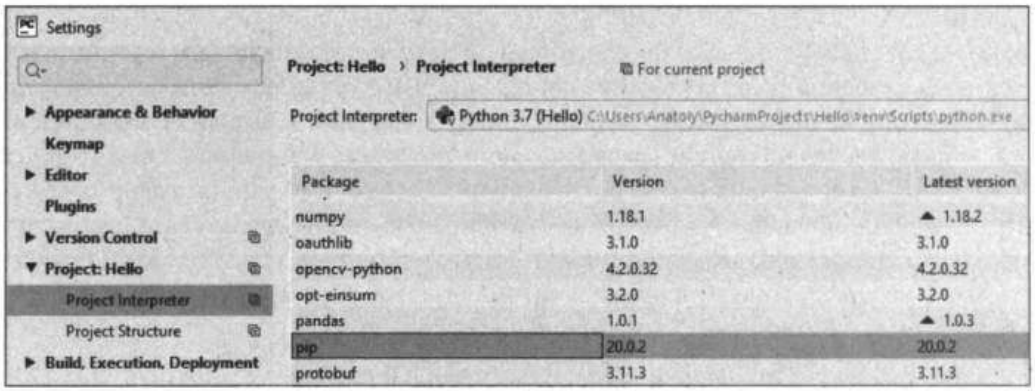


Рис. 1.22. Проверка наличия в проекте модуля pip

меню **File | Settings | Project Interpreter**[1]. Модуль pip должен присутствовать в списке загруженных пакетов и библиотек (рис. 1.22).

В случае отсутствия этого модуля последнюю его версию можно загрузить, нажав на значок + в правой части окна и выбрав модуль pip из списка (рис. 1.23).

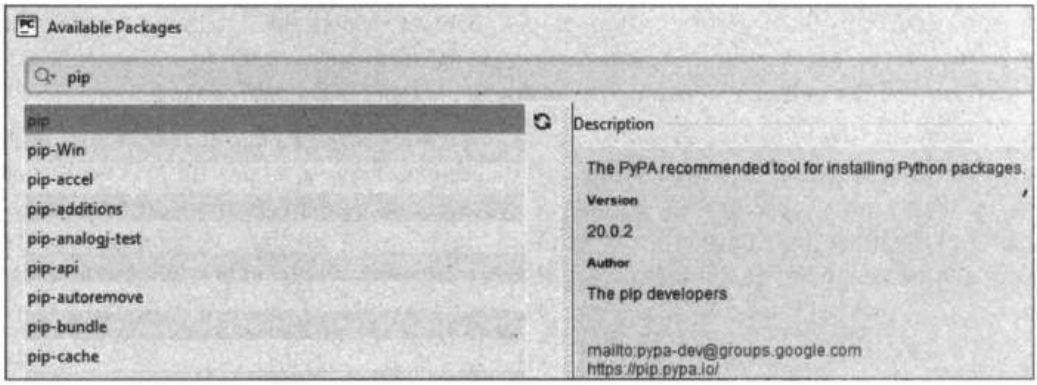


Рис. 1.23. Загрузка модуля pip

1.3.3. Использование pip

Далее рассмотрим основные варианты использования pip: установку пакетов, их удаление и обновление.

Установка пакета

Pip позволяет установить самую последнюю версию пакета, конкретную версию или воспользоваться логическим выражением, через которое можно определить,

[1] Эта запись выбора команд меню означает, что в меню **File** надо выбрать команду **Settings**, а затем команду **Project Interpreter**. Именно так в дальнейшем в книге будет указываться последовательность выбора команд.

что вам, например, нужна версия, не ниже указанной. Также есть поддержка установки пакетов из репозитория. Рассмотрим, как использовать эти варианты (здесь *Name* — это имя устанавливаемого пакета):

- установка последней версии пакета:

```
> pip install Name
```

- установка определенной версии:

```
> pip install Name==3.2
```

- установка пакета с версией не ниже 3.1:

```
> pip install Name>=3.1
```

Удаление пакета

Для того чтобы удалить пакет, воспользуйтесь командой:

```
> pip uninstall Name
```

Обновление пакетов

Для обновления пакета используйте ключ `-upgrade`:

```
> pip install --upgrade Name
```

Просмотр установленных пакетов

Для вывода списка всех установленных пакетов применяется команда:

```
> pip list
```

Если вы хотите получить более подробную информацию о конкретном пакете, то используйте аргумент `show`:

```
> pip show Name
```

Поиск пакета в репозитории

Если вы не знаете точное название пакета или хотите посмотреть на пакеты, содержащие конкретное слово, вы можете это сделать, используя аргумент `search`:

```
> pip search "test"
```

Если вы запускаете pip в терминале Windows, то терминальное окно автоматически закроется после того, как утилита завершит свою работу. При этом вы просто не успеете увидеть результаты ее работы. Для того чтобы терминальное окно не закрывалось автоматически, его нужно запускать с ключом `/k`. Например, запуск процедуры установки пакета tensorflow будет выглядеть так, как показано на рис. 1.24.

Если пакет pip запускается из терминального окна PyCharm, то в использовании дополнительных ключей нет необходимости, т. к. терминальное окно после завершения работы программ не закрывается. Пример выполнения той же команды в терминальном окне PyCharm показан на рис. 1.25.

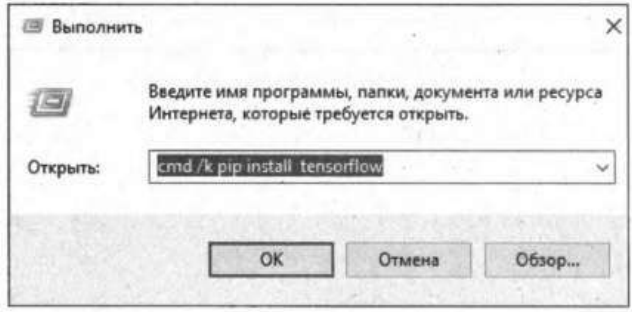


Рис. 1.24. Выполнение модуля pip в терминальном окне Windows

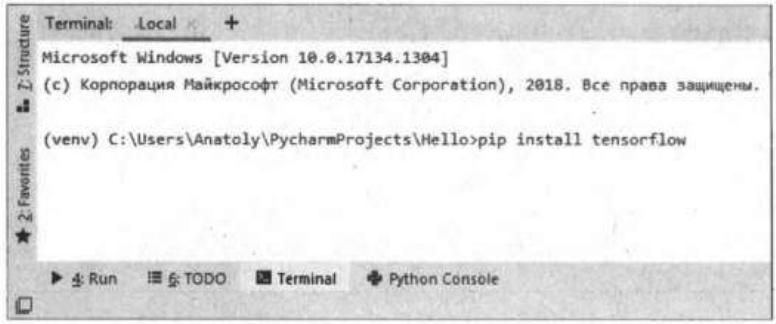


Рис. 1.25. Выполнение модуля pip в терминальном окне PyCharm

1.4. Интерактивная среда разработки интерфейса PyQt

Итак, основной инструментарий для разработки программ на языке Python установлен, и мы можем перейти к инсталляции дополнительных библиотек — PyQt5Designer и PyQt5, позволяющих разрабатывать и запускать приложения с графическим интерфейсом. Библиотека PyQt5Designer представляет программистам простой интерфейс перетаскивания для размещения компонентов — таких как кнопки, текстовые поля, поля со списком и многое другое.

Для установки библиотеки достаточно в окне терминала среды PyCharm выполнить команду:

```
pip install PyQt5Designer
```

После этого произойдут автоматическое скачивание дистрибутива и установка необходимых модулей. По завершении указанных процессов мы получим сообщение об успешном завершении установки (рис. 1.26).

В папке проекта, из которой была запущена инсталляция (см. последнюю строку на рис. 1.26), появится файл designer.exe. Если запустить его, то загрузится программа Qt Designer для разработки внешнего интерфейса приложений (рис. 1.27).

Для установки библиотеки PyQt5 можно выбрать команду **File | Settings**, в окне PyCharm (рис. 1.28).

```
(venv) C:\Users\Anatoly\PycharmProjects\Hello>pip install PyQt5Designer
Collecting PyQt5Designer
  Downloading https://files.pythonhosted.org/packages/ec/14/bd55fc528ad76(
    100% |████████████████████████████████| 40.8MB 305kB/s
Installing collected packages: PyQt5Designer
Successfully installed PyQt5Designer-5.14.1

(venv) C:\Users\Anatoly\PycharmProjects\Hello>
```

Рис. 1.26. Установки библиотеки PyQt5Designer в среде разработки PyCharm

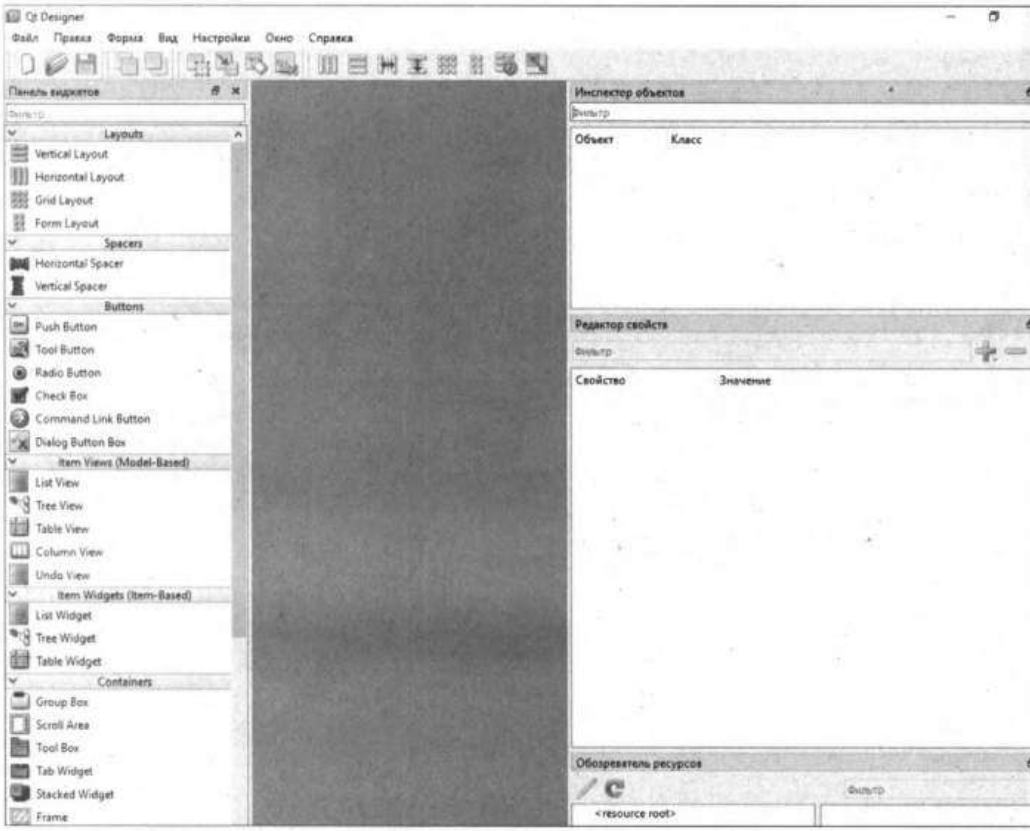


Рис. 1.27. Интерфейс программы Qt Designer

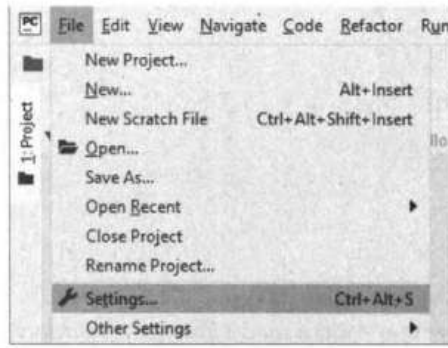


Рис. 1.28. Вызов окна настройки параметров проекта **Settings**

В левой части открывшегося окна настроек выберите опцию **Project Interpreter**, после этого в правой части окна будет показана информация об интерпретаторе языка Python и подключенных к нему библиотеках (рис. 1.29).

Для того чтобы добавить новую библиотеку, нужно нажать на значок + в правой части окна, после чего будет отображен полный список доступных библиотек (рис. 1.30).

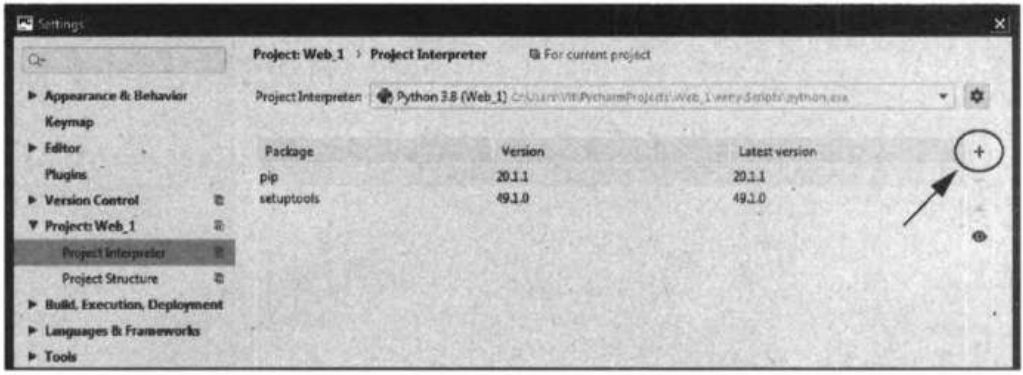


Рис. 1.29. Перечень установленных библиотек в окне настройки параметров проекта **Settings**

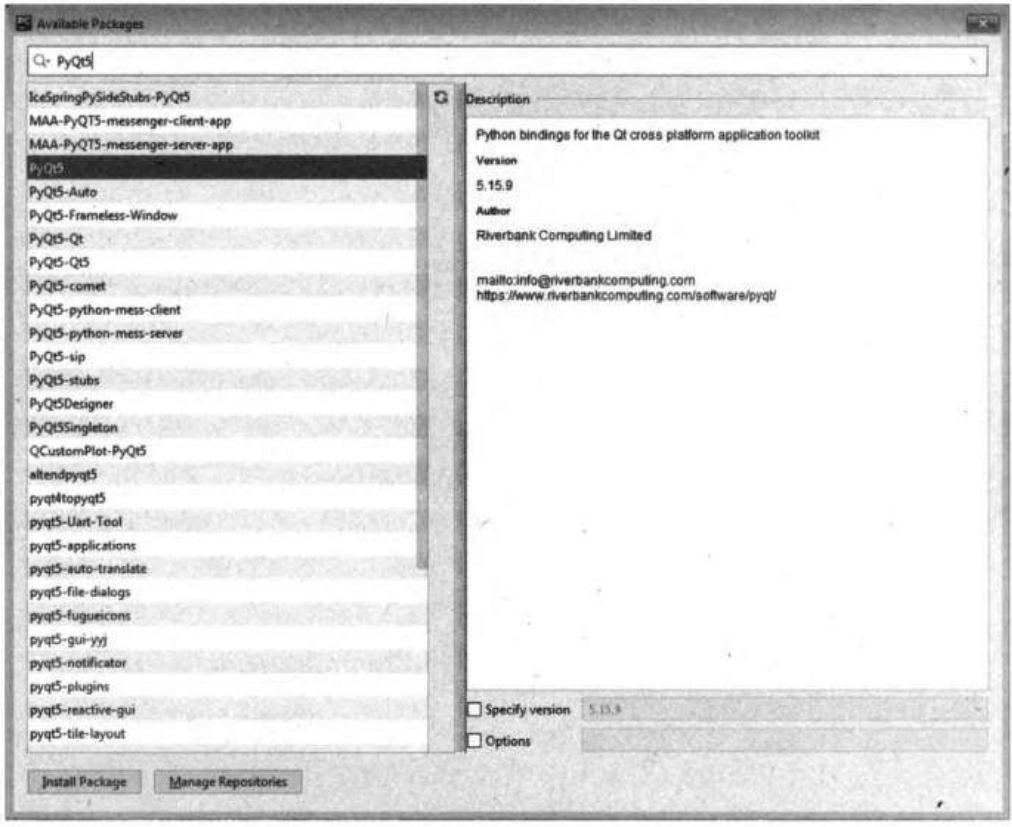


Рис. 1.30. Поиск библиотеки PyQt5 в списке доступных библиотек

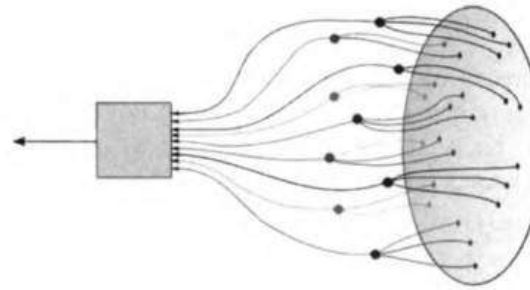
Здесь можно либо пролистать весь список и отыскать библиотеку PyQt5, либо набрать наименование этой библиотеки в верхней строке поиска, и она будет найдена в списке. После нажатия кнопки **Install Package** (в левом нижнем углу окна) выбранная библиотека будет добавлена в ваш проект.

Теперь у нас есть минимальный набор инструментальных средств, который необходим для разработки приложений на языке Python. В процессе расмотрения конкретных примеров нам понадобится загрузка еще ряда дополнительных пакетов и библиотек. Их описание и процедуры подключения будут представлены в последующих главах.

1.5. Краткие итоги главы

В этой главе мы познакомились с основными инструментальными средствами, с помощью которых можно разрабатывать приложения искусственного интеллекта. Это по сути дела интерпретатор Python, интерактивная среда разработки программного кода PyCharm, интерактивная среда разработки интерфейса PyQt5Designer. Установив на свой компьютер эти инструментальные средства, теоретически можно приступать к написанию программного кода. Однако наличие инструмента — это необходимое, но недостаточное условие для того, чтобы приступить к программированию. Нужно еще и уметь использовать этот инструментарий. Те специалисты, которые имеют богатый опыт программирования на Python, могут пропустить следующую главу, т. к. в ней даны элементарные представления об этом языке программирования. Для тех, кто только начинает знакомиться с миром Python, следующую главу рекомендуется не просто прочитать, но и сесть за компьютер и самостоятельно создать небольшие программные модули, о которых будет в ней говориться. А после того как будут получены элементарные навыки работы с Python, постараться накопить более глубокие знания и навыки работы с этим языком программирования, воспользовавшись предназначенной для этих целей специальной литературой. Итак, переходим к следующей главе и знакомимся с Python.

ГЛАВА 2



Основы языка программирования Python

Согласно индексу TIOBE (ежемесячный индикатор популярности языков программирования на базе подсчета результатов поисковых запросов), Python три раза становился языком года: в 2007, 2010 и 2018 годах. Эта награда присуждается тому языку программирования, который имеет самый высокий рост рейтинга за год. В 2023 году Python опять находится в первой строчке рейтинга TIOBE.

Как и любой другой язык программирования, Python имеет достоинства и недостатки. Однако количество разработчиков, увлеченных этим языком программирования, растет, как и число проектов, взаимно требующих Python-специалистов. Python развивается и не померкнет еще долго. В многочисленных обзорах и рейтингах язык занимает высокие позиции. Согласно рейтингу языков программирования DOU, он находится на пятом месте и занимает третью позицию в веб-технологиях. Для решения большинства задач — в частности, для веб-разработки, для обработки научных данных, для создания нейронных сетей и машинного обучения, для работы с большими данными, — это один из лучших языков.

Python — это объектно-ориентированный язык программирования общего назначения, который разработан с целью повышения продуктивности программиста. Он имеет следующие достоинства:

- низкий порог вхождения (синтаксис Python более понятный для новичков);
- это логичный, лаконичный и понятный язык программирования (разве что Visual Basic может сравниться с ним по этим параметрам);
- это кросс-платформенный язык, подходящий для разных платформ (Linux, Windows, MacOS);
- в нем предусмотрена реализация интерпретаторов для мобильных устройств и непопулярных систем (с фреймворком Kivy+KivyMD для Android и iOS);
- имеет широкое применение (используется для разработки веб-приложений с фреймворком Django, игр, математических вычислений, машинного обучения, в области Интернета вещей);
- у него сильная поддержка в Интернете со стороны приверженцев этого языка;
- имеется мощная поддержка компаний-гигантов IT-индустрии (Google, Dropbox, Spotify, Quora, Netflix);

- высокая потребность в программистах этого направления на рынке труда;
- в мире Python много качественных библиотек, так что не нужно «изобретать велосипед», если требуется срочно решить какую-то коммерческую задачу;
- для обучения есть много литературы — в первую очередь на английском языке, но и в переводе также издано большое количество книг;
- отличается строгим требованием к написанию кода (обязательны отступы), что является преимуществом (язык приучает записывать код организованно и красиво).

В этой главе мы не будем основательно рассматривать все тонкости и возможности Python. Остановимся только на тех операторах и процедурах, которые нам понадобятся в процессе написания примеров, реализующих элементы искусственного интеллекта. В частности, будут рассмотрены следующие базовые элементы этого языка:

- переменные;
- функции;
- массивы;
- условия и циклы;
- классы и объекты;
- пример создания собственного класса и объектов на его основе;
- программные модули;
- оконная форма как основа интерфейса;
- подключение Windows-формы к программе на Python;
- сборка исполняемого файла на Python под Windows.

Итак, совершаем небольшое погружение в мир Python, при этом будем использовать инструментальное средство, позволяющее облегчить написание программного кода, — PyCharm.

2.1. Первая программа в среде интерпретатора Python

В этом разделе описывается последовательность создания простейшего приложения с использованием программной оболочки PyCharm.

Для создания приложения выполните следующие действия.

1. Запустите PyCharm и в строке главного меню выберите команду **File | New Project** (рис. 2.1).
2. В открывшемся окне в строке **Location** задайте имя проекта **MyProject**. (рис. 2.2).
3. Разверните раздел **Project interpreter: New virtualenv environment** и убедитесь, что в создаваемой виртуальной среде программирования определен интерпретатор языка Python (рис. 2.3).

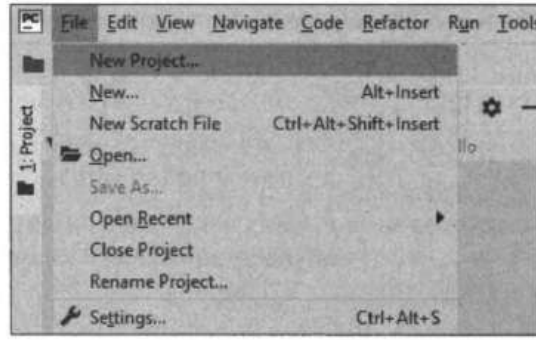


Рис. 2.1. Создание нового проекта в среде программирования PyCharm

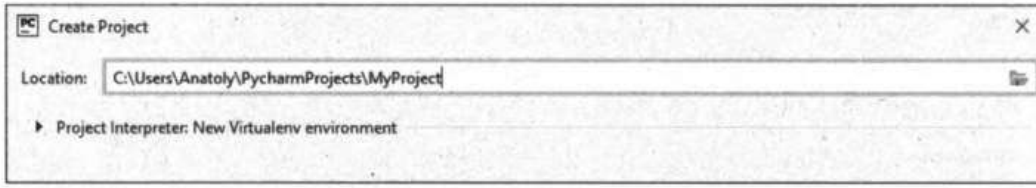


Рис. 2.2. Задание имени новому проекту в среде программирования PyCharm

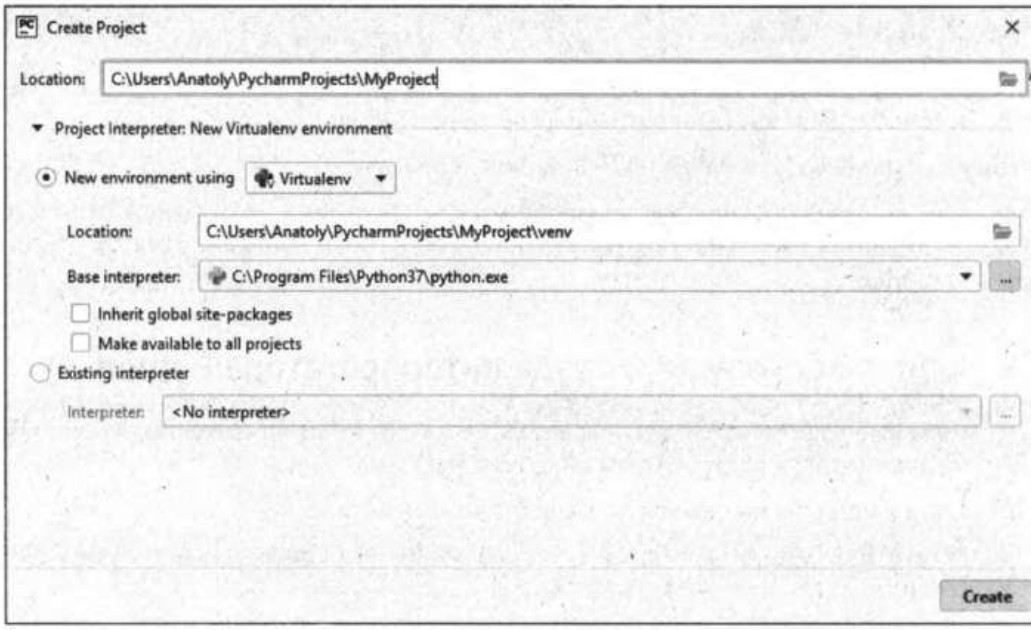


Рис. 2.3. Задание интерпретатора языка Python в виртуальной среде программирования

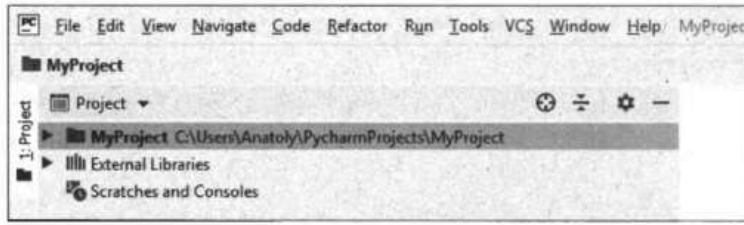


Рис. 2.4. Новый проект в среде программирования PyCharm

В нашем случае определен интерпретатор языка Python версии 3.7. После нажатия кнопки **Create** будет создан новый проект (рис. 2.4).

4. На следующем шаге необходимо создать файл, в котором будет содержаться программный код на языке Python. Для этого щелкните правой кнопкой мыши в строке **MyProject** и в контекстном меню выберите команду **New | Python File** (рис. 2.5).

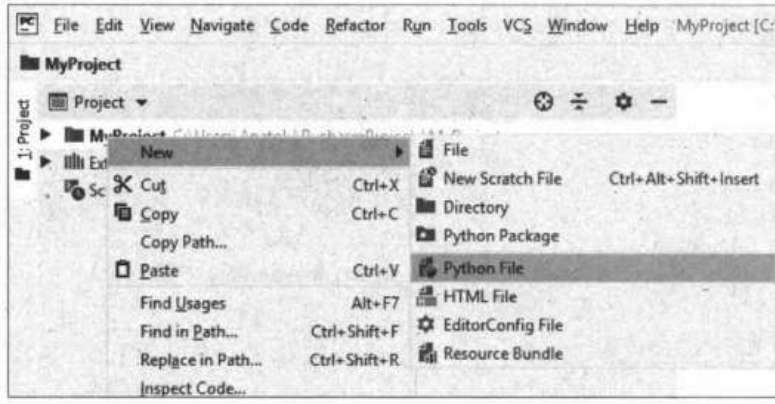


Рис. 2.5. Создание файла с программным кодом на языке Python

5. В открывшемся окне (рис. 2.6) выберите вариант **Python file** и задайте имя создаваемой программы. Назовем ее **FirstProgram**.

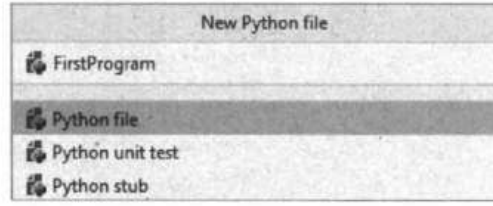


Рис. 2.6. Задание имени файла с программным кодом на языке Python

После нажатия клавиши <Enter> будет создан соответствующий файл и откроется окно для ввода и редактирования программного кода (рис. 2.7).

Напишем в окне редактора одну строку программного кода на языке Python:

```
print("Привет Python")
```

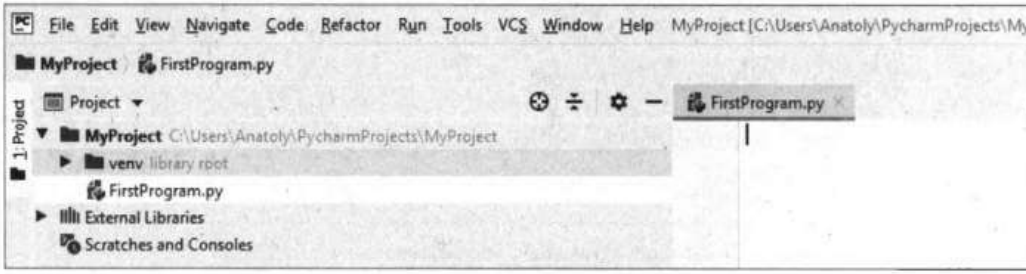


Рис. 2.7. Окно для ввода и редактирования программного кода на языке Python

Запустить программу на исполнение можно двумя способами:

- либо выбрать в главном меню команду **Run | Run 'FirstProgram'** (рис. 2.8);
- либо щелкнуть правой кнопкой мыши в окне редактора программного кода и в контекстном меню выбрать команду **Run 'FirstProgram'** (рис. 2.9).

В обоих случаях программный код будет запущен на исполнение и в нижней части экрана отобразятся результаты его работы (рис. 2.10).

Рис. 2.8. Запуск программы через главное меню PyCharm

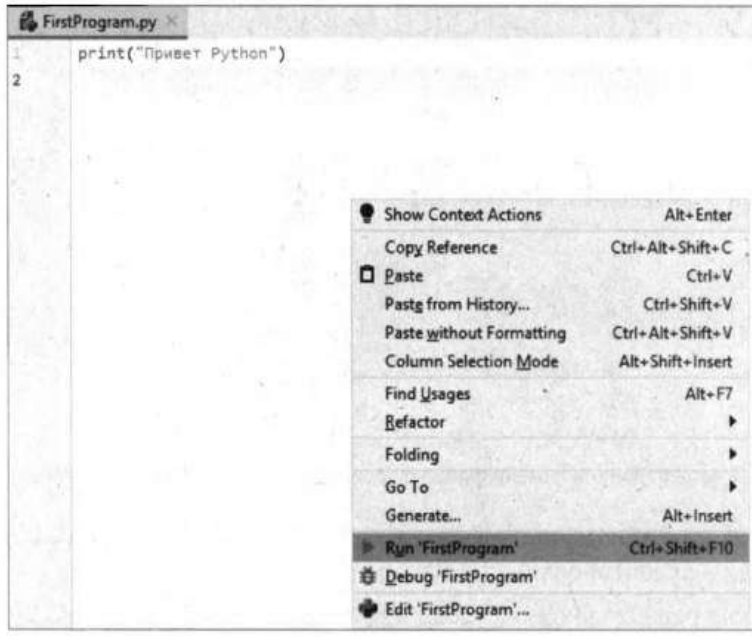


Рис. 2.9. Запуск программы через контекстное меню PyCharm

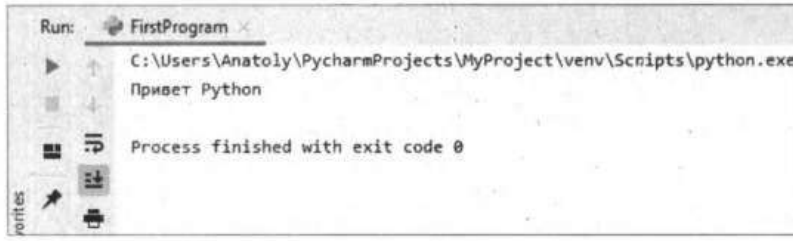


Рис. 2.10. Результаты работы первой программы на языке Python

Итак, мы сделали первые шаги по освоению языка программирования Python — загрузили на свой компьютер пакет необходимых инструментальных средств и написали первую программу. Однако наша программа работает только в инструментальной среде разработчика и не имеет пользовательского интерфейса. Следующим шагом попробуем выполнить тот же программный код, но с использованием оконного интерфейса пользователя.

2.2. Оконная форма как основа интерфейса

Рассмотрим, как построить простую пользовательскую форму в среде Windows.

Для создания Windows-формы выполните следующие действия.

1. Запустите Qt Designer. Откроется диалоговое окно, в котором предлагается создать новую пользовательскую форму (рис. 2.11).

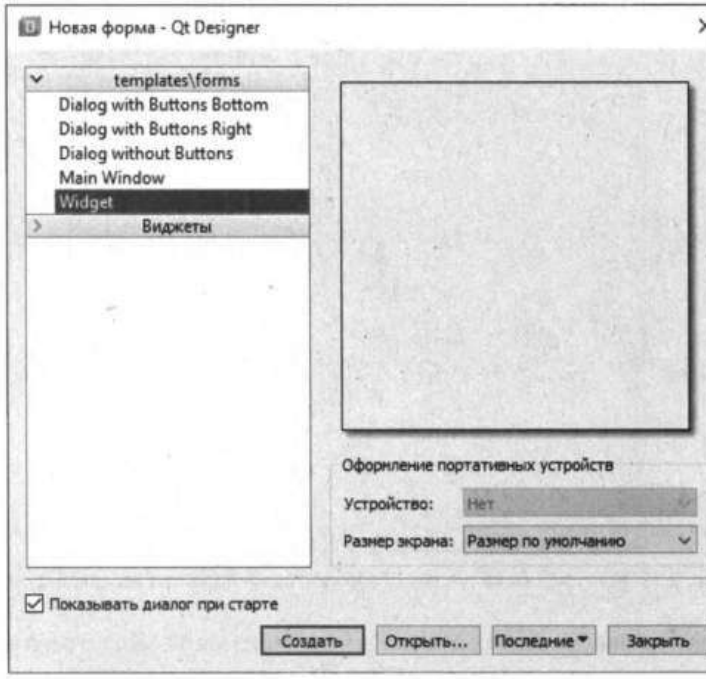


Рис. 2.11. Диалоговое окно создания формы

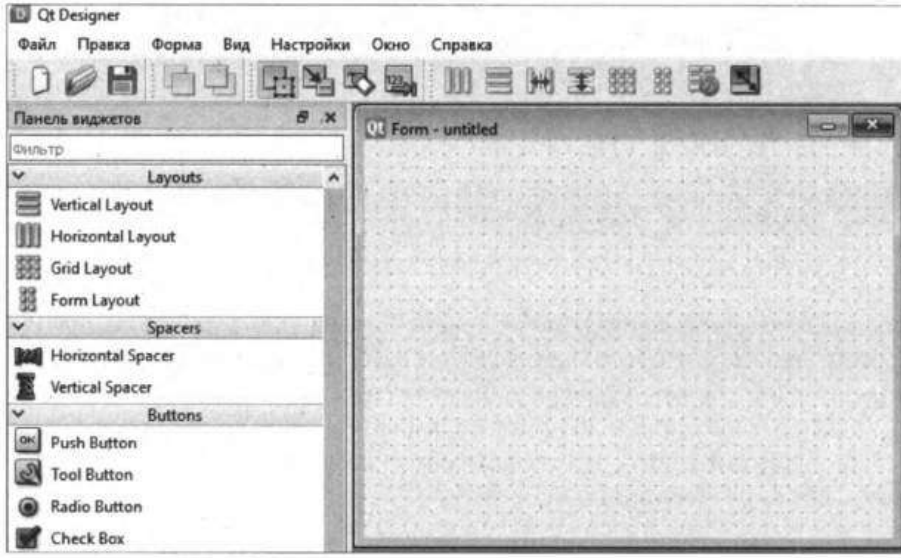


Рис. 2.12. Пустая пользовательская форма, созданная в Qt Designer

2. В этом окне выберите тип формы **Widget** (элемент управления) и нажмите кнопку **Создать**. Откроется новая пустая форма (рис. 2.12).
3. Добавим на форму надпись и кнопку. Для этого на панели **Панель виджетов** в группе **Display Widgets** нажмите левую кнопку мыши на пункте **Label** и, не отпуская кнопку мыши, перетащите компонент на форму. Проделайте аналогичную операцию с компонентом **Push Button** из группы **Buttons** и поместите его ниже надписи (рис. 2.13).

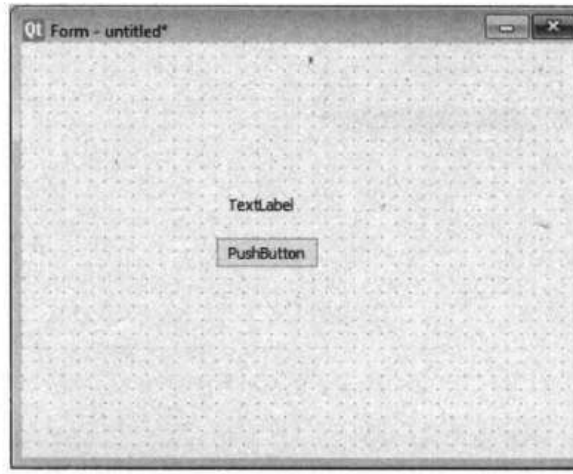


Рис. 2.13. Компоновка визуальных элементов на пользовательской форме

4. Теперь изменим некоторые свойства созданного окна. Для этого в правой части интерфейса Qt Designer найдите панель **Инспектор объектов** и выделите на ней первый пункт — **Form** (рис. 2.14).

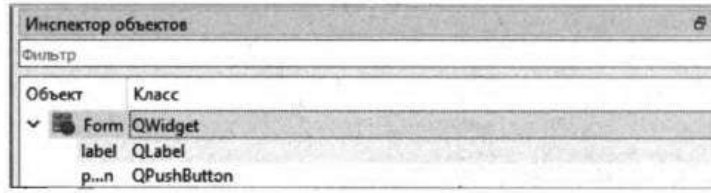


Рис. 2.14. Выбор объекта **Form** в панели **Инспектор объектов**

5. В правой части интерфейса Qt Designer перейдите в панель **Редактор свойств**, найдите свойство **objectName** и справа от свойства измените значение **Form** на **MyForm** (рис. 2.15).

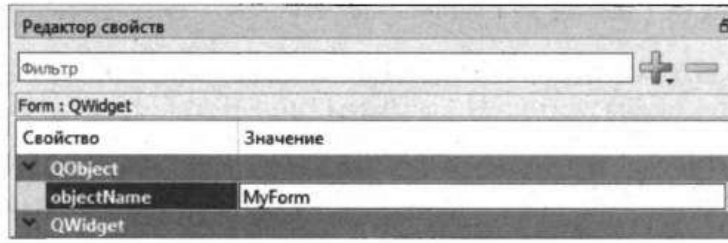


Рис. 2.15. Изменение имени формы в панели **Редактор свойств**

6. Для того чтобы изменить текст, который будет отображаться в заголовке окна, в панели **Редактор свойств** у свойства **windowTitle** (это заголовок нашей формы) измените значение на **Моя первая форма** (рис. 2.16).

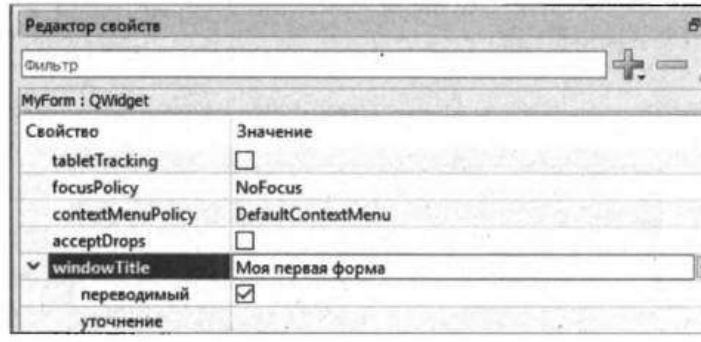


Рис. 2.16. Изменение заголовка формы в панели **Редактор свойств**

7. Для того чтобы изменить свойства надписи в элементе **Label**, следует выделить этот компонент с помощью мыши или выбрать соответствующий ему пункт в панели **Инспектор объектов**. Для нашего примера значение свойства **text** замените значением **Здесь будет текст** (рис. 2.17).

   На форме будет видно, что эта надпись не помещается в отведенный по умолчанию размер поля **Label**. Для его увеличения нужно выделить элемент **Label** и, удерживая правую границу поля левой кнопкой мыши, растянуть его до нужного размера.

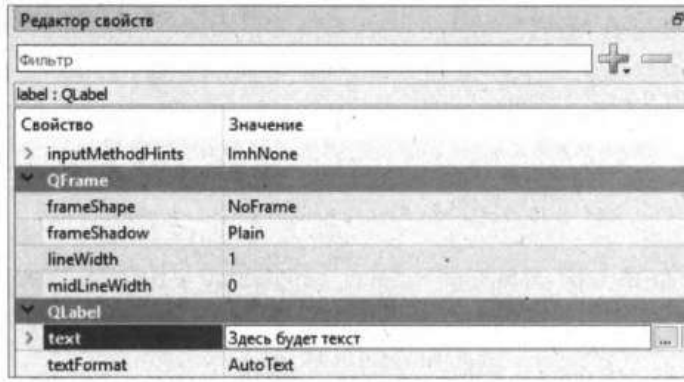


Рис. 2.17. Изменение свойства элемента **Label** в панели **Редактор свойств**

8. Теперь выделите на форме кнопку **Push Button** и в панели **Редактор свойств** замените значение свойства **text** значением **Приветствие** (рис. 2.18).

   Результаты наших действий представлены на рис. 2.19.

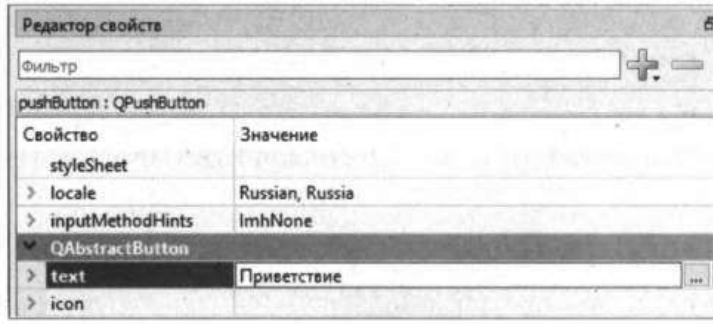


Рис. 2.18. Изменение свойства элемента **Push Button** в панели **Редактор свойств**

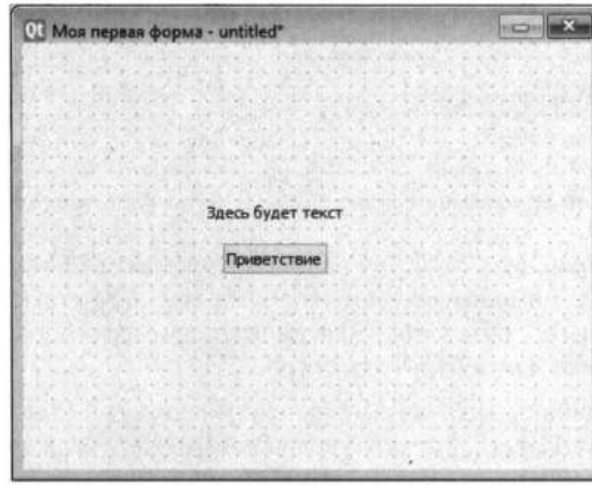


Рис. 2.19. Внешний вид формы с измененными свойствами

9. Закончив компоновку формы, сохраните ее в виде файла. Для этого в главном меню Qt Designer выберите команду **Файл | Сохранить** и ведите имя файла MyForm.ui. При необходимости внести в этот файл какие-либо изменения, его можно открыть в программе Qt Designer, выбрав в главном меню программы команду **Файл | Открыть**.

2.3. Подключение Windows-формы к программе на Python

Если открыть содержимое файла MyForm.ui, то можно убедиться, что внутри ui-файла содержится текст в XML-формате, а не программный код на языке Python (рис. 2.20).

Рис. 2.20. Фрагмент содержимого файла MyForm.ui

Следовательно, подключить этот файл к программе на Python с помощью инструкции `import` не получится. Чтобы обойти это неудобство, в среде PyCharm создадим новый файл с именем SimpleForm.py и напишем несколько строк программного кода на Python (листинг 2.1).

Листинг 2.1

```
from PyQt5 import QtWidgets, uic
import sys
app = QtWidgets.QApplication(sys.argv)
window = uic.loadUi("MyForm.ui")
window.show()
sys.exit(app.exec_())
```

Рассмотрим этот программный код. Для того чтобы использовать ui-файл внутри программы на Python, следует воспользоваться модулем uic, который входит в состав библиотеки PyQt. Именно поэтому на первом шаге его необходимо подключить к нашей программе с помощью инструкции:

```
from PyQt5 import QtWidgets, uic
```

Кроме того, следует подключить системную библиотеку `sys`:

```
import sys
```

Каждое приложение PyQt5 должно создать свой объект приложения (или экземпляр класса `QApplication`). Поэтому в следующей строке мы создаем объект «приложение» — `app`:

```
app = QtWidgets.QApplication(sys.argv)
```

Для загрузки созданного ранее ui-файла предназначена функция `loadUi()`. Для этого используем переменную `window`:

```
window = uic.loadUi("MyForm.ui")
```

Далее отображаем на экране созданную в дизайнере форму:

```
window.show()
```

Метод `show()` отображает виджет на экране. Виджет сначала создается в памяти и только потом (с помощью метода `show()`) показывается на экране.

Наконец, мы попадаем в основной цикл приложения:

```
sys.exit(app.exec_())
```

Обработка событий начинается с этой точки. Основной цикл получает события от оконной системы и распределяет их по виджетам приложения. Основной цикл заканчивается, если мы вызываем метод `exit()` или уничтожаем главный виджет. Метод `sys.exit()` гарантирует чистый выход. Метод `exec_()` имеет подчеркивание, поскольку `exec` является ключевым словом в Python.

Итак, все готово для запуска приведенного в листинге 2.1 программного кода (рис. 2.21).

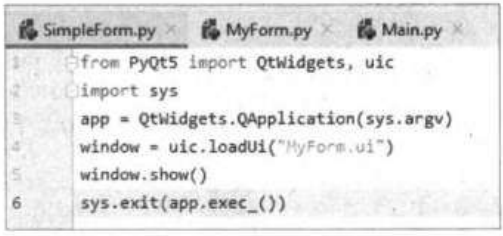


Рис. 2.21. Программа SimpleForm.py в среде PyCharm

Запускаем этот программный модуль с использованием команды **Run** (рис. 2.22). Итог работы программы представлен на рис. 2.23.

В результате мы увидели ту пользовательскую форму, которая была сформи́рована в программной оболочке Qt Designer.

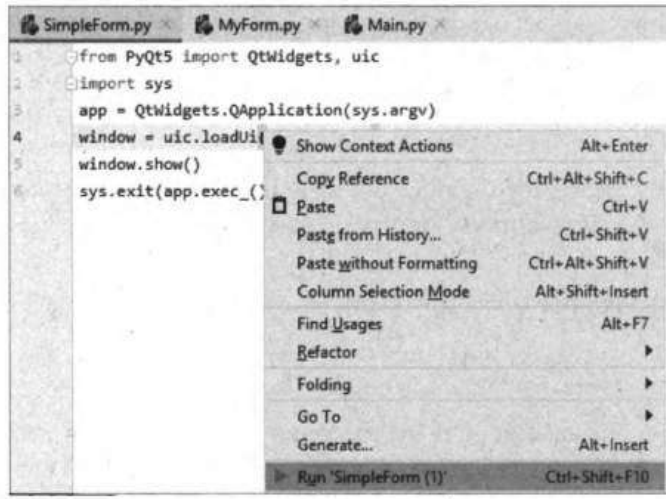


Рис. 2.22. Запуск программы SimpleForm.py в среде PyCharm

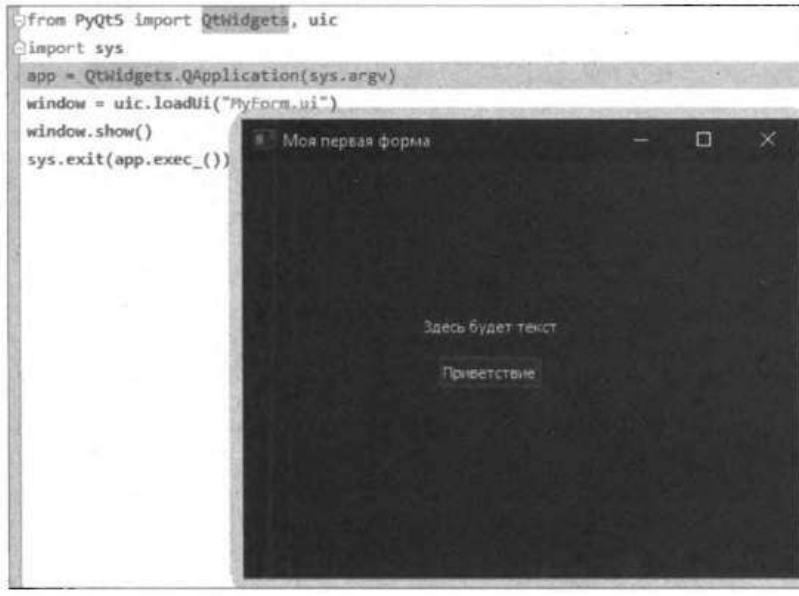


Рис. 2.23. Отображение экранной формы, содержащейся в файле MyForm.ui

На этой форме нажатие кнопки не вызовет каких-либо действий, поскольку еще не написан обработчик события нажатия кнопки. Однако если нажать значок крестика в правом верхнем углу формы, то программа завершит свою работу и пользовательское окно будет закрыто.

Существует еще один способ подключения ui-файла к программе на Python — это преобразовать ui-файл в py-файл. Иными словами, вместо подключения ui-файла можно сгенерировать на его основе программный код на языке Python. Для этого служит утилита pyuic5.bat. Проще всего это сделать через интерфейс PyCharm, в ко-

тором создано виртуальное окружение для каждого проекта, и все необходимые библиотеки имеются в этом окружении. Для выполнения такой конвертации достаточно открыть терминал в PyCharm и там ввести команду:

```
pyuic5 file.ui -o file.py
```

где *file.ui* — имя исходного файла с формой, созданной в модуле Qt Designer; *file.py* — имя сгенерированного программного модуля той же формы на языке Python.

Но перед этим необходимо загрузить библиотеку PyQt5-stubs следующей командой:

```
pip install PyQt5-stubs==5.14.2.2
```

Выполним это преобразование, для чего скопируем файл MyForm.ui в каталог нашего проекта и в окне терминала PyCharm отдадим команду (рис. 2.24):

```
pyuic5 MyForm.ui -o MyForm.py
```

По завершении этого процесса появится новый файл с именем MyForm.py (рис. 2.25).

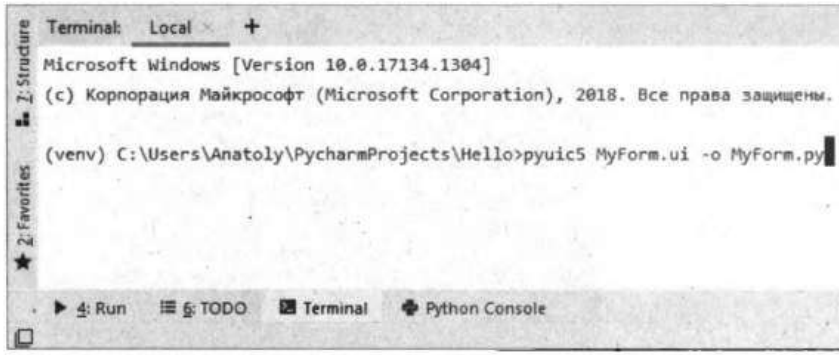


Рис. 2.24. Преобразование файла MyForm.ui в файл MyForm.py

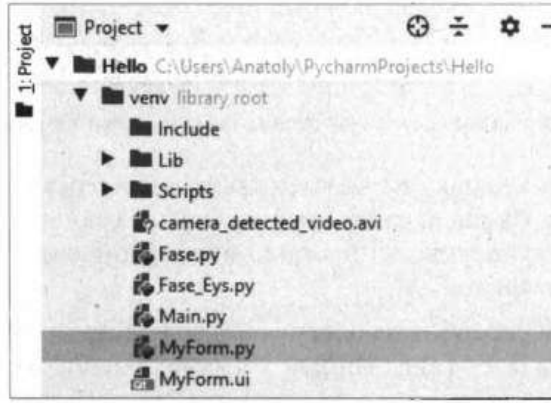


Рис. 2.25. Файл MyForm.py, созданный из файла MyForm.ui

Теперь, если в PyCharm открыть содержимое файла MyForm.py, мы увидим, что, действительно, файл MyForm.ui формата XML был преобразован в файл MyForm.py на языке Python (рис. 2.26).

```
# -*- coding: utf-8 -*-
# Form implementation generated from reading ui file 'MyForm.ui'
## Created by: PyQt5 UI code generator 5.14.1
## WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MyForm(object):
    def setupUi(self, MyForm):
        MyForm.setObjectName("MyForm")
        MyForm.resize(400, 300)
        self.pushButton = QtWidgets.QPushButton(MyForm)
        self.pushButton.setGeometry(QtCore.QRect(140, 140, 75, 23))
        self.pushButton.setObjectName("pushButton")
        self.label = QtWidgets.QLabel(MyForm)
        self.label.setGeometry(QtCore.QRect(130, 110, 111, 16))
        self.label.setObjectName("label")

        self.retranslateUi(MyForm)
        QtCore.QMetaObject.connectSlotsByName(MyForm)

    def retranslateUi(self, MyForm):
        _translate = QtCore.QCoreApplication.translate
        MyForm.setWindowTitle(_translate("MyForm", "Моя первая форма"))
        self.pushButton.setText(_translate("MyForm", "Приветствие"))
        self.label.setText(_translate("MyForm", "Здесь будет текст"))
```

Рис. 2.26. Содержимое файла MyForm.py на языке Python

Как использовать этот программный код в своих приложениях, будет показано в последующих главах — после того, как мы более детально ознакомимся с объектами и командами языка Python.

2.4. Сборка исполняемого файла на Python под Windows

Когда программа написана и отлажена, она может работать только на том компьютере, где установлен интерпретатор Python. Каждый более или менее опытный разработчик понимает, что без интерпретатора Python-код на других компьютерах просто не запустить. А хотелось бы дать возможность каждому пользователю запускать разработанную программу. Вот здесь к нам на помощь и приходят специальные библиотеки, позволяющие собирать проекты в исполняемые exe-файлы, которые можно без проблем запустить на любом компьютере любого пользователя. В этом разделе мы как раз и научимся собирать проекты в исполняемые приложения.

В стандартную библиотеку Python уже входит библиотека tkinter, позволяющая создавать GUI-приложения с интерфейсом пользователя. Но проблема tkinter заключается в том, что ей в технической литературе уделено мало внимания, и обучающий курс, книгу или иную документацию по ней довольно-таки сложно найти. Поэтому для написания интерфейса пользователя мы использовали более мощную, современную и функциональную библиотеку Qt, которая имеет привязки к языку программирования Python в виде библиотеки PyQt5.

Существует большое количество библиотек, позволяющих скомпилировать исполняемый exe-файл, среди которых самые популярные: cx_Freeze, py2exe, nuitka, PyInstaller и др. Про каждую написано довольно много. Однако надо сказать, что многие из этих решений позволяют запускать код на компьютере только с предустановленными интерпретатором и PyQt5. Вряд ли пользователь будет ставить себе дополнительные библиотеки и пакеты, чтобы запустить на своем компьютере ваш программный модуль. Запуск программы на компьютере программиста и у конечного пользователя не одно и то же.

Помочь в этом плане может пакет Pyinstaller, позволяющий собрать полностью работоспособное Python-приложение и все зависимости в один пакет. Такое приложение пользователь сможет запускать без установки интерпретатора Python или каких-либо модулей. Модуль Pyinstaller поддерживает Python 2.7, Python 3.3 и более поздние версии, а также такие библиотеки, как numpy, PyQt, Django, wxPython и многие другие.

К тому же после сборки приложение «весит» всего несколько десятков мегабайтов. Это, к слову, и является преимуществом Pyinstaller, поскольку он не собирает все подряд, а соединяет только необходимое для работы конечного приложения. Аналогичные библиотеки выдают результат на сотни мегабайтов.

Модуль Pyinstaller тестировался на Windows, macOS и Linux. Однако, как бы то ни было, это не кросс-платформенный компилятор, — чтобы создать приложение под Windows, нужно делать это на компьютере с операционной системой Windows, приложение под Linux требует наличия именно этой операционной системы и т. д.

Приступаем к сборке приложения. Прежде всего, мы должны установить необходимые библиотеки, а именно — pywin32 и собственно Pyinstaller. Для этого в окне терминала PyCsharm выполните команды:

```
pip install pypiwin32
pip install pyinstaller
```

Проверить успешность установки пакета pypiwin32 можно в окне настроек проекта, вызвав его через главное меню **File | Settings | Project Interpreter** (рис. 2.27).

Другой способ убедиться, что все установилось нормально, — это ввести следующую команду в окне терминала PyCharm:

```
pip show pyinstaller
```

В этом случае в окне терминала будут показаны версия и другие сведения об установленном пакете — в нашем случае о pyinstaller (рис. 2.28).

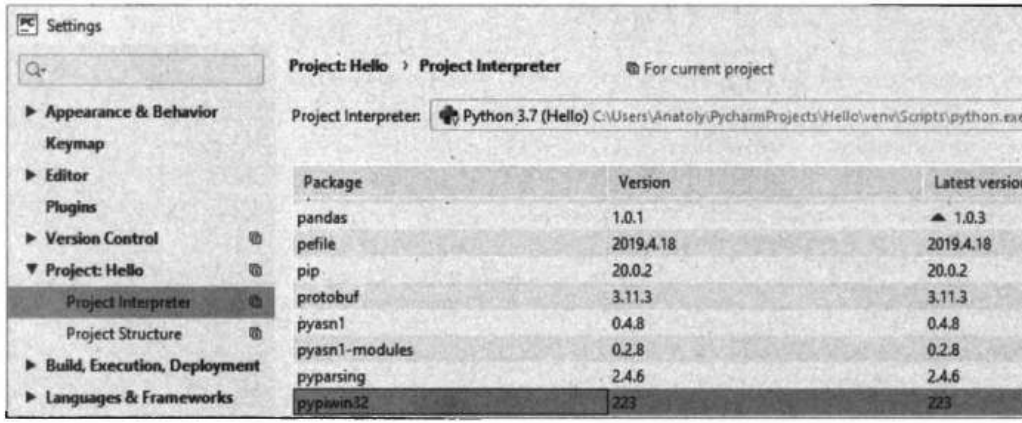


Рис. 2.27. Контроль корректности установки пакета pypiwin32 через интерфейс PyCharm

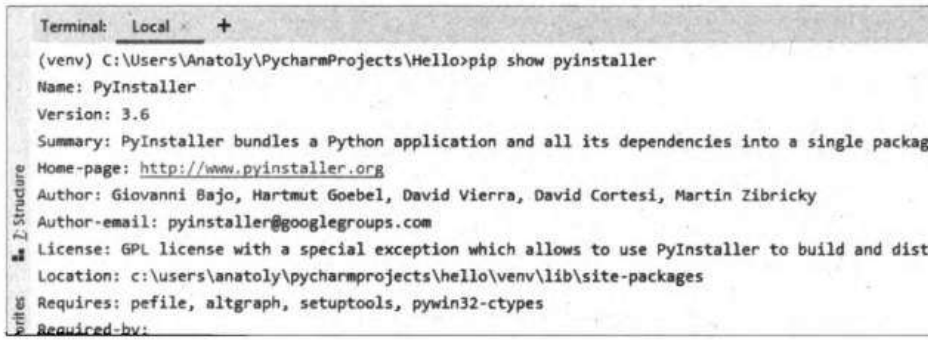


Рис. 2.28. Контроль корректности установки пакета pyinstaller через окно терминала PyCharm

Команда `pyinstaller` имеет следующий синтаксис:

```
pyinstaller [options] script [script ...] | specfile
```

Наиболее часто используемые опции (параметр `options`):

- `--onefile` — сборка в один файл, т. е. файлы .dll не пишутся;
- `--windowed` — при запуске приложения будет появляться консоль;
- `--noconsole` — при запуске приложения консоль появляться не будет;
- `--icon=app.ico` — добавление значка в окно;
- `--paths` — дает возможность вручную прописать путь к необходимым файлам, если `pyinstaller` не может их найти (например, D:\python35\Lib\site-packages\PyQt5\Qt\bin).

Для того чтобы скомпилировать программу с нашей первой пользовательской формой MyForm.py, нужно войти в каталог нашего проекта и в окне терминала PyCharm набрать команду:

```
pyinstaller --onefile --windowed SimpleForm.py
```

Собственно, это и есть простейшая команда, которая соберет наш проект (рис. 2.29).

```
Terminal: Local +

(venv) C:\Users\Anatoly\PycharmProjects\Hello>pyinstaller --onefile --windowed SimpleForm.py
98 INFO: PyInstaller: 3.6
99 INFO: Python: 3.7.7
99 INFO: Platform: Windows-10-10.0.17134-SP0
100 INFO: wrote C:\Users\Anatoly\PycharmProjects\Hello\SimpleForm.spec
105 INFO: UPX is not available.
112 INFO: Extending PYTHONPATH with paths
['C:\\Users\\Anatoly\\PycharmProjects\\Hello',
 'C:\\Users\\Anatoly\\PycharmProjects\\Hello']
113 INFO: checking Analysis
146 INFO: checking PYZ
166 INFO: checking PKG
167 INFO: Building because toc changed
167 INFO: Building PKG (CArchive) PKG-00.pkg
13614 INFO: Building PKG (CArchive) PKG-00.pkg completed successfully.
13620 INFO: Bootloader c:\users\anatoly\pycharmprojects\hello\venv\lib\site-packages\PyInstaller\bootloader\Windows-64bit\runw.exe
13620 INFO: checking EXE
13621 INFO: Rebuilding EXE-00.toc because SimpleForm.exe missing
13621 INFO: Building EXE from EXE-00.toc
13622 INFO: Appending archive to EXE C:\Users\Anatoly\PycharmProjects\Hello\dist\SimpleForm.exe
13690 INFO: Building EXE from EXE-00.toc completed successfully.
```

Рис. 2.29. Компиляция программы SimpleForm.py пакетом pyinstaller в окне терминала PyCharm

Как работает пакет PyInstaller? Он анализирует файл SimpleForm.py и делает следующее:

1. Пишет файл SimpleForm.spec в той же папке, где находится скрипт.
2. Создает папку build в той же папке, где находится скрипт.
3. Записывает некоторые рабочие файлы в папку build.
4. Создает папку dist в той же папке, где находится скрипт.
5. Пишет исполняемый файл в папку dist.

В итоге после работы программы вы найдете две папки: dist и build. Собственно, в папке dist и находится наше приложение (рис. 2.30).

Впоследствии папку build можно спокойно удалить — она не влияет на работоспособность приложения.

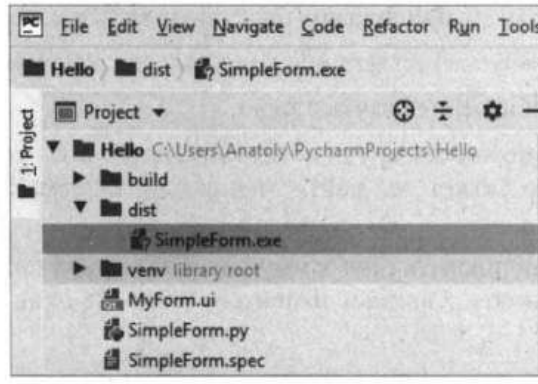


Рис. 2.30. Исполняемый файл SimpleForm.exe, созданный пакетом pyinstaller в окне терминала PyCharm

2.5. Базовые конструкции языка Python

Нейронные сети создают и обучают в основном на языке Python. Поэтому очень важно иметь базовые представления о том, как писать на нем программы. В этом разделе мы остановимся только на основных операторах и функциях языка Python, т. е. тех, которые нам понадобятся для подключения различных пакетов и библиотек, организации циклов и разветвлений, создания объектов, вызова функций и вывода итогов работы программ. Этого будет вполне достаточно для того, чтобы с минимальными трудозатратами на написание программного кода создать нужную функциональность, реализованную в уже готовых библиотеках и пакетах, созданных сообществом программистов на Python.

Материал, представленный в следующих разделах, рассчитан на читателей, только начинающих знакомиться с языками программирования.

2.5.1. Переменные

Переменная — ключевое понятие в любом языке программирования. Проще всего представить переменную в виде коробки с ярлыком. В этой коробке хранится что-то (число, матрица, объект и т. п.). Например, мы хотим создать две переменные: `x` и `y`, которые должны хранить значения 2 и 3. На Python код создания этих переменных будет выглядеть так:

```
x=2
y=3
```

В левой части выражений мы объявляем переменные с именами `x` и `y`. Это равносильно тому, что мы взяли две коробки и приклеили на них именные ярлыки. Далее идет знак равенства и числа `2` и `3`. Знак равенства здесь играет необычную роль. Он *не означает*, что «`x` равно `2`, а `y` равно `3`». Равенство в этом случае означает, что в коробку с именем `x` положили два предмета, а в коробку с именем `y` — три предмета (рис. 2.31).

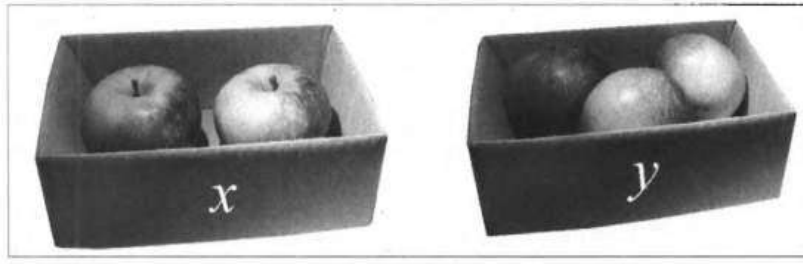


Рис. 2.31. Именные хранилища переменных `x` и `y`

Если говорить более корректно, то мы *присваиваем* переменной `x` число `2`, а переменной `y` — число `3`. Теперь в программном коде мы можем обращаться к этим переменным и выполнять с ними различные действия. Можно, например, просто вывести значения этих переменных на экран. Программный код в таком случае будет выглядеть следующим образом:

```
x=2
y=3
print(x, y)
```

Здесь команда `print(x, y)` представляет собой *вызов функции*. Мы будем рассматривать функции далее. Сейчас же важно, что эта функция выводит в терминальное окно то, что расположено между скобками. Между скобками у нас стоят переменные `x`, `y`. Ранее мы переменной `x` присвоили значение `2`, а переменной `y` — значение `3`. Значит, именно эти значения и будут выведены в окне терминала, если вы выполните нашу программу (рис. 2.32).

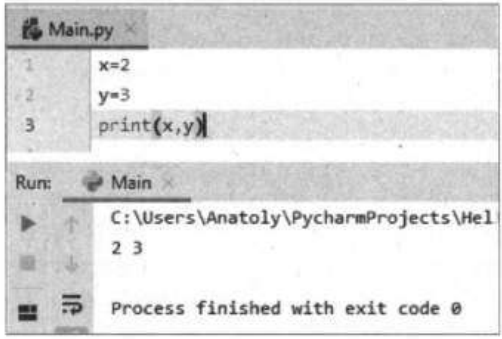


Рис. 2.32. Вывод значений переменных `x` и `y` в окне терминала PyCharm

С переменными, которые хранят числа, можно выполнять различные простейшие действия: складывать, вычитать, умножать, делить и возводить в степень. Текст программы выполнения этих действий приведен в листинге 2.2.

Листинг 2.2

```
x = 2
y = 3
# Сложение
z = x + y
print(z)  # Результат 5
# Вычитание
z = x - y
print(z)  # Результат -1
# Умножение
z = x * y
print(z)  # Результат 6
# Деление
z = x / y
print(z)  # Результат 0.66666...
# Возведение в степень
z = x ** y
print(z)  # Результат 8
```

В этой программе содержатся и *невыполняемые команды*, т. е. текстовая информация, которая позволяет вставить в программу некоторые *комментарии*, облегчающие обозначение смысла тех или иных команд и выполняемых в программе действий. В языке Python можно закомментировать любую строку знаком #. Эти строки *не выполняются* при работе программ — они служат только для облечения понимания смысла фрагментов программного кода. Выполнив программу из листинга 2.2, мы получим результат, показанный на рис. 2.33.

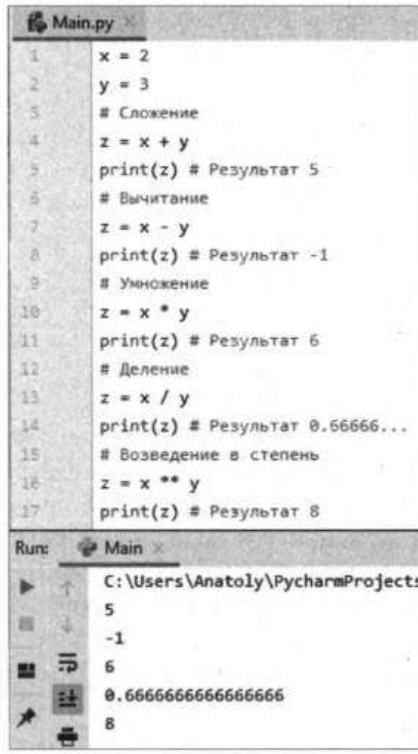


Рис. 2.33. Вывод значений различных действий с переменными x и y в окне терминала PyCharm

В этом программном коде мы вначале создаем две переменные, содержащие значения 2 и 3. Затем создаем переменную z, в которую заносим результат операции с переменными x, y, и выводим значение переменной z в консоль. На приведенном примере хорошо видно, что переменная может многократно менять свое значение в ходе выполнения программы. Так, наша переменная z меняет свое значение 5 раз.

2.5.2. Функции

Функция — это программный код, в котором запрограммированы некоторые часто повторяющиеся действия. Различают системные функции и функции пользователя. К *системным функциям* относятся те, которые уже имеются в структуре языка программирования. В примере, приведенном в листинге 2.2, мы использовали системную функцию `print()`. Кроме системных функций, программист имеет возможность написать свои *пользовательские функции* и многократно обращаться к ним из

любой точки программного кода. Их основное назначение заключается в том, чтобы избежать многократного повторения одного и того же программного кода. Задается функция с помощью ключевого слова `def`, далее следует название функции, затем скобки и двоеточие. В скобках через запятую можно указать параметры, которые будет принимать функция. Далее с отступом надо задать действия, которые будут выполнены при вызове функции. Если функция возвращает результат вычислений, то последней ее строкой будет оператор `return` с указанием возвращаемого результата. Для вызова функции нужно указать ее имя и в скобках записать передаваемые в нее аргументы. В листинге 2.3 приведен пример программного кода описания и вызова функции.

Листинг 2.3

```
def NameFun(p1, p2):
    s = p1 + p2
    return s

a1 = 1
a2 = 2
Itog = NameFun(a1, a2)
```

В приведенном коде `a1` и `a2` — это аргументы, передаваемые в функцию `NameFun()`. А в самой функции `NameFun()` переменные `p1` и `p2` — это принимаемые параметры. Другими словами, перéменные, которые мы передаем функции при ее вызове, называются *аргументами*. А вот внутри функции эти переданные переменные называются *параметрами*. По сути, это два названия одного и того же, но путать их не стоит. Кроме того, следует иметь в виду, что функция может либо иметь, либо не иметь параметров. Может возвращать, а может и не возвращать результаты своих действий. Если функция не возвращает результаты, то в ней будет отсутствовать последний оператор `return`.

В примере, приведенном в листинге 2.2, мы уже использовали функцию `print()`. Она, как отмечалось ранее, относится к системным функциям, которые уже встроены в язык программирования. Мы самостоятельно не писали программный код, который обеспечивает вывод данных, а просто обратились к готовому программному коду. Вернемся к этому примеру (листинг 2.4).

Листинг 2.4

```
x = 2
y = 3
# Сложение
z = x + y
print(z) # Результат 5
# Вычитание
z = x - y
print(z) # Результат -1
```

```
# Умножение
z = x * y
print(z) # Результат 6
```

Здесь мы многократно меняем значение переменной z, а затем передаем это значение в функцию print() в виде аргумента, — т. е. пишем print(z). В результате функция print() многократно выводит пользователю рассчитанное значение переменной z. Сам по себе программный код функции не выполняется до тех пор, пока не будет вызван на исполнение с помощью специальной команды.

Для того чтобы выдавать более осмысленные результаты вычислений, вернемся к программному коду предыдущего примера (листинг 2.5).

Листинг 2.5

```
x = 2
y = 3

# Сложение
z = x + y
print('Мы выполнили сложение переменных X и Y')
print("Результат Z=", z) # Результат 5

# Вычитание
z = x - y
print('Мы выполнили вычитание переменных X и Y')
print("Результат Z=", z) # Результат -1

# Умножение
z = x * y
print('Мы выполнили умножение переменных X и Y')
print("Результат Z=", z) # Результат 6

# Деление
z = x / y
print('Мы выполнили деление переменных X и Y')
print("Результат Z=", z) # Результат 0.6

# Возведение в степень
z = x ** y
print('Мы выполнили возведение в степень переменных X и Y')
print("Результат Z=", z) # Результат 8
```

Как можно здесь видеть, после каждого действия над переменными x и y повторяется один и тот же программный код вызова функции print() с разными аргументами. Значит, имеет смысл создать собственную функцию, в которой будут выполняться эти действия.

Создадим такую функцию с именем `MyPrint` (листинг 2.6).

Листинг 2.6

```
def MyPrint(d, r):
    print('В функции мы выполнили печать ', d, 'переменных X и Y')
    print('Результат Z=', r)
```

Эта функция принимает два параметра: `d` и `r`, и в следующих двух строках выполняется вывод значений этих двух параметров с помощью команды `print()`. Обратиться к функции можно, указав ее имя с указанием в скобках тех аргументов, которые нужно передать в функцию. В листинге 2.7 приведен фрагмент программы обращения к функции `MyPrint()`.

Листинг 2.7

```
x = 2
y = 3
z = x + y
MyPrint('сложения', z)
z = x - y
MyPrint('вычитания', z)
z = x * y
MyPrint('умножения', z)
z = x / y
MyPrint('деления', z)
z = x ** y
MyPrint('возведения в степень', z)
```

Объединим код листингов 2.6 и 2.7 в один модуль. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_7_1) в сопровождающем книгу файловом архиве. Его можно скачать с сервера издательства «БХВ» по ссылке: **https://zip.bhv.ru/9785977518185.zip**, а также со страницы книги на сайте **https://bhv.ru** (см. *приложение*). Полный листинг этой программы и результаты работы функции `MyPrint()` представлены на рис. 2.34. Как можно здесь увидеть, программный код получился короче, а отображение итогов работы программы более понятно конечному пользователю.

Функция может не иметь параметров — в этом случае после ее имени в скобках ничего не указывается. Например, если в созданную нами функцию не нужно было бы передавать параметры, то ее заголовок выглядел бы так: `def MyPrint()`.

Написанная нами функция `MyPrint()` выполняет только вывод информации на печать и не возвращает никаких значений. Просто повторять в функции некоторые действия, конечно, удобно. Но это еще не все. Иногда требуется не только передать в функцию значения переменных, но и вернуть из нее результаты сделанных там операций. Таким образом, функция может не только принимать данные и использовать их в процессе выполнения команд, но и возвращать результат.

Рис. 2.34. Вывод значений различных действий с переменными x и y с использованием функции

Напишем простую функцию, которая складывает два переданных ей числа и возвращает результат (листинг 2.8).

Листинг 2.8

```
def f_sum(a, b):
    result = a + b
    return result
```

Первая строка выглядит почти так же, как и для обычных функций. Между скобок записаны две переменные: `a` и `b` — это *параметры функции*. Наша функция имеет два параметра (т. е. принимает значения двух переменных). Параметры можно использовать внутри функции как обычные переменные. Во второй строке кода мы создаем переменную `result`, которая равна сумме параметров `a` и `b`. В третьей строке мы возвращаем значение переменной `result`.

Теперь в программном коде мы можем писать обращение к этой функции (листинг 2.9).

Листинг 2.9

```
s = f_sum(2, 3)
print(s)
```

Мы вызываем здесь функцию `f_sum()` и передаем ей *два аргумента*: `2` и `3`. Аргумент `2` становится значением переменной `a`, а аргумент `3` — значением переменной `b`. Наша функция складывает переменные (`a + b`), присваивает итог этой операции переменной `result` и возвращает рассчитанное значение в точку вызова. То есть в точке вызова переменной `s` будет присвоен результат работы функции, который оператором `print()` будет выведен пользователю. Объединим код листингов 2.8 и 2.9. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_9_1) в сопровождающем книгу файловом архиве (см. *приложение*). Выполним этот программный код и посмотрим на результат (рис. 2.35).

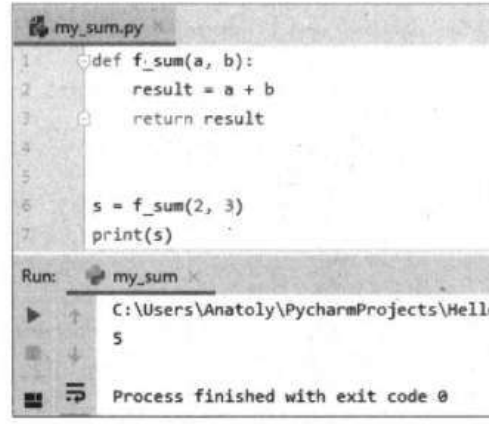


Рис. 2.35. Результаты работы функции `f_sum()`

В языке Python важно строго соблюдать порядок описания и обращения к функции. Сначала функция должна быть описана оператором `def`, и только после этого можно к ней обращаться. Если обращение к функции будет выполнено до ее описания, то вы получите сообщение об ошибке.

2.5.3. Массивы (списки)

Если переменную можно представлять как коробку, которая что-то хранит (не обязательно число), то *массивы* можно представить в виде шкафа со множеством полок. На рис. 2.36 представлен такой шкаф с шестью полками. Шкаф (в нашем случае — массив) имеет имя `array`, а каждая полка — свой номер от 0 до 5 (нумерация полок начинается с нуля).

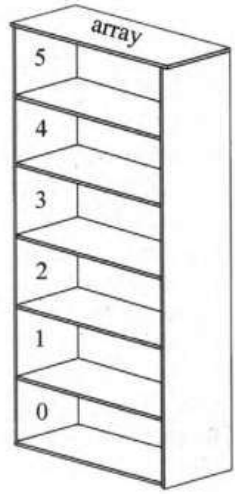


Рис. 2.36. Представление массива из шести элементов

Положим на все эти полки различную информацию. На языке Python это будет выглядеть следующим образом:

```
array = [10, 11, 12, 13, 14, "Это текст"]
```

В отличие от других языков программирования, в Python в качестве термина для упорядоченной коллекции элементов используется термин не «массив», а «список». Списки в Python — упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы данных в списке могут отличаться). В рассматриваемом случае наш массив, а вернее — список, содержит разные элементы: пять чисел и текстовую строку. Попробуем вывести какой-нибудь элемент массива:

```
array = [10, 11, 12, 13, 14, "Это текст"]
print(array[1])
```

В консоли будет выведено число 11. Но почему 11, а не 10? Все дело в том, что в Python, как и во многих других языках программирования, нумерация массивов *начинается с 0*. Поэтому `array[1]` выдает нам *второй* элемент массива, а не первый. Для вызова первого надо было написать `array[0]`.

Если выполнить следующий программный код:

```
array = [10, 11, 12, 13, 14, "Это текст"]
print(array[5])
```

в консоли будет выведено `Это текст`.

Иногда бывает очень полезно получить *количество элементов* в массиве. Для этого можно использовать функцию `len()`. Она подсчитает количество элементов и вернет их число:

```
array = [10, 11, 12, 13, 14, "Это текст"]
print(len(array))
```

В консоли выведется число `6`.

2.5.4. Условия и циклы

По умолчанию любые программы выполняют все команды подряд — от первой строки до последнего оператора. Но есть ситуации, когда необходимо проверить какое-то условие, и в зависимости от того, правдиво оно или нет, выполнить разные действия. Кроме того, часто возникает необходимость много раз повторить практически одинаковую последовательность команд. В первой ситуации помогают операторы, обеспечивающие выполнение различных фрагментов программы по результатам соблюдения или несоблюдения некоторого условия, а во второй — организация многократно повторяющихся действий.

Условия

Условия (или структуры принятия решений) нужны для того, чтобы выполнить два разных набора действий в зависимости от того, истинно или ложно проверяемое утверждение (рис. 2.37).

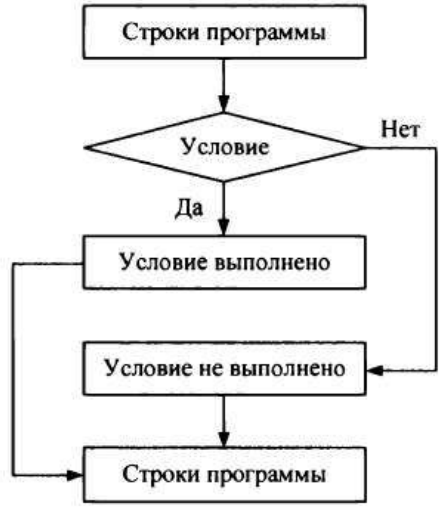


Рис. 2.37. Условный переход к разным частям программного кода

В Python условия можно записывать с помощью инструкции `if`:

```
if: <программный_код_1> else: <программный_код_2>
```

Пусть у нас есть некоторая переменная `x`, которой в ходе работы программы было присвоено некоторое значение. Если `x` меньше 10, то мы делим его на 2. Если же `x` больше или равно 10, то мы умножаем его на 2. В листинге 2.10 показано, как будет выглядеть программный код при `x=8`.

Листинг 2.10

```
x = 8
if x < 10:
    x = x / 2
```

```
else:
    x = x * 2
print(x)
```

А результатом работы программы будет значение x=4.

Теперь изменим этот программный код и зададим значение x=12 (листинг 2.11).

Листинг 2.11

```
x = 12
if x < 10:
    x = x / 2
else:
    x = x * 2
print(x)
```

В этом случае результатом работы программы будет значение x=24.

Рассмотрим этот программный код. После создания переменной x и присвоения ей некоторого значения записывается условие. Начинается все с ключевого слова if (в переводе с английского «если»). В скобках мы указываем проверяемое выражение. В нашем случае мы проверяем, действительно ли переменная x меньше 10. Если она меньше 10, то мы делим ее на 2. Затем идет ключевое слово else, после которого начинается блок действий, которые будут выполнены, если выражение в скобках после if ложное. Если значение переменной x больше или равно 10, то мы умножаем эту переменную на 2. Последним оператором выводим значение x в консоль.

Циклы

Циклы нужны для многократного повторения действий. Предположим, мы хотим вывести таблицу квадратов первых 10 натуральных чисел. Это можно сделать так, как представлено в листинге 2.12.

Листинг 2.12

```
print("Квадрат 1 равен " + str(1**2))
print("Квадрат 2 равен " + str(2**2))
print("Квадрат 3 равен " + str(3**2))
print("Квадрат 4 равен " + str(4**2))
print("Квадрат 5 равен " + str(5**2))
print("Квадрат 6 равен " + str(6**2))
print("Квадрат 7 равен " + str(7**2))
print("Квадрат 8 равен " + str(8**2))
print("Квадрат 9 равен " + str(9**2))
print("Квадрат 10 равен " + str(10**2))
```

Здесь в каждой строке программного кода мы формируем текстовую строку "Квадрат ... равен " и добавляем к ней знаком + новую строку: str(1**2). В ней используется функция преобразования чисел в текстовую информацию: str. В скобках этой функции мы возводим число в квадрат. И таких строк у нас 10. Если мы запустим эту программу на выполнение, то получим следующий результат (рис. 2.38).

Рис. 2.38. Результат работы программы возведения в квадрат 10 натуральных чисел

А что если нам надо вывести квадраты первых 100 или 1000 чисел? Неужели нужно будет писать 100 или даже 1000 строк программного кода. Совсем нет, именно для таких случаев и существуют циклы. Всего в Python два вида циклов: while и for. Разберемся с ними по очереди.

Цикл while повторяет необходимые команды до тех пор, пока не остается истинным некоторое условие. В листинге 2.13 показано, как тот же самый программный код будет выглядеть с использованием цикла while.

Листинг 2.13

```
x = 1
while x <= 100:
    print("Квадрат числа " + str(x) + " равен " + str(x**2))
    x = x + 1
```

Сначала мы создаем переменную x и присваиваем ей значение 1. Затем создаем цикл while и проверяем, меньше или равен 100 наш x. Если меньше или равен, то выполняем два действия:

- возводим x в квадрат;
- увеличиваем x на 1.

После выполнения второй команды программа возвращается к проверке условия. Если условие снова истинно, мы опять выполняем эти два действия. И так до тех пор, пока x не станет равным 101. Тогда условие вернет значение ложь, и цикл больше не будет выполняться. Результат работы программы возведения в квадрат 100 натуральных чисел с использованием цикла while приведен на рис. 2.39.

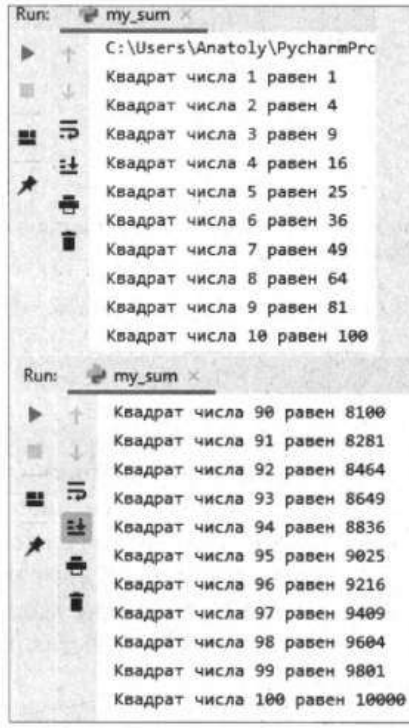


Рис. 2.39. Результат работы программы возведения в квадрат 100 натуральных чисел с использованием цикла `while`

Цикл `for` предназначен для того, чтобы перебирать массивы. Запишем тот же пример с формированием таблицы квадратов первой сотни натуральных чисел, но уже через цикл `for` (листинг 2.14).

Листинг 2.14

```
for x in range(1, 101):
    print("Квадрат числа " + str(x) + " равен " + str(x**2))
```

Здесь в первой строке мы используем ключевое слово `for` для создания цикла. Далее мы указываем, что хотим повторить определенные действия для всех `x` в диапазоне от `1` до `100`. Функция `range(1, 101)` создает массив из 100 чисел, начиная с 1 и заканчивая 100.

Вот еще пример перебора массива с помощью цикла `for` (листинг 2.15).

Листинг 2.15

```
for i in [1, 10, 100, 1000]:
    print(i * 2)
```

Этот код выводит четыре числа: `2`, `20`, `200` и `2000`, каждое из которых получено умножением на `2` исходных чисел. Тут наглядно видно, что в первой строке программы `for` последовательно перебираются элементы массива, а во второй строке

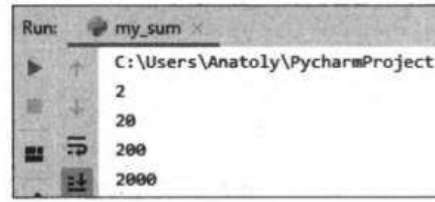


Рис. 2.40. Результат работы программы выполнения действий с элементами массива с использованием цикла `for`

эти элементы умножаются на 2. Результат работы программы представлен на рис. 2.40.

2.5.5. Классы и объекты

В реальной жизни мы оперируем не переменными или функциями, а *объектами*. Светофор, автомобиль, пешеход, кошка, собака, самолет — это все объекты. Рассмотрим более подробно такой объект, как кошка. Абсолютно все кошки обладают некоторыми характерными *свойствами*, или параметрами. К ним можно отнести наличие хвоста, четырех лап, шерсти, усов и т. п. Но это еще не все. Помимо определенных внешних параметров, кошка может выполнять свойственные ей *действия*: мурлыкать, шипеть, царапаться, кусаться.

Теперь возьмем такой объект, как автомобиль. Абсолютно все автомобили имеют кузов, колеса, фары, двигатель, тормоза и т. п. Помимо этих внешних параметров, автомобиль может выполнять свойственные ему действия: двигаться, тормозить, издавать звуковой сигнал, выпускать выхлопные газы и т. п.

Только что мы словесно представили некие общие свойства всех кошек и всех автомобилей в целом, т. е. описали два класса: класс `Кошка` и класс `Автомобиль`. Подобное описание характерных свойств и действий какого-либо объекта на различных языках программирования и называется классом. *Класс* — просто набор переменных и функций, которые описывают какой-то объект.

Если все кошки (как класс) имеют общую характеристику — наличие шерсти, то каждая конкретная кошка (как объект) будет иметь свойственную только ей длину шерсти и ее окрас. То же самое касается и автомобиля: все автомобили (как класс) имеют кузов, но каждый конкретный автомобиль (как объект) имеет свою форму кузова, свой цвет. Все автомобили имеют колеса, но на каждом конкретном автомобиле стоят шины определенного размера и завода-изготовителя. Исходя из сказанного, очень важно понимать разницу между классом и конкретным объектом этого класса. Класс — это некая схема, которая описывает объект *в целом*. Объект класса — это, если можно так сказать, материальное воплощение некого конкретного элемента из этого класса. Класс `Кошка` — это описание ее свойств и действий, он всегда только один. А объектов-кошек может быть великое множество. По аналогии класс `Автомобиль` — это описание свойств и действий автомобиля, такой класс всегда только один. А объектов-автомобилей может очень много (рис. 2.41):

Рис. 2.41. Демонстрация различий между классом и объектом класса

Классы

Для создания класса надо написать ключевое слово `class` и затем указать имя этого класса. Создадим класс кошек с именем `Cat`:

```
class Cat:
```

Далее нужно указать действия, которые будет способен совершать этот класс (в нашем случае — возможные действия кошки). Такие действия реализуются в виде функций, которые описываются внутри класса. Функции, прописанные внутри класса, называются *методами*. Словесно мы можем описать следующие методы кошки (действия, которые она может совершать): мурлыкать (`purr`), шипеть (`hiss`), царапаться (`scrabble`). Теперь реализуем эти методы в классе кошки `Cat` на языке Python (листинг 2.16).

Листинг 2.16

```
# Класс кошки
class Cat:
    # Мурлыкать
    def purr(self):
        print("Муррр!")

    # Шипеть
    def hiss(self):
        print("Шшшш!")

    # Царапаться
    def scrabble(self):
        print("Цап-царап!")
```

Все достаточно просто! Мы создали класс с именем `Cat` и внутри него определили три обычные функции с именами `purr`, `hiss` и `scrabble`. Как видно из приведенного в листинге 2.16 программного кода, каждая из трех описанных функций имеет единственный параметр: `self`. Рассмотрим, зачем нужен и что означает этот параметр в функциях Python, описанных внутри класса.

Классам нужен способ, чтобы ссылаться на самих себя. В методах класса первый параметр функции по соглашению именуют `self`, и он представляет собой ссылку на сам объект этого класса. Помещать такой параметр нужно в каждую функцию, чтобы иметь возможность вызвать ее на текущем объекте. Таким образом, `self` заменяет идентификатор объекта. Все наши кошки (объекты `Cat`) способны мурлыкать (имеют метод `purr`). Теперь мы хотим, чтобы замурлыкал наш конкретный объект-кошка, созданный на основе класса `Cat`. Как это сделать? Допустим, мы в Python напишем следующую команду:

```
Cat.purr()
```

Если мы запустим на выполнение эту команду, то будут мурлыкать сразу все коты и кошки на свете. А если мы воспользуемся командой:

```
self.purr ()
```

то будет мурлыкать только та особь, на которую укажет параметр `self`, т. е. только наша кошка.

Как видите, обязательный для любого метода параметр `self` позволяет нам обращаться к методам и переменным самого класса! Без этого аргумента выполнить подобные действия мы бы не смогли.

Кроме того, что кошки могут шипеть, мурлыкать и царапаться, они имеют еще и ряд свойств: рост, массу тела, окрас шерсти, длину усов и пр. Давайте теперь в классе кошек `Cat` определим для них ряд свойств — в частности окрас шерсти, цвет глаз и кличку. А также зададим статический атрибут «наименование класса»: `Name_Class`. Как это сделать? Мы можем создать переменную и занести в нее наименование класса. Кроме того, в *абсолютно любом классе* нужно определить функцию: `__init__()`. Она вызывается всегда, когда мы создаем реальный объект на основе нашего класса (обратите внимание, что здесь используются символы *двойного подчеркивания*). Итак, задаем нашим кошкам наименование класса: `Name_Class`, и три свойства: окрас шерсти — `wool_color`, цвет глаз — `eyes_color`, кличку — `name` (листинг 2.17).

Листинг 2.17

```
class Cat:
    Name_Class = "Кошки"

    # Действия, которые надо выполнять при создании объекта "Кошка"
    def __init__(self, wool_color, eyes_color, name):
        self.wool_color = wool_color
        self.eyes_color = eyes_color
        self.name = name
```

В приведенном здесь методе `__init__()` мы задаем переменные, в которых будут храниться свойства нашей кошки. Как мы это делаем? Обязательно используем параметр `self` — чтобы при создании конкретного объекта `Кошка` сразу (по умолчанию) задать нашей кошке три нужных свойства. Затем мы определяем в этом методе три переменные, отвечающие за окрас шерсти, цвет глаз и кличку. Также мы задаем кошке, например, окрас шерсти. Вот эта строчка:

```
self.wool_color = wool_color
```

В левой части этого выражения мы создаем атрибут для задания цвета шерсти нашей кошки с именем `wool_color`. Этот процесс почти идентичен обычному созданию переменной. Присутствует только одно отличие — приставка `self`, которая указывает на то, что эта переменная относится к классу `Cat`. Затем мы присваиваем этому атрибуту значение. Аналогичным образом мы формируем еще два свойства: цвет глаз и кличку. Программный код описания класса `Cat` теперь будет выглядеть так, как показано в листинге 2.18.

Листинг 2.18

```
class Cat:
    Name_Class = "Кошки"

    # Действия, которые надо выполнять при создании объекта "Кошка"
    def __init__(self, wool_color, eyes_color, name):
        self.wool_color = wool_color
        self.eyes_color = eyes_color
        self.name = name

    # Мурлыкать
    def purr(self):
        print("Мурррр!")

    # Шипеть
    def hiss(self):
        print("Шшшш!")

    # Царапаться
    def scrabble(self):
        print("Цап-царап!")
```

Объекты

Мы создали класс `Кошки`. Теперь создадим на его основе реальный объект — кошку:

```
my_cat = Cat('Цвет шерсти', 'Цвет глаз', 'Кличка')
```

В этой строке мы создаем переменную `my_cat`, а затем присваиваем ей объект класса `Cat`. Выглядит это все так, как вызов некоторой функции: `Cat(...)`. На самом деле так и есть. Этой записью мы вызываем метод `__init__()` класса `Cat`. Функция

__init__() в нашем классе принимает четыре аргумента: сам объект класса — self, который указывать не надо, а также еще три аргумента, которые затем становятся атрибутами нашей кошки.

Получается, что с помощью приведенной строки мы создали реальный объект — нашу собственную кошку. Для этой кошки зададим следующие *атрибуты*, или свойства: белую шерсть, зеленые глаза и кличку Мурка. Давайте выведем эти атрибуты в консоль с помощью программного кода из листинга 2.19.

Листинг 2.19

```
my_cat = Cat('Белая', 'Зеленые', 'Мурка')
print("Наименование класса - ", my_cat.Name_Class)
print("Вот наша кошка:")
print("Цвет шерсти- ", my_cat.wool_color)
print("Цвет глаз- ", my_cat.eyes_color)
print("Кличка- ", my_cat.name)
```

То есть обратиться к атрибутам объекта мы можем, записав имя объекта, поставив точку и указав имя желаемого атрибута. Объединим код листингов 2.18 и 2.19. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_19_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результаты работы этой программы представлены на рис. 2.42.

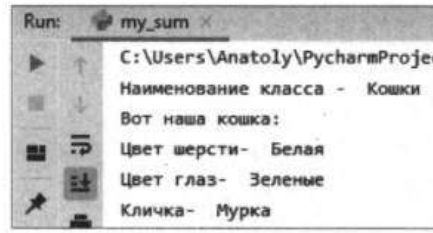


Рис. 2.42. Параметры объекта Кошки (my_cat)

Атрибуты кошки можно менять. Например, давайте сменим кличку нашему питомцу и поменяем цвет (выпустим кошку Ваську черного цвета). Изменим программный код (листинг 2.20).

Листинг 2.20

```
# Это дополнительный код, он не является рабочим
my_cat = Cat('Белая', 'Зеленые', 'Мурка')
my_cat.name = "Васька"
my_cat.wool_color = "Черный"
print("Наименование класса - ", my_cat.Name_Class)
print("Вот наша кошка:")
print("Цвет шерсти- ", my_cat.wool_color)
```

```
print("Цвет глаз- ", my_cat.eyes_color)
print("Кличка- ", my_cat.name)
```

Добавим этот код к листингу 2_18. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_20_1) в сопровождающем книгу файловом архиве (см. *приложение*). В итоге получим следующий результат (рис. 2.43).

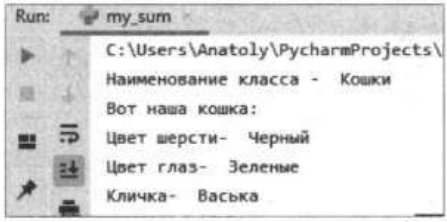


Рис. 2.43. Измененные параметры объекта Кошки (my_cat)

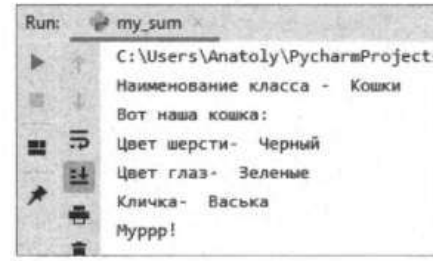


Рис. 2.44. Демонстрация вызова метода объекта Кошки

Теперь вспомним, что в нашем классе Cat запрограммирована возможность выполнять некоторые действия. Попросим нашу кошку Ваську мяукнуть. Добавим в приведенную в листинге 2.20 программу всего одну строчку:

```
my_cat.purr()
```

Добавим эту же строчку кода к листингу 2_20_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_20_2) в сопровождающем книгу файловом архиве (см. *приложение*). Выполнение этой команды выведет в консоль текст **Муррр!** (рис. 2.44).

Как видите, обращаться к методам объекта так же просто, как и обращаться к его атрибутам.

2.5.6. Создание классов и объектов на примере автомобиля

В настоящее время активно развивается беспилотный транспорт — в частности, автомобили. Программные средства, которые управляют беспилотными автомобилями, строят с использованием элементов искусственного интеллекта. К таким элементам относятся нейронные сети, машинное обучение, системы распознавания и идентификации объектов, программные средства приема и обработки большого количества данных, поступающих от различных датчиков и видеокамер, модули привода в действие таких элементов автомобиля, как тормоза, рулевое управление, акселератор и т. п. Исходя из этого, попробуем создать простейший класс Car, описывающий автомобиль (листинг 2.21).

Листинг 2.21

```
class Car(object):
    # Наименование класса
    Name_class = "Автомобиль"
```

```
def __init__(self, brand, weight, power):
    self.brand = brand    # Марка, модель автомобиля
    self.weight = weight  # Вес автомобиля
    self.power = power    # Мощность двигателя

# Метод двигаться прямо
def drive(self):
    # Здесь команды двигаться прямо
    print("Поехали, двигаемся прямо!")

# Метод повернуть направо
def righ(self):
    # Здесь команды повернуть руль направо
    print("Едем, поворачиваем руль направо!")

# Метод повернуть налево
def left(self):
    # Здесь команды повернуть руль налево
    print("Едем, поворачиваем руль налево!")

# Метод тормозить
def brake(self):
    # Здесь команды нажатия на педаль тормоза
    print("Стоп, активируем тормоз")

# Метод подать звуковой сигнал
def beep(self):
    # Здесь команды подачи звукового сигнала
    print("Подан звуковой сигнал")
```

В этом примере мы создали класс `Car` (автомобиль), добавили в него три атрибута и пять методов. Вот атрибуты:

```
self.brand = brand     # Марка, модель автомобиля
self.weight = weight   # Вес автомобиля
self.power = power     # Мощность двигателя
```

Они описывают автомобиль: марка (модель), вес, мощность двигателя. Также у этого класса есть пять методов. Метод описывает, что делает класс, — т. е. автомобиль. В нашем случае автомобиль может двигаться прямо, поворачивать направо, поворачивать налево, подавать звуковой сигнал и останавливаться. Что касается аргумента под названием `self`, то его назначение подробно рассмотрено в предыдущем разделе.

Теперь на основе этого класса создадим объект — автомобиль с именем `MyCar` с атрибутами: Мерседес, вес — 1200 кг, мощность двигателя — 250 лошадиных сил. Выведем параметры созданного объекта (листинг 2.22).

Листинг 2.22

```
# Это дополнительный код, он не является рабочим
MyCar = Car('Мерседес', 1200, 250)
print('Параметры автомобиля, созданного из класса- ', MyCar.Name_class)
print('Марка (модель)- ', MyCar.brand)
print('Вес (кг)- ', MyCar.weight)
print('Мощность двигателя (лс)- ', MyCar.power)
```

Объединим код листингов 2.21 и 2.22 в один модуль. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_22_1) в сопровождающем книгу файловом архиве (см. *приложение*).

Результаты работы этого программного кода представлены на рис. 2.45.

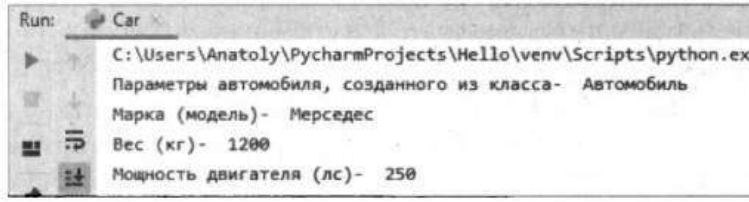


Рис. 2.45. Демонстрация создания объекта `Автомобиль` **—** `MyCar` **из класса** `Car` **и вывода его параметров**

Теперь испытаем созданные в классе `Car` методы и заставим автомобиль двигаться — т. е. обратимся к соответствующим методам созданного нами объекта `MyCar` (листинг 2.23).

Листинг 2.23

```
# Это дополнительный код, он не является рабочим
MyCar.drive()    # Двигается прямо
MyCar.righ()     # Поворачиваем направо
MyCar.drive()    # Двигается прямо
MyCar.left()     # Поворачиваем налево
MyCar.drive()    # Двигается прямо
MyCar.beep()     # Подаем звуковой сигнал
MyCar.brake()    # Тормозим
```

Объединим код листингов 2.22_1 и 2.23 в один модуль. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 2_23_1) в сопровождающем книгу файловом архиве (см. *приложение*).

Результаты работы этого программного кода представлены на рис. 2.46.

Итак, мы познакомились с базовыми понятиями о классах и объектах, создали два класса: `Кошки` и `Автомобиль`, а также заставили эти объекты выполнить некоторые элементарные действия.

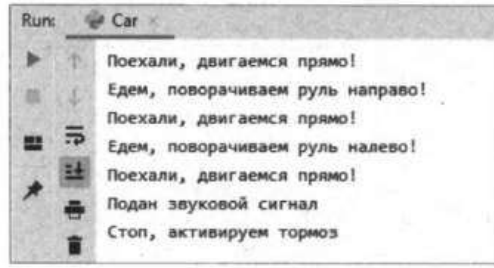


Рис. 2.46. Демонстрация работы методов объекта `Автомобиль` — `MyCar` из класса `Car`

2.5.7. Программные модули

Любой файл с расширением py является модулем. Зачем они нужны? Достаточно много программистов создают приложения с полезными функциями и классами. Другие программисты могут подключать эти сторонние модули и использовать все имеющиеся в них функции и классы, тем самым упрощая себе работу.

Например, вам не нужно тратить время и писать свой программный код для работы с матрицами. Достаточно подключить модуль NumPy и использовать его функции и классы. К настоящему моменту программистами Python написано более 110 тыс. разнообразных модулей. Упоминавшийся ранее модуль NumPy позволяет быстро и удобно работать с матрицами и многомерными массивами. Модуль math предоставляет множество методов для работы с числами: синусы, косинусы, переводы градусов в радианы и пр.

Установка модуля

Интерпретатор Python устанавливается вместе со стандартным набором модулей. В этот набор входит очень большое количество модулей, которые позволяют работать с математическими функциями, веб-запросами, читать и записывать файлы и выполнять другие необходимые действия.

Для того чтобы использовать модуль, который отсутствует в стандартном наборе, необходимо его установить. Для установки модуля в Windows нужно либо нажать сочетание клавиш <Win>+<R>, либо щелкнуть правой кнопкой мыши на кнопке **Пуск** в Windows в левой нижней части экрана и выбрать команду **Выполнить** (рис. 2.47).

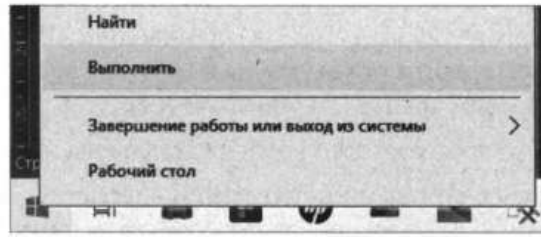


Рис. 2.47. Активирование окна запуска программ на выполнение

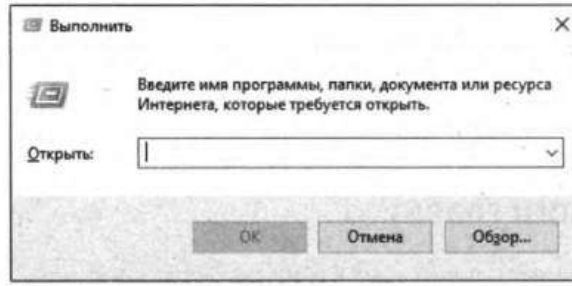


Рис. 2.48. Окно Windows запуска программ на выполнение

В результате этих действий откроется окно Windows запуска программ на выполнение (рис. 2.48).

В текстовом поле **Открыть** введите команду инсталляции модуля:

```
pip install [название_модуля]
```

То же самое можно сделать в окне терминала PyCharm (рис. 2.49).

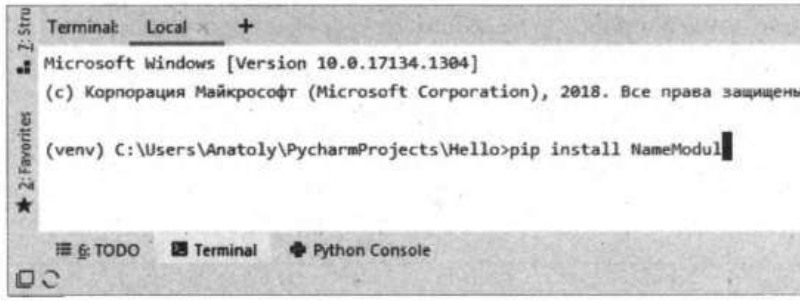


Рис. 2.49. Окно терминала PyCharm для запуска инсталляции модуля

В обоих случаях начнется процесс установки модуля. Когда он завершится, можно использовать установленный модуль в своей программе.

Подключение и использование модуля

Сторонний модуль подключается достаточно просто. Нужно написать всего одну короткую строку программного кода:

```
import [название_модуля]
```

Например, для импорта модуля, позволяющего работать с математическими функциями, надо написать следующее:

```
import math
```

Для обращения к какой-либо функции модуля достаточно написать название модуля, затем поставить точку и указать название функции или класса. Например, вычисление факториала числа 10 будет выглядеть так:

```
math.factorial(10)
```

То есть мы обратились к функции `factorial(a)`, которая определена внутри модуля math. Это удобно, ведь нам не нужно тратить время и вручную создавать функцию, которая считает факториал числа. Можно просто подключить модуль и сразу выполнить необходимое действие.

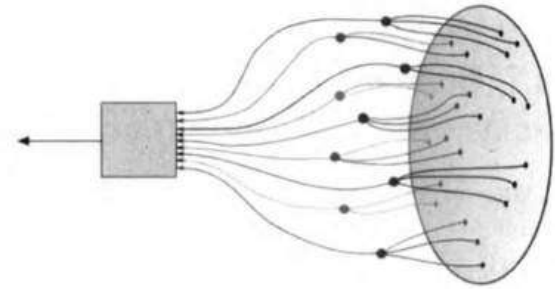
2.6. Краткие итоги главы

Итак, мы установили весь необходимый инструментарий, познакомились с основами языка Python и можем писать элементарные программы. Но прежде чем приступить к разработке элементов искусственного интеллекта с использованием Python, нужно разобраться:

- что такое искусственный интеллект;
- чем работа искусственного интеллекта отличается от мыслительной деятельности человека;
- каким способом можно научить машину мыслить и действовать так, как это может делать человек.

Рассмотрению этих вопросов, т. е. основ искусственного интеллекта, и посвящена следующая глава.

ГЛАВА 3



Элементы искусственного интеллекта

Искусственный интеллект (ИИ — artificial intelligence, AI) — это область информатики, в рамках которой разрабатываются компьютерные программы для решения задач, требующих имитации мыслительной деятельности человека. Такие программы могут делать обобщения и выводы, выявлять взаимосвязи и обучаться с учетом накопленного опыта. Системы искусственного интеллекта ни в коем случае не заменяют человека — они расширяют и дополняют его возможности.

Два ключевых понятия в системах искусственного интеллекта — это нейронные сети и машинное обучение. *Нейронная сеть* по своей сути представляет некую математическую модель и ее программную реализацию, которая в упрощенном виде воссоздает работу биологической нейронной сети человека. А *машинное обучение* — это набор специальных алгоритмов, благодаря которым воплощается ключевое свойство нейронных сетей — способность самообучения на основе получаемых данных. Чем больший объем информации будет представлен в качестве тренировочного массива данных, тем проще обучающим алгоритмам найти закономерности и тем точнее будет выдаваемый результат.

В этой главе в общих чертах будет рассказано о следующих основных элементах искусственного интеллекта:

- базовых понятиях и определениях искусственного интеллекта;
- работе искусственного нейрона как основы нейронных сетей;
- назначении и видах различных функций активации;
- структуре различных типов нейронных сетей;
- основе обучения нейронных сетей;
- видах и назначении обучающих наборов данных;
- понятиях о видах обучения нейронных сетей (обучение с учителем, обучение без учителя).

Итак, приступим к знакомству с основами искусственного интеллекта.

3.1. Основные понятия и определения искусственного интеллекта

Существует несколько видов искусственного интеллекта, среди которых можно выделить три основные категории:

- *ограниченный искусственный интеллект* (artificial narrow intelligence, ANI). Он представляет собой программно-аппаратный комплекс, специализирующийся исключительно в одной конкретной области. Например, компьютерная программа может победить чемпиона мира по шахматам в шахматной партии, но это все, на что она способна;
- *общий искусственный интеллект* (artificial general intelligence, AGI). Он представляет собой программно-аппаратный комплекс, чей интеллект напоминает человеческий, — т. е. он может выполнять все те же задачи, что и человек. Общий искусственный интеллект позволяет копировать мыслительные способности человека: получать данные, выделять из потока данных нужную информацию, сравнивать различные варианты решения задачи, быстро обучаться, использовать накопленный опыт;
- *искусственный суперинтеллект* (artificial superintelligence, ASI). Это интеллект, который превосходит человеческий практически во всех областях, включая научные изобретения, общие познания и социальные навыки.

В настоящее время человечество уже с успехом применяет элементы искусственного интеллекта в различных сферах:

- беспилотные автомобили, которые распознают различные препятствия на своем пути и реагируют на них;
- беспилотные летательные аппараты, которые могут самостоятельно перемещаться по заданному маршруту;
- навигатор, который получает задание движения по маршруту с помощью голосовой команды;
- спам-фильтр в электронной почте, который вначале обучают распознавать спам, а затем, анализируя свой предыдущий опыт и ваши предпочтения, он перемещает письма в специальную папку;
- переводчик — это классический пример применения ИИ, который достаточно хорошо справляется со своей узкой задачей;
- системы распознавания текста, голоса, генерации голоса из текста и т. п.

Остановимся на терминологии: что такое искусственный интеллект, машинное обучение и искусственные нейронные сети? Как они связаны?

Искусственный интеллект (artificial intelligence, AI). Можно сказать, что это наука и технология создания интеллектуальных (умных) машин. Реализуется ИИ в виде программного обеспечения, которое может работать на большом компьютере, на мини-ЭВМ, в смартфоне или ином вычислительном средстве. В любом случае это программно-аппаратный комплекс. Искусственные интеллектуальные системы

выполняют как вычислительные, так и некоторые творческие функции, которые считаются прерогативой человека.

Машинное обучение (machine learning) — подраздел искусственного интеллекта, изучающий различные способы построения обучающихся алгоритмов. Под обучающимися алгоритмами понимаются алгоритмы, которые меняются (обучаются) каким-то образом в зависимости от входных данных и итоговых результатов.

Машинное обучение — очень обширная область знаний. Можно ведь по-разному определять слово «обучение» и каждый раз получать интересные результаты. Однако среди множества парадигм и подходов в машинном обучении выделяется одна очень интересная область — искусственные нейронные сети.

Искусственные нейронные сети (artificial neural networks, ANN) — упрощенные модели биологических нейронных сетей мозга человека.

3.2. Искусственный нейрон как основа нейронных сетей

Теперь в самых общих чертах разберемся с такими понятиями, как искусственная нейронная сеть и искусственный нейрон.

Что такое *биологические* нейронные сети? Если вспомнить школьную программу по биологии, то это что-то, связанное с головой, а вернее, с мозгом. Мозг есть не только у человека, но и у многих других животных. Наш мозг представляет собой сложнейшую биологическую нейронную сеть, которая принимает информацию от органов чувств (глаза, уши, нос), каким-то образом ее обрабатывает (узнает лица, распознает речь, ощущает запахи и т. д.). На основе полученных данных она дает команды различным исполнительным органам (пожать руку знакомому человеку, пойти к доске по просьбе учителя, покинуть помещение при появлении неприятного запаха). Все эти действия выполняет наш мозг, опираясь на биологическую нейронную сеть, состоящую из совокупности нейронов. В головном мозге человека общая нейронная сеть содержит примерно 90 млрд нейронов, соединенных друг с другом миллиардами связей. Это, пожалуй, самый сложный объект, известный нам, людям. Основой этой сети является *нейрон*. Упрощенное строение биологического нейрона показано на рис. 3.1.

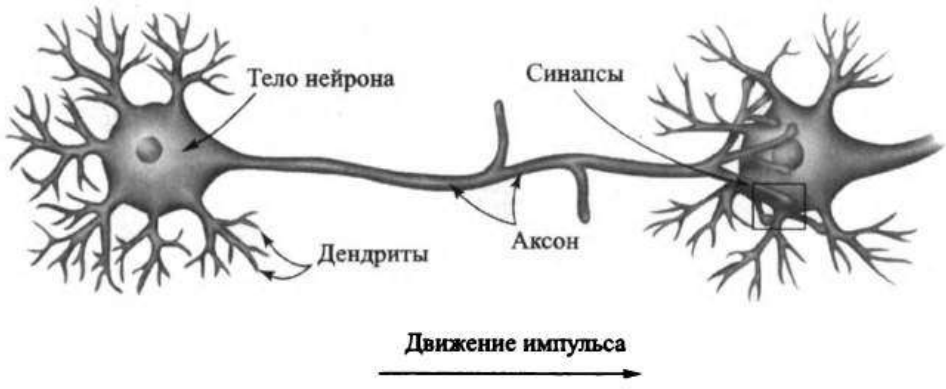


Рис. 3.1. Упрощенное строение биологического нейрона

Биологический нейрон — это нервная клетка, состоящая из тела, дендритов и аксона. От тела отходит множество коротких и толстых отростков, называемых *дендритами*. Это входные отростки — они принимают импульсы с тех нейронов, которые находятся в сети раньше, чем этот нейрон. От тела отходит также один очень длинный и тонкий отросток, называемый *аксоном*. Это выходной отросток — по нему нейрон передает электрохимический импульс следующим нейронам в сети. Окончание аксона имеет разветвления, через которые выходной сигнал может быть передан нескольким следующим нейронам. Таким образом, через множество дендритов нейрон получает входные сигналы, в теле нейрона они обрабатываются, а через единственный аксон выходной импульс от нейрона передается дальше. Реальный биологический нейрон является достаточно сложной системой. Во многом это объясняется тем, что нейрон, помимо обработки сигнала (основное его назначение), вынужден еще выполнять множество других функций, поддерживающих его жизнь. Более того, сам механизм передачи сигнала от нейрона к нейрону тоже очень сложный с биологической и химической точек зрения.

Достаточно упрощенная схема искусственного нейрона с двумя входами и одним выходом представлена на рис. 3.2.

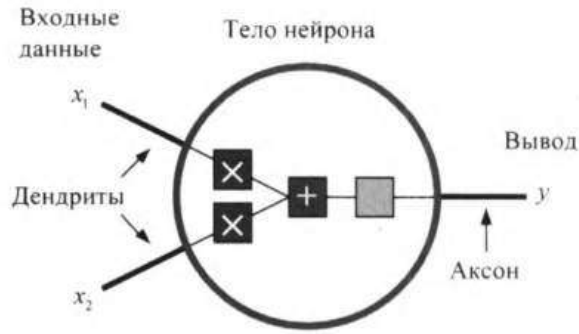


Рис. 3.2. Упрощенная схема искусственного нейрона

Здесь тело нейрона представлено в виде «черного ящика», который принимает входные данные (x_1, x_2), выполняет с ними некоторые операции, а затем выводит результат — y.

Как нейроны взаимодействуют между собой в нейронной сети? Дело в том, что аксон каждого нейрона на своем конце имеет большое количество так называемых синапсов. *Синапс* — это место соединения выходного аксона одного нейрона с входными дендритами другого нейрона (см. рис. 3.1).

Через синапс передается нервный импульс. А сам нервный импульс формируется в теле нейрона, которое фактически выступает сумматором, принимающим все входные импульсы, взвешивающим их, передающим их в некоторую активирующую функцию и принимающим решение, запускать импульс дальше по своему аксону или нет. Решение принимается очень просто — если суммарные входные импульсы превышают некий заданный порог, то выходной импульс запускается.

Для реализации искусственного нейрона нужно построить его упрощенную математическую модель, в которой были бы реализованы его базовые функции без учета функций жизнеобеспечения. В 1943 году первую модель искусственного нейрона и основанную на нем модель нейронной сети предложили американские ученые — нейрофизиолог Уоррен Мак-Каллок и математик Уолтер Питтс. Математическая модель искусственного нейрона представлена на рис. 3.3.

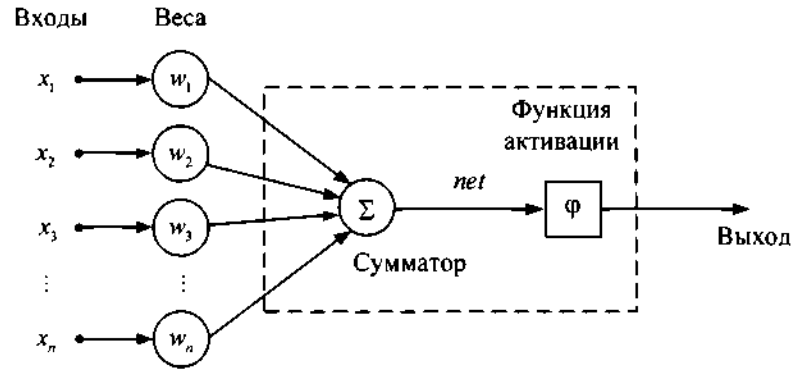


Рис. 3.3. Математическая модель искусственного нейрона

Эта модель довольно простая. На вход математического нейрона поступает некоторое количество входных параметров: x_1, x_2, x_3, ..., x_n, как бы от дендритов, и каждый входной параметр имеет свой вес: w_1, w_2, w_3, ..., w_n. В теле искусственного нейрона имеется *сумматор*, где каждый входной сигнал умножается на некоторый действительный весовой коэффициент, в результате чего формируется итоговая сумма. Полученное значение передается в *функцию активации*. Перед выходом из тела нейрона осуществляется проверка значения этой функции — если оно выше некоторого порога, то на выходе из тела нейрона формируется значение, равное единице. В этом случае нейрон активируется, и в аксон передается соответствующий сигнал. Если же итоговое значение в функции активации ниже порога, то на выход из тела нейрона подается значение, равное нулю, и нейрон считается неактивированным.

Давайте в упрощенном виде рассмотрим, как работает математическая модель искусственного нейрона. У каждого нейрона, в том числе и искусственного, должны быть какие-то входы, через которые он принимает сигнал — X. Для входных сигналов вводится понятие весов — W, на которые умножаются эти сигналы. В теле искусственного нейрона поступившие на входы сигналы умножаются на их веса. Сигнал первого входа x_1 умножается на соответствующий этому входу вес w_1, сигнал x_2 — на w_2, и так до последнего n-го входа. Затем в сумматоре эти произведения суммируются. В итоге мы получаем сумму произведений значений входных сигналов на их веса S:

$$S = x_1w_1 + x_2w_2 + x_3w_3 + ... + x_nw_n.$$

Роль сумматора здесь очевидна: он преобразует все входные сигналы (которых может быть много) в одно число — взвешенную сумму, характеризующую поступив-

ший на нейрон сигнал в целом. Итак, на вход искусственного нейрона было подано множество входных сигналов X с их весами W, а в сумматоре мы получили единственное число — S. Реализуем сумматор в виде следующего программного кода (листинг 3.1).

Листинг 3.1

```
# Модуль Neuron
import numpy as np

# Создание класса "Нейрон"
class Neuron:
    def __init__(self, w):
        self.w = w

    def y(self, x):                 # Сумматор
        s = np.dot(self.w, x)       # Суммируем входы
        return s                    # функция активации

Xi = np.array([2, 3])   # Задание значений входам
Wi = np.array([1, 1])   # Веса входных сенсоров
n = Neuron(Wi)          # Создание объекта из класса Neuron
print('S1= ', n.y(Xi))  # Обращение к нейрону
Xi = np.array([5, 6])   # Веса входных сенсоров
print('S2= ', n.y(Xi))  # Обращение к нейрону
```

Здесь мы создали класс `Neuron`, в теле которого подсчитывается сумма произведений входного параметра на его вес. Если мы запустим этот программный код, то получим следующий ответ:

```
S1=5
S2=11
```

Теперь перейдем к следующему компоненту искусственного нейрона — функции активации. Для понимания принципа его работы рассмотрим простой пример. Допустим, у нас есть один искусственный нейрон, роль которого — определить, идти на рыбалку или нет. Это типичная задача, в которой нужно проанализировать сочетание множества факторов и на основе этого анализа принять итоговое решение. Для простоты учтем всего четыре фактора, влияющих на активность рыб, а значит, на успех вернуться с рыбалки с уловом. На входы в нейрон мы подадим следующие исходные данные и оценим влияние этих факторов на клев рыбы:

- x_1 — скорость ветра;
- x_2 — атмосферное давление;
- x_3 — яркость солнца;
- x_4 — перепад температуры воды.

В зависимости от скорости ветра во многом меняются поведение рыбы в водоеме и ее интерес к приманкам. Небольшой по силе ветер всегда создает на поверхности воды рябь. Из-за нее рыба не видит четко, что находится над водой, а потому не пугается даже заметных и непривычных для водоема ситуаций. При этом даже такие сигналы опасности, как сторонний шум, яркий костюм рыбака, толстая или яркая леска, рыба может проигнорировать и будет активно брать наживку. Однако когда начинается сильный ветер, рыба, предчувствуя возможную бурю или шторм, перестает клевать и уходит на глубину, где ей безопасно. Таким образом, при слабом ветре рыба хорошо клюет, а при сильном — клев плохой.

Известно, что рыба меняет жизненную активность, скорее, не при разном значении атмосферного давления, а при его резких скачках. Наихудшими условиями для рыбалки считаются перепады давления, а также низкое давление, хотя не все виды рыб одинаково реагируют на подобные изменения. Повышенное давление положительно сказывается на «мелочи», которая в поисках пищи перемещается в верхние слои воды и начинает активно клевать. При пониженном давлении она менее активна. Однако при пониженном давлении активизируются хищники. Мелкая рыба становится вялой, поэтому хищники тратят меньше сил и энергии в поисках пропитания. То есть при пониженном давлении не стоит рассчитывать на клев мелкой рыбы, но зато можно поймать крупного хищника. Предположим, что мы берем с собой спиннинг и собираемся ловить щуку или судака. Тогда при низком давлении клев этих видов рыб будет хорошим, а при высоком давлении — плохим.

Яркое солнце не способствует хорошему улову. Поэтому, если выходить на рыбалку в ясный солнечный день, то клёва может и не быть. Но если на небе есть облака, либо вы пойдете на берег ранним утром или вечером, когда солнце не такое яркое, то вероятность иметь хороший улов гораздо выше.

Рыбы чувствительны не столько к температуре воды, сколько к ее резким перепадам, и реагируют даже на колебания в десятые доли градуса. Перепад в 4–5°С (не связанный со сменой дня и ночи) негативно сказывается на активности рыб. Лучше всего рыба чувствует себя при минимальном суточном изменении температуры. Считается, что этот диапазон не должен превышать 1–2°С. Так, если вчера дневная температура воды была 10°С, а сегодня в это же время она составляет от 8 до 12°С, то это нормально, и можно прогнозировать пищевую активность рыбы. В противном случае при перепаде температуры более 4–5°С активность рыбы будет снижена. Таким образом, при стабильной температуре воды клёв будет хороший, и можно идти на рыбалку, а при резком колебании температуры воды лучше остаться дома.

Для упрощения нашей модели присвоим всем этим параметрам логические значения в виде цифр 0 или 1. Например, если ветер умеренный и мы можем идти на рыбалку, то значение входа будет 1, если же на улице сильный ветер и клёв маловероятен, то на входе будет 0. Соответственно если атмосферное давление низкое и щука с судаком выйдут на охоту, то на этот вход подаем 1, а если атмосферное давление высокое, то 0. Аналогично поступаем со всеми остальными параметрами: пасмурно — 1, солнечно — 0, температура воды стабильна — 1, сильный перепад температуры воды — 0.

Если у нейрона есть четыре входа, то должны быть и четыре весовых коэффициента. В нашем примере весовые коэффициенты можно представить как показатели важности каждого входа, влияющие на общее решение нейрона. Чем больше значение коэффициента, тем больше важность входного параметра. Веса входов распределим следующим образом:

- $w_1 = 5$;
- $w_2 = 4$;
- $w_3 = 1$;
- $w_4 = 1$.

По заданным весовым коэффициентам нетрудно заметить, что очень важную роль играют факторы скорости ветра и атмосферного давления (первые два входа). Они же и будут самыми важными при принятии нейроном решения. Вторые два фактора имеют более низкое влияние на принятие решения.

Пусть на входы нашего нейрона мы подаем следующие сигналы (ветер — умеренный, атмосферное давление — высокое, яркость солнца — пасмурно, температура воды — стабильная):

- ветер (умеренный) $x_1 = 1$;
- атмосферное давление (высокое) $x_2 = 0$;
- яркость солнца (солнечно) $x_3 = 0$;
- перепад температуры воды (нет) $x_4 = 1$.

При поступлении этой информации в сумматор он выдаст следующую итоговую сумму:

$$S = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 = 1 \cdot 5 + 0 \cdot 4 + 0 \cdot 1 + 1 \cdot 1 = 5 + 0 + 0 + 1 = 6.$$

Проверим это с помощью нашей программы-сумматора, представленной в листинге 3.2. Для этого изменим значения соответствующих параметров в программном коде сумматора.

Листинг 3.2

```
# Этот фрагмент кода не является рабочим
Xi = np.array([1, 0, 0, 1])  # Задание значений входам
Wi = np.array([5, 4, 3, 1])  # Веса входных сенсоров
n = Neuron(Wi)               # Создание объекта из класса Neuron
print('S= ', n.y(Xi))        # Обращение к нейрону
```

Теперь необходимо объединить листинги 3.1 и 3.2 в один программный модуль. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 3_2_1) в сопровождающем книгу файловом архиве (см. *приложение*). Если запустить программу на выполнение, то мы действительно получим результат S=6.

Итак, сумматор выдаст нам значение S=6, а что делать дальше? Как нейрон должен решить, идти на рыбалку или нет? Очевидно, нам нужно как-то преобразовать взвешенную сумму S в итоговое решение. Эту задачу и решает функция активации.

Просто так выводить пользователю значение взвешенной суммы бессмысленно. Нейрон должен как-то обработать ее и сформировать адекватный выходной сигнал. Иными словами, нужно преобразовать взвешенную сумму S в какое-то число, которое и является выходом нейрона (обозначим выходной сигнал переменной Y).

Для разных типов искусственных нейронов используют самые разные функции активации. Обозначим нашу функцию активации как $f(S)$. В этом выражении указание взвешенного сигнала S в скобках означает, что функция активации принимает его как параметр.

Функция активации (activation function) — это такая функция, которая в качестве входного параметра получает взвешенную сумму S как аргумент, а на выходе формирует значение выходного сигнала из нейрона (Y). В общем виде эту функцию можно описать выражением

$$Y = f(S).$$

Рассмотрим некоторые математические функции, которые могут быть использованы в качестве функции активации.

3.2.1. Функция единичного скачка

Общий вид *функции единичного скачка* (или *функции Хевисайда*) приведен на рис. 3.4.

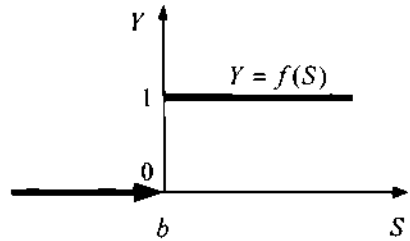


Рис. 3.4. Вид функции единичного скачка

На горизонтальной оси этой функции расположены величины взвешенной суммы S. На вертикальной оси — значения выходного сигнала Y. Это самый простой вид функции активации. При использовании такой функции выход из нейрона Y может получать только два значения: 0 или 1. Какое значение получит выходной сигнал Y на выходе из функции активации, зависит от величины порогового значения b. Если взвешенная сумма S будет больше порога b, то выход нейрона получит значение единица ($Y = 1$). Если сумма S окажется меньше порогового значения b, то выход из нейрона получит значение ноль ($Y = 0$).

Как же можно использовать эту функцию в нашем примере? Предположим, что мы поедем на рыбалку только тогда, когда взвешенная сумма S будет больше или

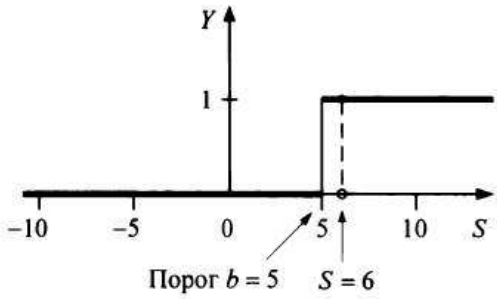


Рис. 3.5. Результаты работы пороговой функции для примера о поездке на рыбалку

равна пороговому значению $b = 5$. В нашем примере взвешенная сумма 6, а это больше 5. Значит, пороговая функция на выходе выдаст значение $Y = 1$ (рис. 3.5).

Проверим это, реализовав работу нашей функции активации следующим программным кодом (листинг 3.3).

Листинг 3.3

```
# Модуль onestep
import numpy as np

def onestep(x):
    b = 5
    if x >= b:
        return 1
    else:
        return 0

# Создание класса "Нейрон"
class Neuron:
    def __init__(self, w):
        self.w = w

    def y(self, x):  # Сумматор
        s = np.dot(self.w, x)    # Суммируем входы
        return onestep(s)        # Функция активации

Xi = np.array([1, 0, 0, 1])    # Задание значений входам
Wi = np.array([5, 4, 3, 1])    # Веса входных сенсоров
n = Neuron(Wi)                 # Создание объекта из класса Neuron
print('Y= ', n.y(Xi))          # Обращение к нейрону
```

В результате работы этой программы получим `Y=1`. Итак, поскольку на выходе искусственный нейрон выдаст значение `Y=1`, то мы принимаем решение — едем ловить рыбу.

Теперь рассмотрим работу того же нейрона при условии, что ветер — сильный, атмосферное давление — высокое, на улице пасмурно, и температура воды стабильная. В этом случае входные параметры будут иметь следующие значения:

- ветер (сильный) $x_1 = 0$;
- атмосферное давление (высокое) $x_2 = 0$;
- яркость солнца (пасмурно) $x_3 = 1$;
- перепад температуры воды (нет) $x_4 = 1$.

При поступлении этой информации в сумматор он выдаст следующую итоговую сумму:

$$S = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 = 0 \cdot 5 + 0 \cdot 4 + 1 \cdot 1 + 1 \cdot 1 = 0 + 0 + 1 + 1 = 2.$$

Мы получили взвешенную сумма $S = 2$, что меньше порогового значения $b = 5$. Значит, на выходе из нейрона получим значение `Y=0`. В этом случае мы никуда не идем и остаемся дома, поскольку клёва не будет. Проверим это с помощью нашей программы (листинг 3.2.3), изменив в ней всего одну строку:

```
Xi = np.array([0, 0, 1, 1])  # Задание значений входам
```

Действительно, на выходе из программы получаем `Y=0`.

Теперь запишем нашу функцию активации (функцию единичного скачка) в виде общего математического выражения:

$$Y = \{0,\ S < b;\ 1,\ S \geq b\}.$$

Читается эта запись составной функции так: выход нейрона (Y) зависит от взвешенной суммы (S) следующим образом: если S (взвешенная сумма) меньше какого-то порога (b), то Y (выход нейрона) равен 0. А если взвешенная сумма S больше или равна порогу b, то Y равен 1.

3.2.2. Сигмоидальная функция активации

На самом деле существует целое семейство сигмоидальных функций, и некоторые из них применяются в качестве функции активации в искусственных нейронах. Все эти функции обладают полезными свойствами, ради которых их и используют в нейронных сетях. Эти свойства станут очевидными, когда мы более детально ознакомится с графиками таких функций. Итак, наиболее часто используемая в нейронных сетях функция — это *сигмоида*, или *логистическая функция* (рис. 3.6).

График этой функции выглядит достаточно просто, а внешний вид имеет некоторое подобие английской буквы S. А вот так она записывается аналитически:

$$Y = \frac{1}{1 + \exp(-aS)}.$$

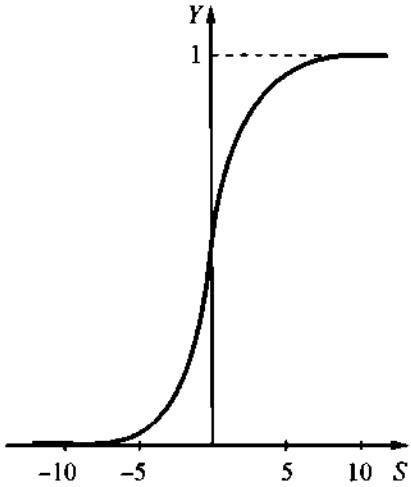


Рис. 3.6. Сигмоида — логистическая функция

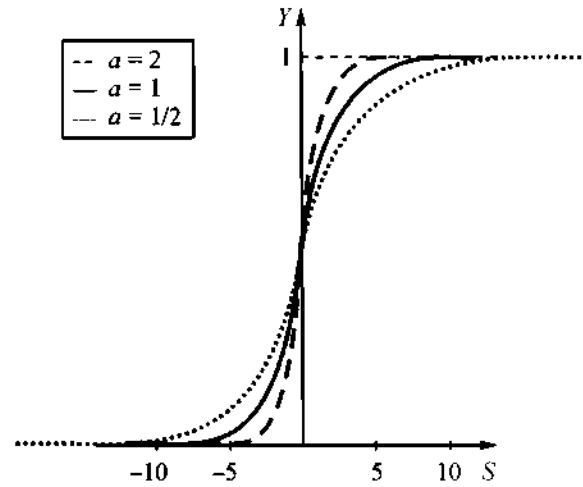


Рис. 3.7. Логистические функции с разными значениями параметра *a*

Какую роль здесь играет параметр *a*? Это некое число, которое характеризует степень крутизны функции. На рис. 3.7 представлены логистические функции с разными значениями параметра *a*.

Вернемся к нашему искусственному нейрону, который определяет успешность похода на рыбалку. В случае с функцией единичного скачка все было очевидно и однозначно: мы либо едем на рыбалку — при $Y = 1$, либо нет — при $Y = 0$. Используя же логистическую функцию, мы получаем вероятностный итоговый результат в виде числа между 0 и 1. Причем чем больше взвешенная сумма S, тем ближе выход Y будет к 1 (но никогда не будет точно ей равен). И наоборот, чем меньше взвешенная сумма S, тем выход нейрона Y будет ближе к 0.

То есть чем ближе значение Y к единице, тем больше вероятность того, что нужно принимать положительное решение. И наоборот, чем ближе значение Y к нулю, тем большая вероятность принятия отрицательного решения. Например, в нашем примере с походом на рыбалку при значении выхода из нейрона $Y = 0,8$, скорее всего, пойти ловить рыбу все-таки стоит. Если же значение $Y = 0,2$, то это означает, что вам почти наверняка нужно отказаться от такого похода. Для того чтобы однозначно определить итоговое решение, нужно задать величину порогового значения. Например, если задать пороговое значение $b = 0,6$, то при рассчитанной величине $Y \geq 0,6$ всегда идем на рыбалку, а при $Y < 0,6$ остаемся дома.

Логистическая функция имеет следующие свойства:

- она является «сжимающей» функцией, т. е. вне зависимости от аргумента (взвешенной суммы S) выходной сигнал Y всегда будет в пределах от 0 до 1;
- она более гибкая, чем функция единичного скачка, — ее результатом может быть не только 0 и 1, но и любое число между ними;
- во всех точках она имеет производную, и эта производная может быть выражена через эту же функцию.

Именно из-за этих свойств логистическая функция чаще всего используется в качестве функции активации в искусственных нейронах. Немного изменим программный код нашего нейрона и задействуем в нем сигмоиду в качестве функции активации (листинг 3.4).

Листинг 3.4

```
# Модуль sigmoid
import numpy as np

# функция активации: f(x) = 1 / (1 + e^(-x))
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Создание класса "Нейрон"
class Neuron:
    def __init__(self, w):
        self.w = w

    def y(self, x):              # Сумматор
        s = np.dot(self.w, x)    # Суммируем входы
        return sigmoid(s)        # функция активации

X1 = np.array([0, 0, 1, 1])    # Задание значений входам
W1 = np.array([5, 4, 3, 1])    # Веса входных сенсоров
n = Neuron(W1)                 # Создание объекта из класса Neuron
print('Y= ', n.y(X1))          # Обращение к нейрону
```

На выходе получим `Y=0.9820137900379085`, что больше $b = 0{,}6$, а это значит, что мы идем на рыбалку.

Изменим входные параметры (самые неблагоприятные погодные условия):

```
X1 = np.array([0, 0, 0, 0])  # Задание значений входам
```

На выходе получим `Y= 0.5`, что меньше $b = 0{,}6$, значит, остаемся дома, клёва не будет.

3.2.3. Гиперболический тангенс

Однако есть и еще одна сигмоида — *гиперболический тангенс*. Эту функцию чаще применяют биологи для более реалистичной реализации модели нервной клетки. Такая функция позволяет получить на выходе значения разных знаков (от -1 до 1), что может быть полезным для ряда сетей.

Функция записывается следующим образом:

$$Y = \text{th}(S).$$

А график этой функции приведен на рис. 3.8.

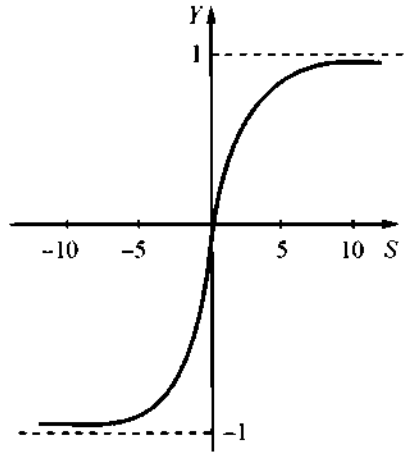


Рис. 3.8. График гиперболического тангенса

3.3. Нейронные сети

У каждого биологического нейрона имеются тысячи входов (дендритов). В свою очередь, каждый выход из нейрона (аксон) через синапсы соединен со входами (дендритами) других нейронов, а это означает тысячи синапсов на каждый нейрон. Каждый синапс индивидуален. Он может либо усиливать, либо ослаблять проходящий через него сигнал. Более того, с течением времени синапсы могут меняться, а значит, будет меняться характер сигнала. Если правильно подобрать параметры синапсов, то входной сигнал после прохода через нейронную сеть будет преобразовываться в правильный выходной сигнал. Именно так и происходит преобразование множества входных сигналов в верное решение на выходе.

Если биологические нейронные сети представляют собой совокупность биологических нейронов (фрагмент биологической нейронной сети представлен на рис. 3.9), то искусственная нейронная сеть — это совокупность взаимодействующих между собой искусственных нейронов. Принципиальная структура искусственной нейронной сети приведена на рис. 3.10.

Вообще говоря, есть несколько способов графического изображения нейронных сетей и нейронов. На рис. 3.10, в частности, искусственные нейроны изображены в виде кружков. А вместо сложного переплетения входов и выходов используются стрелки, обозначающие направление движения сигнала. То есть искусственная нейронная сеть здесь представлена в виде совокупности кружков (искусственных нейронов), связанных стрелками. Во всех нейронных сетях есть входной слой, который выполняет только одну задачу — распределение входных сигналов остальным нейронам. Нейроны этого слоя не производят никаких вычислений. А вот дальше структура сети может иметь самую разную конфигурацию.

В реальной биологической нейронной сети от входов сети к выходам передается электрический сигнал. В процессе прохода по нейронной сети он может изменяться. Меняется величина этого электрического сигнала: сильнее/слабее. А любую величину всегда можно выразить числом: больше/меньше.

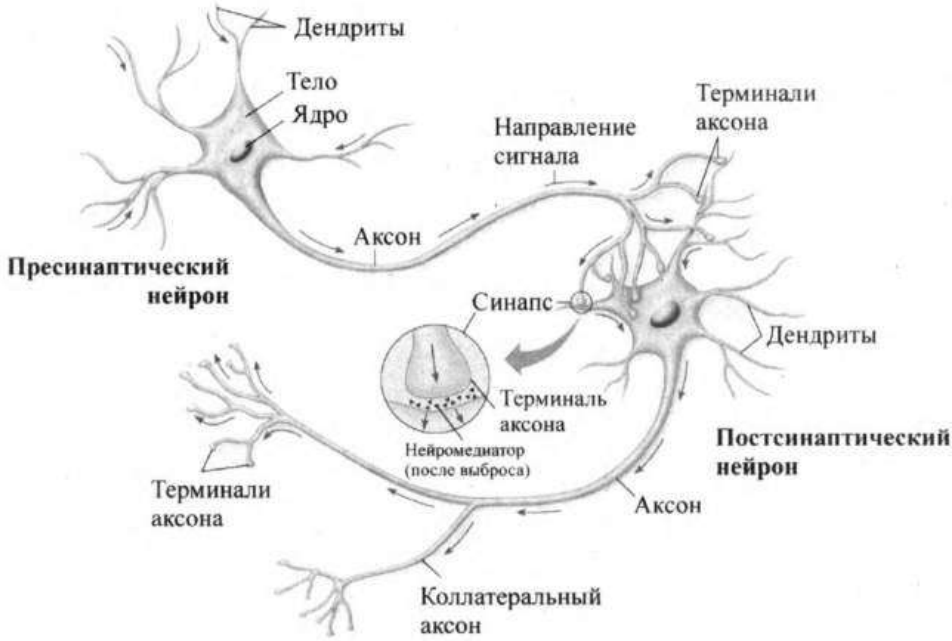


Рис. 3.9. Фрагмент биологической нейронной сети

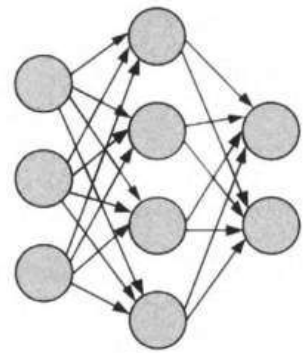


Рис. 3.10. Принципиальная структура искусственной нейронной сети

В модели искусственной нейронной сети совершенно не нужно реализовывать поведение электрического сигнала, т. к. от его реализации все равно ничего зависеть не будет. На входы сети мы будем подавать какие-то числа, символизирующие величины электрического сигнала, как если бы он был. Эти числа будут продвигаться по сети и каким-то образом меняться. На выходе сети мы получим некое результирующее число, являющееся откликом сети.

Вспомним про синапсы, благодаря которым нейроны соединяются между собой. Синапсы могут усиливать или ослаблять проходящий по ним электрический сигнал. Можно характеризовать каждую такую связь определенным числом, называемым *весом связи*. Сигнал, прошедший через связь, умножается на ее вес. Это ключевой момент в концепции искусственных нейронных сетей. Каждая связь между

нейронами сети характеризуется весом связи. Его можно определить некоторым числом. Когда выходной сигнал от одного нейрона передается на вход другого нейрона, то его величина умножается на вес этой связи. Мы уже знакомились с весами входных сигналов, когда рассматривали пример работы нейрона, определяющего целесообразность похода на рыбалку. Так вот, аналогичные веса имеют и связи между нейронами в искусственной нейронной сети. Остановимся подробнее на типах нейронных сетей.

3.3.1. Однослойные нейронные сети

В *однослойных* нейронных сетях сигналы с входного слоя сразу подаются на выходной слой. Он производит необходимые вычисления, результаты которых сразу подаются на выходы. Выглядит однослойная нейронная сеть так, как показано на рис. 3.11.

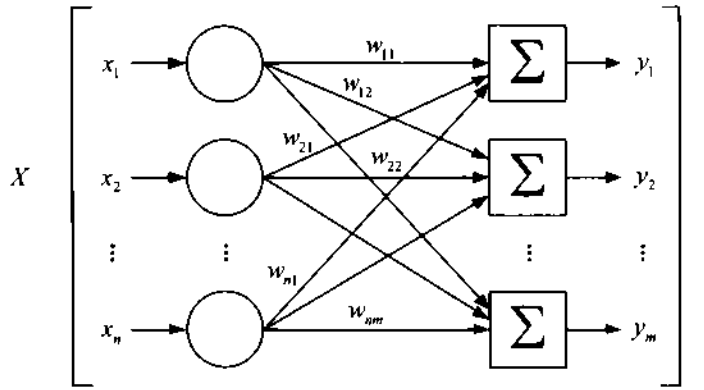


Рис. 3.11. Структура однослойной нейронной сети

Здесь входной слой обозначен кружками, а справа расположен единственный выходной слой нейронов. Нейроны соединены друг с другом стрелками. Над стрелками расположены веса соответствующих связей (весовые коэффициенты).

3.3.2. Многослойные нейронные сети

Многослойная нейронная сеть состоит из входного слоя, нескольких скрытых слоев и выходного слоя (рис. 3.12). На входном слое формируются и передаются в сеть сигналы от каких-либо источников (например, данные от радаров, лидаров, видеокамер беспилотного автомобиля). Внутренние слои нейронной сети обрабатывают входную информацию. Выходной слой передает итоговые решения в виде команд в органы управления (для беспилотного автомобиля это управление рулем, педалями акселератора, тормоза и т. п.).

Скрытые слои получили свое название неслучайно. Дело в том, что только относительно недавно были разработаны методы обучения нейронов скрытого слоя. До этого обходились только однослойными нейронными сетями. Многослойные ней-

ронные сети обладают бóльшими возможностями, чем однослойные. Функционирование скрытых слоев нейронов можно сравнить с работой большого завода. Изделие на заводе собирается за несколько этапов. На каждый станок устанавливается некая заготовка, в результате обработки которой получается сборочная деталь. Эта деталь передается на следующий участок, где из деталей собирается некий узел, который передается в другой цех, и т. д. Результатом работы всего комплекса является готовое изделие. В искусственной нейронной сети происходит то же самое. Входной слой получает исходные сигналы в виде данных. Скрытые слои преобразуют входные сигналы в некоторые промежуточные результаты, которые в итоге доходят до выходного слоя. Таким образом, на выходе будет получен итоговый результат работы всей нейронной сети

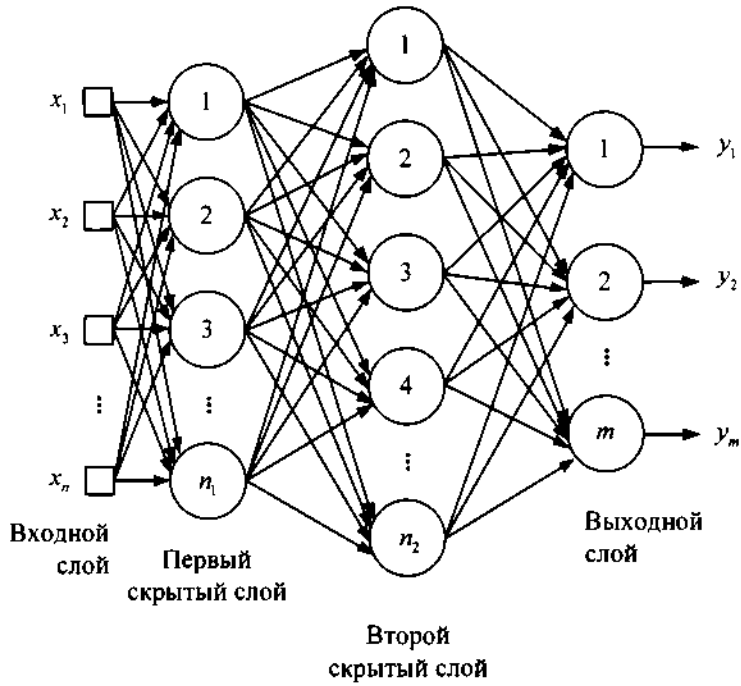


Рис. 3.12. Структура многослойной нейронной сети

На рисунках, отображающих структуру нейронной сети, видно, что все стрелки направлены в одну сторону — слева направо, т. е. сигнал в таких сетях идет от входного слоя к выходному в одном направлении. Такие сети называются *сетями прямого распространения* (feedforward neural network). Эти сети достаточно широко используются и успешно решают определенный класс задач: прогнозирование, кластеризацию, распознавание объектов.

Однако никто не запрещает отправить сигнал и в обратную сторону, т. е. выходной сигнал из нейрона одного слоя вернется на вход нейрона предыдущего слоя (рис. 3.13). Такие сети называют *сетями с обратными связями* (recurrent neural network).

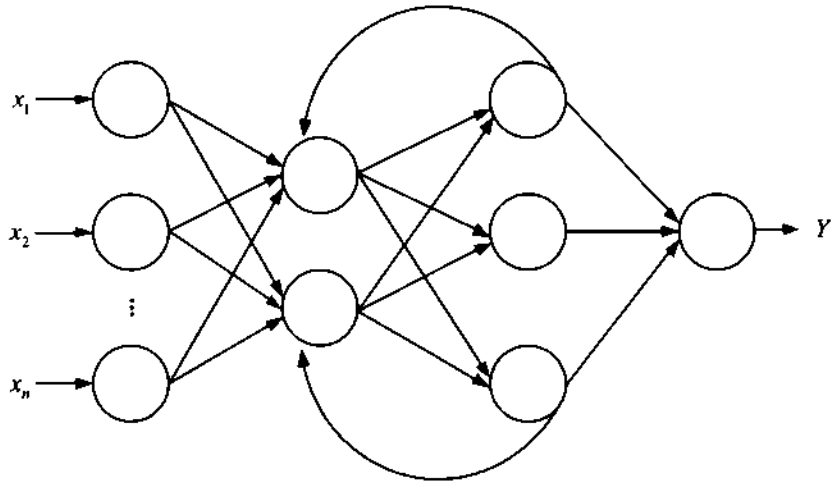


Рис. 3.13. Структура многослойной нейронной сети с обратными связями

В чем здесь преимущество? В сетях с обратными связями выходы нейронов могут возвращаться на входы. Это означает, что выход какого-нибудь нейрона определяется не только его весами и входным сигналом, но еще и выходами из других нейронов последующих слоев. Возможность сигналов циркулировать в сети открывает новые, удивительные перспективы нейронных сетей. Используя эту возможность, можно создавать нейронные сети, восстанавливающие или дополняющие сигналы. Другими словами, такие нейронные сети имеют свойства кратковременной памяти (как у человека).

3.4. Обучение нейронных сетей

Теперь рассмотрим такой важный аспект, как обучение нейронной сети. Что это такое? Каким образом оно происходит?

3.4.1. Что такое обучение сети?

Вернемся к нейронной сети, которая определяла, сто́ит ли идти на рыбалку. В ней, опираясь на свой опыт и знания, мы принудительно задали веса входных параметров (W) и порога (b). Сделать это было несложно, поскольку это была элементарная сеть, состоящая всего из одного нейрона. Теперь рассмотрим сеть, состоящую из 100 нейронов, с множеством скрытых слоев и большим количеством связей. Ясно, что при некорректном задании параметров этой сети, даже при подаче на вход корректных данных мы получим что-то бессмысленное на выходе. Значит, нам надо менять какие-то параметры сети до тех пор, пока входной сигнал не преобразуется в нужный нам выходной.

Что мы можем менять в нейронной сети? Изменять общее количество искусственных нейронов бессмысленно. Если вы соберете вместе 1000 человек, кото́рые не умеют считать (вместо 100), то они все равно не смогут правильно решить постав-

ленную задачу. То есть увеличение количества вычислительных элементов лишь сделает нейронную сеть тяжеловеснее и избыточнее. Сумматор изменить не получится, т. к. он выполняет единственную функцию — сложение. Если мы его заменим чем-нибудь иным или вообще уберем, это уже не будет искусственным нейроном. Менять у каждого нейрона тип функции активации тоже не имеет смысла, поскольку все они выполняют одни и те же действия. Остается только один вариант — менять веса связей.

Обучение нейронной сети (training) — это поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется в нужный нам выходной.

Такой подход к понятию «обучение нейронной сети» соответствует и биологическим нейронным сетям. Наш мозг состоит из огромного количества связанных друг с другом нейронов. Каждый из них имеет функцию активации. Мы обучаемся благодаря изменению синапсов — элементов, которые усиливают или ослабляют передаваемые сигналы.

Однако есть один важный момент. Если обучать сеть, используя только один входной сигнал, то она просто «запомнит» правильный ответ при этом сигнале. Со стороны будет казаться, что сеть очень быстро «обучилась». Но как только будет подан немного измененный входной сигнал, мы вместо правильного ответа увидим бессмыслицу. Например, мы научили сеть распознавать лицо человека на фото по фотографии одного человека. Пока мы будем на вход такой сети подавать изображение именно этого человека, она будет работать корректно — на фотографии, где есть множество людей и других объектов, выделять лицо именно этого человека. Но если мы предъявим такой сети фотографии, на которых совершенно другие люди, она не обнаружит ни одного лица. В самом деле, зачем нам сеть, способная определять лицо только одного конкретного человека. Мы ждем от сети гораздо большего, а именно — способности *обобщать* какие-то признаки и узнавать лица любых людей на любых фотографиях и в видеофайлах. Новорожденные не умеют ни читать, ни считать, ни писать. Их этому учат. А как учат? Показывают примеры и объясняют, что они означают. Например, ребенку говорят: «Это — буква А, а это — буква М. Если их сложить вместе, то получится слово МАМА». Аналогично поступают и с нейронными сетями — им показывают некий набор входных параметров и правильный ответ, который соответствует этим параметрам. Такой набор данных называется *обучающей выборкой*.

3.4.2. Обучающая выборка

Обучающая выборка (training set) — это набор входных сигналов (вместе с правильными выходными сигналами), по которым происходит обучение сети.

Когда сеть обучена — т. е. выдает корректные результаты для всех входных сигналов из обучающей выборки, можно считать, что она прошла учебный курс, и пригодна для использования на практике. Это можно сравнить с процессом обучения студентов и школьников. Допустим, школьнику дали для заучивания таблицу умножения. Он ее вызубрил наизусть и безошибочно продемонстрировал учителю

свои знания у школьной доски. Но это еще не значит, что он сможет правильно использовать свои знания на практике. Через какое-то время школьнику дают контрольную работу, где в различных примерах нужно выполнить разные действия, в том числе и умножение. Здесь уже можно окончательно убедиться, что ученик смог в различных по сложности примерах вычленить такое действие, как умножение, и правильно выполнить его. Аналогично поступают и с нейронными сетями — устраивают для них контрольную работу. Такой контроль качества обучения нейронной сети называется *проверкой на тестовой выборке*.

3.4.3. Тестовая выборка

Тестовая выборка (testing set) — это набор входных сигналов (вместе с правильными выходными сигналами), по которым происходит оценка качества работы сети после обучения на обучающей выборке. В тестовой выборке значения входных сигналов имеют некоторый разброс относительно идеальных сигналов в обучающей выборке. Здесь проверяется, будет ли нейронная сеть выдавать правильные решения в тех случаях, когда входные сигналы имеют некоторое допустимое рассеивание относительно идеальных значений. То есть делается проверка устойчивости работы сети в условиях, приближенных к реальным.

Таким образом, обучение нейронной сети, как и любого ученика, происходит в два этапа: обучение на примерах с идеальными данными, контрольная работа на примерах с реальными данными.

Итак, мы разобрались с понятием «обучение нейронной сети». Это подбор правильного набора весов. Теперь возникает вопрос: как можно обучать сеть? Вернемся к школьникам и студентам. Как у них проходит процесс обучения? Любой ученик получает знания двумя способами: либо слушает учителя и запоминает то, что он говорит, либо сидит за учебниками и самостоятельно изучает учебный материал. По аналогии различают два подхода к обучению нейронных сетей: обучение с учителем и обучение без учителя.

3.4.4. Обучение с учителем

Обучение с учителем (supervised learning) — вид обучения нейронной сети, при котором веса нейронных связей подбираются таким образом, чтобы ответы на выходе из сети минимально отличались от уже готовых правильных ответов.

Суть этого подхода заключается в том, что вы даете на вход сети тестовые сигналы из обучающей выборки, получаете ответ сети, а затем сравниваете его с заведомо правильным ответом. Затем, с помощью специальных алгоритмов, вы меняете веса связей нейронной сети, снова даете ей тестовый входной сигнал и снова сравниваете ее ответ с правильным ответом. Этот процесс повторяется до тех пор, пока сеть не начнет правильно отвечать с приемлемой точностью.

Где взять правильные ответы? Например, если мы хотим, чтобы сеть узнавала лица на фотографиях, мы можем создать обучающую выборку из 1000 фотографий (входные сигналы тестовой выборки) и самостоятельно выделить на ней лица (сформировать правильные ответы).

Если мы хотим, чтобы сеть прогнозировала погоду, то обучающую выборку (входные данные) надо сформировать на основе реальных данных прошлых периодов. В качестве входных сигналов могут быть такие параметры за прошедшие даты, как температура воздуха, атмосферное давление, сила ветра, направление ветра, влажность и т. д. А в качестве правильных ответов — реальная погода на следующий день после указанной даты.

Стоит отметить, что учителем для нейронной сети в большинстве случаев является не человек, а специальная компьютерная программа, поскольку порой сеть приходится тренировать часами и днями, совершая десятки или сотни тысяч циклов обучения (уроков).

3.4.5. Обучение без учителя

Обучение без учителя (unsupervised learning) — вид обучения нейронной сети, при котором сеть самостоятельно классифицирует (разделяет) входные сигналы. При этом правильные (эталонные) выходные сигналы ей не демонстрируются.

Обучение без учителя применяют тогда, когда мы не можем сети предоставить правильные ответы на входные сигналы. В этом случае вся обучающая выборка состоит из набора входных сигналов, а сеть как бы сама делает разбор поступившего на вход материала. Она, по сути, обучается самостоятельно.

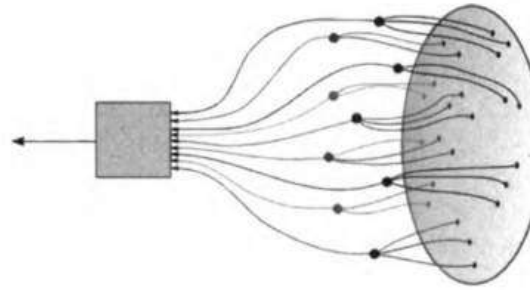
Для того чтобы понять, какой класс задач сможет решать сеть в процессе самообучения, разберем простой пример. Допустим, мы хотим научить нейронную сеть различать на фотографиях некоторые предметы. Для этого подаем на вход в сеть сотни фотографий таких объектов, как автомобили, дорожные знаки и пешеходы. При этом мы не станем вмешиваться в работу сети — мы просто подадим на ее входы сведения о этих объектах. Со временем, после множества циклов самообучения, сеть найдет отличительные признаки каждого объекта, научится их различать и начнет выдавать сигналы трех разных типов, которые и соответствуют каждому объекту на входе. При таком «обучении» сеть начинает разделять подаваемые на вход сигналы на классы. Такой процесс называется *кластеризацией*.

3.5. Краткие итоги главы

В этой главе были приведены начальные теоретические сведения об элементах искусственного интеллекта. Мы узнали, что такое искусственный нейрон и как с помощью Python запрограммировать его действия, что такое функция активации. Выяснили, что такое нейронные сети и каким образом происходит их обучение. То есть к этому моменту у нас есть необходимый инструментарий, небольшой опыт программирования на Python, знания об основах искусственного интеллекта. Это позволяет нам перейти к следующей главе.

В ней рассматриваются некоторые виды нейронных сетей на основе персептрона, конкретные алгоритмы их обучения и практика программирования с использованием Python.

ГЛАВА 4



Программная реализация элементов нейронной сети

В предыдущей главе были представлены общие теоретические сведения об элементах искусственного интеллекта: структура искусственного нейрона и нейронных сетей, общие подходы к их обучению. Тема этой главы — принципы работы простейшего элемента искусственных нейронных сетей — *персептрона*. Здесь будут представлены материалы:

- о персептронах и их классификации;
- о роли персептронов в нейронных сетях;
- о линейной разделимости объектов;
- о способах решения задач классификации объектов на основе логических функций;
- о том, как научить персептрон понимать изображения;
- о том, как научить персептрон подбирать веса связей;
- о дельта-правиле и его использовании при обучении персептрона;
- о линейной аппроксимации и ее использовании для классификации объектов;
- о том, как научить персептрон классифицировать объекты (обучение без учителя);
- об адаптивных линейных нейронах.
- Для запуска программных модулей, приведенных в этой главе, были использованы следующие дополнительные библиотеки:
- Numpy — версия 1.23.5;
- Matplotlib — версия 3.0.2;
- Pandas — версия 1.5.5.

4.1. Персептроны

Понятия искусственного нейрона и искусственной нейронной сети появились достаточно давно, еще в 1943 году. Тогда и была опубликована едва ли не первая статья, в которой предпринимались попытки смоделировать работу мозга. Ее автором был Уоррен Мак-Каллок (рис. 4.1).

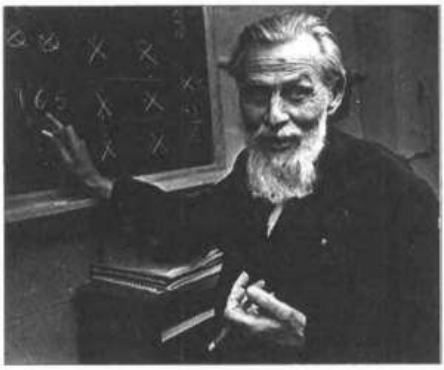


Рис. 4.1. Уоррен Мак-Каллок

Рис. 4.2. Фрэнк Розенблатт

Разитие его идей продолжил нейрофизиолог Фрэнк Розенблатт (рис. 4.2). Он предложил схему устройства, моделирующего процесс человеческого восприятия, и назвал его *персептроном* (от *лат.* perceptio — восприятие). В 1960 году Ф. Розенблатт представил первый нейрокомпьютер — «Марк-1», который был способен распознавать некоторые буквы английского алфавита.

Таким образом, персептрон является первой моделью нейронных сетей, а «Марк-1» — первым в мире нейрокомпьютером.

Персептроны стали очень активно исследовать. На них возлагали большие надежды. Однако, как оказалось, они имели серьезные ограничения. Об этом их недостатке в 1971 году написал целую книгу американский ученый Марвин Ли Минский (рис. 4.3).

Рис. 4.3. Марвин Ли Минский

С тех пор энтузиазм ученых в исследовании персептронов и искусственных сетей поутих. Но потом, через некоторое время, были реализованы алгоритмы обучения нейронных сетей, что вновь возродило интерес к этой области знаний. Впоследствии Минский признал, что его книга нанесла серьезный удар по концепции персеп-

тронов и развитию сфер применения нейронных сетей. Он сам серьезно занялся развитием этого направления и стал одним из основателей лаборатории искусственного интеллекта в Массачусетском технологическом институте.

Теперь подробно рассмотрим структуру персептрона. В его основе лежит математическая модель восприятия информации мозгом. Процесс восприятия информации мы обсудим на примере элементарного персептрона в его физическом воплощении — глазе (рис. 4.4).

В этой модели персептрона можно выделить три элемента: сенсорные элементы (S-элементы), ассоциативные элементы (A-элементы), реагирующие элементы (R-элементы).

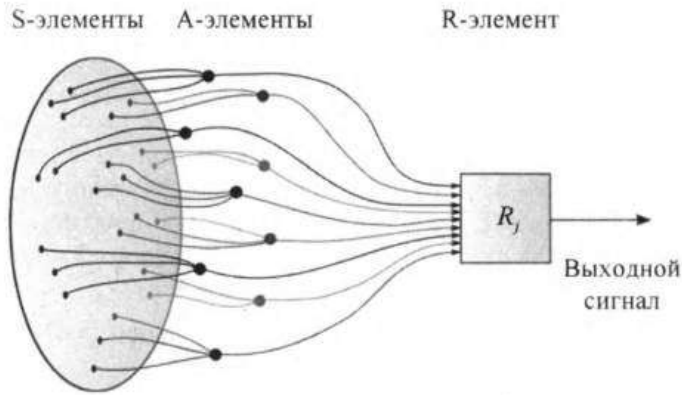


Рис. 4.4. Основные элементы персептрона на примере восприятия информации глазом

S-элементы — это слой сенсоров, или рецепторов. В физическом воплощении они соответствуют, например, светочувствительным клеткам сетчатки глаза или фоторезисторам в матрице камеры цифрового фотоаппарата.

Например, наши глаза состоят из невероятного количества сенсоров, или S-элементов, улавливающих падающий свет (их около 140 млн). В матрице цифровой фотокамеры 5 мегапикселов означает, что в ней есть 5 млн сенсоров, принимающих световой поток, идущий через объектив.

Каждый рецептор (S-элемент) может находиться только в одном из двух состояний: *покоя* (0) или *возбуждения* (1), и только в последнем случае он передает единичный сигнал в следующий слой — ассоциативным элементам (A-элементам).

Элементы следующего слоя (A-элементы) называются *ассоциативными* потому, что каждому такому элементу, как правило, соответствует целый набор S-элементов. A-элемент активизируется, как только количество сигналов от S-элементов на его входе превысит некоторую пороговую величину. Например, мы смотрим на белый лист с изображением буквы «Д». В этом случае набор соответствующих S-элементов будет расположен на сенсорном поле в форме буквы «Д», вокруг которой имеется белый фон. Если мы будем смотреть на пустой белый лист, то на сенсорном поле будет только белый фон и никакого иного изображения. В первом случае A-элемент активизируется (принимает значение 1), т. к. определенное количество

рецепторов сообщило о появлении изображения, ассоциированного с буквой «Д». Во втором случае A-элемент не будет активирован (примет значение 0), поскольку он будет ассоциирован с отсутствием каких-либо букв в определенной части сенсорной области.

Далее сигналы, поступившие от возбудившихся A-элементов, собираются в R-элементе, где определенным образом формируется итоговая картина увиденного изображения.

Рассмотрим теперь логическую структуру математической модели персептрона (рис. 4.5).

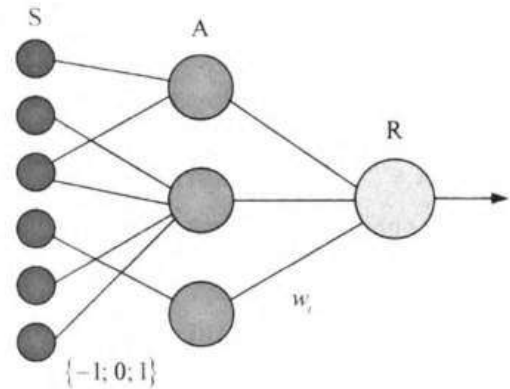


Рис. 4.5. Структура математической модели персептрона

Первыми в работу включаются S-элементы. Они могут находиться либо в состоянии покоя (сигнал равен 0), либо в состоянии возбуждения (сигнал равен 1).

Далее сигналы от S-элементов передаются A-элементам по так называемым S–A-связям. Эти связи имеют вес. Веса S–A-связей могут принимать значения –1, +1 или 0 (т. е. отсутствие связи). Стоит заметить, что одному A-элементу может соответствовать несколько S-элементов. Если сигналы, поступившие на A-элемент, в совокупности превышают некоторый его порог, то этот A-элемент возбуждается и выдает сигнал, равный 1. В противном случае (сигнал от S-элементов не превысил порога), генерируется нулевой сигнал.

Далее сигналы от возбудившихся ассоциативных элементов (A-элементов), передаются в сумматор (R-элемент). При этом сигнал от каждого ассоциативного A-элемента передается с неким коэффициентом w. Этот коэффициент называется *весом A–R-связи* (или *весовым коэффициентом*). Весовой коэффициент может принимать любые значения. Затем R-элемент подсчитывает сумму значений входных сигналов, помноженных на их весовой коэффициент w. После чего полученная сумма сравнивается с неким порогом b и преобразуется при помощи специальной функции в выходной сигнал, который имеет всего два значения (–1 или +1).

Единица будет означать, что персептрон распознал, например, букву «Д», а минус единица — что никаких объектов на пустом белом листе не распознано.

Теперь, после изучения приведенной информации, сформулируем точное определение персептрона.

Персептрон (perceptron) — простейшая модель нейронных сетей. В основе персептрона лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов.

Есть несколько подвидов персептронов, на некоторых из них мы и сконцентрируем все внимание.

4.2. Классификация персептронов

Исторически сложилось так, что разные авторы по-разному классифицируют персептроны. Именно поэтому специалистам, которые только начинают изучение нейронных сетей, бывает трудно разобраться в разновидностях персептронов. Постараемся использовать наиболее часто применимую терминологию.

4.2.1. Персептрон с одним скрытым слоем

Персептрон с одним скрытым слоем — это персептрон, у которого имеется только по одному слою S-, A- и R-элементов. Это именно тот тип персептрона, который часто подразумевают, когда говорят о персептронах в нейронных сетях. Почему слой именно скрытый? Потому что слой A-элементов расположен между слоями S-элементов и R-элементов. Структура персептрона с одним скрытым слоем представлена на рис. 4.6, где нужно обратить внимание на то, что сигнал от одного S-элемента может быть передан нескольким A-элементам.

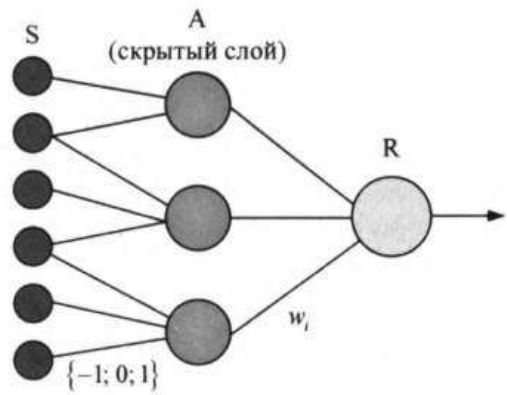


Рис. 4.6. Структура персептрона с одним скрытым слоем

4.2.2. Однослойный персептрон

На первый взгляд *персептрон с одним скрытым слоем* и *однослойный персептрон* — одно и то же. Но это не совсем так. Ключевая особенность однослойного персептрона состоит в том, что каждый S-элемент однозначно соответствует одному A-элементу, все S–A-связи имеют вес, равный +1, а порог A-элементов равен 1.

Структура однослойного персептрона представлена на рис. 4.7, где видно, что каждый сенсор S передает сигнал только одному A-элементу.

Однако при этом допустима ситуация, когда на один A-элемент поступают сигналы от нескольких S-элементов.

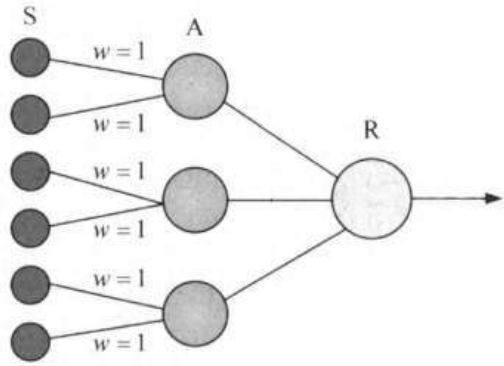


Рис. 4.7. Структура однослойного персептрона

В однослойном персептроне все S–A-связи всегда имеют вес, равный единице ($w = 1$), а порог A-элементов всегда равен +1. При этом сенсоры могут подавать сигнал, равный только 0 или 1.

Рассмотрим первый S-элемент на рис. 4.7. Пусть он генерирует сигнал, равный 1. Сигнал проходит по S–A-связи и не изменяется, т. к. любое число, умноженное на 1, равно самому себе. Порог любого A-элемента равен +1. Так как сенсор произвел сигнал, равный 1, то A-элемент однозначно возбудился. Это означает, что он выдал сигнал, равный +1 (т. к. он тоже может генерировать только 1 или 0 на своем выходе). Далее этот единичный сигнал умножается на произвольный вес A–R-связи и попадает в R-элемент, который суммирует все поступившие на него взвешенные сигналы. Если полученная сумма превышает порог R-элемента, то на выходе он выдаст +1, если не превышает, то выход будет равен –1.

Если не принимать во внимание сенсорные элементы и S–A-связи, то мы повторили описание схемы работы искусственного нейрона. И это неслучайно. Однослойный персептрон действительно представляет собой искусственный нейрон с небольшим отличием. В отличие от искусственного нейрона, у однослойного персептрона входные сигналы могут принимать фиксированные значения: 0 или 1. У искусственного нейрона на вход можно подавать любые значения. Место искусственного нейрона в структуре персептрона показано на рис. 4.8.

В персептроне R-элементы суммируют взвешенные входные сигналы и, если взвешенная сумма выше некоторого порога, выдают 1. Иначе выходы R-элементов были бы равны –1.

Нетрудно догадаться, что такое поведение легко задается функцией активации под названием «функция единичного скачка», которую мы уже рассматривали в *главе 3*. Отличие заключается в том, что функция единичного скачка выдает 0, если порог не превышен, а здесь выдает –1, но это не существенно.

Таким образом, становится ясно, что часть однослойного персептрона (выделена штриховым прямоугольником на рис. 4.8) можно представить в виде искусственного нейрона, но ни в коем случае не путайте эти два понятия. Во-первых, в персептроне имеется слой S-элементов, которого в искусственном нейроне просто нет. Во-вторых, в однослойном персептроне S–A-элементы могут принимать только фиксированные значения 0 и 1, тогда как в искусственном нейроне таких ограничений нет.

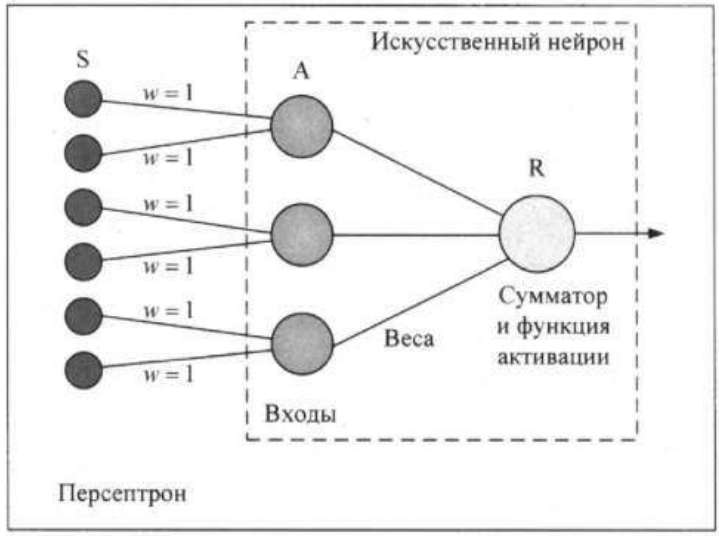


Рис. 4.8. Искусственный нейрон в структуре персептрона

Теперь программно реализуем персептрон на языке Python, но сначала сформируем алгоритм работы программы (рис. 4.9). В листинге 4.1 приведен программный код на Python, реализующий этот алгоритм.

Листинг 4.1

```
# Искусственный нейрон (персептрон)
def perceptron(Sensor):
    b = 7  # Порог функции активации
    s = 0  # Начальное значение суммы
    for i in range(15):  # цикл суммирования сигналов от сенсоров
        s += int(Sensor[i]) * weights[i]

    if s >= b:
        return True  # Сумма превысила порог
    else:
        return False  # Сумма меньше порога
```

В качестве порога срабатывания функции активации здесь задано число 7. Как правильно указать порог для той или иной задачи, будет рассказано позже, когда будут рассматриваться конкретные примеры.

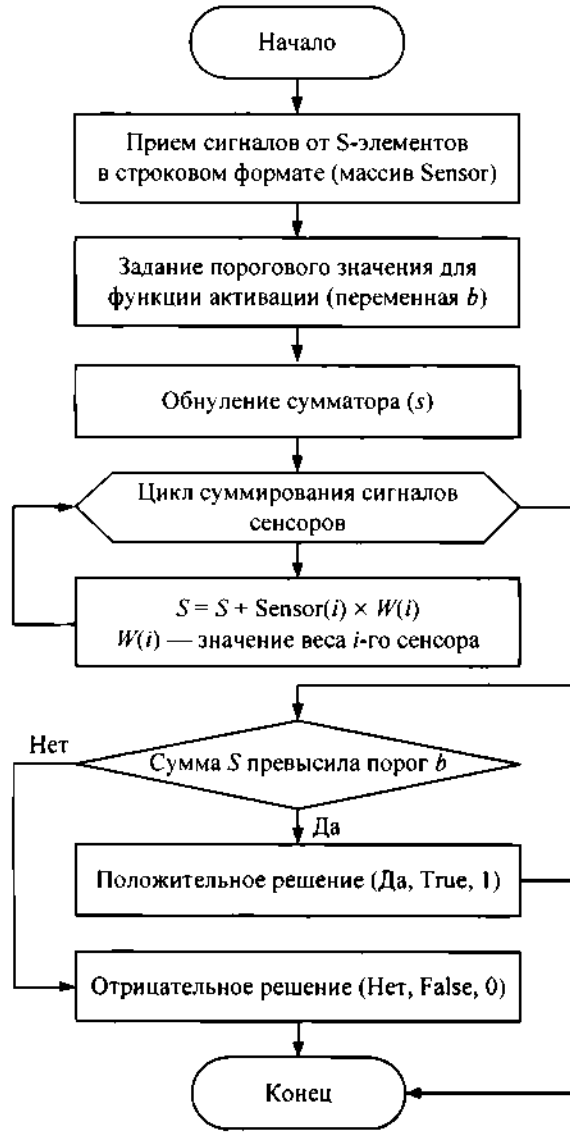


Рис. 4.9. Блок-схема алгоритма работы программы, моделирующей персептрон

Проверим разработку программы. Для этого на вход дадим сигналы от 15 сенсоров в виде двух следующих массивов чисел по 15 значений (нули и единицы) в каждом:

```
num1 = list('001001001001001')
num2 = list('111001111100111')
```

Присвоим весам всех связей значение 1. Мы ранее условились, что в нашей упрощенной модели персептрона веса всех связей одинаковы. Это можно сделать в цикле одним оператором:

```
weights = [1 for i in range(15)]  # Присвоение значений всем связям w=1
```

Теперь нам нужно передать в искусственный нейрон два массива значений сенсоров (num1, num2) и получить результат работы нашего искусственного нейрона. В программном коде это можно сделать следующим образом (листинг 4.2).

Листинг 4.2

```
# Это промежуточный код, он не является рабочим
print(num1)              # Смотрим, какие сигналы от сенсоров получил персептрон
print(perceptron(num1))  # Проверяем, что ответил персептрон
print(num2)              # Смотрим, какие сигналы от сенсоров получил персептрон
print(perceptron(num2))  # Проверяем, что ответил персептрон
```

В листинге 4.3 приведен полный текст программы испытания персептрона.

Листинг 4.3

```
# Модуль Neiron
# Искусственный нейрон (персептрон)
def perceptron(Sensor):
    b = 7  # Порог функции активации
    s = 0  # Начальное значение суммы
    for i in range(15):  # цикл суммирования сигналов от сенсоров
        s += int(Sensor[i]) * weights[i]

    if s >= b:
        return True  # Сумма превысила порог
    else:
        return False  # Сумма меньше порога

# Проверка работы искусственного нейрона (персептрона)
num1 = list('001001001001001')
num2 = list('111001111100111')
weights = [1 for i in range(15)]  # Присвоение значений всем связям w=1

print(num1)              # Смотрим, какие сигналы от сенсоров получил персептрон
print(perceptron(num1))  # Проверяем, что ответил персептрон
print(num2)              # Смотрим, какие сигналы от сенсоров получил персептрон
print(perceptron(num2))  # Проверяем, что ответил персептрон
```

Запустив эту программу на выполнение, получим следующий результат (рис. 4.10).

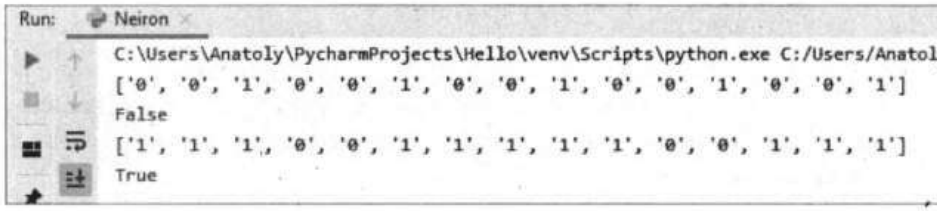


Рис. 4.10. Результаты испытания программы, моделирующей персептрон

В первом случае сумма сигналов, полученных от всех сенсоров, равна 5, что меньше порога 7. На выходе мы получили верный ответ (`False`) — порог не превышен. Во втором случае сумма сигналов, полученных от всех сенсоров, равна 11, что выше порога 7. На выходе мы получили верный ответ (`True`) — порог превышен. Ура! Мы создали первую программу, которая имитирует работу самой упрощенной модели ячейки нашего головного мозга.

4.2.3. Виды персептронов

Теперь рассмотрим различные виды персептронов.

В предыдущем разделе мы уже познакомились с *однослойным персептроном*. Напомним, что это персептрон, каждый S-элемент которого однозначно соответствует одному A-элементу, S–A-связи всегда равны 1, а порог любого A-элемента равен 1. Часть однослойного персептрона соответствует модели искусственного нейрона. Однослойный персептрон может быть и элементарным персептроном, у которого только по одному слою S-, A-, R-элементов.

Под *многослойным персептроном* понимают два разных вида: многослойный персептрон по Розенблатту и многослойный персептрон по Румельхарту.

Многослойный персептрон по Розенблатту — персептрон, у которого имеется более одного слоя A-элементов (рис. 4.11).

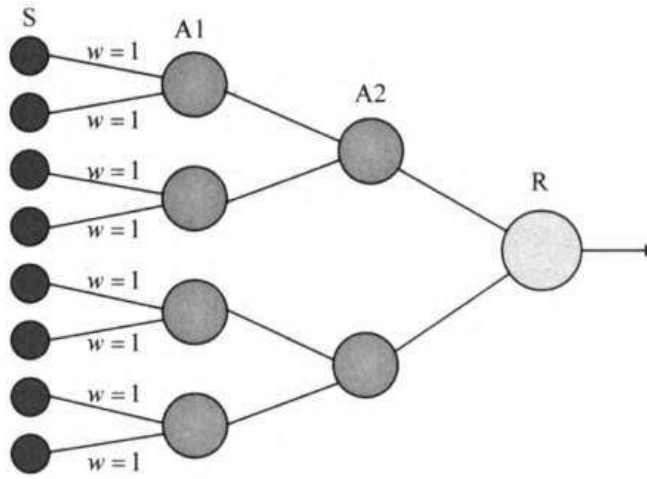


Рис. 4.11. Многослойный персептрон по Розенблатту

Многослойный персептрон по Румельхарту — это многослойный персептрон, у которого обучению подлежат еще и S–A-связи, а обучение производится по методу обратного распространения ошибки. То есть многослойный персептрон по Румельхарту является частным случаем многослойного персептрона по Розенблатту с двумя отличительными особенностями:

- S–A-связи могут иметь произвольные веса и обучаться наравне с A–R-связями;
- обучение производится по специальному алгоритму, который называется *обучением по методу обратного распространения ошибки*.

Этот метод является краеугольным камнем обучения всех многослойных нейронных сетей. Во многом благодаря ему возобновился интерес к нейронным сетям. Но обсуждать мы его будем в других главах. А пока мы разобрались с видами персептронов. Далее в этой главе, чтобы облегчить понимание материала, будет рассматриваться только *однослойный персептрон с одним скрытым слоем*.

4.3. Роль персептронов в нейронных сетях

Какие же задачи решает персептрон? Прежде чем начать практическую реализацию элементов нейронной сети в виде программного кода, нужно четко определить, какие задачи персептрон способен решать, а какие — нет. Мы рассмотрим простые практические примеры, позволяющие понять, возможно ли использовать персептрон в той или иной задаче. Это крайне важно, поскольку персептрон как простейший вид нейронной сети не так уж много и умеет.

Персептроны очень хорошо решают задачи классификации, или разделения объектов на группы. Что это означает? Если у вас есть группы объектов (например, кошки и собаки, светофоры и дорожные знаки), то персептрон после обучения сможет указывать, к какой группе относится объект. То есть если на вход персептрона подать фотографию собаки (неважно, какой породы, в какой позе она находится или с какого ракурса сфотографирована), на выходе мы получим однозначный ответ — это собака. Если на изображении с видеокамеры беспилотного автомобиля появится дорожный знак, то он будет выделен из массы других объектов. Мало того, будет определено, какой это знак и что нужно делать в зоне его действия.

Понятие «решать задачи очень хорошо» весьма растяжимое. А насколько хорошо? Розенблатт доказал несколько теорем, суть которых мы попытаемся уяснить на максимально простых примерах.

Первая теорема Розенблатта доказывает существование элементарного персептрона, способного выполнить любую классификацию заданного множества черно-белых изображений, т. е. она показывает, что персептрон является универсальным устройством для решения любой задачи классификации изображений. Рассмотрим сущность этого вывода на конкретном примере.

Матрица цифрового фотоаппарата представляет собой множество сенсоров, которые улавливают световой сигнал. Если имеются поле сенсоров (матрица) и какая-то классификация, зависящая от того, какой сигнал получен от каждого элемента матрицы, то в соответствии с этой классификацией можно определить, какой объект попал в поле зрения цифрового фотоаппарата.

Рассмотрим только что приведенное утверждение более детально, при этом ведем два понятия: поле сенсоров и классификация объектов. Под *полем сенсоров* будем понимать множество S-элементов. Под *классификацией* — предложенные нами классы или группы объектов (например, кошки или собаки). Все объекты в одном

классе (или группе) имеют некоторые общие признаки, совокупность которых соответствует определенному сочетанию S-элементов.

А как люди могут различать различные объекты? Они этому учатся с детства. Рассмотрим, как родители будут учить маленького ребенка отличать кошку от собаки. Новорожденный, в отличие от родителей, понятия не имеет ни о кошках, ни о собаках. Для того чтобы научить ребенка отличать друг от друга эти два объекта, родители будут показывать ему разные картинки и говорить: «На этой картинке кошка, а вот на этой картинке собака». В результате таких уроков после многократного просмотра десятка картинок (назовем их обучающей выборкой) ребенок выделит на них ряд признаков, по которым можно кошку отличить от собаки, и запомнит эти признаки. После предварительного обучения родители проведут контроль полученных знаний. Ребенку будет показан другой набор картинок с собаками и кошками (назовем их тестовой выборкой), которые ребенок должен самостоятельно разделить на две кучки: в одну сложить всех кошек, в другую — всех собак. Это и есть процесс классификации объектов, т. е. разделения их на группы, в каждой из которых все объекты имеют общие признаки. Если ребенок справился с этим тестовым заданием, значит, его обучение прошло успешно, и он может эти знания применять на практике. Через какое-то время, увидев любую кошку или собаку (даже ту, которую он не видел ни среди обучающих, ни среди тестовых картинок), он выделит в ней запомненные признаки, найдет в памяти соответствие комбинации этих признаков тому или иному объекту, сделает заключение и скажет: «Это кошка, а это собака».

Аналогичные процедуры мы должны совершить и с персептроном. Допустим, мы решили создать персептрон, который будет различать кошек и собак. Изначально персептрон, как и маленький ребенок, этого делать не умеет. С «новорожденным» персептроном мы должны выполнить те же действия, что и родители с маленьким ребенком, — научить. На первый взгляд кажется, что эта задача достаточно трудная или даже невыполнимая. Но ее можно решить, если существенно упростить саму задачу и разложить ее на последовательность простых шагов. Персептрон, как и ребенок, должен знать, по каким признакам можно отделить кошек от собак. Их достаточно много, но для упрощения задачи нашему персептрону выделим всего три таких признака. Ребенок при обучении обратил внимание, что собаки намного выше кошек, и у них длинные лапы, а у кошек лапы короткие. На показанных картинках все собаки имели ровный, однотонный окрас шерсти, а у кошек шерсть была многоцветной (были разноцветные пятна и полосы). Головы у кошек имеют округлую форму, а у собак головы сильно вытянуты. По этим трем характерным признакам наш новорожденный персептрон и будет учиться различать кошек и собак.

Сформируем у персептрона три сенсора (три S-элемента): длина лап — S1, окрас — S2, форма головы — S3. Так как S-элементы могут принимать значения 0 или 1, то условимся, что сенсоры будут принимать следующие значения:

- S1 = 1 (лапы длинные), S1 = 0 (лапы короткие);
- S2 = 1 (шерсть имеет ровный, однотонный окрас), S2 = 0 (шерсть разноцветная);
- S3 = 1 (голова имеет вытянутую форму), S3 = 0 (голова имеет округлую форму).

Вот мы и получили сенсорное поле. Его можно представить в виде множества возможных значений 0 и 1 у каждого S-элемента. В нашем примере для собаки идеальный набор выходов S-элементов — {1, 1, 1}, т. е. длинные лапы, шерсть имеет однотонный окрас, голова — вытянутую форму. Для кошки идеальный набор выходов S-элементов — {0, 0, 0}, т. е. короткие лапы, шерсть разноцветная, голова имеет округлую форму (рис. 4.12).

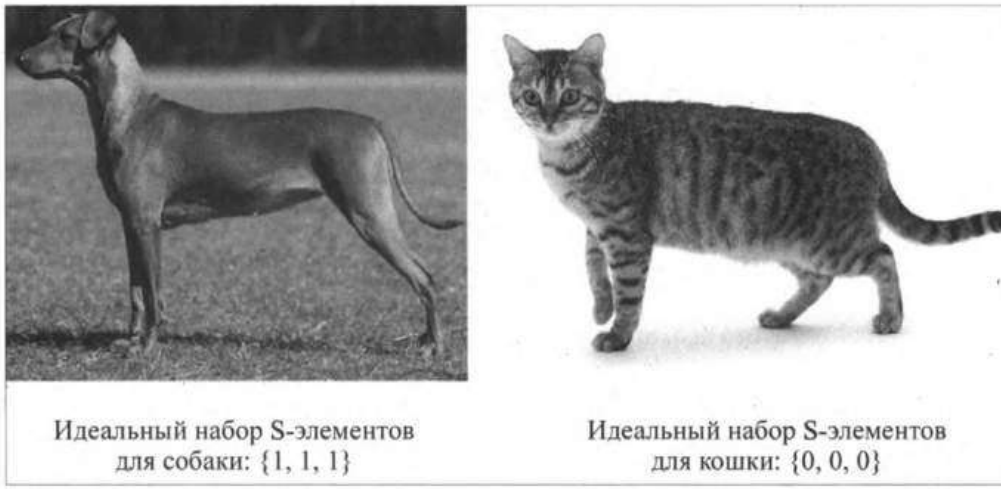


Рис. 4.12. Идеальные наборы выходов S-элементов, отличающих собаку от кошки

Набор сенсоров может иметь совершенно разный смысл. В нашем примере набор выходов сенсоров описывает два объекта: кошку или собаку. Мы фактически создали некоторую классификацию объектов — сформировали два класса объектов. С точки зрения математики сущность персептрона можно описать следующим образом: это функция, которая принимает набор выходов S-элементов, а ее выходным значением является 0 или 1. В нашем примере эти значения функции имеют следующий физический смысл: 0 — это кошка, 1 — это собака.

Из приведенной ранее теоремы следует, что значение, полученное от множества персептронов, правильно проводящих классификацию, не является пустым. На практике можно сформировать любой набор S-элементов и на их основе создать любую классификацию. И множество «решений» все равно не будет пустым, т. е. всегда будет получен некий однозначный результат. Это означает, что теоретически персептроны способны решать любую задачу, связанную с классификацией объектов, т. е. разделения различных объектов на группы. В нашем примере мы разделяли кошек и собак.

Однако есть два важных замечания:

- здесь речь идет об элементарных персептронах;
- объекты, которые мы хотим классифицировать, должны обладать свойством линейной разделимости, т. е. они должны иметь набор уникальных свойств, на основе которых в принципе возможно один объект отличить от другого.

Но есть и *вторая теорема, доказанная Фрэнком Розенблаттом*. В ней утверждается, что если имеется персептрон (т. е. поле сенсоров и какая-то классификация, связанная с ним), то процесс обучения персептрона методом коррекции ошибки, независимо от начального состояния весовых коэффициентов и последовательности появления стимулов, всегда приведет к достижению решения за конечный промежуток времени. Под произвольным исходным состоянием тут понимается персептрон с произвольными S–A- и A–R-весами связей. Под решением в теореме понимается персептрон с определенными весами, успешно решающий нашу задачу на классификацию.

Эта теорема гарантирует успех решения задач на классификацию. Теперь нам известно, что мы всегда сможем решить нашу задачу за конечный промежуток времени. Единственный нюанс заключается в том, что никто не говорит о длительности «конечного промежутка времени». Секунда, минута, час, год или тысяча лет?

Обе теоремы имеют доказательства. Однако нам нет необходимости углубляться в эти математические дебри. Поверим Фрэнку Розенблатту и будем пользоваться его доказанными заключениями. Тем более что эти заключения были основаны не только на уровне математических выражений, но и реализованы на практике, когда Розенблатт создал первый в мире нейронный компьютер «Марк-1», который можно было научить решать практические задачи, не поддающиеся традиционной алгоритмизации.

Итак, элементарные персептроны гарантированно решают задачи на классификацию линейно разделимых объектов. Для их обучения используется метод коррекции ошибок — один из алгоритмов изменения весов. В этой главе мы его также изучим более детально.

4.4. Линейная разделимость объектов

Что же такое линейная разделимость объектов? С точки зрения математики, это выражается следующим образом: два множества точек в двумерном пространстве называются *линейно разделимыми*, если они могут быть полностью отделены друг от друга единственной прямой. Визуальная демонстрация этого определения представлена на рис. 4.13.

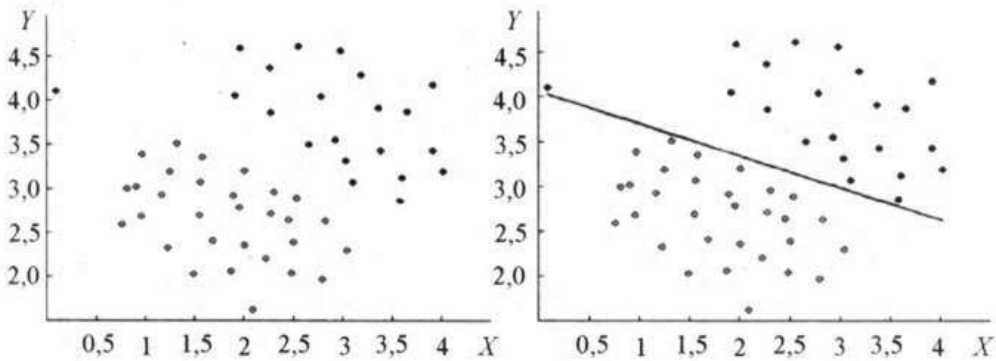


Рис. 4.13. Демонстрация определения линейной разделимости объектов

В левой части рисунка мы видим набор точек. Каждая точка на координатной плоскости имеет собственные координаты (X, Y) и цвет. При этом мы имеем две группы (два класса) точек: синие (вверху) и зеленые (внизу). Можем ли мы каким-то образом отделить одну группу точек от другой? Попробуем провести прямую линию так, чтобы группа синих точек была расположена по одну сторону от прямой линии, а группа зеленых точек — по другую сторону. На правой части рисунка видно, что нам это удалось. То есть мы можем сказать, что этот массив точек имеет линейную разделимость. В дальнейшем, имея параметры уравнения этой линии, мы сможем по координате точки определить ее цвет. Как это можно сделать, показано на рис. 4.14.

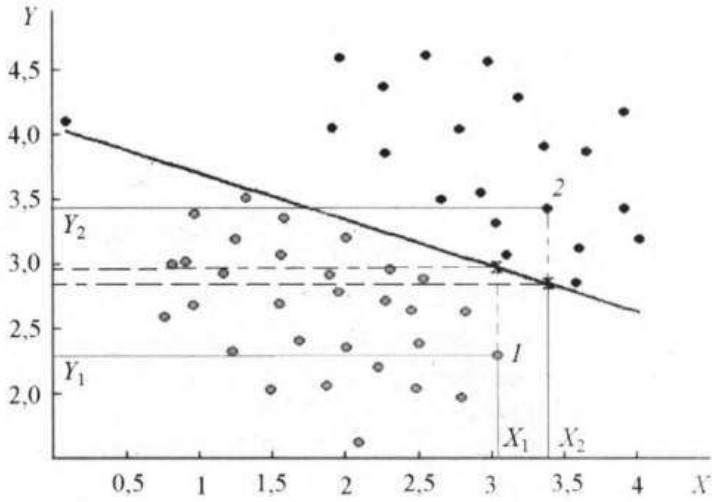


Рис. 4.14. Определение цвета точки по ее координатам

Рассмотрим точку *1* с координатами (X_1, Y_1). Ее координата Y_1 меньше, чем координата $Y_{л1}$ такой же точки, расположенной на линии, разделяющей объекты. Значит, наша точка *1* зеленого цвета. Теперь рассмотрим точку *2* с координатами (X_2, Y_2). Ее координата Y_2 больше, чем координата $Y_{л2}$ такой же точки, расположенной на линии, разделяющей объекты. Значит, наша точка *2* синего цвета.

Давайте теперь визуализируем понятие линейной разделимости объектов на более реальных объектах: на кошках и собаках. Для того чтобы их различать (классифицировать), с целью упрощения задачи введем всего два параметра, которые будут характеризовать наши объекты: рост и вес. Для обеспечения разделимости данных объектов обозначим условие, что все собаки имеют бóльший рост и вес, а кошки — меньший. При таком условии два объекта, имеющие одинаковые рост и вес, не могут быть одновременно и кошкой, и собакой. Наша будущая нейронная сеть должна по значению только этих двух параметров принять решение: кошка это или собака. Для чего ей потребуются всего два S-элемента, определяющих рост и вес.

Поскольку рост и вес животных могут быть совершенно разными и имеют значения, отличные от нуля и единицы, то давайте немного отойдем от принятого в мо-

дели персептрона определения S-элемента. Представим, что наши сенсорные элементы могут выдавать не только 0 (низкий рост, малый вес) или 1 (высокий рост, большой вес), но и реальные значения роста и веса. Так как у нас есть два сенсорных элемента, то их значения можно расположить вдоль двух координатных осей. На получившейся координатной плоскости можно размещать точки, каждая из которых будет характеризовать одну из кошек или собак. Как вы знаете, у каждой точки есть две координаты — в нашем случае этими координатами будут рост и вес. Так вот, задача персептрона здесь — провести некоторую прямую, которая максимально точно разделит два множества точек, — в нашем случае кошек и собак (рис. 4.15).

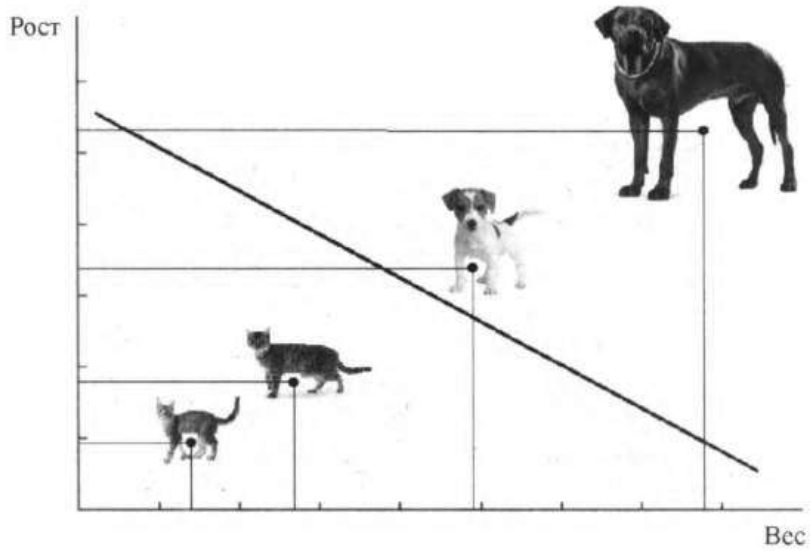


Рис. 4.15. Разделение кошек и собак по признакам роста и веса

Вы заметили, что на рис. 4.15 в качестве разделителя используется прямая линия? А ведь мы могли бы в качестве разделителя использовать кривую или ломаную линию. Но с прямой линией с точки зрения математической формулы работать проще. Так что в этой главе мы будем рассматривать задачи разделения (классификации) объектов на основе их *линейной* разделимости (от слова «линия»). Именно такие задачи способны решать элементарные персептроны.

Однако не всегда множество точек, расположенных на плоскости, можно разделить прямой линией. И если два множества объектов в пространстве невозможно разделить прямой линией или плоскостью, то такие множества называются *линейно неразделимыми*. Пример линейно неразделимого множества представлен на рис. 4.16.

Немного усложним условия нашей задачи разделения кошек и собак и приблизим ее к реальности. В действительности, есть очень большие кошки и очень маленькие собаки. Причем рост и вес маленьких декоративных собак гораздо меньше роста и веса больших сибирских котов (рис. 4.17).

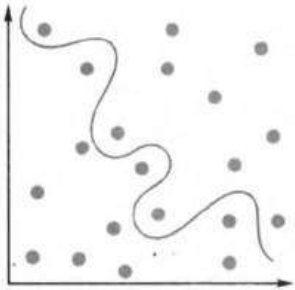


Рис. 4.16. Пример линейно неразделимых множеств

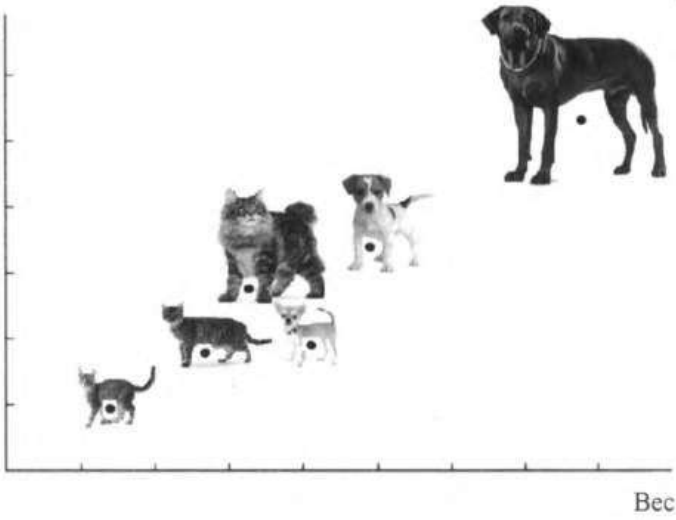


Рис. 4.17. Линейно неразделимое множество на примере классификации кошек и собак по росту и весу

На рис. 4.17 показано, что, как бы мы ни пытались провести прямую линию, отделяющую кошек от собак, у нас ничего не получится. Такие линейно неразделимые множества в этой главе мы рассматривать не будем, поскольку теоремы о сходимости элементарных персептронов на них не распространяются. Однако это совершенно не значит, что в этих условиях наша задача не имеет решения. Введем еще один признак, отличающий кошек от собак, — сенсор S3. Например, способ общения объекта с окружающим миром: кошки мяукают, а собаки лают. Дадим значения сенсору $S3 = 0$ в том случае, если объект издает звук «гав», и значение $S3 = 1$, если объект издает звук «мяу». По этому признаку, если даже два объекта имеют одинаковые рост и вес, мы всегда сможем отличить кошку от собаки.

Итак, если у нас может быть больше признаков, то может быть больше и сенсорных элементов. В случае трех признаков будут три S-элемента, т. е. мы имеем уже трехмерное пространство. Тогда между точками, каждая из которых соответствует определенным значениям всех трех S-элементов, проводилась бы плоскость. И так далее. В общем случае для множества S-элементов в n-мерном пространстве строится так называемая гиперплоскость с размерностью $n-1$.

4.5. Решение задач классификации объектов на основе логических функций

Часто на практике возникает необходимость решения логических задач, т. е. требуется принять решение на основе анализа множества произошедших событий.

Рассмотрим пример. Ваша программа управляет беспилотным автомобилем. В блок управления автомобилем поступают сигналы от двух сенсоров: от радара, который определяет наличие объектов перед автомобилем (пешеходы), и от видеокамеры,

которая показывает сигналы светофора. Когда автомобиль подъезжает к перекрестку с пешеходным переходом и горит красный свет светофора, он останавливается и стоит на месте, есть на переходе пешеходы (рис. 4.18, *а*) или он пуст (рис. 4.18, *б*). Автомобиль может начать движение только в том случае, если горит зеленый свет и перед автомобилем никого нет (рис. 4.18, *в*). Если горит зеленый свет, но на переходе есть пешеходы, то автомобиль стоит на месте (рис. 4.18, *г*). Другими словами, автомобиль может начать движение только в том случае, если пешеходный переход пуст и горит зеленый свет светофора, как и показано на рис. 4.18, *в*.

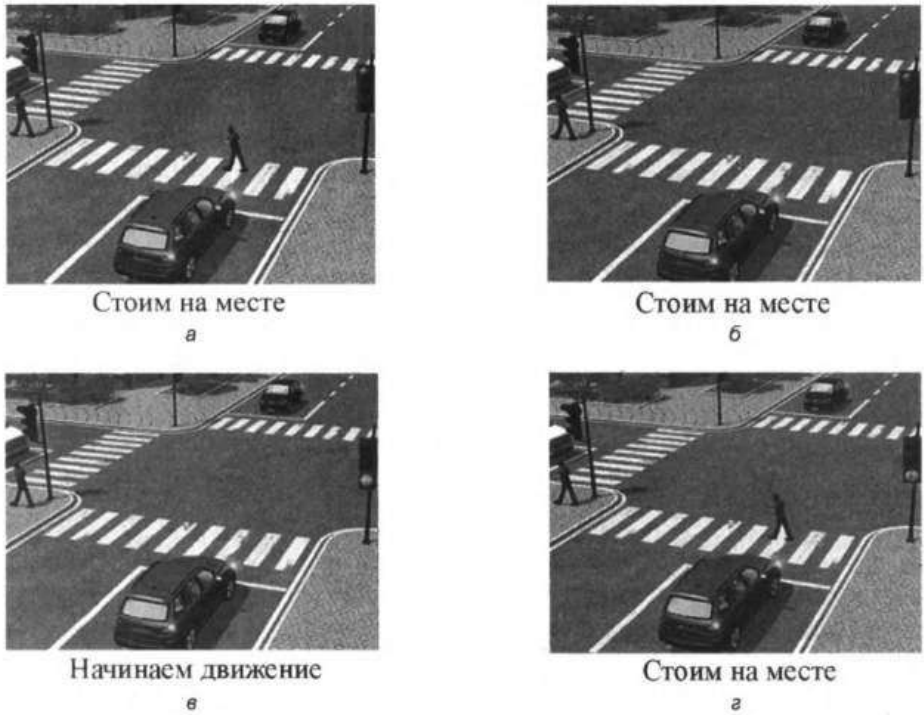


Рис. 4.18. Условия, при которых автомобиль может начать движение от светофора

В этом примере нужно проаннотировать информацию, поступившую в блок управления, и в зависимости от соблюдения или несоблюдения некоторых условий принять то или иное решение. Выразим эти действия в несколько ином виде:

- **ЕСЛИ** свет красный **И** есть пешеходы, **ТО** стоим на месте;
- **ЕСЛИ** свет зеленый **И** есть пешеходы, **ТО** стоим на месте;
- **ЕСЛИ** свет красный **И** нет пешеходов, **ТО** стоим на месте;
- **ЕСЛИ** свет зеленый **И** нет пешеходов, **ТО** начинаем движение.

Здесь нужно обратить внимание на ключевые слова, выделенные жирным шрифтом, — **ЕСЛИ**, **И**, **ТО**. Они говорят о том, что нужно выполнить те или иные действия в зависимости от соблюдения или несоблюдения некоторых условий. Такие задачи нужно решать с использованием *логических функций*.

Что такое логические функции? Это функции от некоторого числа переменных, причем как сами переменные, так и значения логических функций могут принимать только фиксированные (дискретные) значения: 1 или 0 (истина или ложь). Рассмотрим типы логических функций: И и ИЛИ.

Для лучшего понимания принципа работы логической функции используют *таблицы истинности*, где в первых двух столбцах располагают возможные комбинации входных переменных, а в третьем столбце — итоговое значение функции. В табл. 4.1 приведена таблица истинности логической функции И.

***Таблица 4.1.** Таблица истинности логической функции И*

X_1	X_2	X_1 И X_2
0	0	0
1	0	0
0	1	0
1	1	1

Применительно к нашему примеру с автомобилем нужно задать следующие значения входных параметров для этой логической функции: красный свет светофора — 0, зеленый свет светофора — 1, есть пешеходы на переходе — 0, нет пешеходов — 1. Тогда формулировка условий нашей задачи будет выглядеть так:

- **ЕСЛИ** свет красный (0) **И** есть пешеходы (0), **ТО** стоим на месте (0);
- **ЕСЛИ** свет зеленый (1) **И** есть пешеходы (0), **ТО** стоим на месте (0);
- **ЕСЛИ** свет красный (0) **И** нет пешеходов (1), **ТО** стоим на месте (0);
- **ЕСЛИ** свет зеленый (1) **И** нет пешеходов (1), **ТО** начинаем движение (1).

То есть мы можем дать автомобилю команду на начало движения только в том случае, если на выходе из логической функции получим значение 1.

Есть еще логическая функция — ИЛИ. Вернемся к нашему беспилотному автомобилю. Автомобиль подъезжает к перекрестку со светофором. Он должен остановиться перед перекрестком, если загорелся красный цвет (рис. 4.19, *а*). Однако он должен будет затормозить даже при зеленом сигнале светофора, если пешеход, нарушив правила, неожиданно выбежал на дорогу (рис. 4.19, *б*). То есть автомобиль должен остановиться, если: или горит красный сигнал светофора (рис. 4.19, *а* и *в*), или пешеход присутствует на пешеходном переходе (см. рис. 4.19, *б*). Только если горит зеленый свет и пешеходный перекресток пуст, автомобиль продолжает движение (рис. 4.19, *г*).

Тут должно вступить в работу логическое ИЛИ. Значением функции логического ИЛИ будет 0 только тогда, когда значения обеих переменных тоже равны 0. Во всех остальных случаях значение этой логической функции равно 1. Таблица истинности для логического ИЛИ представлена в табл. 4.2.

Для нашего примера с автомобилем нужно задать следующие значения входных параметров для этой логической функции: зеленый свет светофора — 0, красный

Останавливаемся

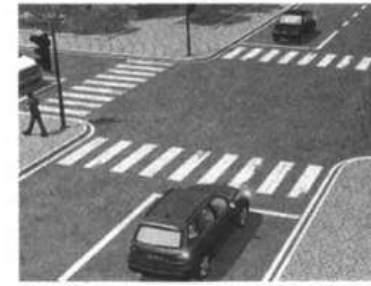
а

Останавливаемся

б

Останавливаемся

в



Продолжаем движение через перекресток

г

Рис. 4.19. Условия, при которых автомобиль может проехать перекресток без остановки

Таблица 4.2. *Таблица истинности логической функции ИЛИ*

X_1	X_2	X_1 **ИЛИ** X_2
0	0	0
1	0	1
0	1	1
1	1	1

свет светофора — 1, нет пешеходов на переходе — 0, есть пешеходы — 1. Тогда формулировка условий остановки автомобиля перед перекрестком будет выглядеть так:

- **ЕСЛИ ИЛИ** свет зеленый (0), **ИЛИ** нет пешеходов (0), **ТО** продолжаем движение (0);
- **ЕСЛИ ИЛИ** свет красный (1), **ИЛИ** нет пешеходов (0), **ТО** тормозим (1);
- **ЕСЛИ ИЛИ** свет зеленый (0), **ИЛИ** есть пешеходы (1), **ТО** тормозим (1);
- **ЕСЛИ ИЛИ** свет красный (1), **ИЛИ** есть пешеходы (1), **ТО** тормозим (1).

То есть мы можем дать автомобилю команду на продолжение движения только в том случае, если на выходе из логической функции получим значение 0. Во всех других случаях автомобиль должен остановиться.

Логические функции очень наглядно иллюстрируют возможность их использования для классификации объектов. Любая такая функция принимает на вход два аргумента:

- **ЕСЛИ** «аргумент 1» **И** «аргумент 2», **ТО** «значение функции»;
- **ЕСЛИ** «аргумент 1» **ИЛИ** «аргумент 2», **ТО** «значение функции».

Любая точка на плоскости тоже имеет два параметра: координату X и координату Y. Но логические функции могут принимать только дискретные аргументы (0 или 1). В итоге получается, что для изображения любой логической функции на плоскости достаточно четырех точек с координатами (0, 0), (1, 0), (0, 1), (1, 1). Вот как это выглядит на координатной сетке (рис. 4.20).

Рассмотрим логическую функцию И. Ее значение равно нулю для любого набора входных аргументов, кроме набора $X_1 = 1, X_2 = 1$ (см. табл. 4.1).

Для такой функции задачу линейной классификации можно выразить следующим образом: «Имеются 4 точки на плоскости. Необходимо провести прямую так, чтобы по одну сторону от прямой оказались точки, для которых значения логической функции равны 1, а по другую — точки, для которых эти значения равны 0».

В случае с логической функцией И эту прямую, например, можно провести так, как показано на рис. 4.21.

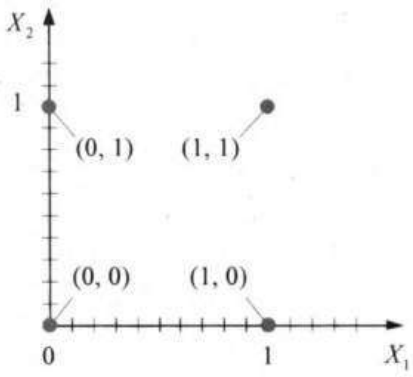


Рис. 4.20. Возможные положения значений логической функции на сетке координат

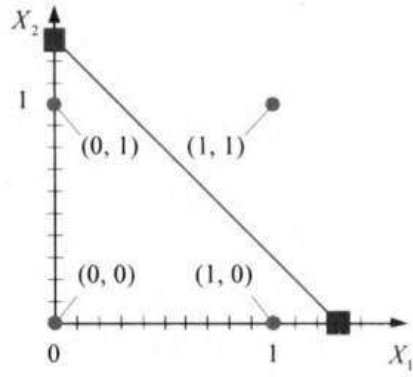


Рис. 4.21. Пример использования логической функции И для линейной классификации объектов

Все точки, находящиеся *под* этой прямой, приводят к нулевому значению функции. Единственная точка *над* этой прямой приводит к значению логического И, равному 1.

Похожим образом ведет себя логическая функция ИЛИ. Ее значение равно 1 для любого набора входных аргументов, кроме набора $X_1 = 0, X_2 = 0$ (см. табл. 4.2).

В случае с логической функцией ИЛИ разделяющую прямую линию можно провести так, как показано на рис. 4.22.

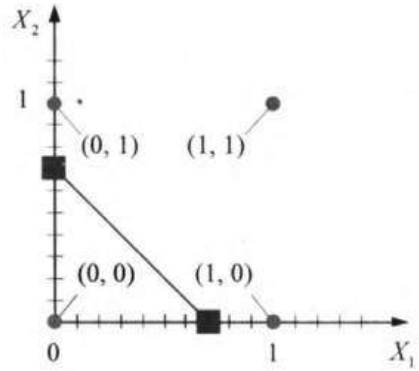


Рис. 4.22. Пример использования логической функции ИЛИ для линейной классификации объектов

На этом мы завершаем рассмотрение задач классификации. Материалы этого раздела дают общее представление о возможности использования математических методов для классификации объектов. На практике достаточно большое количество задач можно переформулировать, представив их виде задач на классификацию, и решить с использованием искусственного интеллекта.

Теперь мы переходим к краеугольному камню нейронных сетей — к их обучению. Ведь без способности обучаться их интеллект был бы на уровне новорожденного.

4.6. Урок 1. Учим персептрон понимать изображения

Здесь мы более подробно рассмотрим вопрос обучения нейронной сети. Что это такое? Каким образом это происходит?

Искусственная нейронная сеть — это совокупность искусственных нейронов. Давайте с помощью программных средств создадим 100 нейронов и соединим их друг с другом. Теперь подадим на вход сети набор входных сигналов и посмотрим, что получилось на выходе. Да, что-то получится. Но что конкретно получится, что именно это значит, какое решение нужно принять — совершенно непонятно. Такая сеть похожа на новорожденного — он есть, но пока еще ничего не умеет делать, он даже не знает, что вообще можно делать в этом мире. Родители должны всему его научить. Так же ведет себя и новоиспеченная нейронная сеть — она не знает ни того, что нужно делать, ни того, как вообще можно что-то делать. Она тоже должна всему учиться. Для начала сеть должна понять, какую задачу ей нужно решить, и только потом ее надо научить способам решения этой задачи.

Какую задачу надо решать нашему искусственному интеллекту, частично станет понятно после того, как мы сообщим сети набор входных сигналов, и то, что мы должны получить на выходе. А вот как преобразовать поданные входные сигналы в правильное решение, нам нужно будет ее научить, опираясь на имеющиеся у нас опыт и знания. Процесс обучения сети будет заключаться в том, что мы будем менять какие-то параметры сети до тех пор, пока входной сигнал не преобразуется в нужный нам выходной.

Под *обучением нейронной сети* понимается процесс корректировки весовых коэффициентов связи между элементами сети таким образом, чтобы в результате при поступлении на вход сети определенного сигнала она выдавала нам правильный ответ.

Упростим поставленную перед нейронной сетью задачу и начнем обучение с самого простого случая. Для этого создадим простейший однослойный персептрон с одним скрытым слоем (рис. 4.23).

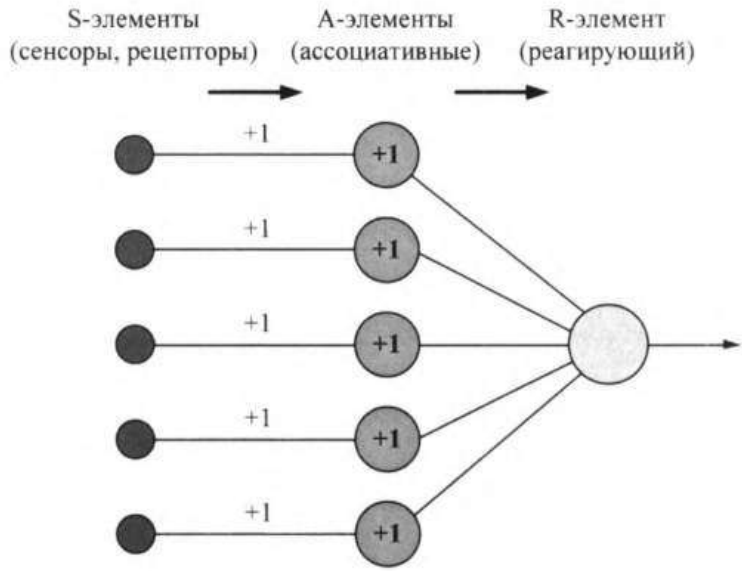


Рис. 4.23. Простейший однослойный персептрон с одним скрытым слоем

Такой простейший персептрон имеет следующую структуру и параметры:

- каждый S-элемент соединен только с одним A-элементом;
- веса всех связей S–A равны +1;
- пороги всех A-элементов равны +1;
- имеется только один R-элемент;
- все A–R-связи могут принимать только целые значения (..., −2, −1, 0, 1, 2, ...).

Однако при такой структуре персептрона получается, что слой A-элементов не выполняет никакие функции. Фактически он эквивалентен S-слою. Поэтому мы делаем еще одно упрощение — убираем слой сенсоров (передаем функции сенсоров A-элементам). Роли сенсоров будут играть ассоциативные A-элементы (рис. 4.24).

Итак, мы до предела упростили однослойный персептрон с одним скрытым слоем. Это может показаться странным, но даже в таком виде персептрон будет работать, решать задачи на классификацию и ряд других задач. Чему в первую очередь учат детей в школе? На первый взгляд, читать, писать и считать. Однако это не совсем

верно: чтобы читать и считать, нужно знать и понимать, что такое буквы и цифры. То есть детей сначала учат этим понятиям. Давайте также поступим с нашим простейшим, очень юным персептроном, который еще не знает, что такое буква и что такое цифра. Поскольку букв в алфавите 33, а цифр всего 10, то сначала попробуем научить нашего юного персептрона распознавать цифры. Следует отметить, что этот упрощений персептрон полностью копирует модель искусственного нейрона.

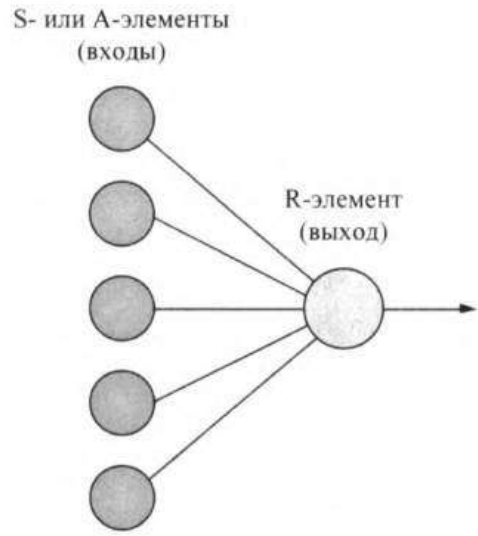


Рис. 4.24. Простейший однослойный персептрон с объединенным входом от S- и A-элементов

4.6.1. Распознавание цифр

Как учат детей распознавать цифры? Им показывают картинки с изображением цифр и говорят: «Это цифра 1. Это цифра 2» и т. д. Так же поступим и с нашим персептроном — будем показывать ему изображения цифр и пояснять, какой цифре соответствует каждое изображение. Как же передать смысл изображения нашему персептрону? Картинка, как известно, состоит из пикселов. Значение каждого пиксела можно передавать одному сенсору персептрона. В этом случае сколько пикселов будет на картинке, столько нужно будет создать S-элементов для персептрона. А какие сигналы будут поступать на каждый сенсор (S-элемент)? Для того чтобы упростить понимание процесса обучения персептрона, упростим и поставленную перед ним задачу. А именно:

- научим его распознавать только черно-белые цифры от 0 до 9;
- все цифры будут одного размера и вписаны черными квадратами в таблицу размером 3 квадрата в ширину и 5 квадратов в высоту;
- персептрон на этом этапе обучения должен распознать только одну цифру.

Далее сформируем обучающую выборку, т. е. создадим набор картинок, которые мы будем демонстрировать персептрону на этапе обучения (рис. 4.25).

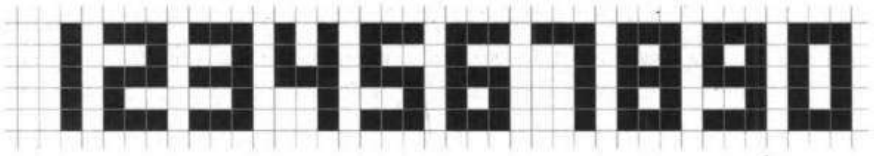


Рис. 4.25. Изображения цифр для обучения персептрона

Для решения нашей задачи потребуется 15 S-элементов (они же A-элементы), каждый из которых будет получать сигнал от одного квадратика таблицы размером 3×5. Черный цвет квадрата будет соответствовать возбуждению S-элемента. Белый цвет выхода — нахождению сенсора в состоянии покоя. На рис. 4.26 показано, в каком состоянии окажутся все 15 сенсорных элементов в том случае, если им будет предоставлена цифра 4 из обучающей выборки.

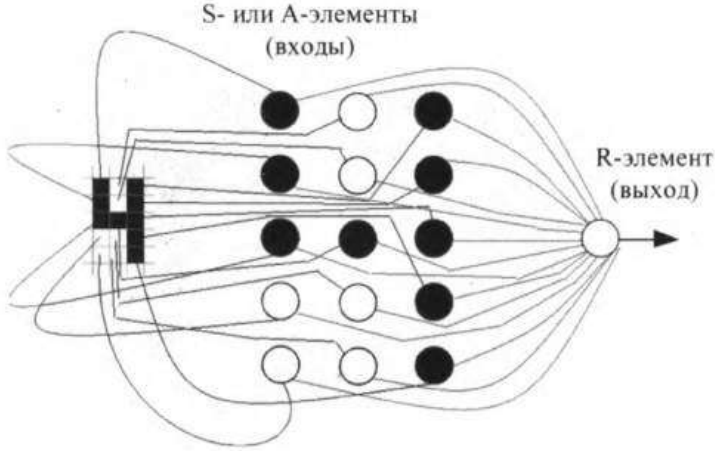


Рис. 4.26. Состояние сенсорных элементов при получении сигналов от тестового изображения цифры 4

Для того чтобы работать с персептроном, мы должны подавать на его входы сигналы в виде чисел. Условимся, что черный цвет квадрата будет соответствовать возбуждению S-элемента (значение такого сигнала будет равно 1). Белый цвет квадрата будет соответствовать нахождению сенсора в состоянии покоя (значение такого сигнала будет равно 0). В этом случае состояние сенсоров для четверки в цифровом виде можно представить таблицей, как показано на рис. 4.27.

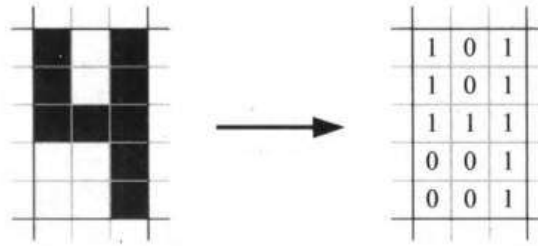


Рис. 4.27. Состояние сенсоров персептрона для цифры 4 в цифровом виде

В нашем примере каждая цифра представляет собой пятнадцать квадратиков, причем только двух возможных цветов. Как мы условились, за белый квадратик отвечает 0, а за черный квадратик — 1. Поэтому все десять цифр от 0 до 9 в табличном формате будут выглядеть так, как представлено на рис. 4.28.

У персептрона данные от сенсоров через А-элементы поступают на сумматор (R-элемент). У нашего упрощенного персептрона А-элементы и S-элементы — это одно и то же. Посмотрим, что мы получим на сумматоре, когда он примет все сигналы от сенсоров. Результаты работы сумматора представлены в табл. 4.3.

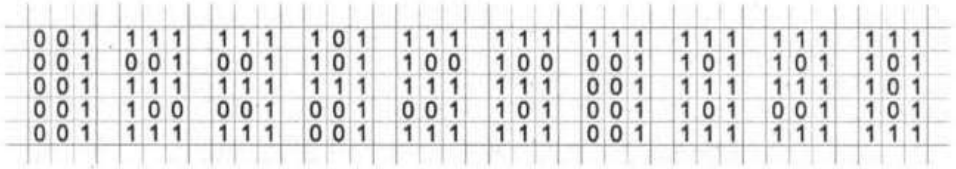


Рис. 4.28. Состояние сенсоров персептрона для 10 цифр в табличном формате

***Таблица 4.3.** Результаты работы сумматора при арифметическом сложении сигналов от сенсоров*

Цифра	Сигналы от сенсоров	Значение R-элемента (арифметическое сложение)
1	0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1	5
2	1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1	11
3	1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1	11
4	1 + 0 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 0 + 0 + 1	9
5	1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1	11
6	1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1	12
7	1 + 1 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1	7
8	1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1	13
9	1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1	12
0	1 + 1 + 1 + 1 + 0 + 1 + 1 + 0 + 1 + 1 + 0 + 1 + 1 + 1 + 1	12

Теперь посмотрим, сможем ли мы, имея только значение R-элемента, определить, какой цифре соответствует это значение? Из табл. 4.3 видно, что не сможем. Например, значение R-элемента 11 соответствует трем цифрам — 2, 3 и 5. Значение R-элемента 12 тоже соответствует трем цифрам — 6, 9 и 0. Значит, по арифметической сумме сигналов от сенсоров сумматор не сможет однозначно распознать цифру. Неужели мы зашли в тупик и не сможем научить персептрон повторить эффективную работу нейрона человека? Вовсе нет.

Давайте поставим сумматору несколько иную задачу — получить не арифметическую сумму сигналов от сенсоров, а символьную. Для записи каждой цифры мы использовали таблицу из 5 строк и 3 столбцов. Просуммируем строки, т. е. объеди-

ним все строки таблицы в одну длинную строку, содержащую сочетание нулей и единиц, которое будет соответствовать каждой цифре. Результат такой работы сумматора представлен в табл. 4.4.

Если проанализировать все строки табл. 4.4, то можно убедиться, что они уникальны — ни в одной строке нет одинаковых комбинаций нулей и единиц. Вот такие строки мы уже сможем использовать в персептроне для распознавания цифр на изображениях. Итак, мы добились своей цели — научили персептрон распознавать цифры. Теперь результаты обучения (содержимое табл. 4.4) нужно сохранить в памяти. В дальнейшем наш обученный персептрон, увидев, например, изображение цифры 4, получит от сенсоров сигналы и преобразует их в строку: 101101111001001. Дальше найдет в своей памяти такую же строку и выдаст ответ: «Такая комбинация нулей и единиц соответствует цифре 4».

Таблица 4.4. *Результаты работы сумматора при символьном сложении сигналов от сенсоров*

Цифра	Сигналы от сенсоров	Значение R-элемента (посимвольное сложение)
1	001 + 001 + 001 + 001 + 001	001001001001001
2	111 + 001 + 111 + 100 + 111	111001111100111
3	111 + 001 + 111 + 001 + 111	111001111001111
4	101 + 101 + 111 + 001 + 001	101101111001001
5	111 + 100 + 111 + 001 + 111	111100111001111
6	111 + 100 + 111 + 101 + 111	111100111101111
7	111 + 001 + 001 + 001 + 001	111001001001001
8	111 + 101 + 111 + 101 + 111	111101111101111
9	111 + 101 + 111 + 001 + 111	111101111001111
0	111 + 101 + 101 + 101 + 111	111101101101111

Однако нужно иметь в виду, что мы сильно упростили условия задачи: на вход персептрону даем цифры одинакового размера (таблицы размером 3×5), и все клетки этих таблиц заполнены в соответствии с образцами. Такой персептрон будет успешно работать лишь в том случае, если ему станут демонстрировать изображения цифр, которые идеально соответствуют обучающей выборке. Но ведь в реальной жизни цифры имеют разный размер, написаны разным шрифтом, и еще в большей степени различается одна и та же цифра, написанная разными людьми от руки. На рис. 4.29 представлены разные варианты изображения цифры 5.

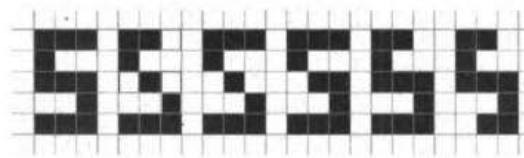


Рис. 4.29. Состояние сенсоров персептрона для возможных изображений цифры 5 в табличном формате

Даже в рамках одного размера цифра 5 имеет небольшие различия в написании. Человек без труда проигнорирует эти отличия и правильно определит значение этой цифры. Но если подать эти изображения на вход нашего упрощенного персептрона, то он не найдет правильного ответа, поскольку в его памяти отсутствуют символьные строки, соответствующие такой комбинации черных и белых точек. Да, наш упрощенный персептрон чему-то научился, но этих уроков оказалось недостаточно. Нужно продолжить обучение. В частности, в упрощенной модели персептрона мы задали такие условия, при которых значения весов всех связей одинаковы и равны 1. А вот если мы теперь научим персептрон самостоятельно подбирать веса связей для различных комбинаций значений сенсоров, то фактически научим его различать разный стиль написания той же цифры 5, и не только этой цифры. Переходим к следующему этапу обучения.

4.7. Урок 2. Учим персептрон подбирать веса связей

Уже было упомянуто в предыдущих разделах, что процесс обучения заключается в подборе весов в цепочке связей между сенсорными элементами и сумматором. Мы знаем, что важность тем или иным входам (в нашем случае S-элементам) придают веса, связывающие их с R-элементом. Чем сильнее вес какой-то связи, тем сильнее этот сенсор влияет на конечный результат. Но как менять вес, в какую сторону, на какую величину? Если мы станем случайным образом изменять веса и смотреть на результаты, то процесс обучения может затянуться на годы. Вот тут мы подошли к главному — мы должны научить нейронную сеть обучаться самостоятельно. Иными словами, мы должны создать программу-учителя, которая будет давать уроки нейронной сети.

Алгоритм обучения нейронных сетей в 1949 году предложил канадский физиолог Дональд Хебб (рис. 4.30). В этом алгоритме использованы так называемые *правила Хебба* (Hebb's rule, Hebbian learning rule). Исследуя работу мозга, физиолог сформулировал следующий постулат: если аксон клетки *A* находится достаточно близко,

Рис. 4.30. Дональд Олдинг Хебб

чтобы возбуждать клетку *B*, и неоднократно или постоянно принимает участие в ее возбуждении, то наблюдается некоторый процесс роста или метаболических изменений в одной или обеих клетках, ведущий к увеличению эффективности *A* как одной из клеток, возбуждающих *B*. На основе этого постулата были сформулированы два правила.

- **Правило 1.** Если сигнал персептрона неверен и равен 0, то необходимо увеличить веса тех входов, на которые была подана единица.
- **Правило 2.** Если сигнал персептрона неверен и равен 1, то необходимо уменьшить веса тех входов, на которые была подана единица.

По сути, на этих двух правилах и основан алгоритм обучения персептронов для решения простейших задач классификации, когда входы могут быть равны только 0 или 1.

Попробуем написать простенькую программу, которая будет обучать нейронную сеть распознавать цифры с использованием этих правил. Распишем для начала, какие шаги нужно сделать, чтобы нейронная сеть научилась правильно распознавать, например, цифру 5:

1. Если нейронная сеть правильно распознала или отвергла цифру 5, то мы ничего не предпринимаем (она справилась с задачей).
2. Если нейронная сеть ошиблась и неверно распознала предъявленную цифру (например, цифру 3 приняла за 5, — первый тип ошибки), то мы уменьшаем веса тех связей, через которые прошел положительный сигнал. Другими словами, нужно уменьшить влияние тех сенсоров, которые возбудились и привели к неправильному выводу.
3. Если нейронная сеть ошиблась и не распознала предъявленную цифру (например, цифру 5 приняла за другую цифру, — второй тип ошибки), то мы должны увеличить все веса, через которые прошел положительный сигнал. Другими словами, нужно увеличить влияние тех сенсоров, которые возбудились и привели к неправильному выводу.

Алгоритм самообучения нейронной сети, пожалуй, самый сложный для понимания, поскольку логика работы этого алгоритма, на первый взгляд, противоречит логике обработки ошибок. Все привыкли, что ошибка — это некий единственный элемент, который может иметь только два состояния: 1 — ошибка есть, 0 — ошибки нет. А здесь при составлении алгоритма работы программы нужно учитывать два типа ошибок, т. е. с точки зрения ошибок нейронная сеть может быть в четырех состояниях: первый тип ошибки есть/нет, второй тип ошибки есть/нет.

Составим алгоритм работы такой программы на примере обучения распознавания цифры 5 (рис. 4.31).

Вот как работает этот алгоритм обучения:

1. Формируем обучающую выборку (набор идеальных сигналов S-сенсоров) от всех цифр от 0 до 9.
2. Получаем двумерный массив размером 10×15. Здесь 10 — количество цифр (от 0 до 9), 15 — количество сенсоров, характеризующих каждую цифру.

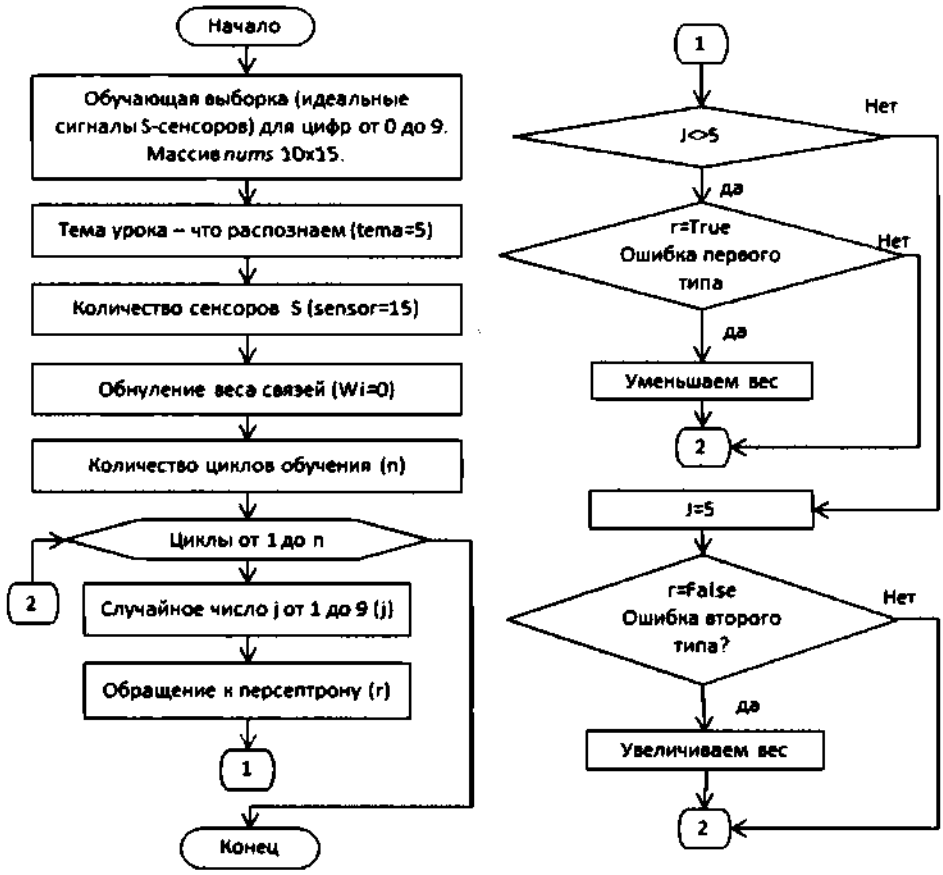


Рис. 4.31. Алгоритм программы обучения персептрона распознавать цифры (на примере цифры 5)

3. Задаем тему урока. В нашем случае тема урока: «Распознавание цифры 5».
4. Обнуляем веса связей для всех 15 сенсоров, которые характеризуют цифры.

   В предыдущем разделе мы определили, что каждая цифра представляет собой таблицу из пяти строк и трех столбцов, в которой каждая ячейка имеет черный или белый цвет (рис. 4.27). Создаем такую же ячейку, в которой будем хранить веса сенсорных элементов. Поскольку наша сеть еще не училась, то она не знает значения этих весов. Поэтому для такого нового ученика в каждую ячейку положим вес, равный 0 (рис. 4.32).

   В этой таблице 15 ячеек — в программе мы можем хранить эти веса в виде массива $W(i)$, состоящего из 15 чисел. Перед началом обучения всему массиву присвоим значение 0.
5. Задаем количество уроков обучения n. Известно, что чем больше уроков посетил ученик, тем лучше он усвоил учебный материал. Вряд ли можно выучить длинное стихотворение с одного раза. Только после многократного прочтения и повторения его можно выучить наизусть. Так же и с нейронной сетью — ей нужно преподать достаточно много уроков.

0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Рис. 4.32. Таблицы с нулевыми значениями весов для сенсорных элементов

6. Организуем цикл, состоящий из *n* уроков.
7. На каждом уроке (в каждом цикле) будем с помощью генератора случайных чисел получать одну цифру (*j*) от 0 до 9 и пытаться угадать — пятерка это или нет. Угадывать будет наш персептрон, которому будет передана эта цифра — *j*. Персептрон вернет ответ — *r*, который может дать только два значения: да — угадал (True), нет — не угадал (False).
8. Далее идет разветвление программы: если *j* не равно пяти, то идем в блок проверки наличия ошибки первого типа, если *j* равно пяти, то идем в блок проверки наличия ошибки второго типа.
9. **Проверки наличия ошибки первого типа**. Если персептрон ошибся — например, на цифру 3, выданную генератором чисел, указал, что это цифра 5, то фиксируется ошибка первого типа. В этом случае вычитаем единицу из всех связей, связанных с возбудившимися сенсорами (S-элементами). То есть мы уменьшаем влияние тех сенсоров, которые возбудились и привели к неправильному выводу (как бы подсказываем ученику, что в этом случае нужно давать отрицательное решение). Если персептрон не ошибся, то делается возврат в заголовок цикла.
10. **Проверки наличия ошибки второго типа**. Если персептрон ошибся — например, на цифру 5, выданную генератором чисел, сказал, что это другая цифра, то фиксируется ошибка второго типа. В этом случае прибавляем единицу ко всем связям, связанным с возбудившимися сенсорами (S-элементами). Другими словами, мы увеличиваем влияние тех сенсоров, которые возбудились и привели к неправильному выводу. То есть мы как бы подсказываем ученику, что в этом случае нужно давать положительное решение. Если персептрон не ошибся, то делается возврат в заголовок цикла.

Почему в алгоритме мы прибавляем или отнимаем именно единицу? Потому что тогда в конце всех уроков мы сможем увидеть, сколько раз ученик правильно угадывал или ошибался в процессе обучения. На самом деле эту величину можно задать любой. Сколько уроков должен посетить наш ученик — это отдельный вопрос, который будет рассматриваться позднее.

Теперь напишем нашу программу «учитель» на языке Python. Если кто-то предпочитает Visual Basic, C, C# или любой другой язык программирования, то можно реализовывать программу и на них. Никаких принципиальных различий нет и алго-

ритм один и тот же. Однако для языка Python имеется большое количество готовых библиотек, реализующих функции элементов нейронных сетей, которые можно просто подключить. Это значительно облегчает работу программистов. Поэтому мы будем использовать Python.

Первыми шагами в программах на Python подключаются необходимые библиотеки. Поскольку мы используем генератор случайных чисел, то для нашей программы нужно импортировать модуль для работы со случайными числами:

```
import random
```

Теперь сформируем обучающую выборку с идеальным изображением цифр — запишем все цифры от 0 до 9 в виде символьной строки (значения таких строк берем из колонки значений R-элементов в табл. 4.4). Строки соответствующего фрагмента программного кода представлены в листинге 4.4.

Листинг 4.4

```
# Это промежуточный код, он не является рабочим
# Обучающая выборка (идеальное изображение цифр от 0 до 9)
num0 = list('111101101101111')
num1 = list('001001001001001')
num2 = list('111001111100111')
num3 = list('111001111001111')
num4 = list('101101111001001')
num5 = list('111100111001111')
num6 = list('111100111101111')
num7 = list('111001001001001')
num8 = list('111101111101111')
num9 = list('111101111001111')
```

Здесь функция `list(*)` позволяет создать список (массив), состоящий из отдельных символов, на которые разбивается длинная строка. Далее все эти 10 цифр упакуем в один общий список `nums` (для быстрого доступа к каждой цифре):

```
# Список всех цифр от 0 до 9 в едином массиве
nums = [num0, num1, num2, num3, num4, num5, num6, num7, num8, num9]
```

Зададим тему уроков (распознавание — какой цифре будем обучать) и количество сенсоров (для нашей задачи мы получаем входной сигнал от 15 сенсоров). Эти параметры можно задать с помощью следующих строк:

```
tema = 5        # какой цифре обучаем
n_sensor = 15   # количество сенсоров
```

Теперь необходимо создать список весов. Так как у нас 15 сенсоров (они же 15 A-элементов), соединенных с одним R-элементом, то нам потребуется 15 связей. Пусть все веса на момент начала работы программы будут равны 0. Для этой цели можно использовать обычный цикл, повторяющийся 15 раз, с присвоением всем элементам нашего списка весов значения 0:

```
# Инициализация весов для связей сенсоров с сумматором
weights = []
for i in range(15):
    weights.append(0)
```

Но можно использовать и генератор списков, тогда приведенный код запишется одной строкой:

```
weights = [0 for i in range(15)] # Обнуление весов
```

Теперь создадим простейшую функцию с именем `perceptron`, которая будет вычислять взвешенную сумму и сравнивать ее с порогом. Фактически эта функция представляет собой единичный шаг работы нашей нейронной сети (фрагмент этой программы приведен в листинге 4.5).

Листинг 4.5

```
# Это промежуточный код, он не является рабочим
# Функция определяет, является ли полученное изображение числом 5
# Возвращает Да, если признано, что это 5. Возвращает Нет, если отвергнуто, что это 5
def perceptron(Sensor):
    b = 7  # Порог функции активации
    s = 0  # Начальное значение суммы
    for i in range(n_sensor):  # цикл суммирования сигналов от сенсоров
        s += int(Sensor[i]) * weights[i]
        if s >= b:
            return True   # Сумма превысила порог
        else:
            return False  # Сумма меньше порога
```

Так как мы используем пороговую функцию активации, то нам необходимо установить какое-то число (порог). Если взвешенная сумма будет больше или равна ему, то сеть выдаст «Да» (фактически `True`), и это будет означать, что она считает предоставленную цифру пятеркой. Порог можно выбрать любым. Пусть будет равен 7:

```
B = 7  # Порог функции активации
```

Результат работы последних строк функции: либо `True` (Да), либо `False` (Нет). Это значение будет возвращено в точку вызова функции.

Теперь определим еще две вспомогательные функции: обработки ошибок первого типа и обработки ошибок второго типа.

Первая функция вызывается тогда, когда сеть принимает за цифру 5 заведомо неверную цифру. Она же уменьшает на единицу все веса, связанные с возбужденными входами (листинг 4.6). Вспомним, что мы считаем вход возбужденным, если он получил черный пиксел (т. е. 1).

Листинг 4.6

```
# Это промежуточный код, он не является рабочим
# Уменьшение значений весов
# Если сеть ошиблась и выдала Да при входной цифре, отличной от пятерки
def decrease(number):
    for i in range(n_sensor):
        if int(number[i]) == 1:  # Если вход возбужден
            weights[i] -= 1      # Уменьшаем связанный с входом вес на единицу
```

Здесь используется функция `int(number[i])` для преобразования символа `'1'` в цифру `1`. Если этого не сделать, то возникнет ошибка в строке `if int(number[i])==1`. Это произошло бы от того, что Python не умеет сравнивать символы (текст) с цифрами.

Вторая функция вызывается тогда, когда сеть не смогла распознать цифру 5 при условии, что на вход персептрону была действительно передана пятерка. Эта же функция увеличивает на 1 все веса, связанные с возбужденными входами (листинг 4.7).

Листинг 4.7

```
# Это промежуточный код, он не является рабочим
# Увеличение значений весов
# Если сеть ошиблась и выдала Нет при поданной на вход цифре 5
def increase(number):
    for i in range(n_sensor):
        if int(number[i]) == 1:  # Если вход возбужден
            weights[i] += 1      # Увеличиваем связанный с входом вес на единицу
```

Теперь начнем обучать нашу нейронную сеть. Делать мы это будем в цикле, повторяемом многократно (фрагмент этой программы приведен в листинге 4.8).

Листинг 4.8

```
# Это промежуточный код, он не является рабочим
# Тренировка сети
n = 1000  # количество уроков
for i in range(n):
    j = random.randint(0, 9)  # Генерируем случайное число j от 0 до 9
    r = perceptron(nums[j])   # Результат обращения к сумматору
                              # (ответ - Да или НЕТ)

    if j != tema:  # Если генератор выдал случайное число j, не равное 5
        if r:  # Если сумматор сказал True (ДА)- это пятерка,
               # а j - это не пятерка
            decrease(nums[j])  # Ошибка первого типа,
                               # уменьшаем значимые веса
```

```
    else:  # Если генератор выдал случайное число j, равное 5
        if not r:  # Если сумматор сказал False (НЕТ)- это не пятерка,
                   # а на самом деле j=5
            increase(nums[tema])  # Ошибка второго типа,
                                  # увеличиваем значимые веса
```

В первой строке здесь задается количество уроков n. На каждом уроке будет распознаваться значение одной из десяти цифр, выдаваемых генератором случайных чисел. В процессе каждого урока мы передаем персептрону значения сенсоров, которые характеризуют сгенерированную цифру. Далее в зависимости от полученного ответа либо уменьшаем значимые веса, либо увеличиваем значимые веса. По завершении всех уроков выводим результаты обучения:

```
print(weights)  # Вывод значений весов
```

В листинге 4.9 представлен полный текст этой программы.

Листинг 4.9

```
# Модуль Urok2
import random

# Обучающая выборка (идеальное изображение цифр от 0 до 9)
num0 = list('111101101101111')
num1 = list('001001001001001')
num2 = list('111001111100111')
num3 = list('111001111001111')
num4 = list('101101111001001')
num5 = list('111100111001111')
num6 = list('111100111101111')
num7 = list('111001001001001')
num8 = list('111101111101111')
num9 = list('111101111001111')
# Список всех цифр от 0 до 9 в едином массиве
nums = [num0, num1, num2, num3, num4, num5, num6, num7, num8, num9]

tema = 5                                  # какой цифре обучаем
n_sensor = 15                             # количество сенсоров
weights = [0 for i in range(n_sensor)]  # обнуление весов

# Функция определяет, является ли полученное изображение числом 5
# Возвращает Да, если признано, что это 5. Возвращает Нет, если отвергнуто, что это 5
def perceptron(Sensor):
    b = 7  # Порог функции активации
    s = 0  # Начальное значение суммы
    for i in range(n_sensor):  # цикл суммирования сигналов от сенсоров
        s += int(Sensor[i]) * weights[i]
```

```
        if s >= b:
            return True   # Сумма превысила порог
        else:
            return False  # Сумма меньше порога

# Уменьшение значений весов
# Если сеть ошиблась и выдала Да при входной цифре, отличной от пятерки
def decrease(number):
    for i in range(n_sensor):
        if int(number[i]) == 1:  # Если вход возбужден
            weights[i] -= 1      # Уменьшаем связанный с входом вес на единицу

# Увеличение значений весов
# Если сеть ошиблась и выдала Нет при поданной на вход цифре 5
def increase(number):
    for i in range(n_sensor):
        if int(number[i]) == 1:  # Если вход возбужден
            weights[i] += 1      # Увеличиваем связанный со входом вес на единицу

# Тренировка сети
n = 1000  # количество уроков
for i in range(n):
    j = random.randint(0, 9)  # Генерируем случайное число j от 0 до 9
    r = perceptron(nums[j])   # Результат обращения к сумматору
                              # (ответ - Да или НЕТ)

    if j != tema:  # Если генератор выдал случайное число j, не равное 5
        if r:  # Если сумматор сказал True (ДА)- это пятерка,
               # а j - это не пятерка
            decrease(nums[j])  # Ошибка первого типа,
                               # уменьшаем значимые веса

    else:  # Если генератор выдал случайное число j, равное 5
        if not r:  # Если сумматор сказал False (НЕТ)- это не пятерка,
                   # а на самом деле j=5
            increase(nums[tema])  # Ошибка второго типа,
                                  # увеличиваем значимые веса

print(j)
print(weights)  # Вывод значений весов
'''
# проверка работы программы на обучающей выборке
print("0 это 5? ", perceptron(num0))
print("1 это 5? ", perceptron(num1))
print("2 это 5? ", perceptron(num2))
print("3 это 5? ", perceptron(num3))
print("4 это 5? ", perceptron(num4))
print("5 это 5? ", perceptron(num5))
print("6 это 5? ", perceptron(num6))
print("7 это 5? ", perceptron(num7))
```

```
print("8 это 5? ", perceptron(num8))
print("9 это 5? ", perceptron(num9))

# Тестовая выборка (различные варианты изображения цифры 5)
num51 = list('111100111000111')
num52 = list('111100010001111')
num53 = list('111100011001111')
num54 = list('110100111001111')
num55 = list('110100111001011')
num56 = list('111100101001111')

print("++++++++++++")
# Прогон по тестовой выборке
# print("Узнал 5 в 5? ", perceptron(num5))
print("Узнал 5 в 51? ", perceptron(num51))
print("Узнал 5 в 52? ", perceptron(num52))
print("Узнал 5 в 53? ", perceptron(num53))
print("Узнал 5 в 54? ", perceptron(num54))
print("Узнал 5 в 55? ", perceptron(num55))
print("Узнал 5 в 56? ", perceptron(num56))
'''
```

Обратите внимание, что после инструкции `print(weights)` заключительные строки программы закомментированы (тройными кавычками `'''`). Чуть позже мы будем снимать комментарии с этих строк.

Протестируем работу нашей программы. Зададим количество уроков 1, а результаты выведем следующими командами:

```
print(j)
print(weights)  # Вывод значений весов
```

При каждом запуске будет генерироваться одна цифра в интервале 0–9. Обратим внимание, что для всех чисел, кроме 5, веса для всех 15 сенсоров будут нулевыми. Например:

```
8
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Но когда будет сгенерирована цифра 5, то результат станет иной:

```
5
[1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1]
```

Расположим значения этих весов в таблице размером 3×5, а потом в такую же таблицу вместо единиц подставим кружочки черного цвета, а вместо нулей — белого. Результат представлен на рис. 4.33.

То есть мы значениями весовых коэффициентов фактически нарисовали цифру 5. Значит, при распознавании цифры 5 влияние сенсоров с весовыми коэффициентами 1 — максимально, а с нулевыми значениями — минимально.

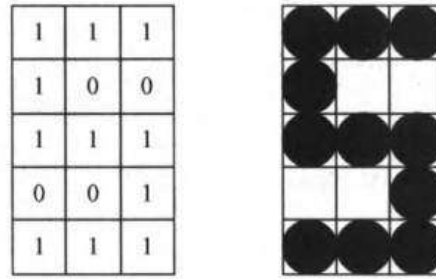


Рис. 4.33. Таблицы с нулевыми значениями весов для сенсорных элементов (цифра 5)

Изменим задание нашей программе — теперь пусть она значениями весовых коэффициентов попробует нарисовать цифру 7. Для этого в программе изменим тему урока:

```
tema = 7 # какой цифре обучаем
```

Вновь будем запускать программу до тех пор, пока генератор случайных чисел не выдаст цифру 7. В результате получим следующие значения весовых коэффициентов для этой цифры:

```
[1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1]
```

Снова поместим значения этих весов в таблицу размером 3×5 (рис. 4.34).

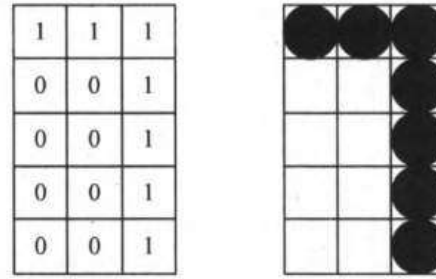


Рис. 4.34. Таблицы с нулевыми значениями весов для сенсорных элементов (цифра 7)

Теперь значениями весовых коэффициентов была нарисована цифра 7. Отметим, что мы многократно запускали программу всего с одним уроком обучения. Однако чем больше будет циклов обучения (больше уроков), тем умнее будет становиться сеть (веса наиболее значимых сенсоров увеличатся, наименее значимых — уменьшатся). Это наглядно продемонстрировано на рис. 4.35.

Следует отметить, что при увеличении числа циклов обучения в какой-то момент времени веса перестанут наращиваться или уменьшаться. Это произойдет в тот момент, когда сеть станет достаточно «умной» и перестанет ошибаться (не будут возникать ошибки первого и второго типа).

Для того чтобы в этом убедиться, запустим программу на выполнение 5 раз, меняя при каждом пуске количество уроков (т. е. циклов обучения), придав им следующие значения: 10, 100, 1000, 10 000, 100 000. Выведем на печать значения весов, полученных для каждого сенсора. Результаты расчетов приведены в табл. 4.5.

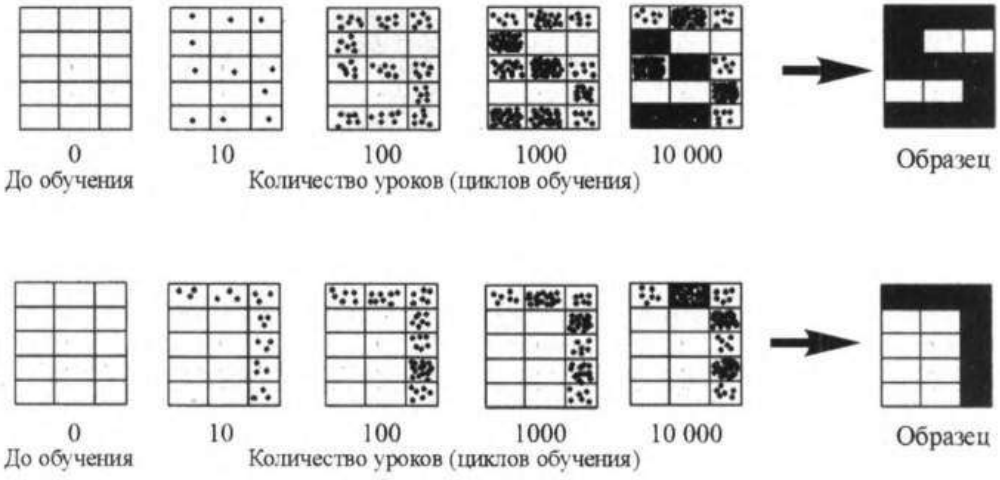


Рис. 4.35. Влияние количества циклов обучения на достоверность получаемых выводов

Таблица 4.5. *Значения весов S–R-связей при разном количестве циклов обучения (распознавание цифры 5)*

Циклы обучения	Весовые коэффициенты S–R-связей для 15 сенсоров														
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
10	0	0	0	0	0	–1	0	0	0	–1	0	0	0	0	0
100	0	1	0	1	0	–6	0	2	0	–3	0	0	1	1	0
1000	0	1	0	3	0	–4	0	0	0	–2	0	2	1	1	0
10 000	0	1	0	4	0	–5	0	0	0	–3	0	0	1	1	0
100 000	1	2	1	3	0	–11	1	2	1	–11	0	1	2	2	1

Из этой таблицы видно, что с увеличением циклов обучения (больше уроков) сеть становится умнее (веса наиболее значимых сенсоров увеличиваются, наименее значимых — уменьшаются). Обратите внимание, что у вас значения весовых коэффициентов будут отличаться от тех, которые приведены в табл. 4.5, поскольку при каждом запуске программы генератор случайных чисел будет выдавать разные последовательности. Однако общая тенденция накопления положительных и отрицательных значений весов сохранится, и чем больше циклов обучения провести, тем умнее будет становиться нейронная сеть.

Для распознавания другой цифры по тому же алгоритму (с тем же учителем) мы по логике должны получить совершенно другой набор весовых коэффициентов. Давайте теперь попробуем научить нашу искусственную ячейку мозга распознавать цифру 7. Поменяем тему урока:

```
tema = 7 # какой цифре обучаем
```

Опять запустим программу 5 раз, меняя количество уроков, т. е. циклов обучения: 10, 100, 1000, 10 000, 100 000. Получим следующие результаты (табл. 4.6).

Таблица 4.6. *Значения весов S–R-связей при разном количестве циклов обучения (распознавание цифры 7)*

Циклы обучения	Весовые коэффициенты S–R-связей для 15 сенсоров														
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**
10	0	0	0	0	0	0	–1	–1	0	0	0	0	–1	–1	0
100	1	1	1	–1	0	1	–1	0	1	–1	0	1	–1	–1	1
1000	1	1	1	0	0	1	–1	–1	1	0	0	1	–1	–1	1
10 000	1	1	1	–1	0	1	–1	–1	1	–1	0	1	–1	–1	1
100 000	1	1	1	–1	0	1	–1	–1	1	0	0	1	–1	–1	1

Для цифры 7 мы получили совершенно иные веса важности каждого сенсора. Таким образом, мы можем последовательно подобрать веса сенсорных связей для любой цифры. Учитель один, но он может обучить ученика разным навыкам.

Итак, мы обучили сеть распознаванию двух цифр. Теперь для нашего ученика необходимо провести контрольную работу — проверить, как он усвоил пройденный материал. Проверку сделаем в два этапа. Сначала выясним, как наш простейший нейрон научился распознавать цифры из обучающей выборки (идеальное начертание цифр, на которых его учили), а потом из тестовой выборки (цифры с несколько искаженными очертаниями).

Контрольную работу проведем следующим образом. Запустим программу с разным количеством циклов обучения (10, 100, 1000, 10 000, 100 000) и получим значения весов всех сенсорных элементов. После этого дадим программе задание сравнить с цифрой 5 все цифры от 0 до 9 из обучающей выборки. Затем проверим, как разработанный нами программный модуль справился со своей задачей. Такая контрольная работа уже запрограммирована в листинге 4.9, но эти строки временно закомментированы. Меняем комментарии в строках программы листинга 4.9 — на две строки ставим комментарии, а с ряда строк снимаем комментарии, переместив тройные кавычки (листинг 4.10).

Листинг 4.10

```
# Измененный фрагмент кода листинга 4.9
# print(j)
# print(weights)  # Вывод значений весов
# проверка работы программы на обучающей выборке
print("0 это 5? ", perceptron(num0))
print("1 это 5? ", perceptron(num1))
print("2 это 5? ", perceptron(num2))
print("3 это 5? ", perceptron(num3))
print("4 это 5? ", perceptron(num4))
print("5 это 5? ", perceptron(num5))
print("6 это 5? ", perceptron(num6))
print("7 это 5? ", perceptron(num7))
```

```
print("8 это 5? ", perceptron(num8))
print("9 это 5? ", perceptron(num9))

'''
```

Здесь в первой строке выводим на печать рассчитанные значения весов. В последующих строках передаем в персептрон идеальные образы чисел от 0 до 9 и печатаем результат.

Не забудем правильно указать тему урока:

```
tema = 5  # тема урока (какую цифру распознаем)
```

Результаты такой контрольной работы приведены в табл. 4.7.

Таблица 4.7. *Результаты контрольной работы (ответы на вопрос: "Это пятерка?")*

Цифры	Циклы обучения				
	10	100	1000	10 000	100 000
0	0 это 5? False	0 это 5? False	0 это 5? False	0 это 5? False	0 это 5? False
1	1 это 5? False	1 это 5? False	1 это 5? False	1 это 5? False	1 это 5? False
2	2 это 5? False	2 это 5? False	2 это 5? False	2 это 5? False	2 это 5? False
3	3 это 5? False	3 это 5? False	3 это 5? False	3 это 5? False	3 это 5? False
4	4 это 5? False	4 это 5? False	4 это 5? False	4 это 5? False	4 это 5? False
5	5 это 5? False	5 это 5? False	**5 это 5? True**	**5 это 5? True**	**5 это 5? True**
6	6 это 5? False	6 это 5? False	6 это 5? False	6 это 5? False	6 это 5? False
7	7 это 5? False	7 это 5? False	7 это 5? False	7 это 5? False	7 это 5? False
8	8 это 5? False	8 это 5? False	8 это 5? False	8 это 5? False	8 это 5? False
9	9 это 5? False	9 это 5? False	9 это 5? False	9 это 5? False	9 это 5? False

Как видно из этой таблицы, наш персептрон начал безошибочно отличать пятерку от всех остальных цифр после 1000 уроков обучения.

Во второй контрольной работе усложним задачу нашему персептрону. Проверим, сможет ли он распознавать различные варианты написания цифры 5. Проверять будем на тестовой выборке, в которой содержатся цифры 5 с небольшими искажениями. Создадим такую тестовую выборку, в которой для упрощения задачи все изображения останутся того же размера, что и в обучающей выборке (рис. 4.36).

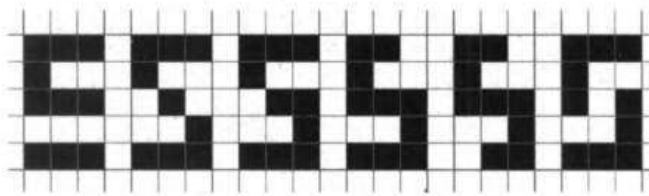


Рис. 4.36. Тестовая выборка вариантов написания цифры 5

Преобразуем эти изображения в строковый формат (табл. 4.8).

***Таблица 4.8.** Значения вариантов изображений цифры 5 в строковом формате*

№ п/п	Цифра	Сигналы от сенсоров	Значение R-элемента (посимвольное сложение)
1	5	111 + 100 + 111 + 000 + 111	111100111100111
2	5	111 + 100 + 010 + 001 + 111	111100010001111
3	5	111 + 100 + 011 + 001 + 111	111100011001111
4	5	110 + 100 + 111 + 001 + 111	110100111001111
5	5	110 + 100 + 111 + 001 + 011	110100111001011
6	5	111 + 100 + 101 + 001 + 111	111100101001111

Из этой таблицы видно, что в строковом формате цифра 5 может иметь совершенно разный набор значений нулей и единиц. Аналогичная ситуация и с другими цифрами. Теперь запишем шесть возможных вариантов изображения цифры 5 в виде текстовой строки (значение строк возьмем из табл. 4.8). В нашей программе (листинг 4.9) это находится в следующих строках, которые пока закомментированы (листинг 4.11).

Листинг 4.11

```
# Измененный фрагмент кода листинга 4.9
# Тестовая выборка (различные варианты изображения цифры 5)
num51 = list('111100111000111')
num52 = list('111100010001111')
num53 = list('111100011001111')
num54 = list('110100111001111')
num55 = list('110100111001011')
num56 = list('111100101001111')
```

Эту контрольную работу мы проведем следующим образом. Запускаем программу с разным количеством циклов обучения (10, 100, 1000, 10 000, 100 000) и получаем значения весов всех сенсорных элементов. После этого даем программе задание сравнить с цифрой 5 все шесть вариантов ее написания из тестовой выборки. Затем проверим, как разработанный нами программный модуль справился со своей задачей. Такая контрольная работа уже запрограммирована в листинге 4.9 в следующих строках (листинг 4.12).

Листинг 4.12

```
# Измененный фрагмент кода листинга 4.9
# Прогон по тестовой выборке
print("Узнал 5 в 51? ", perceptron(num51))
print("Узнал 5 в 52? ", perceptron(num52))
```

```
print("Узнал 5 в 53? ", perceptron(num53))
print("Узнал 5 в 54? ", perceptron(num54))
print("Узнал 5 в 55? ", perceptron(num55))
print("Узнал 5 в 56? ", perceptron(num56))
```

Снимем все комментарии с этих строк (удалим тройные кавычки) и запустим программу несколько раз, меняя количество циклов обучения: 10, 100, 1000, 10 000, 100 000. Результаты такой контрольной работы приведены в табл. 4.9.

***Таблица 4.9.** Результаты контрольной работы (ответы на вопрос: "Это пятерка?")*

Варианты цифры 5	Циклы обучения				
	10	100	1000	10 000	100 000
51	Узнал 5 в 51? False	Узнал 5 в 51? False	Узнал 5 в 51? False	**Узнал 5 в 51? True**	**Узнал 5 в 51? True**
52	Узнал 5 в 52? False	**Узнал 5 в 52? True**	**Узнал 5 в 52? True**	**Узнал 5 в 52? True**	**Узнал 5 в 52? True**
53	Узнал 5 в 53? False	**Узнал 5 в 53? True**	**Узнал 5 в 53? True**	**Узнал 5 в 53? True**	**Узнал 5 в 53? True**
54	Узнал 5 в 54? False	**Узнал 5 в 54? True**	**Узнал 5 в 54? True**	**Узнал 5 в 54? True**	**Узнал 5 в 54? True**
55	Узнал 5 в 55? False	Узнал 5 в 55? False	Узнал 5 в 55? False	**Узнал 5 в 55? True**	**Узнал 5 в 55? True**
56	Узнал 5 в 56? False	Узнал 5 в 56? False	**Узнал 5 в 56? True**	**Узнал 5 в 56? True**	**Узнал 5 в 56? True**

Как видно из этой таблицы, наш персептрон после 10 уроков не справился с задачей (не смог узнать ни одной пятерки). После 100 уроков сделал три ошибки, после 1000 уроков — две ошибки. А вот после 10 000 циклов обучения стал безошибочно решать поставленную задачу.

Итак, мы убедились, что даже на предельно упрощенном однослойном персептроне с одним скрытым слоем, который, по сути, представляет собой фактически один искусственный нейрон, мы смогли добиться эффекта обобщения. Наш искусственный нейрон никогда не видел искаженные изображения цифры 5, но смог их распознать.

Мы также убедились, что с ростом циклов обучения нейрон становится более грамотным и, в конце концов, перестает ошибаться. Но как определить, какого количества уроков будет достаточно, чтобы завершить процесс обучения? Мы рассмотрели очень упрощенный вариант сети, связи и входы которой могли быть только целыми числами. А как быть в тех случаях, когда эти величины будут отличны от единицы и нуля? Для этого обратимся к следующему разделу.

4.8. Дельта-правило

Теперь попробуем распространить правила Хебба на произвольные значения входов (сенсоров) и связей между элементами нейронной сети (не только целые числа и не только 0/1). Пусть мы заранее знаем правильный выход нашей сети (обозначим его d) и фактический ответ, выданный сетью (обозначим ответ сети y). Если в процессе обучения сеть ошиблась, то мы можем рассчитать значение ошибки (погрешность сети). Величина ошибки (δ) может быть рассчитана как разница между правильным и реальным ответом:

$$\delta = d - y.$$

Мы знаем, что решающую роль в преобразовании сигнала играют связи. Значит, необходимо каким-то образом их изменять. Классический алгоритм изменения связей — дельта-правило, или алгоритм обучения персептронов.

Дельта-правило (delta rule) — метод обучения персептрона по принципу градиентного спуска по поверхности ошибки. Его дальнейшее развитие привело к созданию метода обратного распространения ошибки.

Дельта-правило можно выразить следующей формулой:

$$w_i(t+1) = w_i(t) + \delta \cdot x_i \cdot \eta,$$

где w_i — вес i-й связи; t — шаг обучения (номер урока); δ — величина ошибки; x_i — значение входного сигнала от i-го сенсора; η — коэффициент пропорциональности (скорость, или норма, обучения).

По этой формуле в процессе обучения персептрона рассчитывается новое значение i-го веса связи на $(t+1)$-м шаге обучения. Рассмотрим эту формулу более детально.

Для обозначения количества шагов обучения (количества уроков) здесь используется переменная t. Очевидно, что наша цель — получить из старого значения веса связи $w_i(t)$ новое значение $w_i(t+1)$. Для этого мы, в соответствии с правилами Хебба, должны прибавить какое-то число к весу связи. Как раз эта добавка и вычисляется в выражении $\delta \cdot x_i \cdot \eta$. Рассмотрим это выражение более детально.

Переменная δ — это ошибка нейронной сети. В соответствии с правилами Хебба, если нейронная сеть ответила правильно (ожидаемый и реальный результаты равны), то ошибки нет и, значит, $\delta = 0$, т. е. вся добавка к весу связи равна 0 и вес связи менять не нужно.

В случае, если значение ошибки положительное ($\delta > 0$), а это будет при условии $d > y$ (т. е. правильный ответ должен быть выше, чем тот, который дала сеть), то значение добавки к весу будет положительным и вес связи нужно увеличить (первое правило Хебба). Это соответствует случаю, когда сеть получила на вход цифру 5, но не узнала ее.

В случае, если значение ошибки отрицательное ($\delta < 0$), а это будет при условии $d < y$ (т. е. правильный ответ должен быть ниже, чем тот, который дала сеть), то значение добавки к весу будет отрицательным и вес связи нужно уменьшить (вто-

рое правило Хебба). Это соответствует случаю, когда сеть неверно приняла представленное ей число за 5.

Теперь перейдем к следующему элементу рассматриваемого выражения — x_i. Это значение, которое пришло на i-й вход сети, т. е. от одного из сенсоров. Чем более сильный сигнал поступил на вход, тем сильнее изменится вес, связанный с этим входом, что вполне логично. А если на вход вообще не поступил сигнал ($x_i = 0$), то и соответствующий вес не должен будет измениться (добавка будет равна нулю).

Теперь перейдем к самому интересному — параметру η, который характеризует скорость обучения. Его еще называют *коэффициентом скорости обучения*. Попробуем разобраться, для чего в формулу ввели этот параметр?

Корректность работы сети зависит от правильно подобранных весов связей. А значит, от весов зависит и величина ошибки. Чем больше отклонился выданный сетью ответ от правильного ответа, тем больше ошибка. Представим в виде графика зависимость погрешности сети δ от значения веса одной из связей w_i (рис. 4.37).

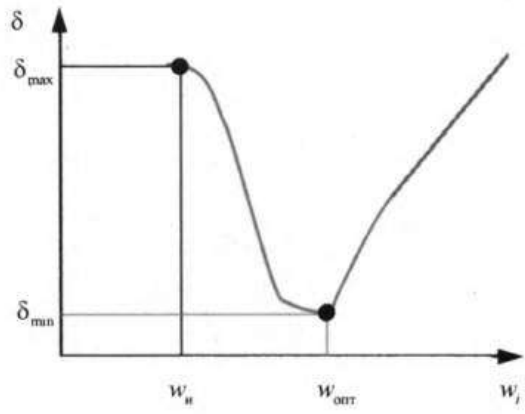


Рис. 4.37. Зависимость погрешности сети δ от значения веса связи w_i

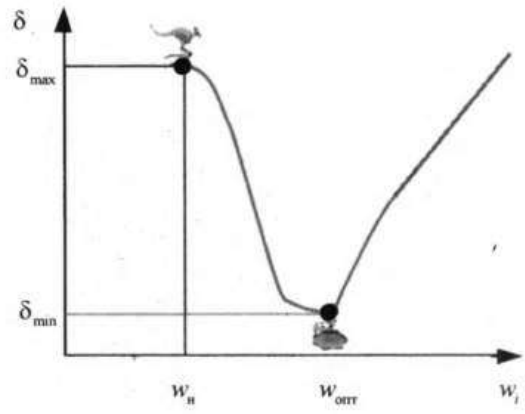


Рис. 4.38. Положение кенгуру перед началом движения к водопою

Как видно из этого рисунка, при каком-то начальном назначенном значении веса $w_н$ мы имеем максимальную погрешность в работе сети δ_{max}, т. е. мы будем постоянно получать от сети ошибочные результаты. Наша задача — найти такое оптимальное значение веса $w_{опт}$, при котором значение ошибки будет минимальным — δ_{min}. В этой конкретной ситуации нужно увеличить значение i-го веса, т. е. надо с каким-то шагом увеличивать начальное значение веса $w_н$ до тех пор, пока погрешность не станет минимальной. Вот как раз величина этого шага и задается коэффициентом пропорциональности. От его значения зависит и скорость обучения, и вообще возможность определения оптимального значения веса $w_{опт}$.

Поясним это примером — как кенгуру спускается с горки к водопою. Предположим, что кенгуру находится на пригорке, а в самой низине имеется водопой (рис. 4.38).

Положение кенгуру означает конкретный вес связи, который был назначен перед обучением нейронной сети. Кенгуру надо спуститься с горки в самый низ, т. к. именно в этой точке находится водопой (в ней ошибка сети минимальна). Однако кенгуру может только прыгать. Прыжок может быть очень коротким или очень длинным. И именно за «длину прыжка» и отвечает коэффициент η в формуле добавки к весу.

Предположим, что коэффициент η очень маленький. Тогда наш кенгуру станет продвигаться к низине очень маленькими прыжками, и этот путь будет весьма долгим (рис. 4.39, *слева*). Тогда сделаем коэффициент η очень большим — ведь большими прыжками кенгуру быстрее доскачет до низины. Однако нет. При большом значении прыжка (высокой скорости обучения) есть опасность не доскакать до самого низа из-за того, что кенгуру будет постоянно прыгать вперед-назад, практически оставаясь на одной и той же высоте (рис. 4.39, *справа*).

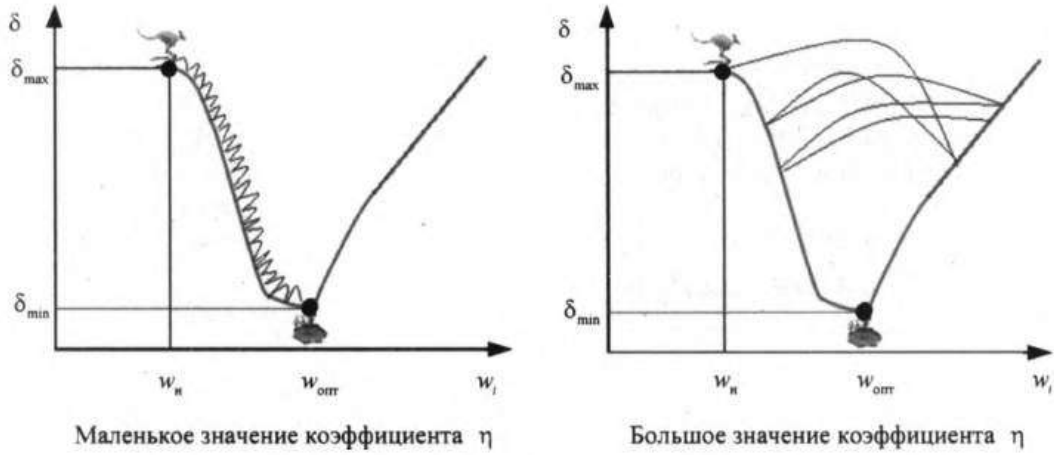


Рис. 4.39. Влияние значения коэффициента η на процесс минимизации ошибок нейронной сети

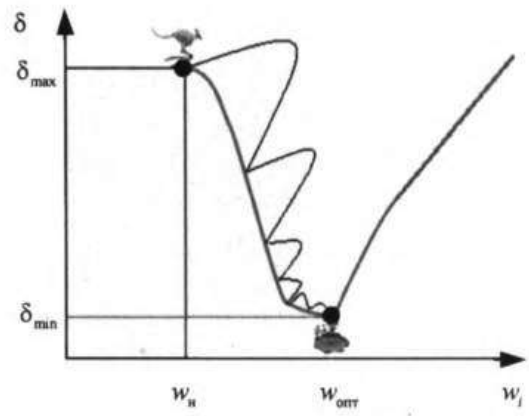


Рис. 4.40. Изменение значения коэффициента η в процессе минимизации ошибок нейронной сети

Поэтому кенгуру должен двигаться к водопою рационально: сначала большими прыжками, а по мере приближения к воде длину прыжка уменьшать. То есть нужно использовать некую функцию, которая будет уменьшать параметр η по мере приближения к минимальному значению ошибки (рис. 4.40).

Вот мы и разобрались в предназначении коэффициента скорости обучения. Мы можем изменять вес связи только скачками, и этот коэффициент определяет величину скачков.

4.9. Линейная аппроксимация

Теперь рассмотрим другую нестандартную задачу, которая относится не к классификации объектов (как в примере с распознаванием цифр), а к *аппроксимации*, т. е. к поиску промежуточных значений при заданном наборе конкретных значений. И тут нам как раз и пригодится дельта-правило. Криминалистам приходится часто сталкиваться с такой задачей, как определение роста преступника по отпечатку оставленного им следа. Если мы будем знать зависимость между ростом и длиной отпечатка обуви, то сможем, имея длину отпечатка обуви, сказать, примерно какого роста преступник. Возьмем 10 человек разного роста (мужчин), замерим у них отпечатки от носимой обуви и занесем полученные значения в таблицу (табл. 4.10).

***Таблица 4.10.** Зависимость роста мужчины от длины следа обуви*

Длина следа от обуви X, см	Рост Y, см
22	150
23	155
24	160
25	162
26	171
27	174
28	180
29	183
30	189
31	192

Почему мы взяли только мужчин и не включили в выборку женщин? Потому что у женщин несколько иное соотношение между ростом и размером обуви. Теперь представим эти данные в виде графика (рис. 4.41).

Мы видим, что полученные данные подозрительно напоминают прямую линию. Наша задача состоит в том, чтобы как можно более точно провести ту линию, которая будет повторять заданный набор точек. В этом и состоит *принцип аппроксимации*. Например, у нас нет данных о Y-координате точки с $X = 23{,}5$, и задача

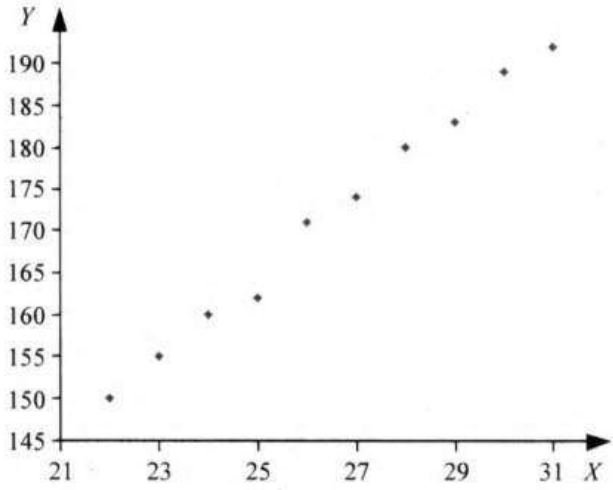


Рис. 4.41. Зависимость роста мужчины от размера отпечатка следа обуви

аппроксимации — с наибольшей вероятностью предсказать, какой будет Y для этой точки. Иными словами, какой рост будет у человека, который оставил след размером 23,5 см.

Для решения этой задачи можно использовать метод наименьших квадратов. Его действительно применяют для решения задач такого класса. Однако для нас важно сделать это с помощью нейронной сети. Сделать это можно следующим образом. Нам нужна прямая линия. Из алгебры известно, что любая прямая в декартовых координатах задается уравнением

$$Y = kX + C.$$

Коэффициент k отвечает за крутизну наклона прямой, а C указывает точку на оси y, через которую проходит эта прямая.

Мы будем искать уравнение прямой линии, аппроксимирующее наши данные (зависимость роста человека от размера обуви). Значит, параметр Y у нас будет выходом сети. Теперь определимся с входами. Совершенно точно, что одним входом будет являться переменная X. Однако в уравнении прямой фигурирует еще одно слагаемое — C. О нем тоже нельзя забывать. C — постоянная величина. Поэтому мы добавим в нашу сеть второй вход, на который всегда будет подаваться этот параметр (в нашей задаче он будет равен 1). Таким образом, произведение этого входа на вес всегда будет равно только этому весу вне зависимости от входа. Функции активации в этом случае не будет. Взвешенная сумма и станет выходом нашей сети. Вот такой упрощенный персептрон представлен на рис. 4.42.

Наш персептрон по факту является искусственным нейроном. Мы уже говорили об этом, когда рассматривали математическую модель искусственного нейрона. Давайте вспомним, как математически определяется выход нашей сети:

$$Y = x_1 w_1 + x_2 w_2 + x_3 w_3 + \ldots + x_n w_n.$$

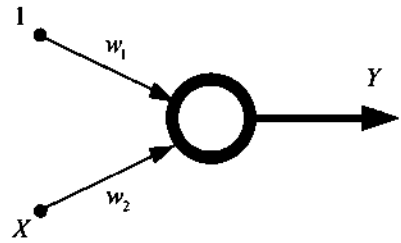


Рис. 4.42. Упрощенный персептрон для определения параметров аппроксимации

Это была запись в общем виде. А для нашего случая, согласно схеме, показанной на рис. 4.42, имеем:

$$Y = w_2 X + w_1.$$

Ничего эта запись вам не напоминает? Да это же уравнение прямой линии:

$$Y = kX + C,$$

где

$$w_2 = k,\ w_1 = C.$$

Фактически мы построили персептрон так, что в процессе обучения его весовые коэффициенты станут коэффициентами прямой линии, которую мы и ищем. Теперь напишем программу, которая установит зависимость роста человека от размера его обуви.

Для начала нам понадобится модуль для работы со случайными числами. Импортируем его следующей командой:

```
import random
```

Далее мы создаем переменную, являющуюся весом связи w_2 для входа X (это размер обуви). По сути, это и есть коэффициент крутизны наклона прямой:

```
# Коэффициент при x
k = random.uniform(-5, 5)
```

Функция `uniform(from, to)` генерирует случайное вещественное число от `from` и до `to` включительно. Заметьте, что мы уже не работаем с целыми числами.

Затем создаем переменную, являющуюся весом связи при всегда единичном входе (а по совместительству отвечающую за свободный член в уравнении прямой). То есть это вес связи w_1:

```
# Свободный член уравнения прямой - c
c = random.uniform(-5, 5)
```

Задав эти два параметра, мы определили прямую линию. Так как оба параметра заданы случайным образом, то нами определена случайная прямая линия. Перед началом обучения персептрона положение прямой линии абсолютно несущественно — она может быть какой угодно. Далее мы должны вывести в консоль созданные переменные, т. е. посмотреть, с чего мы начали обучать наш новоиспеченный нейрон:

```
# Вывод данных начальной прямой линии
print('Начальная прямая линия: ', k, '* X + ', c)
```

Теперь задаем данные о наборе точек:

```
# Набор точек X:Y
data = {22: 150, 23: 155, 24: 160, 25: 162, 26: 171, 27: 174, 28:
    180, 29: 183,30: 189, 31: 192 }
```

Здесь используется не список, а словарь. Это практически то же самое, что и список, только используются не числовые индексы, а собственноручно заданные имена — ключи. В нашем словаре ключи — это X (размер обуви), а соответствующие им значения — это Y (рост мужчины).

Теперь нам необходимо задать коэффициент скорости обучения. Это делается экспериментально. Первоначально для нашей задачи была задана скорость обучения — `0.1`, но при этом сеть не обучалась. По мере роста шагов ошибка сети только увеличивалась. Этот случай соответствует картинке с кенгуру, делающем слишком большие прыжки.

После ряда понижений коэффициента скорости обучения до значения `0.0001` персептрон начал обучаться, т. е. ошибка по мере выполнения шагов обучения стабильно уменьшалась.

Теперь нужно создать функцию, рассчитывающую ответ нашего персептрона:

```
# Расчет  Y
def proceed(x):
    return x*k+c
```

Как видите, никакой функции активации тут нет. Ответ нашей сети есть взвешенная сумма, где `k` — вес связи при `x`, а `c` — вес связи при входе, всегда равном единице.

Теперь надо обучить нашу сеть (фрагмент этой программы приведен в листинге 4.13). Чем больше шагов заложим, тем лучше. Возьмем, например, 100 000 шагов обучения. Хотя, возможно, для получения результата подошло бы и меньшее количество шагов. Тут есть простор для экспериментирования.

Листинг 4.13

```
# Это промежуточный код, он не является рабочим
# Тренировка сети
for i in range(100000):
    # Получить случайную X-координату точки
    x = random.choice(list(data.keys()))

    # Получить соответствующую Y-координату точки
    true_result = data[x]

    # Получить ответ сети
    out = proceed(x)
```

```
    # Считаем ошибку сети
    delta = true_result - out

    # Меняем вес при x в соответствии с дельта-правилом
    k += delta*rate*x

    # Меняем вес при постоянном входе в соответствии с дельта-правилом
    c += delta*rate
```

Здесь в цикле мы 100 000 раз выбираем из нашего словаря случайный ключ (т. к. ключ — это и есть значение *X*).

После этого создается переменная `true_result`, в которой хранится правильный параметр *Y* для соответствующего значения *X*. Далее в переменную `out` мы помещаем ответ нашей сети для этого значения *X*. Затем мы высчитываем ошибку и, используя дельта-правило, меняем оба веса связи. Осталось только вывести данные о новой прямой линии:

```
# Вывод данных готовой прямой линии
print('Готовая прямая: ', k, '* X + ', c)
```

В листинге 4.14 приведен полный код программы.

Листинг 4.14

```
# МодульUrok22
import random

k = random. Uniform(-5, 5) . # Коэффициент при x
c = random.uniform(-5, 5)    # Свободный член уравнения прямой
print('Начальная прямая: Y =  ', k, '* X + ', c)  # Вывод данных начальной прямой
rate = 0.0001  # Скорость обучения

# Набор точек X:Y
data = {
    22: 150,
    23: 155,
    24: 160,
    25: 162,
    26: 171,
    27: 174,
    28: 180,
    29: 183,
    30: 189,
    31: 192
}

# Высчитать y
def proceed(x):
    return x * k + c
```

```
# Тренировка сети
for i in range(100000):
    x = random.choice(list(data.keys()))  # Получить случайную X-координату точки
    true_result = data[x]          # Получить соответствующую Y-координату точки
    out = proceed(x)               # Получить ответ сети
    delta = true_result - out      # Считаем ошибку сети
    k += delta * rate * x          # Меняем вес при x в соответствии с дельта-правилом
    c += delta * rate              # Меняем вес при постоянном входе
                                   # в соответствии с дельта-правилом

print('Готовая прямая: Y = ', k, '* X + ', c)  # Вывод данных готовой прямой
```

После запуска этой программы получим следующие результаты (рис. 4.43).

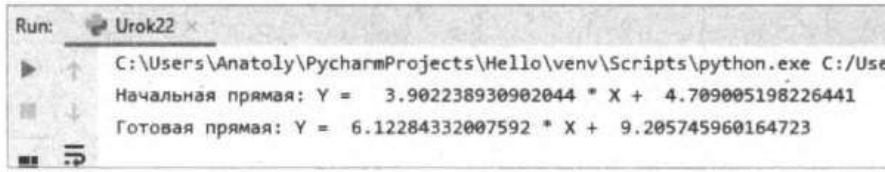


Рис. 4.43. Результаты работы программы поиска зависимости роста человека от размера обуви

Не забываем, что при каждом запуске мы будем получать немного разные результаты, поскольку в программе используется генератор случайных чисел. Изобразим на графике (рис. 4.44) две линии: до обучения (штриховая линия) и после обучения (сплошная).

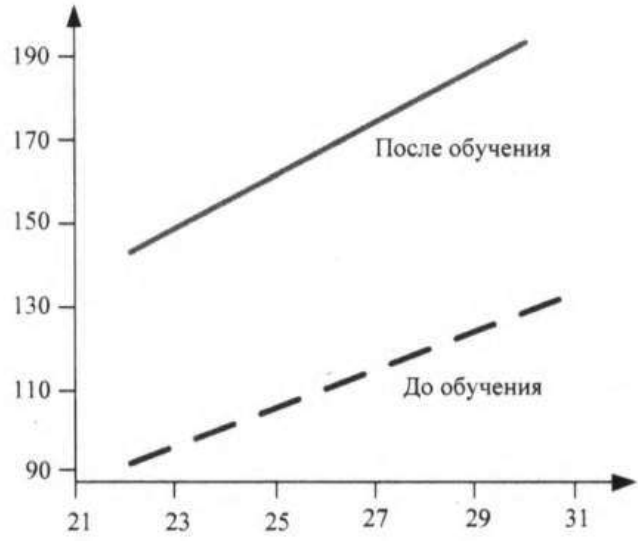


Рис. 4.44. График, построенный по уравнениям прямых линий (до и после обучения персептрона)

На этом рисунке пунктиром показана линия, которую мы случайным образом задали в начале программы. Она может проходить как угодно и где угодно. А вот сплошная прямая линия — результат работы нашего персептрона, свидетельствующий о том, что мы получили линейную аппроксимацию заданного набора точек. То есть наш персептрон построил зависимость роста от размера обуви. Совместим теперь сплошную прямую линию с нашими исходными данными (рис. 4.45).

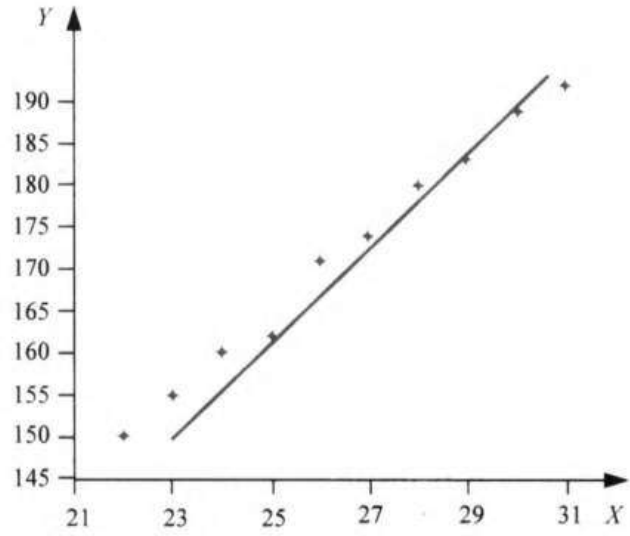


Рис. 4.45. Аппроксимация прямой линией зависимости роста мужчины от размера обуви

Здесь точки — это набор исходных данных. Прямая линия — результат работы нашего персептрона, представляющий собой линейную аппроксимацию этого набора точек. Мы решили поставленную задачу. И приведенный пример хорошо показывает, что персептроны могут выполнять не только задачи на классификацию.

4.10. Учим персептрон классифицировать объекты. Обучение без учителя

В предыдущих разделах мы познакомились с принципом работы персептрона и выполнили его программную реализацию, использовав для этого достаточно упрощенный пример и элементарные команды языка Python. Теперь пойдем дальше — реализуем персептрон на Python в виде класса, после чего обучим его классифицировать объекты на основе их геометрических параметров. Для этого нам понадобится подключить следующие библиотеки для работы с математическими функциями и построения графиков: NumPy, pandas, matplotlib. Мы примем объектно-ориентированный подход, определив интерфейс персептрона как класс языка Python, что позволит нам инициализировать новые объекты персептрона. Внутри этого класса мы создадим два метода: метод `fit()`, который сможет реализовать процесс обучения персептрона на обучающей выборке данных, и метод `predict()`

(прогнозировать), который обеспечит возможность делать прогнозы на обученной модели. В листинге 4.15 приведен программный код класса `Perceptron` (персептрон).

Листинг 4.15

```
# Это промежуточный код, он не является рабочим
# Описание класса Perceptron
class Perceptron(object):

    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta                          # Темп обучения (от 0 до 1)
        self.n_iter = n_iter                    # Количество итераций (уроков)

    '''
    Выполнить подгонку модели под тренировочные данные.
    Параметры
    X - тренировочные данные: массив, размерность - X[n_sam ples, n_features],
    где
                            n_samples - число образцов,
                            n _features - число признаков,
    y - Целевые значения: массив, размерность - y[n_samples]
    Возвращает
    self: object
    '''

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])  # w_: одномерный массив -
                                            # веса после обучения
        self.errors_ = []           # errors_: список ошибок
                                    # классификации в каждой эпохе
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

        ''' Рассчитать чистый вход '''
    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

        ''' Вернуть метку класса после единичного скачка '''
    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

Входными данными для этой модели персептрона является двумерный массив:

```
X[n_sam ples, n_features]
```

где:

- `n_samples` — количество образцов (количество строк в массиве данных обучающей выборки);
- `n_features` — количество признаков (количество входов или сенсоров в обучающей выборке).

Целевые значения в обучающей выборке представлены одномерным массивом `y[n_samples]`.

В качестве результата своей работы такой персептрон будет возвращать объект `self`.

Используя эту реализацию персептрона, мы можем инициализировать новые объекты `Perception`, задавая темп обучения `eta` и число эпох `n_iter`, т. е. циклов (уроков) прохода по тренировочному набору данных. При помощи метода обучения `fit()` мы рассчитываем веса в атрибуте `self.w_`.

Итак, у нас есть персептрон. Теперь нужно определиться: чему же мы его будем учить? Возьмем популярный в литературе пример распознавания вида цветка ириса на основе размера его лепестков и чашелистика. Для обучения нашего персептрона сформируем обучающий набор данных из двух видов ирисов: ирис щетинистый (Iris setosa) и ирис разноцветный (Iris versicolor). Несмотря на то что правило персептрона не ограничивается двумя размерностями, в целях упрощения визуализации мы рассмотрим только два признака: длину чашелистика (sepal length) и длину лепестка (petal length).

Воспользуемся уже готовым набором обучающих данных, который хранится в библиотеке данных для машинного обучения Калифорнийского университета. Для этого прибегнем к помощи библиотеки pandas, чтобы загрузить набор обучающих данных с параметрами ириса в объект `DataFrame`. Для проверки правильности загруженных данных выведем их на печать. В листинге 4.16 приведен фрагмент этого программного кода.

Листинг 4.16

```
# Загрузка из Интернета данных, запись их в объект DataFrame и вывод на печать
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
df = pd.read_csv(url, header=None)
print('Данные об ирисах')
print(df.to_string())
df.to_csv('./CSV/Iris.csv')
```

Здесь мы в переменную `url` записали интернет-ссылку на данные о ирисах. Затем скачали эти данные из Интернета и сохранили их в объекте `df`. Для визуального

контроля корректности полученных данных вывели их на печать. И наконец, последней командой сохранили этот набор данных на своем компьютере в текущем каталоге в файле Iris.csv (чтобы в дальнейшем больше не скачивать эти данные из Интернета, а считывать их из файла своего компьютера). Перед запуском этого программного кода в текущем каталоге нужно создать папку с именем CSV либо закомментировать в коде последнюю строку (тогда скачанные данные просто не будут сохранены на вашем компьютере). Фрагмент результатов работы этого программного кода приведен в табл. 4.11.

Таблица 4.11. *Параметры различных видов ириса*

	0	1	2	3	4
	Чашелистик		Лепесток		Вид ириса
	длина, см	ширина, см	длина, см	ширина, см	
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
...	...	...	...	...	...
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4.0	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
55	5.7	2.8	4.5	1.3	Iris-versicolor
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

В этой таблице 150 строк (по 50 строк сведений о каждом виде цветка). В первой колонке содержатся номера строк (начиная с 0). В первой строке содержатся номера столбцов (начиная с номера 0). В итоге мы получили двумерный массив данных размером 150 строк и 5 столбцов. Здесь мы видим, что набор данных состоит из длины/ширины двух элементов ириса: лепестка и чашелистика (рис. 4.46). Не станем углубляться в ботанику, а будем рассматривать эти четыре колонки как значения неких параметров, которые характеризуют тот или иной вид ириса.

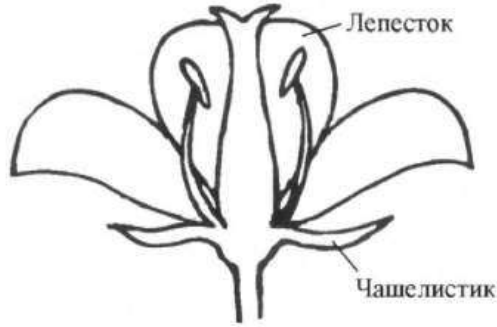


Рис. 4.46. Структура ириса

Теперь на основе этого массива данных сформируем обучающую выборку, для чего сделаем следующие шаги:

1. Выделим из этих 100 тренировочных образцов первый столбец (длина чашелистика) и третий столбец (длина лепестка) и присвоим их матрице признáков (двумерный массив X).
2. Выделим из этих 100 тренировочных образцов пятый столбец (вид ириса), преобразуем символьные значения этого столбца в два целочисленных значения 1 и –1 (1 будет означать вид Iris versicolor, или разноцветный, –1 — вид Iris setosa, или щетинистый), эти значения присвоим вектору y (y — одномерный массив, характеризующий вид цветка).
3. Для контроля сделаем распечатку полученного массива тренировочных данных.

Эти действия можно реализовать с помощью фрагмента программного кода из листинга 4.17.

Листинг 4.17

```
# Это промежуточный код, он не является рабочим
# Выборка из DF 100 строк (столбец 0 и столбец 2), загрузка их в массив X
X = df.iloc[0:100,[0, 2]].values
print('Значение X - 100')
print(X)

# Выборка из DF 100 строк (столбец 4 - название цветков)
# и загрузка их в массив Y
y = df.iloc[0:100,4].values

# Преобразование названий цветков (столбец 4) в массив чисел -1 и 1
y = np.where(y == 'Iris-setosa',-1, 1)
print('Значение названий цветков в виде -1 и 1, Y - 100')
print(y)
```

Таким образом, мы сформировали обучающую выборку, которая содержит два входных параметра: $X1$ и $X2$ (в виде двумерного массива — X), и один выходной

параметр — *Y* (в виде одномерного массива — y). Полученные данные можно представить в графическом виде с помощью следующего фрагмента программного кода (листинг 4.18).

Листинг 4.18

```
# Это промежуточный код, он не является рабочим
# Первые 50 элементов обучающей выборки (строки 0-50, столбцы 0, 1)
plt.scatter(X[0:50, 0], X[0:50, 1], color='red', marker='o', label='щетинистый')
# Следующие 50 элементов обучающей выборки (строки 50-100, столбцы 0, 1)
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x',
label='разноцветный')

# Формирование названий осей и вывод графика на экран
plt.xlabel('Длина чашелистика')
plt.ylabel('Длина лепестка')
plt.legend(loc='upper left')
plt.show()
```

Теперь необходимо объединить листинги 4.15–4.18 в один программный модуль. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 4_18_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результат работы этого объединенного программного кода представлен на рис. 4.47.

Как видно из рис. 4.47, оба признака достаточно хорошо группируются, практически не пересекаются, и на их основе можно классифицировать эти виды цветков.

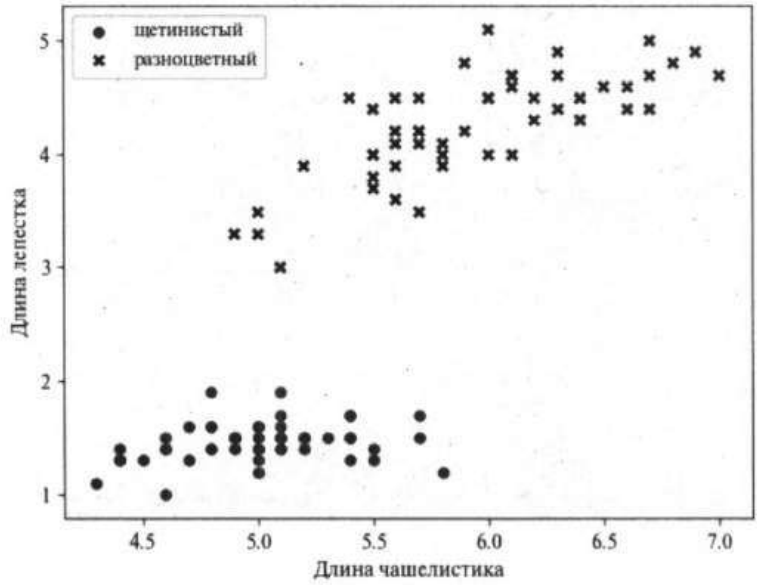


Рис. 4.47. Результат работы программного кода

Теперь можно натренировать наш алгоритм персептрона на подмножестве данных цветков ириса, которые мы только что получили. Мы также можем построить график изменения ошибок в процессе обучения (значение ошибок после каждой эпохи обучения), чтобы удостовериться, что алгоритм персептрона нашел те параметры весов, которые смогут четко разделить эти два вида ирисов. Обучение персептрона реализуем в следующем фрагменте программного кода (листинг 4.19).

Листинг 4.19

```
# Это промежуточный код, он не является рабочим
# Тренировка
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Эпохи')
plt.ylabel('Число случаев ошибочной классификации')
plt.show()
```

Этот фрагмент кода добавим к модулю листинга 4_18_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 4_19_1) в сопровождающем книгу файловом архиве (см. *приложение*). Здесь мы на основе класса `Perceptron` создали объект `ppn`. Этому объекту передали два параметра: темп обучения `eta=0.1` и количество эпох, или уроков (итераций) обучения `n_iter=10`. Затем вызвали метод нашего персептрона `fit()`, который осуществляет запуск процесса обучения, и передали этому методу массивы `X` и `y` (сформированную на предыдущих шагах обучающую выборку). Если запустить модуль листинга 4_19_1, то после десяти циклов обучения мы получим следующую картину (рис. 4.48).

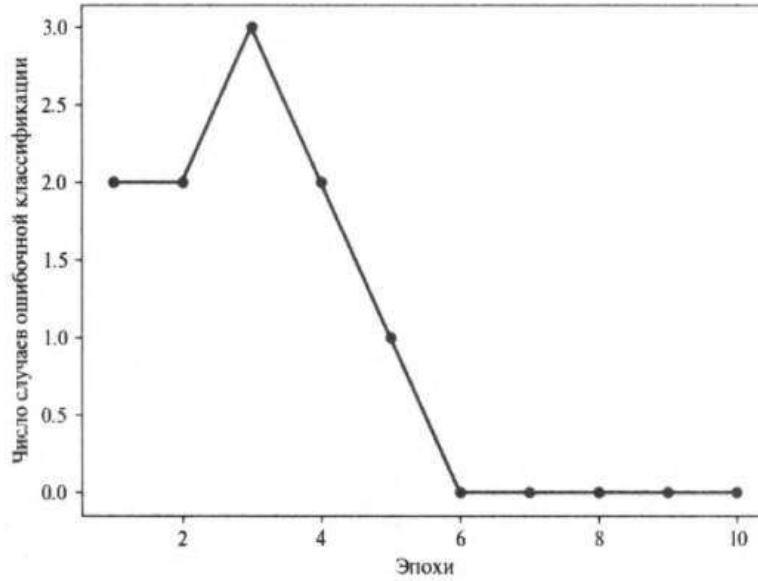


Рис. 4.48. Зависимость числа случаев ошибочной классификации от количества эпох обучения

Как видно из рис. 4.48, наш персептрон стабилизировался уже после шестой эпохи обучения и теперь может идеально классифицировать тренировочные образцы. Проверим это на следующем примере (листинг 4.20).

Листинг 4.20

```
# Это промежуточный код, он не является рабочим
i1=[5.5, 1.6]
i2=[6.4, 4.5]
R1 = ppn.predict(i1)
R2 = ppn.predict(i2)
print('R1=', R1, '  R2=', R2)
```

Здесь мы задали размеры чашелистика и лепестка для двух цветков `i1` и `i2` (цветков с такими размерами нет в обучающей выборке), передали эти параметры в функцию предсказания нашего персептрона (`predict`), а полученные результаты записали в переменные `R1` и `R2`. Добавим этот фрагмент кода к модулю листинга 4_19_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 4_20_1) в сопровождающем книгу файловом архиве (см. *приложение*).

В результате работы этого объединенного программного кода получим ответ:

```
R1= -1  R2= 1
```

Значит, в первом случае это оказался цветок вида Iris setosa, а во втором — цветок вида Iris versicolor. Немного дополним наш код (листинг 4.21).

Листинг 4.21

```
# Это промежуточный код, он не является рабочим
if R1 == 1:
  print('R1= Вид Iris setosa')
else:
  print('R1= Вид Iris versicolor')
```

Этот фрагмент кода добавим к модулю листинга 4_20_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 4_21_1) в сопровождающем книгу файловом архиве (см. *приложение*). В результате работы этого объединенного программного кода получим ответ в виде текстового сообщения:

```
R1= Вид Iris versicolor
```

Мы добились своей цели — обучили наш персептрон, и он после успешного обучения начал давать правильные ответы на поставленные вопросы.

Реализуем небольшую вспомогательную функцию, которая визуально показывает границу, разделяющую две области, характеризующие различные виды ирисов (листинг 4.22).

Листинг 4.22

```
# Это промежуточный код, он не является рабочим
# Визуализация разделительной границы
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # Настроить генератор маркеров и палитру
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # Вывести поверхность решения
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
        np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Показать образцы классов
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8,
            c=cmap(idx), marker=markers[idx], label=cl)

# Нарисовать картинку
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('Длина чашелистика, см')
plt.ylabel('Длина лепестка, см')
plt.legend(loc='upper left')
plt.show()
```

Сначала мы определяем цвета и маркеры и затем в строке `ListedColormap` создаем из списка цветов палитру (карту цветов). Далее мы определяем минимальные и максимальные значения для двух признаков и на основе массивов `X`, содержащих эти признаки, создаем пару матричных массивов `xx1` и `xx2`, используя для этого функцию `mesh-grid()` библиотеки NumPy. Учитывая, что мы тренировали наш персептрон на двух признаках, мы должны сгладить исходные матричные массивы и создать новую матрицу с тем же самым числом столбцов, что и у тренировочного подмножества цветков ириса. Для этого здесь применяется метод `predict()`, позволяющий идентифицировать метки классов `Z` для соответствующих точек матрицы. После преобразования формы идентифицированных меток классов `Z` в матрицу с таким же размером, что и у `xx1` и `xx2`, можно начертить контурный график, используя для этого функцию `contourf()` библиотеки matplotlib, которая ставит в соот-

ветствие различным областям решения разный цвет по каждому идентифицированному классу в матричном массиве.

Этот фрагмент кода добавим к модулю листинга 4_21_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 4_22_1) в сопровождающем книгу файловом архиве (см. *приложение*). После выполнения этого программного кода будут цветом выделены области, разделяющие два вида наших цветков (рис. 4.49).

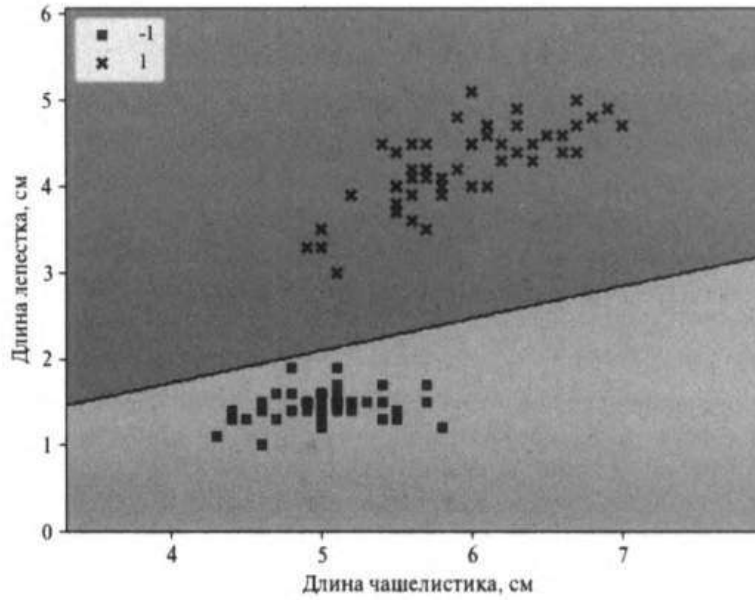


Рис. 4.49. Цветовое выделение областей, разделяющих два вида ирисов

Из рис. 4.49 видно, что персептрон в процессе обучения нашел ту границу, которая идеально разделила все образцы тренировочной выборки цветков ириса. Это был идеальный случай. Однако на практике такого результата можно добиться не всегда. *Сходимость*, или возможность определить четкие границы между разными классами объектов, представляет для персептрона одну из самых больших проблем. Фрэнк Розенблатт математически доказал, что правило обучения персептрона сходится, только если два класса могут быть разделены линейной гиперплоскостью. Однако если полностью разделить классы такой линейной границей решения невозможно, то без установления максимального числа эпох веса никогда не прекратят обновляться (обучение никогда не закончится).

4.11. Адаптивные линейные нейроны

В предыдущих разделах мы рассматривали персептрон Розенблатта. Теперь изучим другой тип однослойной нейронной сети — так называемый *адаптивный линейный нейрон* (ADAptive Linear Neuron, ADALINE). Концепция ADALINE была предло-

жена американским профессором Бернардом Уидроу (рис. 4.50) спустя всего несколько лет после создания алгоритма персептрона Ф. Розенблатта. Алгоритм обучения адаптивного нейрона интересен тем, что он реализует более продвинутый алгоритм машинного обучения для задач классификации.

Рис. 4.50. Бернард Уидроу

Основное отличие правила обучения ADALINE (также известного как правило Уидроу — Хоффа, или дельта-правило) от правила обучения персептрона Розенблатта в том, что в нем для обновления весов используется линейная функция активации, а не единичная ступенчатая, как в персептроне. В ADALINE эта функция активации $f(z)$ представляет собой просто тождественное отображение чистого входа.

Помимо линейной функции активации, которая используется для извлечения весов, далее с целью распознавания меток классов применяется так называемый *квантизатор*, т. е. функция, аналогичная встречавшейся ранее единичной ступенчатой функции. Структура адаптивного нейрона представлена на рис. 4.51.

Если сравнить рис. 4.51 с иллюстрацией алгоритма обучения персептрона, который мы видели ранее, то ADALINE отличается тем, что вместо бинарных значений

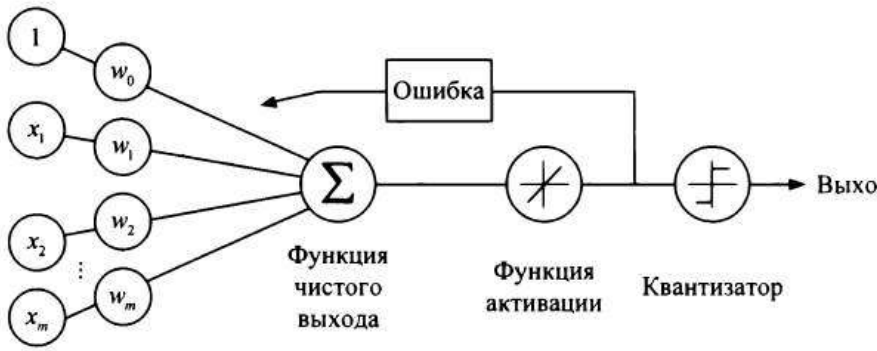


Рис. 4.51. Структура адаптивного нейрона

соответствия объектов тому или иному классу теперь для вычисления ошибки предсказания и обновления весов используется непрерывный выход из линейной функции активации.

Одна из ключевых составляющих алгоритмов машинного обучения с учителем состоит в задании целевой функции, которая подлежит оптимизации во время процесса обучения. Эта целевая функция часто является функцией стоимости, или функцией потерь, которую мы хотим минимизировать. В случае с ADALINE можно задать *функцию стоимости*, которая извлекает веса в виде суммы квадратичных ошибок или суммы отклонений расчетных результатов от истинных меток классов.

В отличие от единичной ступенчатой функции, основное преимущество этой непрерывной линейной функции активации состоит в том, что функция стоимости становится дифференцируемой. Еще одно хорошее свойство этой функции стоимости заключается в том, что она выпуклая, — следовательно, нам удастся использовать простой и одновременно мощный алгоритм оптимизации — *алгоритм градиентного спуска*. С его помощью мы можем найти оптимальные веса связей, которые минимизируют ошибку при решении ранее рассмотренной задачи классификации ирисов. Сущность алгоритма градиентного спуска представлена на рис. 4.52.

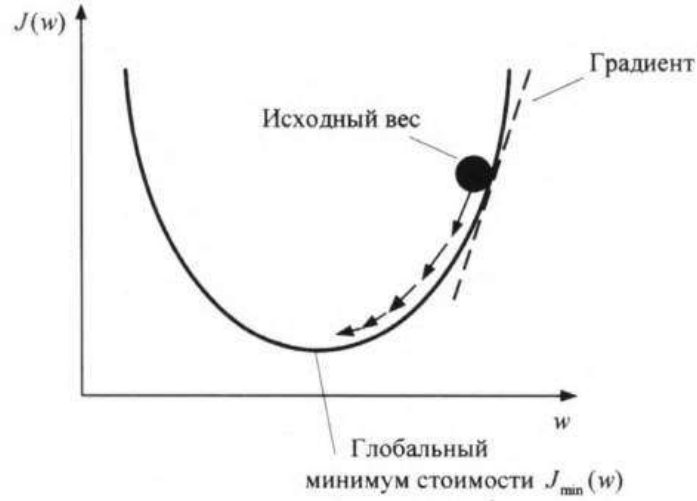


Рис. 4.52. Сущность алгоритма градиентного спуска

Идея градиентного спуска заключается в спуске вниз по склону оврага до тех пор, пока не будет достигнут локальный или глобальный минимум (минимизируем потери правильных решений или стоимость ошибок). На каждой итерации обучения мы делаем один шаг в противоположную от градиента сторону, при этом размер шага определяется значением темпа обучения и наклоном градиента (его угловым коэффициентом). Не будем углубляться в математическую сущность этого метода, отметим только, что для определения стоимости ошибки необходимо вычислить частную производную функции стоимости относительно каждого веса.

Учитывая, что правило персептрона и правило ADALINE очень похожи, мы возьмем определенную ранее реализацию персептрона и изменим метод `fit()` таким образом, чтобы веса обновлялись путем минимизации функции стоимости методом градиентного спуска. В листинге 4.23 приведен программный код, реализующий адаптивный линейный нейрон.

Листинг 4.23

```
# Это вспомогательный программный модуль
# Адаптивный линейный нейрон
class AdaptiveLinearNeuron(object):

    def __init__(self, rate=0.01, niter=10):
         self.rate = rate   # rate - темп обучения (между 0.0 и 1.0)
         self.niter = niter # niter - проходы по тренировочному набору данных

    def fit(self, X, y):
        self.weight = np.zeros(1 + X.shape[1])
        self.cost = []
        for i in range(self.niter):
            output = self.net_input(X)
            errors = y - output
            self.weight[1:] += self.rate * X.T.dot(errors)
            self.weight[0] += self.rate * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost.append(cost)
        return self

    def net_input(self, X):
        # Вычисление чистого входного сигнала
        return np.dot(X, self.weight[1:]) + self.weight[0]

    def activation(self, X):
        # Вычисление линейной активации
       return self.net_input(X)

    def predict(self, X):
        # Возвращает метку класса после единичного шага (предсказание)
        return np.where(self.activation(X) >= 0.0, 1, -1)
```

Входными данными для этой модели персептрона является двумерный массив:

```
X[n_sam ples, n_features]
```

где:

- `n_samples` — количество образцов (количество строк в массиве данных обучающей выборки);

- `n_features` — количество признаков (количество входов или сенсоров в обучающей выборке).

Целевые значения в обучающей выборке представлены одномерным массивом `y[n_samples]`.

В качестве результата своей работы этот адаптивный нейрон будет возвращать объект `self`.

Используя такую реализацию адаптивного нейрона, мы теперь можем инициализировать новые объекты `AdaptiveLinearNeuron`, задавая темп обучения `rate` и число эпох `n_iter`, т. е. циклов (уроков) прохождения по тренировочному набору данных. При помощи метода обучения `fit()` мы рассчитываем веса в атрибуте `self.w_`.

В процессе обучения формируются следующие значения:

- `w_` — одномерный массив весов (веса связей после обучения);
- `errors_` — массив или список ошибок в тренировочном наборе данных в каждой эпохе обучения.

Вместо того чтобы обновлять веса после каждого образца тренировочной выборки (как в персептроне), здесь вычисляется градиент на основе всего тренировочного набора входных данных, для чего используются специализированные функции библиотеки NumPy. Аналогично предыдущей реализации персептрона мы аккумулируем значения стоимости ошибки в списке `self.cost_`, чтобы после тренировки удостовериться, что алгоритм сходится.

На практике часто требуется сначала поэкспериментировать, чтобы найти хороший темп обучения `rate` для оптимальной сходимости. Поэтому для начала мы проверим эффективность работы двух темпов обучения: `rate = 0.01` и `rate = 0.0001`. Для оценки результатов обучения построим график зависимости стоимости ошибок от числа эпох (циклов обучения), чтобы увидеть, насколько хорошо адаптивный персептрон обучился на тренировочных данных. Реализуем это в следующем фрагменте программного кода (листинг 4.24).

Листинг 4.24

```
# Это промежуточный код, он не является рабочим
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))
# Обучение при rate = 0.01
aln1 = AdaptiveLinearNeuron(0.01, 10).fit(X,y)
ax[0].plot(range(1, len(aln1.cost) + 1), np.log10(aln1.cost), marker='o')
ax[0].set_xlabel('Эпохи')
ax[0].set_ylabel('log(Сумма квадратичных ошибок)')
ax[0].set_title('ADALINE. Темп обучения 0.01')

# Обучение при rate = 0.0001
aln2 = AdaptiveLinearNeuron(0.0001, 10).fit(X,y)
ax[1].plot(range(1, len(aln2.cost) + 1), aln2.cost, marker='o')
ax[1].set_xlabel('Эпохи')
```

```
ax[1].set_ylabel('Сумма квадратичных ошибок')
ax[1].set_title('ADALINE. Темп обучения 0.0001')
plt.show()
```

Здесь на основе класса `AdaptiveLinearNeuron` мы создали два адаптивных нейрона, или объекта: `aln1` и `aln2`:

```
aln1 = AdaptiveLinearNeuron(0.01, 10).fit(X,y)
aln2 = AdaptiveLinearNeuron(0.0001, 10).fit(X,y)
```

Первому нейрону мы задали темп обучения `0.01`, второму — `0.0001`. Далее обоим нейронам определили количество циклов обучения — `10` и передали в модуль обучения `fit()` обучающую выборку в виде двух массивов: (`X`, `y`). Для построения графиков использовали набор команд библиотеки matplotlib.

Объединим листинги 4.23 и 4.24 с модулями ввода исходных данных. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 4_24_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результаты работы объединенного программного модуля представлены на рис. 4.53.

Из графиков видно, что мы можем получить две проблемы. Левая диаграмма показывает, что именно произойдет, если выбрать слишком высокий темп обучения. Вместо минимизации функции стоимости ошибки после каждой новой эпохи (цикла обучения) ошибка увеличивается, потому что мы промахиваемся по глобальному минимуму и вряд ли когда-нибудь в него попадем. Когда мы смотрим на правый график, то видим, что стоимость уменьшается. Вместе с тем выбранный темп обучения `0.0001` настолько низок, что через 10 шагов обучения мы будем все еще достаточно далеко от минимума ошибки. Для полной сходимости алгоритм потребовал бы очень большого количества эпох.

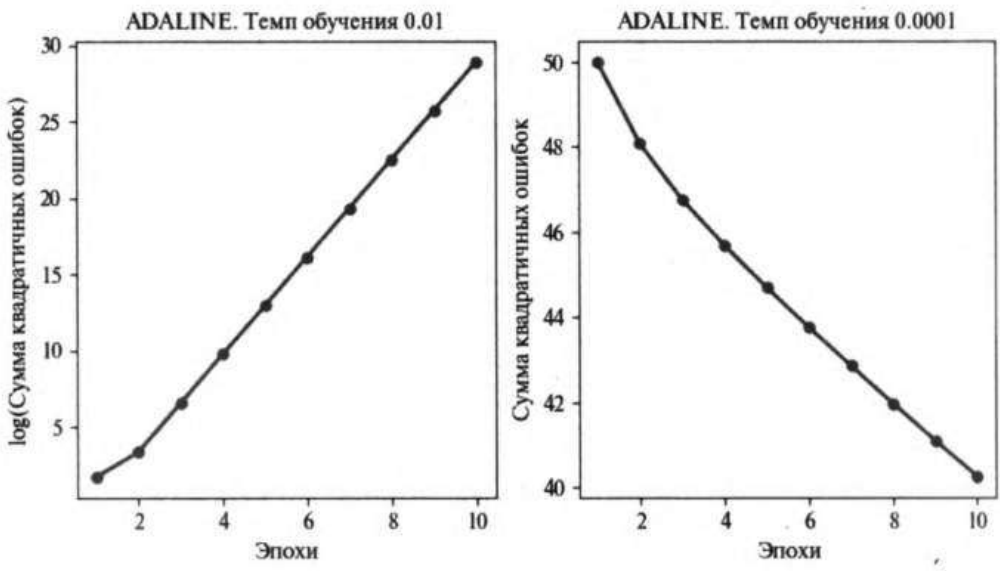


Рис. 4.53. Характер зависимости стоимости ошибок обучения от темпа обучения

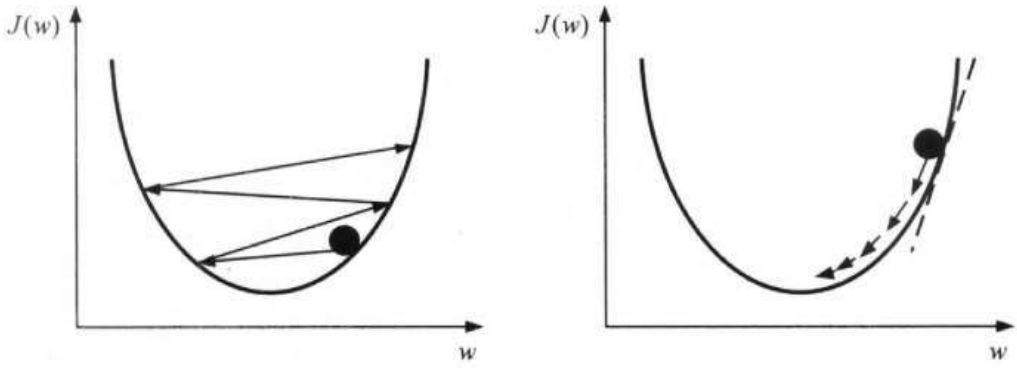


Рис. 4.54. Варианты минимизации ошибок обучения при разных темпа обучения: *слева* — высокий; *справа* — низкий

Более наглядно эти два варианта минимизации стоимости ошибки показаны на рис. 4.54.

Многие алгоритмы для получения оптимального качества машинного обучения требуют выполнения масштабирования исходных признаков. Алгоритм градиентного спуска является одним из таких алгоритмов. Достаточно часто используется метод масштабирования признаков, называемый *стандартизацией*, который придает нашим данным свойство стандартного нормального распределения. Среднее значение каждого признака центрируется в значении 0, а столбец признака имеет единичное стандартное отклонение, т. е. равное 1. Стандартизацию нашей обучающей выборки можно легко получить при помощи методов `mean()` и `std()` библиотеки NumPy. Напишем программный код, который выполнит стандартизацию нашей обучающей выборки и проведет обучение адаптивного нейрона уже на ее основе. В листинге 4.25 приведен текст этой программы.

Листинг 4.25

```
# Это промежуточный код, он не является рабочим
# Стандартизуем обучающую выборку
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

# Обучение на стандартизованной выборке при rate = 0.01
aln = AdaptiveLinearNeuron(0.01, 10)
aln.fit(X_std,y)

# Строим график зависимости стоимости ошибок от эпох обучения
plt.plot(range(1, len(aln.cost) + 1), aln.cost, marker='o')
plt.xlabel('Эпохи')
plt.ylabel('Сумма квадратичных ошибок')
plt.show()
```

```
# Строим области принятия решений
plot_decision_regions(X_std, y, classifier=aln)
plt.title('ADALINE (градиентный спуск)')
plt.xlabel('Длина чашелистика [стандартизованная]')
plt.ylabel('Длина лепестка [стандартизованная]')
plt.legend(loc='upper left')
plt.show()
```

Здесь на основе класса `AdaptiveLinearNeuron` мы создали адаптивный нейрон, или объект `aln`:

```
aln = AdaptiveLinearNeuron(0.01, 10)
aln.fit(X_std,y)
```

Мы задали темп обучения `0.01`, определили количество циклов обучения — `10` и передали в модуль обучения `fit()` обучающую выборку в виде массивов — (`X_std`, `y`). Для построения графиков использовали набор команд библиотеки matplotlib.

Объединим листинг 4_24_1 с листингом 4.25. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (4_25_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результаты работы объединенного программного кода представлены на рис. 4.55.

На стандартизированной обучающей выборке адаптивный нейрон сходится (обучается) даже при темпе обучения `0.01`. При этом даже десяти циклов обучения оказалось достаточно, чтобы корректно построить область разделения цветков двух видов (рис. 4.56).

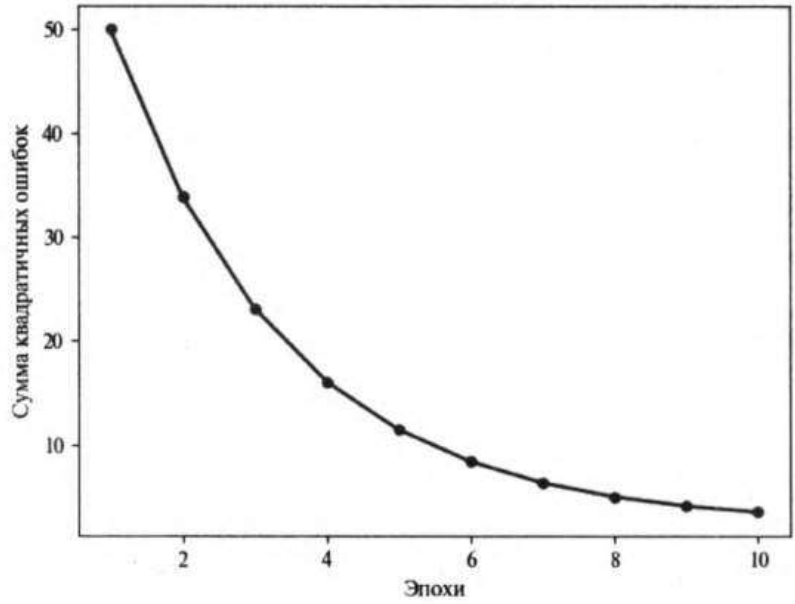


Рис. 4.55. Характер зависимости стоимости ошибок обучения от темпа обучения , на основе стандартизованной обучающей выборки и адаптивного линейного нейрона

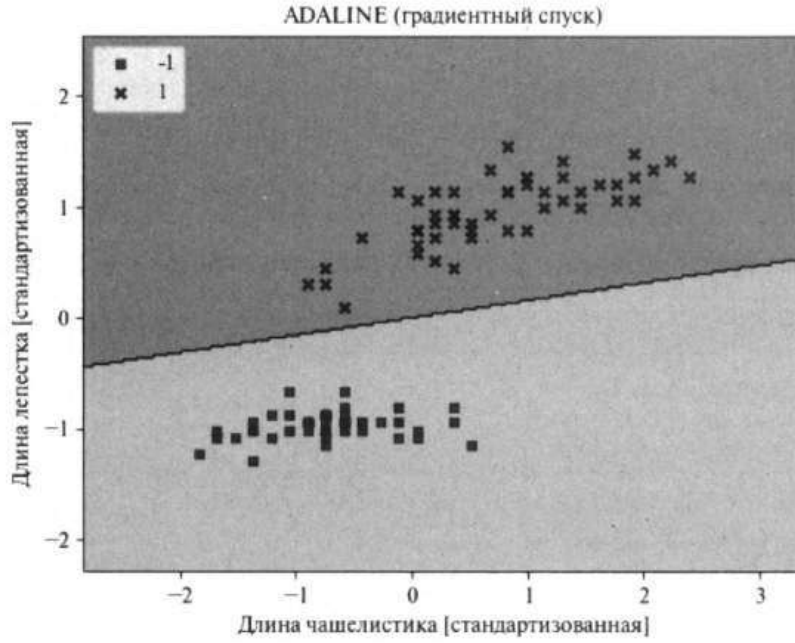


Рис. 4.56. Цветовое выделение областей, разделяющих два вида ирисов на основе стандартизованной обучающей выборки и адаптивного линейного нейрона

Проверим, насколько успешно обучился наш адаптивный персептрон. Напишем и запустим следующий программный код, в котором передадим на вход персептрона значения параметров цветков, отсутствующих в обучающей выборке (листинг 4.26).

Листинг 4.26

```
# Это промежуточный код, он не является рабочим
i1 = [-1.5, -0.75]
# i1 = [0.25, 1.1]
R1 = aln.predict(i1)
print('R1=', R1)

if (R1 == 1):
    print('R1= Вид Iris setosa')
else:
    print('R1= Вид Iris versicolor')
```

При входных параметрах `i1 = [-1.5, -0.75]` получим следующий ответ:

```
R1= -1
R1= Вид Iris versicolor
```

Цветок с такими параметрами будет отнесен к виду Iris versicolor.

При входных параметрах `i1 = [0.25, 1.1]` получим следующий ответ:

```
R1= 1
R1= Вид Iris setosa
```

Цветок с такими параметрами будет отнесен к виду Iris setosa.

Отсюда можно сделать вывод, что обучение адаптивного персептрона прошло успешно.

Полный текст программ из *разд. 4.10* и *4.11* приведен в листинге 4.27.

Листинг 4.27

```
# Модуль Lerning_Iris
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Загрузка из Интернета массива из 150 элементов
# Загрузка их в объект DataFrame и печать
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
df = pd.read_csv(url, header=None)
print('Массив')
print(df.to_string())

# Выборка из объекта DF 100 элементов (столбец 4 - название цветков),
# загрузка его в одномерный массив Y и печать
y = df.iloc[0:100, 4].values
print('Значение четвертого столбца Y - 100')
print(y)

# Преобразование названий цветков (столбец 4)
# в одномерный массив (вектор) из -1 и 1
y = np.where(y == 'Iris-setosa', -1, 1)
print('Значение названий цветков  в виде -1 и 1, Y - 100')
print(y)

# Выборка из объекта DF массива 100 элементов (столбец 0 и столбец 2),
# загрузка его в массив X (матрица) и печать
X = df.iloc[0:100, [0, 2]].values
print('Значение X - 100')
print(X)
print('Конец X')

# Формирование параметров значений для вывода на график
# Первые 50 элементов (строки 0-50, столбцы 0, 1)
plt.scatter(X[0:50, 0], X[0:50, 1], color='red', marker='o', label='щетинистый')
# Следующие 50 элементов (строки 50-100, столбцы 0, 1)
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='разноцветный')
```

```
# Формирование названий осей и вывод графика на экран
plt.xlabel('Длина чашелистика')
plt.ylabel('Длина лепестка')
plt.legend(loc='upper left')
plt.show()

# Описание класса Perceptron
class Perceptron(object):
    '''
    Классификатор на основе персептрона.
    Параметры
    eta:float  - Темп обучения (между 0.0 и 1.0)
    n_iter:int - Проходы по тренировочному набору данных.
    Атрибуты
    w_: 1-мерный массив - Весовые коэффициенты после подгонки.
    errors_: список - Число случаев ошибочной классификации в каждой эпохе.
    '''

    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

    '''
    Выполнить подгонку модели под тренировочные данные.
    Параметры
    X : массив, форма = [n_samples, n_features] тренировочные векторы,
    где
                                    n_samples - число образцов и
                                    n_features - число признаков,
    y : массив, форма = [n_samples] Целевые значения.
    Возвращает
    self: object
    '''
    def fit(self, x, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.errors_ = []
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    ''' Рассчитать чистый вход '''
    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]
```

```
    ''' Вернуть метку класса после единичного скачка '''
    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)

# Тренировка
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Эпохи')
# число ошибочно классифицированных случаев во время обновлений
plt.ylabel('Число случаев ошибочной классификации')
plt.show()

# Визуализация границы решений
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # настроить генератор маркеров и палитру
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # вывести поверхность решения
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
        np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # показать образцы классов
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8,
            c=cmap(idx), marker=markers[idx], label=cl)

# Нарисовать картинку
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('Длина чашелистика, см')
plt.ylabel('Длина лепестка, см')
plt.legend(loc='upper left')
plt.show()

# Адаптивный линейный нейрон
class AdaptiveLinearNeuron(object):
```

```
    '''
    Классификатор на основе ADALINE (ADAptive Linear NEuron).
    Параметры
    eta : float - Темп обучения (между 0.0 и 1.0)
    n iter : in - Проходы по тренировочному набору данных
    Атрибуты
    w_ : 1-мерный массив - Веса после подгонки.
    errors_ : список - Число случаев ошибочной классификации в каждой
    эпохе.
    '''
    def __init__(self, rate=0.01, niter=10):
         self.rate = rate
         self.niter = niter

    def fit(self, X, y):
        ''' Выполнить подгонку под тренировочные данные.
        Параметры
        X : (массив), форма = [n_samples, n_features] - тренировочные векторы,
                          где n_samples - число образцов,
                              n_features - число признаков.
        y : массив, форма = [n_samples] - Целевые значения.
        Возвращает
        self : объект
        '''

        self.weight = np.zeros(1 + X.shape[1])
        self.cost = []
        for i in range(self.niter):
            output = self.net_input(X)
            errors = y - output
            self.weight[1:] += self.rate * X.T.dot(errors)
            self.weight[0] += self.rate * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost.append(cost)
        return self

    def net_input(self, X):
        # Вычисление чистого входного сигнала
        return np.dot(X, self.weight[1:]) + self.weight[0]

    def activation(self, X):
        # Вычислительная линейная активация
        return self.net_input(X)

    def predict(self, X):
        # Вернуть метку класса после единичного скачка
        return np.where(self.activation(X) >= 0.0, 1, -1)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))
```

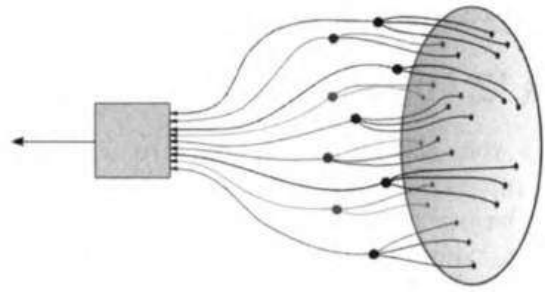
```
# Обучение при rate = 0.01
aln1 = AdaptiveLinearNeuron(0.01, 10).fit(X,y)

ax[0].plot(range(1, len(aln1.cost) + 1), np.log10(aln1.cost), marker='o')
ax[0].set_xlabel('Эпохи')
ax[0].set_ylabel('log(Сумма квадратичных ошибок)')
ax[0].set_title('ADALINE. Темп обучения 0.01')

# Обучение при rate = 0.01
aln2 = AdaptiveLinearNeuron(0.0001, 10).fit(X,y)

ax[1].plot(range(1, len(aln2.cost) + 1), aln2.cost, marker='o')
ax[1].set_xlabel('Эпохи')
ax[1].set_ylabel('Сумма квадратичных ошибок')
ax[1].set_title('ADALINE. Темп обучения 0.0001')
plt.show()

# Стандартизуем обучающую выборку
X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()

# Обучение на стандартизованной выборке при rate = 0.01
aln = AdaptiveLinearNeuron(0.01, 10)
aln.fit(X_std, y)

# Строим график зависимости стоимости ошибок от эпох обучения
plt.plot(range(1, len(aln.cost) + 1), aln.cost, marker='o')
plt.xlabel('Эпохи')
plt.ylabel('Сумма квадратичных ошибок')
plt.show()

# Строим область принятия решений
plot_decision_regions(X_std, y, classifier=aln)
plt.title('ADALINE (градиентный спуск)')
plt.xlabel('Длина чашелистика [стандартизованная]')
plt.ylabel('Длина лепестка [стандартизованная]')
plt.legend(loc='upper left')
plt.show()

i1 = [-1.5, -0.75]
# i1 = [0.25, 1.1]
R1 = aln.predict(i1)
print('R1=', R1)

if (R1 == 1):
    print('R1= Вид Iris setosa')
else:
    print('R1= Вид Iris versicolor')
```

4.12. Краткие итоги главы

В этой главе было показано, что персептрон — это простейший вид искусственных нейронных сетей. Детально рассмотрен очень важный аспект, а именно — каким образом происходит процесс самообучения персептрона. Этот материал надо детально изучить и хорошо усвоить, т. к. без знаний этих основ практически невозможно будет реализовать более сложные многослойные сети.

Следующая глава будет посвящена вопросам построения нейронных сетей на основе простейших нейронов. Мы по-прежнему будем использовать только Python без применения специализированных библиотек. Это позволит более глубоко изучить все механизмы работы нейронных сетей.

ГЛАВА 5



Построение многослойных нейронных сетей

В этой главе будет представлено максимально простое объяснение того, как работают нейронные сети, а также показаны способы их реализации в Python. Сам термин «нейронные сети» многих пугает — кажется, что это что-то запутанное и непонятное. Нейронные сети не такие уж и сложные, если в них разобраться на простых примерах.

Пояснения, приведенные в этой главе, позволят преодолеть страх перед искусственным интеллектом. Принцип работы нейронных сетей будет показан на примерах их реализации на языке Python с использованием некоторых уже готовых библиотек. В главе будут представлены следующие материалы:

- структура простейшего искусственного нейрона;
- программирование простейшего искусственного нейрона;
- построение сети из нейронов;
- обучение нейронной сети;
- подведение некоторых итогов.

Для запуска используемых в главе программных модулей были задействованы следующие дополнительные библиотеки:

- Numpy — версия 1.23.5;
- Matplotlib — версия 3.0.2.

5.1. Исследуем простейший искусственный нейрон

Для начала необходимо вспомнить, что представляет собой базовый компонент нейронной сети — нейрон. Нейрон принимает вводные данные, выполняет с ними определенные математические операции, а затем выводит результат. Простейший нейрон с двумя входными данными представлен на рис. 5.1.

Здесь каждый вход умножается на вес. Затем все взвешенные входы складываются вместе со смещением b, и все это передается в функцию активации Y:

$$Y = f\left(x_1 w_1 + x_2 w_2 + b\right).$$

Функция активации используется для сопоставления несвязанных входных данных с выводом, у которого простая и предсказуемая форма. Как правило, в качестве функции активации берут сигмоиду (рис. 5.2).

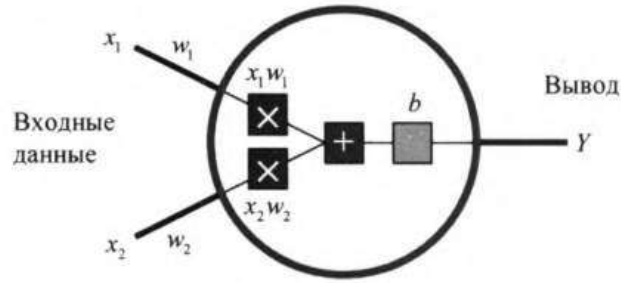


Рис. 5.1. Простейший искусственный нейрон

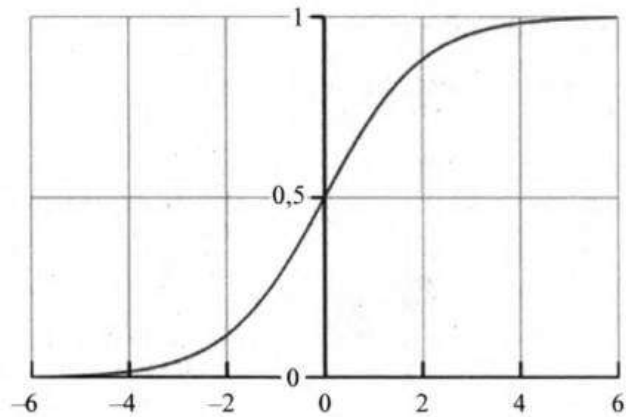


Рис. 5.2. Графическое представление функции активации (сигмоида)

Замечательное свойство этой функции заключается в том, что независимо от числа, поданного на вход, на выходе мы получим значение в диапазоне от 0 до 1. Это можно представить как сжатие чисел из диапазона $(-\infty; +\infty)$ в диапазон (0, 1). Значения крупных отрицательных чисел приближаются к нулю, а значения крупных положительных чисел — к единице.

С входными параметрами x_i и их весами w_i вроде все понятно. А что же такое параметр b — коэффициент смещения? Это коэффициент, с помощью которого мы можем дополнительно усилить или ослабить вес (значимость) какого-либо нейрона. Коэффициент смещения можно трактовать как еще один вход в нейрон, у которого значение самого сигнала всегда равно единице ($x_b = 1$), а вес w_b может принимать любые значения (рис. 5.3).

В этом случае функция активации будет принимать следующие значения:

$$Y = f\left(x_1 w_1 + x_2 w_2 + x_b w_b\right),$$

где $b = x_b w_b$

Посмотрим, как будут влиять значения параметров простейшего нейрона на принимаемое решение. Предположим, что мы имеем на входе в нейрон один сигнал x_1, который может принимать значения в интервале от -8 до $+8$. Посмотрим, как будет меняться форма сигмоиды при разных значениях весов: $w = 0,5; w = 1; w = 2$. Структура такого нейрона представлена на рис. 5.4.

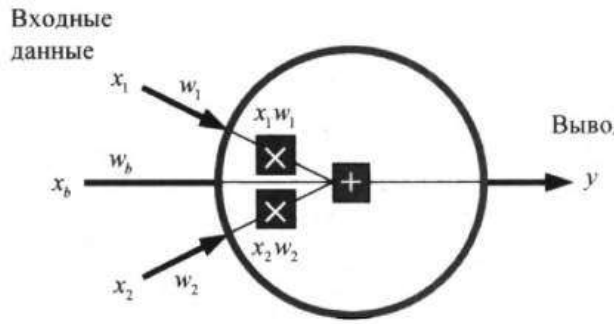


Рис. 5.3. Роль параметра смещения *b* в простейшем нейроне

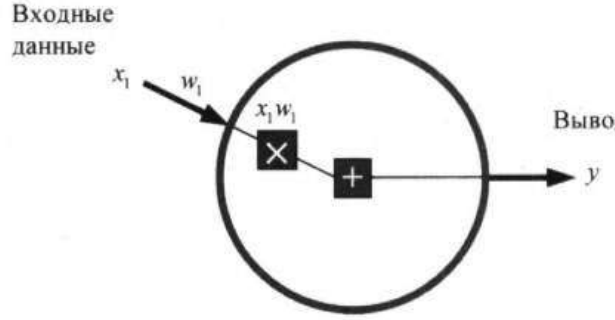


Рис. 5.4. Простейший нейрон с одним входным сигналом

Для исследования работы такого нейрона напишем следующую программу (листинг 5.1).

Листинг 5.1

```
# Модуль Grafik_Vesa
import matplotlib.pylab as plt  # библиотека формирования графиков
import numpy as np              # библиотека математических функций

x = np.arange(-8, 8, 0.1)       # интервал и шаг изменения сигнала от сенсора X
   # Значения весов
w1 = 0.5
w2 = 1.0
w3 = 2.0
```

```
    # Подписи значений весов для отображения в легенде на графике
l1 = 'Вес w = 0.5'
l2 = 'Вес w = 1.0'
l3 = 'Вес w = 2.0'
    # цикл построения графиков для 3 значений весов
for w, l in [(w1, l1), (w2, l2), (w3, l3)]: # организация цикла
                                            # для трех значений весов
    f = 1 / (1 + np.exp(-x * w))            # функция сигмоиды
    plt.plot(x, f, label=l)                 # построение графика для i-го веса (Wi)

plt.xlabel('x')         # подпись оси x
plt.ylabel('Y = f(x)') # подпись оси y
plt.legend(loc=4)       # Место легенды на графике (4 - правый нижний угол)
plt.show()              # вывод графиков
```

Эту программу можно разбить на четыре блока. В первом блоке подключаются необходимые библиотеки. Во втором — задаются начальные значения всем параметрам. Потом организуется цикл расчетов для формирования графиков. И наконец, в четвертом блоке осуществляется вывод результатов. В комментариях, приведенных в тексте программы, даны более подробные пояснения действий в каждой строке. Результаты работы программы представлены на рис. 5.5.

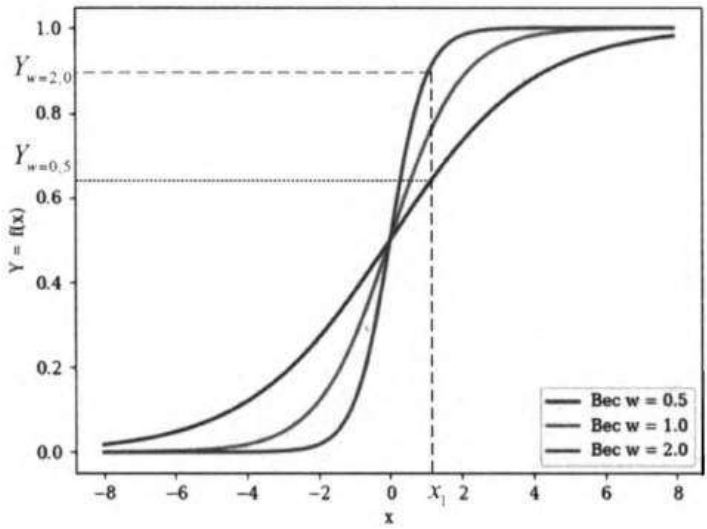


Рис. 5.5. Влияние веса связи на значение функции активации

Значение веса будет менять форму сигмоиды. Чем больше вес, тем круче график изменения выходного сигнала, а с уменьшением веса мы получаем более пологую кривую. Из приведенных на рис. 5.5 графиков видно, что при одной и той же величине входного сигнала x_1 его влияние на выходной параметр функции активации будет выше для большего веса ($Y_{w=2,0} > Y_{w=0,5}$).

Теперь проанализируем роль коэффициента смещения b. Предположим, что мы имеем на входе в нейрон один сигнал x_1, который может принимать значения в интервале от -8 до $+8$. Посмотрим, как будет меняться форма сигмоиды при разных значениях смещения b: $b=-2$, $b=0$, $b=2$. Структура такого нейрона представлена на рис. 5.6.

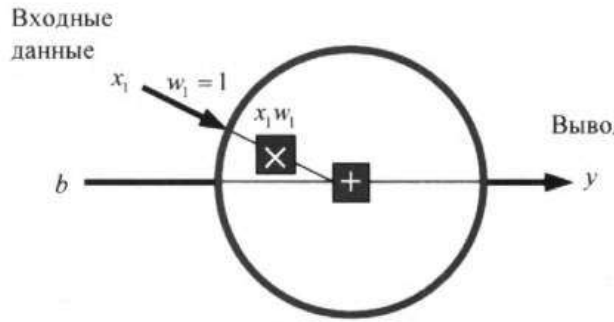


Рис. 5.6. Простейший нейрон с одним входным сигналом и смещением *b*

Для исследования работы этого нейрона напишем следующую программу (листинг 5.2).

Листинг 5.2

```
# Модуль Grafik_Smestil
import matplotlib.pylab as plt  # библиотека формирования графиков
import numpy as np              # библиотека математических функций

x = np.arange(-8, 8, 0.1)       # интервал и шаг изменения сигнала от сенсора X

# Значения смещения
b1 = -2
b2 = 0
b3 = 2
# Подписи значений смещения для отображения в легенде на графике
l1 = 'Смещение b = -2'
l2 = 'Смещение b = 0'
l3 = 'Смещение b = 2'
# цикл построения графиков для трех значений смещения
for b, l in [(b1, l1), (b2, l2), (b3, l3)]:  # организация цикла для трех значений
                                             # смещения
    f = (1 / (1 + np.exp((-x+b) * 1)))       # функция сигмоиды
    plt.plot(x, f, label=l)  # построение графика для i-го смещения (bi)

plt.xlabel('x')          # подпись оси x
plt.ylabel('Y = f(x)')   # подпись оси y
```

```
plt.legend(loc=4)          # место легенды на графике (4 - правый нижний угол)
plt.show()                 # вывод графиков
```

Эту программу также можно разбить на четыре блока. В первом блоке подключаются необходимые библиотеки. Во втором — задаются начальные значения всем параметрам. Потом организуется цикл расчетов для формирования графиков. И наконец, в четвертом блоке осуществляется вывод результатов. В комментариях, приведенных в тексте программы, даны более подробные пояснения действий в каждой строке. Результаты работы программы представлены на рис. 5.7.

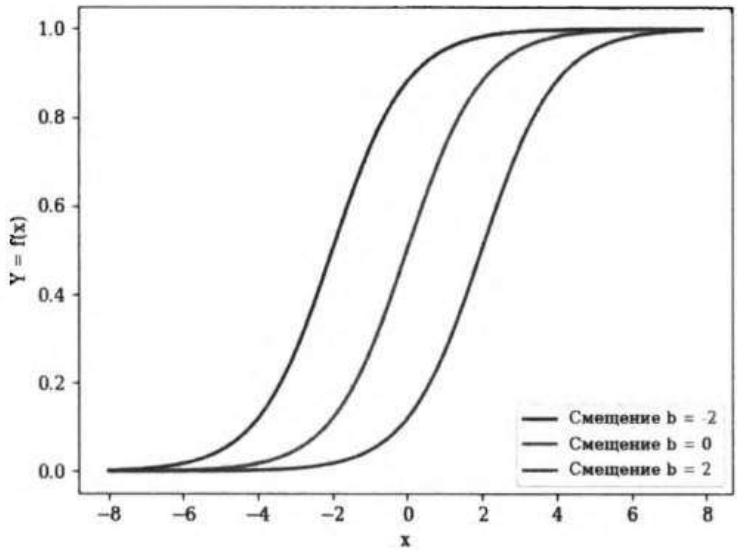


Рис. 5.7. Влияние коэффициента смещения *b* на значение функции активации

При разных значениях смещения форма сигмоиды не изменяется, но ее положение смещается вправо или влево по оси x. Чем больше b, тем правее будет расположен график изменения выходного сигнала, а с уменьшением величины b мы получаем смещение в левую сторону (рис. 5.7). Ну вот, мы наконец разобрались в предназначении параметра b. Теперь реализуем простейший нейрон в виде программного модуля.

5.2. Программируем простейший искусственный нейрон

Предположим, у нас есть нейрон с двумя входами, который использует сигмоиду в качестве функции активации и имеет следующие параметры:

- входы $x_1 = 2, x_2 = 3$;
- веса входов $w_1 = 0, w_2 = 1$;
- смещение $b = 4$.

При таких входных параметрах наш простейший нейрон должен выполнить следующие действия:

$$S = (x_1 w_1 + x_2 w_2) + b = (2 \cdot 0 + 3 \cdot 1) + 4 = (0 + 3) + 4 = 7;$$
$$Y = f(S) = f(7),$$

где f — функция активации.

Приступим к программированию простейшего нейрона (листинг 5.3). Для этого потребуется задействовать стандартную библиотеку NumPy. Это мощная вычислительная библиотека Python, в которой реализованы различные математические операции и функции.

Листинг 5.3

```
# Модуль Prostoy_Neiron
import numpy as np

# функция активации: f(x) = 1 / (1 + e^(-x))
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Создание класса Нейрон
class Neuron:
    def __init__(self, w, b):
        self.w = w
        self.b = b

    def y(self, x):                    # Сумматор
        s = np.dot(self.w, x) + self.b  # Суммируем входы
        return sigmoid(s)               # обращение к функции активации

Xi = np.array([2, 3])  # Задание значений входам x1 = 2, x2 = 3
Wi = np.array([0, 1])  # Веса входных сенсоров w1 = 0, w2 = 1
bias = 4               # Смещение b = 4
n = Neuron(Wi, bias)   # Создание объекта из класса Neuron
print('Y=', n.y(Xi))   # Обращение к нейрону
```

В первой строке мы подключили библиотеку NumPy. В ней мы воспользуемся функцией скалярного произведения `dot()`:

```
s = np.dot(self.w, x) + self.b
return sigmoid(s)
```

Здесь мы с помощью функции `dot()` получили значение `S` (скалярное произведение всех входов на их веса) и прибавили итог к смещению `b`, затем передали значение полученной суммы в функцию активации. Следующие действия программы описаны в комментариях. Мы присвоили значения входным сигналам, их весам и смещению. Затем создали объект из класса `Neuron`, передали в него исходные данные и в последней строке вывели результат. Запустив эту программу, мы получим следующее значение:

```
Y = 0.9990889488055994
```

Проверим, как будет меняться результат работы нейрона для разных значений смещения от –4 до +4 (итог представлен в табл. 5.1).

***Таблица 5.1.** Зависимость результатов работы простейшего нейрона от значения коэффициента смещения*

№ п/п	Коэффициент смещения *b*	Результат, выданный нейроном
1	–4	Y= 0.268
2	–3	Y= 0.500
3	–2	Y= 0.731
4	–1	Y= 0.880
5	0	Y= 0.952
6	1	Y= 0.982
7	2	Y= 0.993
8	3	Y= 0.997
9	4	Y= 0.999

Видно, что с ростом величины смещения значимость входного параметра *Y* увеличивается. А как это будет влиять на принятие решения? Если мы увеличим параметр *b*, значение выходной функции тоже увеличится, т. е. мы увеличиваем влияние персептрона на принятие решения. И наоборот, если мы уменьшаем параметр *b*, то значение выходной функции уменьшается, т. е. мы в итоге уменьшаем влияние персептрона на принятие решения.

5.3. Строим сеть из нейронов

Нейронная сеть по своей сути представляет собой группу связанных между собой нейронов. На рис. 5.8 показано, как выглядит простая нейронная сеть с тремя слоями.

Она содержит: один вводный слой, имеющий два входа: x_1 и x_2, один скрытый слой, состоящий из двух нейронов: h_1 и h_2, и один слой вывода, на котором находится один нейрон — o_1.

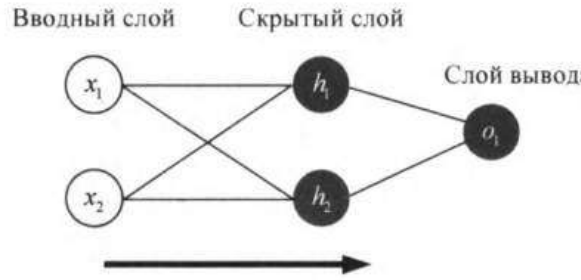


Рис. 5.8. Схема однослойной нейронной сети прямого распространения

Скрытым слоем называется любой слой между вводным слоем и слоем вывода, которые являются первым и последним слоями соответственно. Скрытых слоев может быть несколько. Обратите внимание на то, что входные данные для o_1 являются результатами вывода h_1 и h_2. По такому принципу и строятся нейронные сети. Направление передачи сигналов — слева направо. Такие сети относятся к сетям прямого распространения (feed forward).

Давайте воспользуемся показанной на рис. 5.8 схемой сети и представим, что в качестве ввода будет задано значение `x = [2, 3]`, а нейроны имеют вес `w = [0, 1]` и одинаковое смещение $b = 0$. В качестве функции активации примем сигмоиду. Схематично сеть с такими параметрами представлена на рис. 5.9.

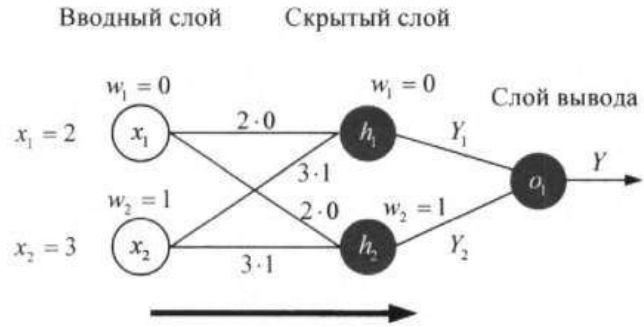


Рис. 5.9. Пример однослойной нейронной сети прямого распространения

Подсчитаем, какие результаты выдаст сеть с такими параметрами. Нейроны h_1 и h_2 на свои входы получат одинаковые значения: $x_1 = 2, x_2 = 3$. На выходе они сформируют сигналы Y_1 и Y_2, которые примут следующие значения:

$$Y_1 = f\left(x_1 w_1 + x_2 w_2 + b\right) = f(2 \cdot 0 + 3 \cdot 1 + 0) = f(3);$$

$$Y_2 = f\left(x_1 w_1 + x_2 w_2 + b\right) = f(2 \cdot 0 + 3 \cdot 1 + 0) = f(3),$$

где f — функция активации.

Далее выходы из нейронов скрытого слоя Y_1 и Y_2 станут входными сигналами на нейрон слоя вывода o_1. Затем нейрон o_1 сформирует общий выходной сигнал из сети Y:

$$Y = f\left(Y_1 w_1 + Y_2 w_2 + b\right) = f\left(Y_1 \cdot 0 + Y_2 \cdot 1 + 0\right).$$

Программный код, реализующий эту сеть, представлен в листинге 5.4.

Листинг 5.4

```
# Модуль Net1
import numpy as np
```

```
# функция активации
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Описание класса Нейрон
class Neuron:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def feedforward(self, inputs):
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)

# Описание класса Нейронная сеть из трех слоев
class OurNeuralNetwork:

    def __init__(self):
        weights = np.array([0, 1])  # веса (одинаковы для всех нейронов)
        bias = 0                    # смещение (одинаково для всех нейронов)

        # формируем сеть из трех нейронов
        self.h1 = Neuron(weights, bias)
        self.h2 = Neuron(weights, bias)
        self.o1 = Neuron(weights, bias)

    def feedforward(self, x):
        out_h1 = self.h1.feedforward(x)  # Формируем выход Y1 из нейрона h1
        out_h2 = self.h2.feedforward(x)  # Формируем выход Y2 из нейрона h2
        out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))  # Формируем выход Y
                                                                  # из нейрона o1
        return out_o1

network = OurNeuralNetwork()   # Создаем объект СЕТЬ из класса "Наша нейронная сеть"
x = np.array([2, 3])           # формируем входные параметры для сети X1=2, X2=3
print("Y=", network.feedforward(x))     # Передаем входы в сеть и получаем результат
```

Эта программа содержит несколько блоков.

1. Подключение библиотеки для работы с математическими функциями NumPy:

   ```
   import numpy as np
   ```

   Из этой библиотеки здесь используются следующие функции:

   - `array()` — формирование одномерного массива;
   - `dot()` — определение скалярного произведения элементов массива.

2. Описание класса `Neuron` (нейрон):

   ```
   class Neuron:
   ```

3. Описание класса `OurNeuralNetwork` (нейронная сеть из трех слоев):

```
class OurNeuralNetwork:
```

4. Создание объекта `СЕТЬ` из класса `"Наша нейронная сеть"`:

```
network = OurNeuralNetwork()
```

5. Формирование входных параметров для сети и передача их в нашу новоиспеченную сеть:

```
x = np.array([2, 3])
print(network.feedforward(x))
```

Меняя входные значения x, мы будем получать итоговый ответ Y от сети. Запустив программу на выполнение, мы получим результат, показанный на рис. 5.10.

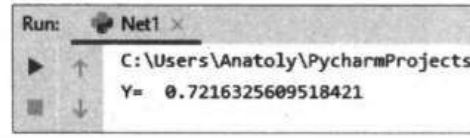


Рис. 5.10. Итог работы однослойной нейронной сети прямого распространения

5.4. Обучаем нейронную сеть

В предыдущем разделе мы показали, как из нейронов построить простейшую сеть, и реализовали ее в виде программного кода на Python. Затем на вход сети мы по́дали набор входных параметров и на выходе получили какой-то ответ. Сеть выполнила свою задачу на наборе тестовых цифр, абсолютно не связанных с реальной действительностью. Попробуем теперь поставить перед нашей сетью близкую к жизни задачу и научить ее находить правильное решение. Для того чтобы понять, как это делается, максимально упростим условия задачи. Попробуем научить нейронную сеть определять пол человека в зависимости от его веса и роста. Для этого сначала нужно сделать замеры этих параметров. Результаты такой работы приведены в табл. 5.2.

***Таблица 5.2.** Замеры роста и веса людей для подготовки обучающей выборки*

Имя	Вес, фунты	Рост, дюймы	Пол
Alice	133	65	Ж
Bob	160	72	М
Charlie	152	70	М
Diana	120	60	Ж

Здесь видно, что мужчины имеют бо́льшие рост и вес, чем женщины. Изменим символьное обозначение пола с М и Ж на 0 и 1, т. е. ответ сети 0 будем понимать как «мужчина», а ответ 1 — как «женщина». Также сделаем смещение роста и веса

на некоторую постоянную величину. Обычно для смещения выбираются средние показатели, поэтому мы обозначим следующие величины смещения: рост уменьшим на 135 футов (–135), вес уменьшим на 66 футов (–66). На основе этих данных сформируем обучающую выборку (табл. 5.3).

Таблица 5.3. Обучающая выборка для определения пола человека по его росту и весу

Имя	Вес (–135)	Рост (–66)	Пол
Alice	–2	–1	1
Bob	25	6	0
Charlie	17	4	0
Diana	–15	–6	1

Перед тренировкой нейронной сети выберем способ оценки того, насколько хорошо сеть справляется с задачами (ошибается или нет). Введем такое понятие, как *потери* (потеря правильного решения). В нашем примере для оценки потери будет использоваться расчет среднеквадратичной ошибки. Не будем углубляться в дебри математики, а рассмотрим, как делается расчет этой ошибки на простом примере. Допустим, мы имеем два значения: заведомо верное решение Y_{true} и предположительное решение Y_{pred} (полученный от сети ответ). Тогда величину ошибки сети E можно подсчитать следующим образом:

$$E = Y_{true} - Y_{pred}.$$

Давая сети задание решить нашу задачу многократно (n раз) и для каждого раза делая расчет ошибки (E_i), среднеквадратичную ошибку подсчитаем следующим образом:

$$E_{ср} = \frac{E_1^2 + E_2^2 + E_3^2 + ... + E_n^2}{n}.$$

Давайте разберемся с обозначениями в этих формулах:

- n — число рассматриваемых объектов, которое в нашем случае равно 4. Это Alice, Bob, Charlie и Diana;
- Y_{true} — истинное значение правильного ответа (для Alice значение $Y_{true} = 1$, она женщина);
- Y_{pred} — предполагаемое значение переменной. Это результат вывода сети (мужчина или женщина);
- $E_{ср}$ — средняя квадратичная ошибка.

Из всего здесь сказанного важно понять: чем меньше сеть будет ошибаться, тем значение этого показателя будет ниже, т. е. «лучшие предсказания = меньшие потери». Давайте реализуем на Python функцию, которая будет делать расчет этой ошибки. В листинге 5.5 приведен текст соответствующей программы.

Листинг 5.5

```
# Это вспомогательный программный модуль
# расчет среднеквадратической ошибки
def mse_loss(y_true, y_pred):
    # y_true и y_pred являются массивами numpy с одинаковой длиной
    return ((y_true - y_pred) ** 2).mean()
```

Обучение нейронной сети будет заключаться в подборе таких значений параметров нейронов, при которых ошибочность в принятии решений окажется минимальной, — т. е. нужно свести к минимуму потерю правильных решений. Ясно, что повлиять на предсказания сети можно при помощи изменения весов связей и смещений. Однако каким способом можно минимизировать потери? Рассмотрим это на примере с максимальным упрощением задачи. Оставим в обучающей выборке только одного человека — Alice (табл. 5.4).

***Таблица 5.4.** Упрощенная обучающая выборка для определения пола человека по его росту и весу*

Имя	Вес (–135)	Рост (–66)	Пол
Alice	–2	–1	1

Для Alice потеря правильного ответа будет просто квадратичной ошибкой, которую при значении правильного ответа ($Y_{true} = 1$) можно подсчитать по следующей формуле:

$$L = \left(1 - Y_{pred}\right)^2.$$

Еще один способ понимания потери правильного решения — это представление потери в виде функции от таких параметров сети, как веса связей и смещения (не нужно путать веса связей нейронов с весом наших героев). То есть

$$L = f(W, b).$$

С помощью этой функции можно, меняя веса связей и смещения, найти минимальное значение ошибки. Давайте обозначим индивидуальным индексом каждый вес связи и смещения в рассматриваемой сети (рис. 5.11).

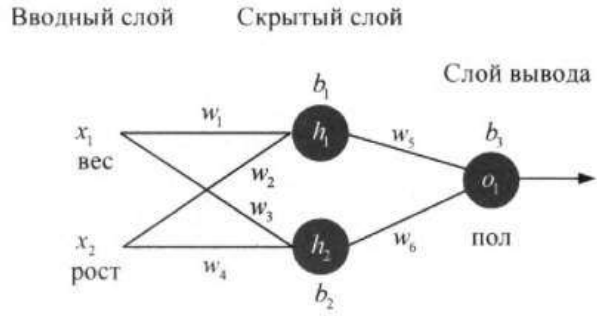


Рис. 5.11. Перечень параметров, влияющих на ошибочность решения нейронной сети

Теперь можно записать в общем виде многовариантную функцию зависимости потери правильного решения от параметров нейронной сети:

$$L = f(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3).$$

Представим, что нам нужно изменить вес w_1. В таком случае как изменится потеря L после внесения поправок в w_1? На этот вопрос может ответить частная производная dL/dw_1. А как ее вычислить? Здесь нужно использовать *систему подсчета частных производных*, или так называемый *метод обратного распространения ошибки* (backprop). Понять суть этого метода, выраженного языком математики, достаточно сложно. Но если не углубляться в математические рассуждения и положиться на выводы великих математиков, то частную производную (dL/dw_1) можно получить из следующих выражений:

$$\frac{dL}{dw_1} = \frac{dL}{dY_{\text{pred}}} \cdot \frac{Y_{\text{pred}}}{dh_1} \cdot \frac{dh_1}{dw_1};$$

$$\frac{dL}{dY_{\text{pred}}} = -2 \cdot (1 - Y_{\text{pred}});$$

$$\frac{dY_{\text{pred}}}{dh_1} = w_5 \cdot f(h_1 w_5 + h_2 w_6 + b3);$$

$$\frac{dh_1}{dw_1} = x_1 \cdot f(x_1 w_1 + x_2 w_2 + b_1).$$

Без глубокого знания основ высшей математики в этих методах и формулах легко запутаться. Поэтому мы и не станем в них углубляться, а для лучшего понимания принципа их работы рассмотрим простой пример. Будем по-прежнему использовать сведения только об одном человеке с именем Alice (см. табл. 5.4). Посмотрим, что получится, если мы сведения об Alice прогоним через нашу сеть при начально заданных параметрах: все веса связей в сети $w_i = 1$, все смещения $b_i = 0$. Работая с обычным калькулятором, получим следующие результаты:

$$h_1 = f(x_1 w_1 + x_2 w_2 + b_1) = f(1 \cdot (-2) + 1 \cdot (-1) + 0) = f(-2 - 3 + 0) = f(-3) = 0{,}0474;$$

$$h_2 = f(x_1 w_3 + x_2 w_4 + b_2) = f(1 \cdot (-2) + 1 \cdot (-1) + 0) = f(-2 - 3 + 0) = f(-3) = 0{,}0474;$$

$$o_1 = f(h_1 w_5 + h_2 w_4 + b_3) = f(0{,}0474 + 0{,}0474 + 0) = 0{,}524.$$

Примечание

Следует напомнить, что в качестве функции активации $f(x)$ в приведенных здесь выражениях использовалась сигмоида. Например, значение $f(-3) = 0{,}0474$ получено с применением именно этой функции, что можно легко проверить, если обратить внимание на следующий фрагмент кода программы:

```
mport numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
print('h1= ', sigmoid(-3))
```

Результат работы фрагмента этого программного кода: `h1= 0.04742587317756678`.

Итак, расчеты с калькулятором показали, что на выходе из нейронной сети мы получим результат:

$$Y_{\text{pred}} = 0,524.$$

Однако это дает нам слабое представление о том, является Alice мужчиной или женщиной, поскольку значение предполагаемого решения $Y_{\text{pred}} = 0,524$ находится почти в середине интервала 0–1 (0 — мужчина, 1 — женщина).

Давайте теперь подсчитаем частную производную dL/dw_1, выполнив соответствующие расчеты на основе данных нашей Alice. Результаты будут следующие:

$$dL/dY_{\text{pred}} = -2 \cdot \left(1 - Y_{\text{pred}}\right) = -2 \cdot (1 - 0,524) = -0,952;$$

$$dY_{\text{pred}}/dh_1 = w_5 \cdot f\left(h_1 w_5 + h_2 w_6 + b_3\right) = 1 \cdot f\left(0,0474 + 0,0474 + 0\right) = 0,249;$$

$$dh_1/dw_1 = x_1 \cdot f\left(x_1 w_1 + x_2 w_2 + b_1\right) = -2 \cdot f\left(-2 + (-1) + 0\right) = -0,0904;$$

$$dL/dw_1 = -0,952 \cdot 0,249 \cdot (-0,0904) = 0,0214.$$

Результаты получили, а что с ними делать дальше? А дальше нужно использовать алгоритм оптимизации под названием «стохастический градиентный спуск», который скажет нам, как именно поменять веса связей и смещения для минимизации потерь. Суть этого метода отражается в следующем выражении:

$$w_1 = w_1 - \left(\eta \cdot dL/dw_1\right),$$

где η — константа, которую можно трактовать как скорость обучения.

Все, что мы делаем, так это вычитаем dL/dw_1 из w_1. При этом:

- если dL/dw_1 имеет положительное значение, то w_1 уменьшится, что приведет к уменьшению потери L;
- если dL/dw_1 имеет отрицательное значение, то w_1 увеличится, что приведет к увеличению потери L.

Если мы правильно применим эти действия на каждый вес связи и на каждое смещение в сети, то добьемся снижения потерь L, а тем самым и улучшения эффективности работы сети (она станет меньше ошибаться). То есть сеть будет постепенно обучаться и принимать все более правильные решения.

В нашем конкретном случае с Alice мы с помощью обыкновенного калькулятора получили следующее значение $dL/dw_1 = 0,0214$ (при весе $w_1 = 1$). Поскольку рассчитанное значение частной производной оказалось положительным, то значение

веса w_1 возрастет, а значит, будет улучшена эффективность работы сети. А на какую величину возрастет или уменьшится вес w_1, определит коэффициент скорости обучения η. Чем больше будет значение этого коэффициента, тем быстрее сеть будет обучаться, и наоборот. Вспомните пример с кенгуру из *главы 4*: η — это как раз длина прыжка кенгуру при движении к водопою. Его прыжок не должен быть очень маленьким, но и не слишком большим. Оптимизация этого параметра в процессе обучения сети — отдельный вопрос. А пока сформулируем итог: что же нужно делать для того, чтобы сеть смогла обучиться выполнять заданные ей действия?

1. Выбираем один пункт из набора данных (обучающей выборки).
2. Подсчитываем все частные производные потери по весу или смещению.
3. Выполняем обновления каждого веса и смещения.
4. Возвращаемся к первому пункту.

Давайте посмотрим, как это работает на практике, и реализуем приведенный укрупненный алгоритм обучения сети на примере задачи распознавания людей по их росту и весу.

В листинге 5.6 приведен полный код соответствующей программы на Python.

Листинг 5.6

```
# Модуль Rost_Ves
import numpy as np

def sigmoid(x):
    # Функция активации sigmoid - f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))

def deriv_sigmoid(x):
    # Производная от sigmoid - f'(x) = f(x) * (1 - f(x))
    fx = sigmoid(x)
    return fx * (1 - fx)

# расчет среднеквадратичной ошибки
def mse_loss(y_true, y_pred):
    # y_true и y_pred являются массивами numpy с одинаковой длиной
    return ((y_true - y_pred) ** 2).mean()

class OurNeuralNetwork:
    def __init__(self):
        # Вес
        self.w1 = np.random.normal()
        self.w2 = np.random.normal()
        self.w3 = np.random.normal()
        self.w4 = np.random.normal()
        self.w5 = np.random.normal()
        self.w6 = np.random.normal()
```

```
        # Смещения
        self.b1 = np.random.normal()
        self.b2 = np.random.normal()
        self.b3 = np.random.normal()

    def feedforward(self, x):
        # x является массивом numpy с двумя элементами
        h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
        h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
        o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
        return o1

    def train(self, data, all_y_trues):
        learn_rate = 0.1
        epochs = 1000  # количество циклов во всем наборе данных

        for epoch in range(epochs):
            for x, y_true in zip(data, all_y_trues):
                # --- Выполняем обратную связь (нам понадобятся эти значения
                #     в дальнейшем)
                sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
                h1 = sigmoid(sum_h1)

                sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
                h2 = sigmoid(sum_h2)

                sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
                o1 = sigmoid(sum_o1)
                y_pred = o1

                # --- Подсчет частных производных
                # --- Наименование: d_L_d_w1 представляет "частично L / частично w1"
                d_L_d_ypred = -2 * (y_true - y_pred)

                # Нейрон o1
                d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
                d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
                d_ypred_d_b3 = deriv_sigmoid(sum_o1)

                d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
                d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

                # Нейрон h1
                d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
                d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
                d_h1_d_b1 = deriv_sigmoid(sum_h1)

                # Нейрон h2
                d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
```

```
                d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
                d_h2_d_b2 = deriv_sigmoid(sum_h2)

                # --- Обновляем вес и смещения
                # Нейрон h1
                self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
                self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
                self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1

                # Нейрон h2
                self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
                self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
                self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

                # Нейрон o1
                self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
                self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
                self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

            # --- Подсчитываем общую потерю в конце каждой фазы
            if epoch % 10 == 0:
                y_preds = np.apply_along_axis(self.feedforward, 1, data)
                loss = mse_loss(all_y_trues, y_preds)
                print("Epoch %d loss: %.3f" % (epoch, loss))

# Определение набора данных
data = np.array([
    [-2, -1],  # Alice
    [25, 6],   # Bob
    [17, 4],   # Charlie
    [-15, -6], # Diana
])

all_y_trues = np.array([
    1,  # Alice
    0,  # Bob
    0,  # Charlie
    1,  # Diana
])

# Тренируем нашу нейронную сеть!
network = OurNeuralNetwork()
network.train(data, all_y_trues)
```

Рассмотрим основные блоки этой программы.

1. В первой строке подключаем библиотеку стандартных математических функций NumPy.

2. Определяются следующие функции:
   - `sigmoid()` — функция активации;
   - функция расчета производной от `sigmoid`;
   - функция расчета среднеквадратичной ошибки.
3. Создается класс `OurNeuralNetwork`, реализующий нейронную сеть, изображенную на рис. 5.1.
4. Внутри класса `OurNeuralNetwork` создается функция, которая реализует алгоритм обучения нашей нейронной сети. В этой функции мы задаем скорость и количество циклов обучения:

```
learn_rate = 0.1
epochs = 1000
```

5. Формируем массив обучающей выборки `data` (вес и рост наших героев см. в табл. 5.3).
6. Формируем массив правильных ответов `all_y_trues` (пол наших героев см. в табл. 5.3).
7. Последним шагом запускаем нашу сеть на учебу (тренируем искать правильные ответы):

```
network = OurNeuralNetwork()
network.train(data, all_y_trues)
```

Вот, собственно, и всё. Запускаем нашу программу. Через каждые 10 уроков она будет выводить уровень потерь правильных решений. Если посмотреть на эти результаты, то будет видно, что после каждых десяти циклов обучения сеть делает все меньше и меньше ошибок. Для того чтобы продемонстрировать результаты расчетов, изменим в программе всего одну строку:

```
if epoch % 100 == 0
```

Тогда мы получим уровень потерь правильных решений через каждые 100 циклов обучения (рис. 5.12).

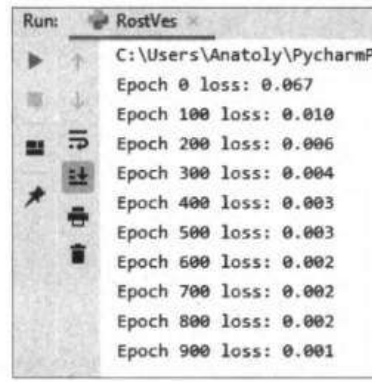


Рис. 5.12. Уменьшение ошибочности решений нейронной сети в процессе обучения

Примечание

Обратите внимание, что результаты расчетов, проведенных вами, будут несколько отличаться от тех, что показаны на рис. 5.12, поскольку процессы предсказания в нейронных сетях носят вероятностный характер.

Если теперь на основе выданных программой данных построить график зависимости потерь правильных решений от количества циклов обучения, то получится картина, показанная на рис. 5.13.

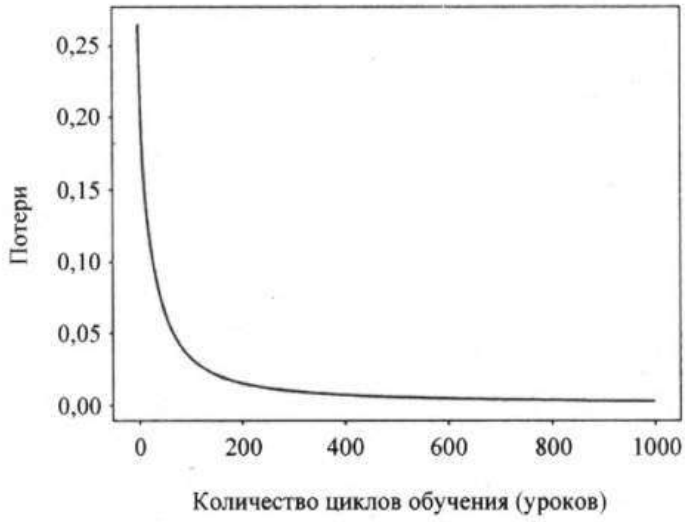


Рис. 5.13. Зависимость ошибочности решений нейронной сети от количества циклов обучения

Здесь видно, что уже после 400 уроков наша сеть получила достаточно опыта для того, чтобы справляться с поставленной задачей.

Теперь дадим нашей обученной сети задание — определить пол людей, которых не было в обучающей выборке. Добавим в программу следующие строки (листинг 5.7).

Листинг 5.7

```
# Это промежуточный код, он не является рабочим
# Делаем предсказания
emily = np.array([-7, -3])  # 128 фунтов, 63 дюйма
frank = np.array([20, 2])   # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily))
print("Frank: %.3f" % network.feedforward(frank))
```

Здесь у нас записаны два человека:

- женщина `Emily` с параметрами: вес — 128 фунтов, рост — 63 дюйма;
- мужчина `Frank` с параметрами: вес — 15 фунтов, рост — 68 дюймов.

Объединим листинги 5.6 и 5.7. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 5_7_1) в сопровождающем книгу файловом архиве (см. *приложение*).

Запустив объединенную программу, получим следующий ответ (рис. 5.14).

При обучении нейронной сети мы условились, что правильный ответ 1 будет означать «женщина», а ответ 0 — «мужчина». Для `Emily` мы получили ответ — `0.996`, что близко к единице, — значит, это женщина. Для `Frank` мы получили ответ `0.054`, что близко к нулю, — значит, это мужчина. То есть обученная сеть справилась с поставленной задачей. Не забываем, что процессы предсказания в нейронных сетях носят вероятностный характер, и ваши результаты будут несколько отличаться от тех, что приведены на рис. 5.14.

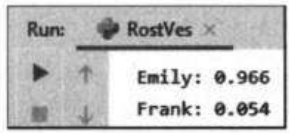


Рис. 5.14. Результат работы обученной нейронной сети

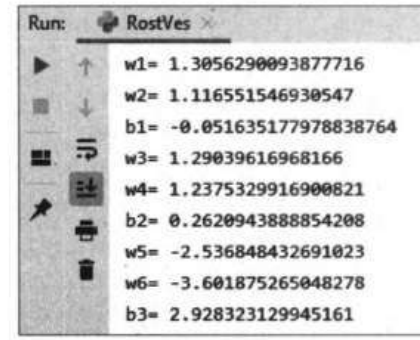


Рис. 5.15. Значения весов связей и смещений после обученной нейронной сети

Теперь посмотрим, какие же значения получили веса связей и смещений после завершения процесса обучения. Для этого после строки программы:

```
print("Epoch %d loss: %.3f" % (epoch, loss))
```

добавим следующие строки:

```
print('w1=', self.w1)
print('w2=', self.w2)
print('b1=', self.b1)
print('w3=', self.w3)
print('w4=', self.w4)
print('b2=', self.b2)
print('w5=', self.w5)
print('w6=', self.w6)
print('b3=', self.b3)
```

Объединим листинг программы 5_7_1 с этими строками. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 5_7_2) в сопровождающем книгу файловом архиве (см. *приложение*). После такой модификации программа из листинга 5_7_2 выдаст результат, показанный на рис. 5.15.

Это, по сути, и есть итоги обучения. Не забываем также, что после каждого запуска программы у вас будут получены результаты, отличные от результатов, приведен-

ных на рис. 5.15, поскольку процессы обучения и принятия решений в нейронных сетях носят вероятностный характер.

Если полученные параметры сохранить, а потом загрузить в структуру сети перед выполнением задания, то сеть справится с поставленной задачей уже без предварительного обучения. То есть нейронная сеть, однажды натренированная на решение какого-либо класса задач, в дальнейшем будет выполнять все порученные задания уже без обучения. Например, если сеть однажды научили распознавать символы, то она будет читать и понимать любые тексты. Если сеть научили выделять лицо человека на фотографии, то она сможет выделить лицо любого человека на любой фотографии. И не только на фотографии, но и на изображении с видеокамеры!

5.5. Последовательность шагов проектирования нейронных сетей

В предыдущем разделе мы спроектировали простейшую нейронную сеть, имеющую:

- два входа;
- один скрытый слой с двумя нейронами (h_1, h_2);
- один слой вывода с одним нейроном (a_1).

Затем написали программный код, который позволил с помощью этой сети решить очень упрощенную задачу. Важно отметить, что этот код достаточно простой и далек от оптимальности. Его можно использовать лишь для изучения основ и принципов работы нейронных сетей. Программный код настоящей нейронной сети будет намного сложнее и выглядеть несколько иначе. Но не стоит огорчаться и бояться трудностей. Мы уже упоминали о готовых модулях и библиотеках, которые буквально одной командой можно подключить к своей программе. Все математические хитрости и их программная реализация спрятаны внутри этих модулей. Нужно просто научиться использовать потенциал, заложенный в уже готовых библиотеках.

Попробуем теперь описать шаги, которые нужно сделать для того, чтобы создать нейронную сеть, способную решать практические задачи разного класса. Последовательность таких шагов представлена на рис. 5.16.

На первом шаге нужно четко сформулировать задачу, которая требует решения. Для этой задачи надо определить перечень входных параметров (сколько таких параметров, в каких пределах они изменяются), а также что является конечным результатом и в каком виде он должен быть представлен на выходе из сети. Сама сеть на этом этапе представляет собой некий «черный ящик», в котором происходят какие-то процессы (рис. 5.17).

Итак, с задачей определились. Теперь нужно сформировать структуру нейронной сети, т. е. определить содержимое этого «черного ящика». А содержимым его будет структура нейронной сети (сколько нейронов, сколько слоев, сколько связей между

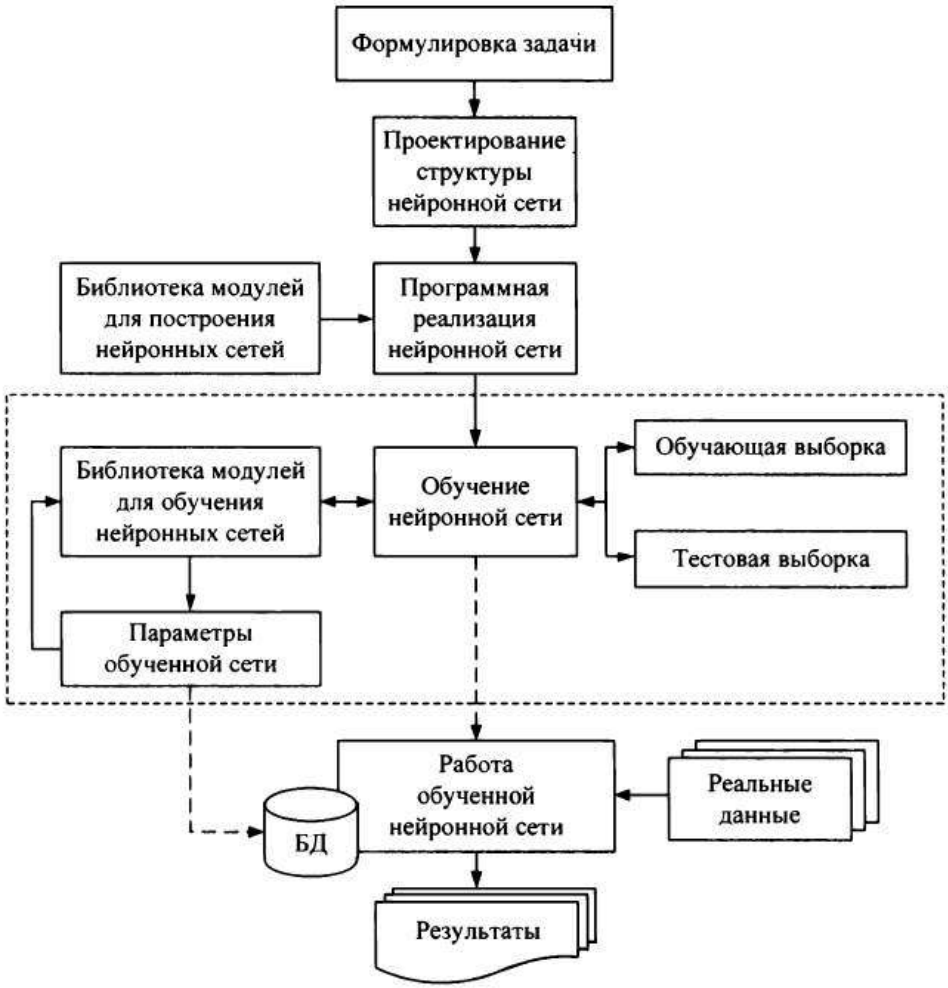


Рис. 5.16. Последовательность шагов проектирования нейронных сетей

Рис. 5.17. Этап «Формулировка задачи» нейронной сети

нейронами и каковы эти связи). На этом этапе «черный ящик» становится уже не абсолютно черным, он немного посветлел (рис. 5.18).

Теперь мы эту сеть отправляем учиться. Для этого нам понадобятся учитель, учебники, задачники. Обучение проходит циклично. Сначала нашей сети показывают материалы из учебника, в котором находятся задачи с готовыми решениями. Сеть сопоставляет условия каждой задачи с правильным ответом и, получая по́дсказки от учителя, формирует алгоритм поиска правильных ответов. Этот алгоритм запо-

минается в виде весов связей между нейронами и коэффициентами смещения. После всех уроков результаты успешного обучения запоминаются в виде оптимальных параметров сети.

Но на этом обучение не заканчивается — нужно теперь выполнить контрольную работу, чтобы убедиться в том, что обучение не прошло впустую. Для этого сети дают задания из задачника. В задачнике содержатся задачи, аналогичные тем, которые были в учебнике, но с несколько иными условиями. Если сеть успешно их решила, то обучение считается завершенным и такой сети можно поручать решать практически важные задачи из реальной жизни. При этом сеть всегда носит с собой багаж полученных на уроках знаний в виде базы данных (БД), в которой хранятся параметры обученной сети. Процессы обучения нейронных сетей схематично показаны на рис. 5.19.

После обучения нейронная сеть уже не «черный ящик», а искусственный интеллект, натренированный на решение задач определенного класса. Теперь на входы в сеть можно подавать реальные данные, а на выходе она будет выдавать готовое

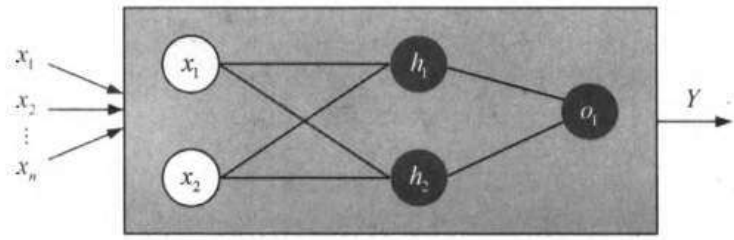


Рис. 5.18. Этап проектирования структуры нейронной сети

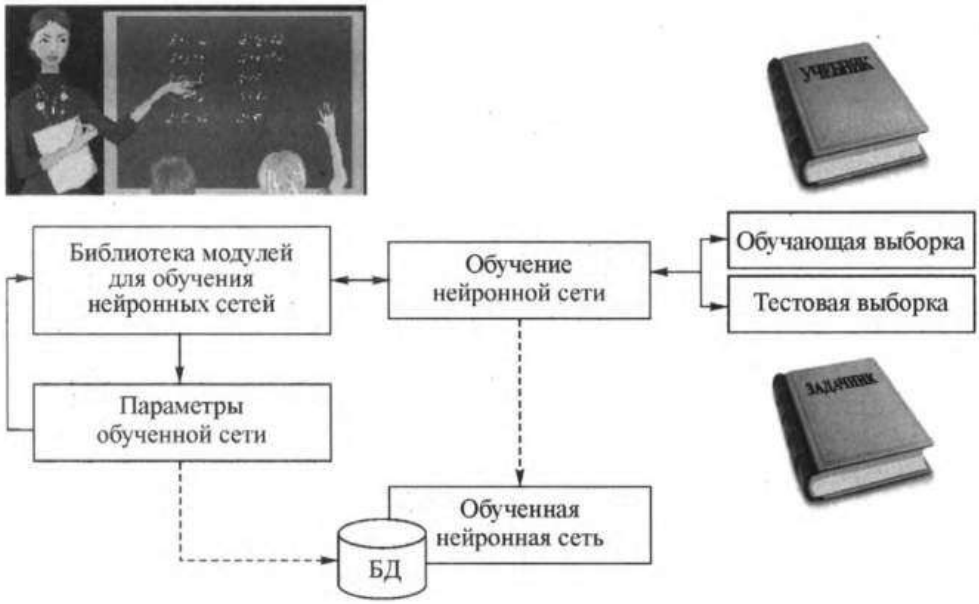


Рис. 5.19. Этап обучения нейронной сети

решение поставленной задачи. Например, если мы научили сеть распознавать лица на фотографиях, то, подав на вход в сеть любую фотографию, на выходе мы получим тот же портрет с выделенным на нем лицом (рис. 5.20).

Итак, мы познакомились с теоретическими основами проектирования и обучения нейронных сетей. При этом не углублялись в тонкости математического фундамента, а использовали достаточно упрощенные примеры. Теперь можно перейти к практике построения элементов искусственного интеллекта на основе специально созданных для этих целей модулей и библиотек.

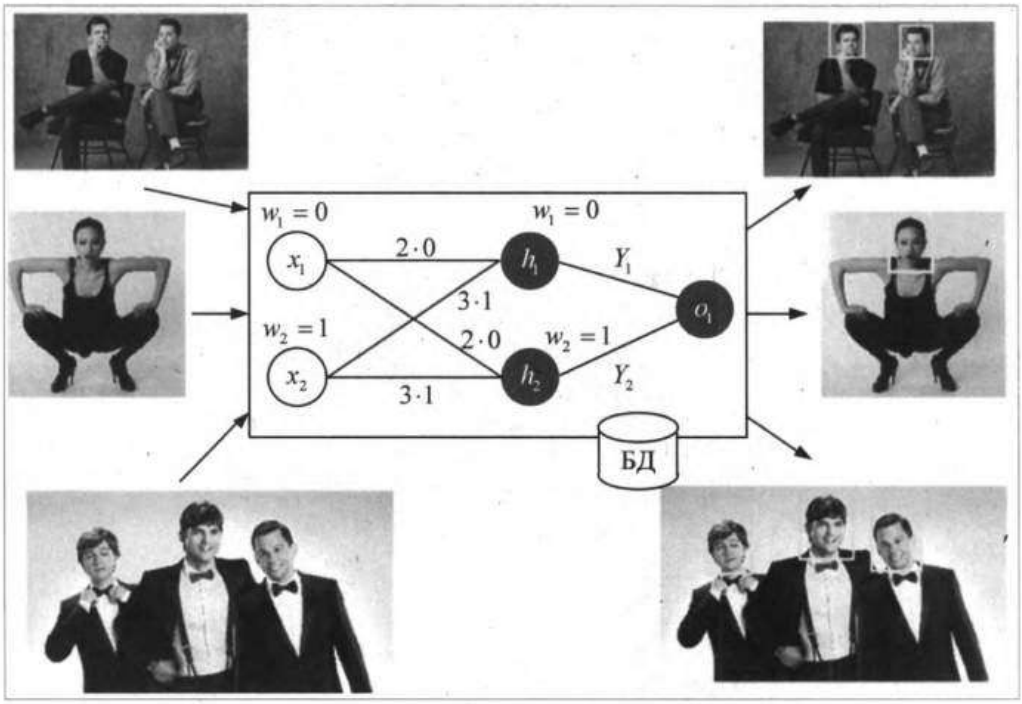


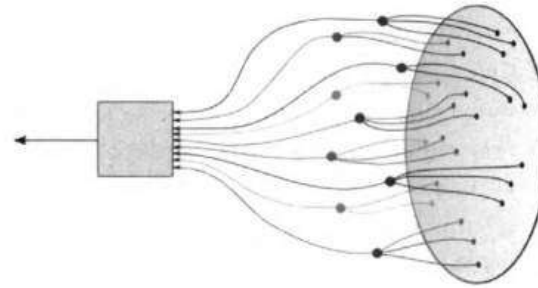
Рис. 5.20. Работа обученной нейронной сети

5.6. Краткие итоги главы

В этой главе были рассмотрены материалы, касающиеся проектирования нейронных сетей: постановка задачи, формирование структуры сети, создание и обучающей, и тестовой выборок, обучение (или тренировка) сети, практическое использование обученной сети. Весь этот процесс был показан на достаточно простом, но понятном примере. И мы по-прежнему использовали только Python и некоторые библиотеки работы с математическими функциями. При этом программный код получался достаточно большим и сложным. Однако такой подход позволяет лучше понять все процессы, которые происходят внутри нейронной сети.

В следующей главе будет показано, как можно ускорить и упростить работу по программированию нейронных сетей, используя для этого специализированные библиотеки.

ГЛАВА 6



Полезные библиотеки для создания нейронных сетей на Python

Язык программирования Python поддерживает разработчиков на протяжении всего цикла программной разработки, что ведет к высокой производительности и дает уверенность в ее конечном результате. У Python много достоинств, имеющих большое значение при разработке проектов, связанных с искусственным интеллектом и машинным обучением. К ним можно отнести:

- встроенные библиотеки;
- простоту интеграции;
- легкость в создании прототипов;
- открытый код;
- объектно-ориентированную парадигму;
- высокую производительность;
- платформенную независимость.

Именно эти свойства еще больше повышают популярность языка. Огромное количество Python-библиотек для искусственного интеллекта и машинного обучения существенно упрощают и ускоряют разработку. Простой синтаксис и читаемость способствуют быстрому тестированию сложных процессов и делают язык понятным для всех. Для реализации алгоритмов машинного обучения и искусственного интеллекта необходимо хорошо структурированное и проверенное окружение — только так можно достичь наилучших результатов. Многочисленные библиотеки Python, предназначенные для машинного обучения, позволяют существенно сократить время создания проектов. В этой главе мы познакомимся с лучшими из них. В частности, будут представлены следующие материалы:

- обзор специализированных библиотек для создания элементов искусственного интеллекта;
- библиотека для построения нейронных сетей PyBrain и примеры работы с ней;
- библиотека scikit-learn (формирование обучающих данных, тренировка модели и оценка качества обучения, классификаторы и примеры их реализации);

- библиотека Keras и сверточные нейронные сети, пример реализации сверточной нейронной сети;
- построение нейронных сетей с библиотекой TensorFlow (простые нейронные сети, нейронные сети для классификации изображений).

Итак, приступим к программированию элементов искусственного интеллекта, используя мощь и функциональность уже готовых специализированных библиотек.

6.1. Виды специализированных библиотек

6.1.1. NumPy

Python SciPy Stack — это набор библиотек, специально предназначенных для научных вычислений. Каждый, кто собрался использовать Python в науке, должен познакомиться с этим стеком.

Самый фундаментальный пакет из этого стека — NumPy. Он позволяет выполнять основные операции над *n*-мерными массивами и матрицами: сложение, вычитание, деление, умножение, транспонирование, вычисление определителя и т. д. Благодаря механизму векторизации NumPy повышает производительность и соответственно ускоряет выполнение операций.

6.1.2. Pandas

Pandas — это пакет, предназначенный для простой и интуитивно понятной работы с «помеченными» и «реляционными» данными. Пакет работает в связке с NumPy и, помимо математических вычислений, обеспечивает их агрегацию и визуализацию.

6.1.3. matplotlib

Для визуализации обработанных данных используется пакет matplotlib — это еще одна библиотека из пакета SciPy Stack. Именно возможности matplotlib позволяют рассматривать Python как полноправного конкурента MATLAB или Mathematica. Библиотека matplotlib является основным инструментом для визуализации данных на языке Python и поддерживается различными платформами и IDE (iPython, Jupyter и пр.).

С помощью этого пакета можно создавать:

- линейные графики;
- графики рассеяния;
- гистограммы;
- круговые диаграммы;
- спектрограммы и т. п.

Библиотека низкоуровневая, что означает большой объем кода для расширенной визуализации. Но производительность и работа с привычным языком позволяют закрыть глаза на этот недостаток.

6.1.4. Theano

Theano — это одна из самых мощных библиотек в нашем списке. Для этого есть несколько причин:

- тесная интеграция с NumPy;
- использование центрального процессора (CPU) и графического процессора (GPU) для повышения производительности;
- встроенные механизмы оптимизации кода;
- расширения для юнит-тестирования и самопроверки.

Theano используется там, где необходимо произвести вычисления с большой точностью максимально быстро, — в частности, в нейронных сетях и машинном обучении.

По своей сути, это научная математическая библиотека, которая позволяет определять, оптимизировать и вычислять математические выражения, в том числе и в виде многомерных массивов. Основой большинства систем машинного обучения и искусственного интеллекта является многократное вычисление сложных математических выражений. Theano позволяет проводить подобные вычисления в сотни раз быстрее, вдобавок она отлично оптимизирована под использование GPU, имеет модуль для символьного дифференцирования, а также предлагает широкие возможности для тестирования кода. Модули этой библиотеки могут работать с очень большими и сложными нейронными сетями. Она обеспечивает уменьшение времени разработки и увеличение скорости выполнения приложений — в частности, основанных на алгоритмах глубоких нейронных сетей. Ее единственный недостаток — не слишком простой синтаксис (по сравнению с TensorFlow), особенно для новичков.

6.1.5. TensorFlow

Эта библиотека от Google была разработана специально для обучения нейронных сетей. Библиотека использует многоуровневую систему узлов для обработки большого количества данных, что расширяет сферу ее применения далеко за пределы научной области. TensorFlow — это система машинного обучения, которая может стать замечательным инструментом, если у вас много данных и имеется желание постичь новейшее достижение в сфере искусственного интеллекта, называемое глубоким обучением. TensorFlow задействуется для поиска новых планет, помогает врачам сканировать диагностические снимки и выявлять болезни, помогает спасать леса, предупреждая власти о признаках незаконной вырубки.

6.1.6. Keras

Библиотека Keras задействует возможности TensorFlow и Theano в качестве компонентов. Простой подход к дизайну и невероятная расширяемость позволяют быстро начать работу с этой библиотекой и не менять ее для серьезного моделирования. Keras используется в построении и обучении нейронных сетей, а также при решении задачи распознавания устной речи.

Библиотека особенно удобна для начинающих разработчиков, которые хотят проектировать и разрабатывать собственные нейронные сети. Также Keras можно использовать при работе со сверточными нейронными сетями. В ней реализованы алгоритмы нормализации, оптимизации и активации слоев. Keras не является библиотекой полного цикла машинного обучения. Однако она функционирует как очень дружественный, расширяемый интерфейс, увеличивающий модульность и выразительность (в том числе использует эффективность и производительность других библиотек).

6.1.7. PyBrian

Библиотека PyBrian написана на языке Python. Она реализует различные топологии нейронных сетей: сети прямого распространения, рекуррентные нейронные сети. При необходимости можно создать топологию собственной структуры. Библиотека предоставляет исследователю гибкие, простые в использовании, но в то же время мощные инструменты для реализации задач из области машинного обучения, тестирования и сравнения эффективности различных алгоритмов.

Итак, переходим к созданию собственных программных модулей с использованием мощи и эффективности описанных в этом разделе библиотек.

6.2. Библиотека для построения нейронных сетей PyBrain

6.2.1. Общие сведения о библиотеке PyBrain

PyBrain — одна из лучших Python-библиотек для изучения и реализации большого количества разнообразных алгоритмов, связанных с нейронными сетями. Она являет собой удачный пример совмещения компактного синтаксиса Python с хорошей реализацией большого набора различных алгоритмов из области машинного обучения. Эта библиотека предназначена для следующих категорий пользователей:

- *исследователей* — предоставляет единообразную среду для реализации различных алгоритмов, избавляя от потребности в использовании десятков различных библиотек. Позволяет сосредоточиться на самом алгоритме, а не особенностях его реализации;
- *студентов и учащихся школ* — с помощью PyBrain удобно выполнять домашнее задание, курсовой проект или вычисления в дипломной работе. Гибкость ее архитектуры позволяет удобно реализовывать разнообразные сложные методы, структуры и топологии;
- *лекторов и преподавателей* — обучение методам машинного обучения было одной из основных целей при создании библиотеки. Ее разработчики будут рады, если результаты их труда помогут в подготовке грамотных школьников, студентов и квалифицированных специалистов;

- *разработчиков* — эта библиотека из серии проектов с открытым кодом (Open Source), поэтому новым разработчикам, готовым внести вклад в ее развитие, всегда рады.

PyBrian представляет собой модульную библиотеку, предназначенную для реализации различных алгоритмов машинного обучения на языке Python. Основным преимуществом ее применения является предоставление исследователю гибких, простых в использовании, но в то же время мощных инструментов для реализации задач из области машинного обучения, тестирования и сравнения эффективности различных алгоритмов.

Название PyBrain является аббревиатурой Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library, что в буквальном переводе с английского языка означает «обучение подкреплению на основе Python, искусственный интеллект и библиотека нейронных сетей». Как сказано на одном из сайтов: «PyBrain — swiss army knife for neural networking» (PyBrain — это швейцарский армейский нож в области нейросетевых вычислений).

Библиотека построена по модульному принципу, что позволяет использовать ее как студентам для обучения основам создания нейронных сетей, так и разработчикам, нуждающимся в реализации более сложных алгоритмов.

Сама библиотека является продуктом с открытым исходным кодом и бесплатна для использования, в ней реализованы различные алгоритмы работы с нейронными сетями:

- алгоритмы обучения с учителем (supervised learning);
- алгоритмы обучения без учителя (black-box optimization/evolutionary methods);
- алгоритмы обучения с подкреплением (reinforcement learning);
- алгоритмы оптимизации методом «черного ящика» (black-box optimization).

Модули библиотеки PyBrain оперируют сетями разных структур. В этих сетях могут быть использованы практически все поддерживаемые библиотекой сложные алгоритмы. В качестве примера можно привести следующие виды нейронных сетей:

- сети прямого распространения, включая deep belief networks и restricted boltzmann machines (RBM);
- рекуррентные нейронные сети (recurrent networks, RNN), включая архитектуру long short-term memory (LSTM);
- многомерные рекуррентные сети (multi-dimensional recurrent networks, MDRNN);
- сети Кохонена, или самоорганизующиеся карты (self-organizing maps);
- нейронная сеть Коско, или двунаправленные сети (bidirectional networks);
- создание топологий собственной структуры.

Кроме того, в этой библиотеке присутствуют дополнительные программные инструменты, позволяющие реализовать сопутствующие задачи:

- построение и визуализацию графиков;
- поддержку netCDF (Network Common Data Form) — машинно-независимый двоичный формат файлов, являющийся стандартом для обмена научными данными;
- запись и чтение файлов форматов XML, CVS.

Библиотека PyBrain имеет ряд неоспоримых достоинств, в частности:

- это бесплатная библиотека с открытым исходным кодом для построения и обучения нейронных сетей (полезна как для новичка, так и профессионала);
- использует Python, что упрощает написание кода по сравнению с такими языками, как Java или C++;
- работает с другими библиотеками Python для визуализации данных;
- поддерживает популярные сети как с прямой связью, так и рекуррентные;
- работает с файлами формата CSV для загрузки наборов данных, что позволяет формировать наборы данных, например в Excel;
- имеет встроенных тренеров (учителей) для обучения и тестирования созданных нейронных сетей.

Общая структура библиотеки PyBrain и процедуры ее использования представлены на рис. 6.1.

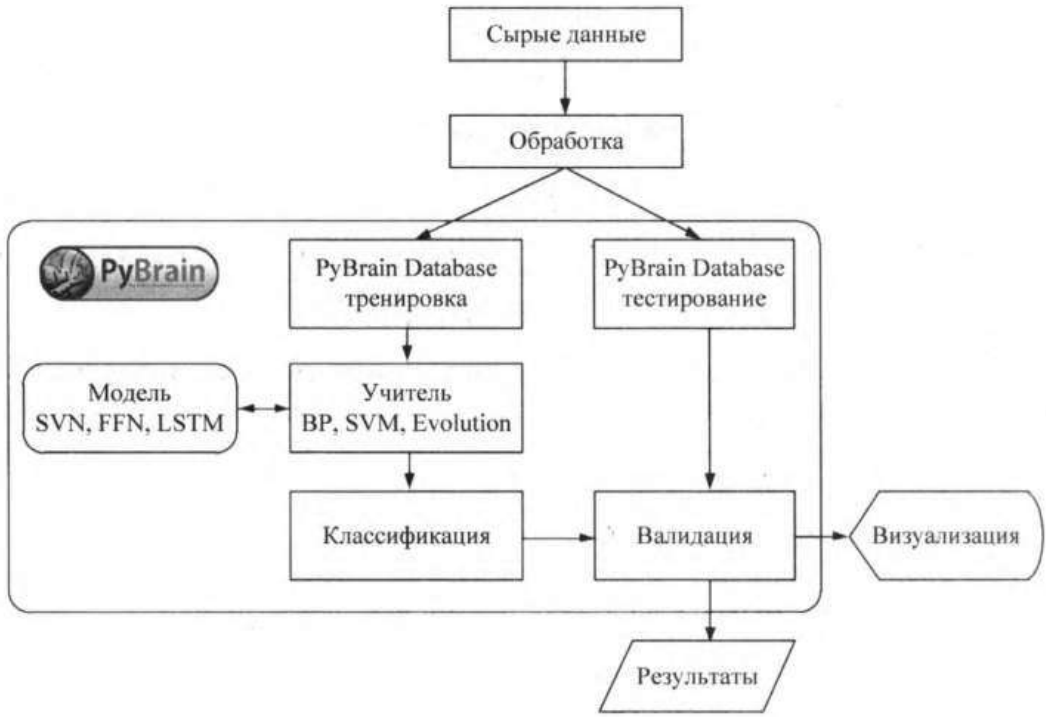


Рис. 6.1. Структура библиотеки PyBrain

Для работы с этой библиотекой берут сырые данные, которые подвергают предварительной обработке. Подготавливают два вида набора данных: обучающую выборку (для тренировки и обучения сети) и тестовую выборку (для тестирования и подведения итогов обучения). С помощью встроенных инструментов формируют структуру нейронной сети (слои и связи между элементами слоев). Далее созданные наборы данных передают тренеру (учителю). Тренер обучает сеть (учит, как получить нужный результат с минимальными ошибками). Это делается на обучающей выборке. Далее тренер проверяет качество обучения на тестовом наборе данных. Здесь проверяется, насколько успешно прошло обучение и можно ли использовать обученную сеть для решения практических задач.

6.2.2. Термины и определения в библиотеке PyBrain

Есть важные термины, которые следует знать при работе с PyBrain для машинного обучения. Ознакомимся с этими терминами и определениями.

- *Общая ошибка* (total error) относится к ошибке, отображаемой после обучения сети. Если ошибка продолжает изменяться на каждой итерации, это означает, что сети все еще нужно время для обучения. Обучаться нужно до тех пор, пока сеть не станет показывать постоянную ошибку между итерациями обучения. Если сеть начинает показывать постоянное значение ошибки, то это означает, что она обучилась и никаких дополнительных циклов обучения не требуется.
- *Обучаемые данные* (trained data) — это данные, используемые для обучения сети PyBrain (обучающая выборка).
- *Данные тестирования* (testing data) — это данные, используемые для тестирования обученной сети PyBrain (тестовая выборка).
- *Тренер* (trainer, учитель). Когда сеть построена (сконфигурирована), ее нужно обучить. Обучается сеть на основе загружаемой в нее обучающей выборки. После обучения нужно проверить, обучена сеть должным образом или нет. Для проверки итогов обучения в нее загружают тестовую выборку и проверяют, насколько верно обученная сеть выдает итоговый результат. Наиболее важной концепцией обучения PyBrain является использование BackpropTrainer и TrainUntilConvergence:
  - *BackpropTrainer* — это способ тренировки сети, при котором формирование параметров модуля происходит в соответствии с обучающими наборами данных под наблюдением или методом обратного распространения ошибок (ClassificationDataset);
  - *TrainUntilConvergence* — это способ тренировки сети, который используется для обучения модуля на основе обучающего набора данных до его схождения.
- *Слои* (layers) — это набор функций, которые используются в скрытых слоях сети.
- *Соединения* (connections) — это некое подобие промежуточных слоев. Единственное отличие от обычного слоя состоит в том, что соединение перемещает данные от одного узла сети к другому.

- *Модули* (modules) — это собственно сами сети, состоящие из входного и выходного слоев.
- *Контролируемое обучение* (supervised learning), или обучение с учителем — это обучение при известных значениях входов и выходов из сети. В этом случае у нас есть известные входы и известные выходы, и мы можем научить сеть использовать определенный алгоритм для сопоставления входа с выходом. Алгоритм предназначен для обучения сети на обучающей выборке. Цикличный процесс обучения останавливается, когда алгоритм начинает выдавать правильные решения.
- *Обучение без присмотра* (unsupervised learning), или обучение без учителя. В этом случае у нас есть только входные данные, и мы не знаем выходных значений. Роль неконтролируемого обучения заключается в том, чтобы как можно больше тренироваться с предоставленными входными данными.

С помощью этой библиотеки сеть строится из модулей, которые связаны с помощью соединений. Библиотека PyBrain поддерживает нейронные сети прямой связи (feed-forward network) и рекуррентные сети (recurrent network):

- *сеть прямой связи* (feed-forward network) — это нейронная сеть, в которой информация между узлами движется в прямом направлении и никогда не перемещается назад. Сеть feed-forward является первой и самой простой среди сетей, доступных в искусственной нейронной сети. Информация передается из входных узлов, рядом со скрытыми узлами, а затем в выходной узел;
- *рекуррентная сеть* (recurrent network) — аналог сети прямой связи с единственным отличием, состоящим в том, что сеть запоминает данные на каждом этапе. История каждого шага в такой сети должна быть сохранена.

Библиотека PyBrain позволяет работать с различными наборами данных (datasets):

- *набор данных* (dataset) — это те данные, которые будут предоставлены для тестирования, проверки и обучения в сетях. Тип используемого набора данных зависит от задач, которые мы собираемся выполнить с помощью машинного обучения. Наиболее часто используемые наборы данных, которые поддерживает PyBrain, — это контролируемый набор данных (SupervisedDataSet) и классификационный набор данных (ClassificationDataSet):
  - *контролируемый набор данных* (SupervisedDataSet) — состоит из полей ввода и цели. Это самая простая форма набора данных, в основном используемая для контролируемых учебных задач;
  - *классификационный набор данных* (ClassificationDataSet) — это набор данных, который в основном используется для решения задач классификации. Он требует ввода целевого поля, а также дополнительного поля, называемого «класс», которое представляет собой автоматическое резервное копирование данных целей. Например, выходные данные будут равны либо 1, либо 0, либо выходные данные будут сгруппированы вместе со значениями на основе заданного входного значения (они попадут в один конкретный класс).

В библиотеке PyBrain есть очень важный пакет — pybrain.tools.shortcuts.buildNetwork. С его помощью достаточно просто сформировать структуру сети. Результаты обу-

чения и тестирования сети не могут быть визуализированы с использованием PyBrain. Но PyBrain для визуализации данных может работать с другими библиотеками, такими как mathplotlib, pyplot.

6.2.3. Установка (подключение) библиотеки PyBrain

Для начала работы с библиотекой нужно установить стандартным способом ряд дополнительных библиотек. Если ранее не устанавливались библиотеки для работы с математическими функциями и визуализации данных, нужно установить следующие библиотеки:

```
pip install numpy
pip install scipy
pip install Matplotlib
```

При работе с библиотекой PyBrain нужно иметь в виду, что библиотека с именем `pybrain` оптимизирована для работы с Python 2. Если вы используете Python 3, то нужно устанавливать адаптированную библиотеку с именем `pybrain3`. Установить эту библиотеку можно следующей командой:

```
pip install pybrain3
```

В инструментальной среде PyCharm это можно сделать через главное меню: **File | Settings | Project Interpreter**. В открывшемся окне нажмите значок + (рис. 6.2) и в списке доступных библиотек выберите вариант **pybrain3** (рис. 6.3).

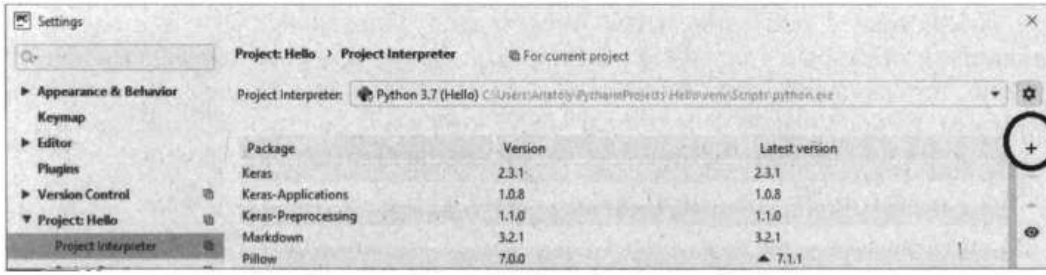


Рис. 6.2. PyCharm: окно **Settings**

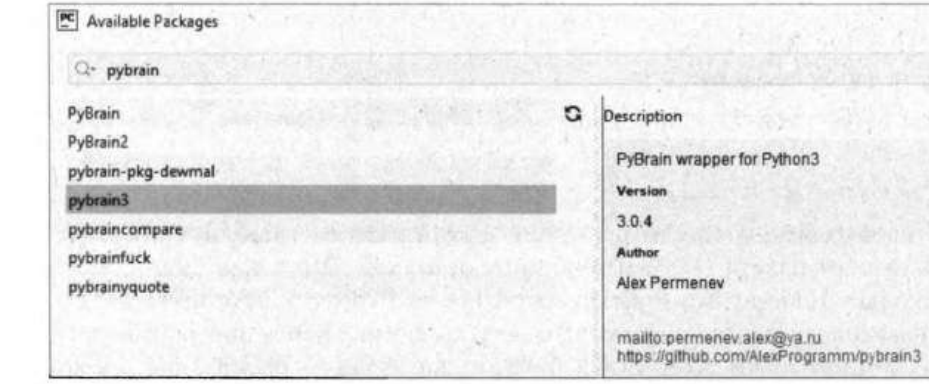


Рис. 6.3. PyCharm: выбор библиотеки **pybrain3** в окне поиска устанавливаемых модулей

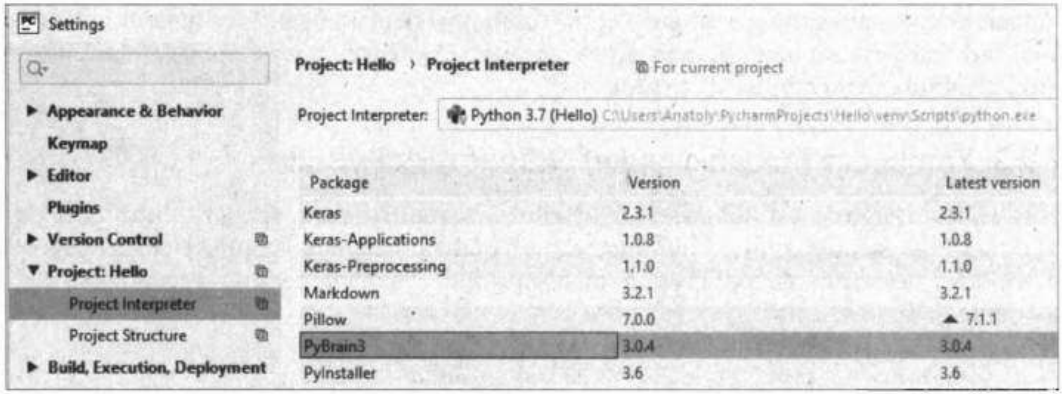


Рис. 6.4. PyCharm: библиотека **PyBrain3** в списке установленных модулей

Если все было сделано правильно, то библиотека появится в списке установленных модулей (рис. 6.4).

Последняя версия библиотека PyBrain — 3.0.4, и она не обновлялась с 2015 года. На тот момент времени она стабильно работала совместно с библиотеками следующих версий:

- numpy — версия 1.18.4;
- scipy — версия 1.4.1;
- matplotlib — версия 3.2.1.

Эти библиотеки в последние годы многократно обновлялись — в частности, из библиотеки scipy был удален модуль `random`, поэтому приложения, основанные на PyBrain, потеряли работоспособность. Однако эта проблема устраняется достаточно просто. Если с PyBrain использовать библиотеку scipy с версией выше, чем 1.4.1, то в модулях библиотеки PyBrain необходимо сделать следующие замены:

1. Модуль **kohonen.py**: `from numpy import random  # from scipy import random;`
2. Модуль **gausinlayer.py**: `from numpy import random  # from scipy import random;`
3. Модуль **statedependenlayer.py**:
   - `from scipy import asarray, zeros, dot`
   - `from numpy import random`
4. Модуль **gaussprocess.py**:
   - `from scipy import r_, exp, zeros, eye, array, asarray, ravel, diag, sqrt, sin, cos, sort, mgrid, dot, floor`
   - `from numpy import random`

То есть необходимо отключить импорт пакета `random` из библиотеки scipy и подключить импорт пакета `random` из библиотеки numpy. Это делается достаточно просто, поскольку библиотека PyBrain написана на Python, а программный код с импортом пакетов находится в самом начале упомянутых ранее программных модулей. На момент подготовки этой книги библиотека PyBrain версии 3.0.4 с рекомендованными здесь изменениями стабильно работает со следующими библиотеками:

- numpy — версия 1.23.5;
- scipy — версия 1.9.3;
- matplotlib — версия 3.6.2.

При этом использовался Python версии 3.8.

6.2.4. Основы работы с библиотекой PyBrain

В PyBrain нейронные сети состоят из модулей, которые связаны между собой. Такую сеть можно представить как ориентированный граф, где узлы — это модули, а ребра — это соединения. Для того чтобы задать схему будущей сети, нужно к нашей программе подключить модуль, отвечающий за создание ее структуры. Это можно сделать следующей командой:

```
from pybrain3.tools.shortcuts import buildNetwork
```

Давайте в качестве примера создадим нейронную сеть с двумя входами, тремя скрытыми слоями и одним выходом. Для этого достаточно всего одной строчки программного кода:

```
net = buildNetwork(2, 3, 1)
```

В результате в объекте `net` будет находиться нейронная сеть с заданными параметрами и случайными значениями весов. В PyBrain слои созданной сети являются объектами `Module`, и эти модули связаны между собой объектами `FullConnection`.

Итак, в объекте `net` сконфигурирована наша сеть, но она пока находится в пассивном состоянии (как бы спит). Для того чтобы заставить сеть работать, ее надо разбудить (активировать). Сделать это нужно методом `activate()`:

```
y = net.activate([2, 1])
print('Y=', y)
```

Количество элементов, передаваемых в сеть, должно быть равно количеству входов. Метод возвращает ответ в виде единственного числа (если текущая сеть имеет один выход) и массива (в случае большего количества выходов). В нашем случае сеть вернет ответ в виде единственного числа, например:

```
Y= [-0.78629107]
```

Поскольку сеть еще не прошла обучение, то ответ сформирован на основании параметров сети, заданных по умолчанию. Можно вывести и посмотреть структуру созданной сети следующей командой:

```
print(net)
```

Подведем итог сказанному. Для того чтобы создать нашу нейронную сеть с помощью библиотеки PyBrain, нам понадобилось всего несколько строчек программного кода на Python:

```
from pybrain3.tools.shortcuts import buildNetwork
net = buildNetwork(2, 3, 1)
y = net.activate([2, 1])
print('Y=', y)
```

Вот и все, нейронная сеть сконфигурирована и запущена в работу. Правда, сеть эта еще ничему не обучена.

В структуре сети каждый ее элемент имеет имя. Оно может быть дано автоматически либо по иным критериям при создании сети. К примеру, в созданной нами сети `net` имена даны автоматически. При необходимости можно получить доступ к модулям слоев и подключения по отдельности, ссылаясь на их имена следующим образом:

```
a = net['bias']
b = net['in']
c = net['hidden0']
d = net['out']
print(a)
print(b)
print(c)
print(d)
```

В листинге 6.1 приведен полный текст программы, которая создает нейронную сеть.

Листинг 6.1

```
# Модуль PyBr_Net1
from pybrain3.tools.shortcuts import buildNetwork
net = buildNetwork(2, 3, 1)
y = net.activate([2, 1])
print('Y=', y)
a = net['bias']
b = net['in']
c = net['hidden0']
d = net['out']
print(a)
print(b)
print(c)
print(d)
```

После работы этого программного кода получим следующий результат (рис. 6.5).

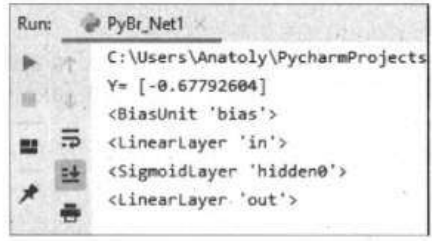


Рис. 6.5. Доступ к отдельным модулям нейронной сети в библиотеке pybrain3

Конечно, в большинстве случаев созданная нейронная сеть должна иметь другие характеристики, нежели заданные по умолчанию. Для их определения существуют разнообразные возможности. К примеру, по умолчанию скрытый слой создается с использованием сигмовидной функции активации, а для задания другого ее типа можно использовать следующие константы:

- BiasUnit;
- GaussianLayer;
- LinearLayer;
- LSTMLayer;
- MDLSTMLayer;
- SigmoidLayer;
- SoftmaxLayer;
- StateDependentLayer;
- TanhLayer.

При задании собственной конфигурации сети можно указать тип скрытого слоя, тип выходного слоя, наличие смещения. Это можно сделать следующими командами:

```
from pybrain3.tools.shortcuts import buildNetwork
from pybrain3.structure import SoftmaxLayer
from pybrain3.structure import TanhLayer
net = buildNetwork(2, 3, 1, hiddenclass=TanhLayer,
    outclass=SoftmaxLayer, bias=True)
net.activate((2, 3))
```

Более подробно с типами слоев можно ознакомиться в оригинальной документации на библиотеку pybrain3.

6.2.5. Работа с наборами данных в библиотеке PyBrain

Созданная сеть должна обрабатывать данные. Типичными наборами данных являются набор входных данных и набор выходных данных. Для работы с ними PyBrain использует модуль pybrain.dataset. Библиотека поддерживает разные классы наборов данных, такие как SupervisedDataset (контролируемый набор данных), SequentialDataset (последовательный набор данных), ClassificationDataSet (классификационный набор данных).

Чтобы создать наш набор данных, мы собираемся использовать SupervisedDataset. Набор данных, с которыми будет работать сеть, зависит от задачи машинного обучения, которую пытается реализовать пользователь. Класс SupervisedDataset — самый простой набор данных для типичного обучения с учителем. Он поддерживает массивы входных и выходных данных. Их размеры задаются при создании экземпляра класса. Запись вида:

```
from pybrain3.datasets import SupervisedDataSet
ds = SupervisedDataSet(2, 1)
```

означает, что создается структура данных для хранения двумерных входных данных и одномерных выходных.

Рассмотрим использование наборов данных при обучении нейронной сети на простом примере. Таким классическим примером является обучение нейронной сети

с применением функции XOR (исключающее ИЛИ). Это функция, которая принимает значение «истина», когда один и только один из ее аргументов имеет значение «истина». При этом истиной является значение 1, ложью — значение 0.

Набор данных SupervisedDataset требует ввода входных параметров и цели (правильного значения для этих входных параметров). Сформируем таблицу истинности для функции XOR (табл. 6.1).

***Таблица 6.1.** Значения функции XOR*

Входные данные для обучения		
параметры функции		цель (правильное значение функции)
A	***B***	***A* XOR *B***
0	0	0
0	1	1
1	0	1
1	1	0

Из этой таблицы видно, что входными данными для обучения нашей сети являются два массива: двумерный массив параметров A и B функции и одномерный массив цели (правильного значения функции). Таким образом, размер входных данных в Dataset для нашей сети будет 2×1. Создать пустой набор данных с такой структурой можно следующими командами:

```
from pybrain3.datasets import SupervisedDataSet
ds = SupervisedDataSet(2, 1)
```

Посмотрим содержимое нашего набора данных. Напишем для этого следующий код (листинг 6.2).

Листинг 6.2

```
# Модуль PyBr_Dset1
from pybrain3.datasets import SupervisedDataSet
ds = SupervisedDataSet(2, 1)
print(ds)
```

После его выполнения получим следующий результат (рис. 6.6).

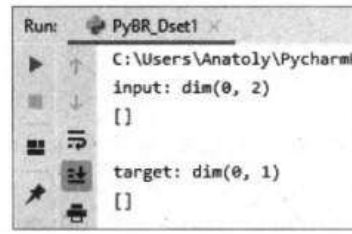


Рис. 6.6. Пустое содержимое объекта Dataset в библиотеке pybrain3

Здесь видно, что в объекте Dataset для хранения обучающей выборки сформированы два пустых массива: входные параметры (input) и правильные значения функции, или цель обучения (target).

Следующим шагом нам нужно эти пустые массивы заполнить обучающими данными из табл. 6.1. Это можно сделать построчно следующими командами:

```
ds.addSample((0, 0), (0,))
ds.addSample((0, 1), (1,))
ds.addSample((1, 0), (1,))
ds.addSample((1, 1), (0,))
```

Или следующим способом — сформировать массив xorModel и в цикле добавлять элементы массива в Dataset:

```
xorModel = [
    [(0, 0), (0,)],
    [(0, 1), (1,)],
    [(1, 0), (1,)],
    [(1, 1), (0,)],
]

for input, target in xorModel:
    ds.addSample(input, target)
```

Независимо от способа заполнения данными объекта Dataset, вывести его содержимое на печать можно командой:

```
print(ds)
```

В листинге 6.3 приведен полный код программы формирования обучающего набора данных.

Листинг 6.3

```
# Модуль PyBr_Dset2
from pybrain3.datasets import SupervisedDataSet
ds = SupervisedDataSet(2, 1)

xorModel = [
    [(0, 0), (0,)],
    [(0, 1), (1,)],
    [(1, 0), (1,)],
    [(1, 1), (0,)],
]

for input, target in xorModel:
    ds.addSample(input, target)

print(ds)
```

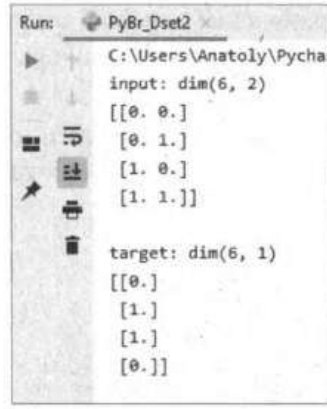


Рис. 6.7. Обучающая выборка данных в объекте Dataset

При выполнении этой программы мы получим результат, представленный на рис. 6.7.

Можно с использованием следующих команд вывести отдельно входные параметры и целевые значения:

```
print(ds['input'])
print(ds['target'])
```

В итоге мы сформировали обучающую выборку и можем перейти к тренировке (обучению) нашей сети. По своей сути, объект `ds` является как бы учебником, который наша сеть должна изучить.

Все готово, чтобы отправить нашу сеть на учебу. Библиотека PyBrain ориентирована на концепцию тренеров (trainers) для обучения сетей с учителем. Тренер получает экземпляр сети (ученика) и экземпляр набора образцов для изучения (учебник) и затем обучает сеть по полученному набору (обучает ученика по учебнику). А кто и каким методом будет проводить обучение? Классический метод обучения — это метод обратного распространения ошибки (BackPropagation). Для обучения этим методом в PyBrain имеется достаточно квалифицированный учитель (тренер). Все его способности собраны в модулях класса `BackpropTrainer`.

Итак, у нас есть ученик (создана сеть `net`), для него имеется учебник (сформирована обучающая выборка `ds`), имеется квалифицированный учитель (тренер `BackpropTrainer`). Теперь эти объекты (ученика с учебником) нужно просто передать на обучение тренеру (т. е. отправить сеть в школу на учебу). Это можно сделать с помощью следующей команды:

```
trainer = BackpropTrainer(net, ds)
```

Здесь мы еще не начали учебу, а просто указали тренеру (`BackpropTrainer`): кто у нас учится (сеть `net`) и чему учится (учебник в виде обучающего набора данных `ds`). Остался один шаг — подать звонок, и урок начнется. Процесс обучения запускается следующей командой:

```
print(trainer.train())
```

В листинге 6.3.1 приведен полный код программы запуска процесса обучения.

Листинг 6.3.1

```
# Listing 6.3.1
from pybrain3.tools.shortcuts import buildNetwork
from pybrain3.datasets import SupervisedDataSet
from pybrain3.supervised import BackpropTrainer

net = buildNetwork(2, 3, 1)
y = net.activate([2, 1])

ds = SupervisedDataSet(2, 1)

xorModel = [
   [(0, 0), (0,)],
   [(0, 1), (1,)],
   [(1, 0), (1,)],
   [(1, 1), (0,)],
]

for input, target in xorModel:
    ds.addSample(input, target)

trainer = BackpropTrainer(net, ds)
print(trainer.train())
```

Но это был всего один урок. Вызов метода `train()` производит одну итерацию (эпоху) обучения и возвращает значение квадратичной ошибки (рис. 6.8).

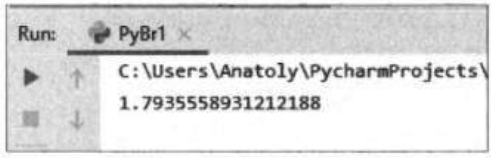


Рис. 6.8. Значение ошибки сети после одной итерации обучения (эпохи)

Однако одного урока обучения для сети недостаточно — нужно организовать серию уроков (эпох) обучения, чтобы свести к минимуму ошибки получения результатов. Организовать множество циклов обучения можно методом обучения сети до сходимости:

```
trainer.trainUntilConvergence()
```

Этот метод возвратит массив ошибок для каждой эпохи. Можно построить график и проследить, как уменьшаются ошибки сети в процессе обучения с помощью следующего программного кода:

```
trainer = BackpropTrainer(net)
trnerr, valerr = trainer.trainUntilConvergence(dataset=ds, maxEpochs=100)
```

```
plt.plot(trnerr, 'b', valerr, 'r')
plt.show()
```

Здесь мы использовали метод `trainUntilConvergence()` для обучающих данных, которые будут сходиться для эпох, равных 100. Он возвращает ошибку обучения и ошибку проверки.

Не забываем при этом, что в процессе обучения используется генератор случайных чисел, и при каждом запуске программы мы будем получать разные результаты.

В листинге 6.4 приведен полный код этой программы (обратите внимание, что здесь подключено несколько дополнительных библиотек).

Листинг 6.4

```
# Модуль PyBr_Trener
import matplotlib.pylab as plt
from pybrain3.tools.shortcuts import buildNetwork
from pybrain3.datasets import SupervisedDataSet
from pybrain3.supervised.trainers import BackpropTrainer

net = buildNetwork(2, 3, 1)
y = net.activate([2, 1])

ds = SupervisedDataSet(2, 1)
ds.addSample((0, 0), (0,))
ds.addSample((0, 1), (1,))
ds.addSample((1, 0), (1,))
ds.addSample((1, 1), (0,))
print(ds)

trainer = BackpropTrainer(net)
trnerr, valerr = trainer.trainUntilConvergence(dataset=ds, maxEpochs=100)
plt.plot(trnerr, 'b', valerr, 'r')
plt.show()
```

Запустите программу несколько раз и понаблюдайте, как будет меняться график значений ошибок. Результаты работы этой программы для случая, когда сеть достаточно хорошо обучена, представлены на рис. 6.9.

Из рисунка видно, что уже после 50-го цикла обучения значения ошибок практически не меняются. Здесь нижняя линия показывает ошибки обучения, а верхняя — ошибки проверки. После обучения можно с помощью программного кода из листинга 6.5 распечатать итоговые результаты — параметры каждого модуля сети после обучения.

Листинг 6.5

```
# Это промежуточный код, он не является рабочим
for mod in net.modules:
    print("Module:", mod.name)
```

```
if mod.paramdim > 0:
    print("--parameters:", mod.params)
for conn in net.connections[mod]:
    print("-connection to", conn.outmod.name)
    if conn.paramdim > 0:
        print("- parameters", conn.params)

if hasattr(net, "recurrentConns"):
    print("Recurrent connections")
    for conn in net.recurrentConns:
        print("-", conn.inmod.name, " to", conn.outmod.name)
        if conn.paramdim > 0:
            print("- parameters", conn.params)
```

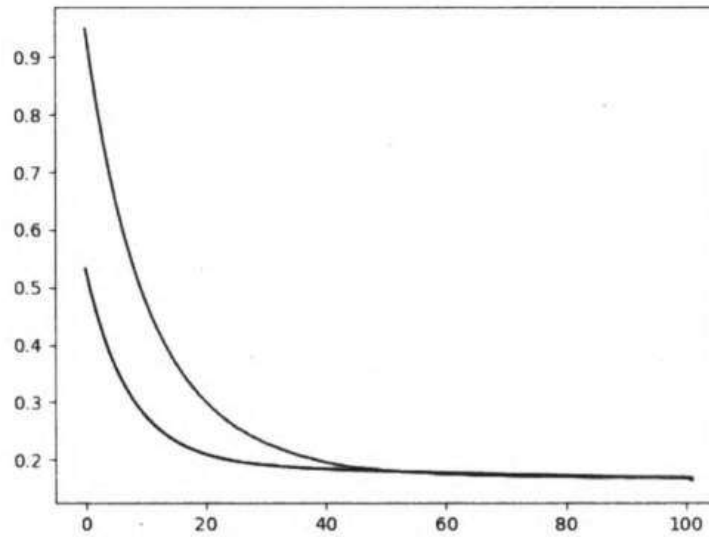


Рис. 6.9. График изменения значений ошибок в процессе обучения сети

Объединим программный код листинга 6.4 с кодом листинга 6.5. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_5_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результаты обучения (итог работы модуля 6_5_1) представлены на рис. 6.10.

Для пользователя распечатка результатов обучения практически не нужна, она ему ни о чем не говорит, и в нашем примере мы их вывели на печать в учебных целях. Однако эти результаты очень важны для обученной сети, и их нужно обязательно сохранить (чтобы в дальнейшем обученная сеть могла решать практические задачи без повторного обучения). Проверим, как будет работать наша сеть после обучения. Добавим в программу следующие строки:

```
y = net.activate([1, 1])
print('Y=', y)
```

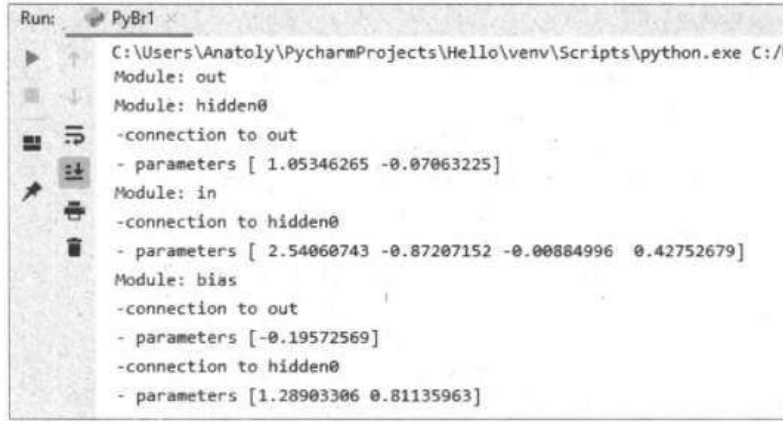


Рис. 6.10. Параметры элементов сети после обучения

Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_5_2) в сопровождающем книгу файловом архиве (см. *приложение*). В результате выполнения объединенного модуля будет получен результат `Y= [-0.00722513]`, что практически соответствует правильному значению функции XOR — 0.

Нейронные сети PyBrain после обучения могут быть сохранены, а потом вызваны для практического использования. Для этого служит встроенный в Python модуль `pickle`. Он подключается следующим образом:

```
import pickle
```

В листинге 6.6 приведен пример программного кода сохранения сети `net` в файл MyNet.txt и загрузки сохраненной сети из этого файла в объект `net2`.

Листинг 6.6

```
# Это промежуточный код, он не является рабочим
fileObject = open('MyNet.txt', 'wb')
pickle.dump(net, fileObject)
fileObject.close()
fileObject = open('MyNet.txt', 'rb'))
net2 = pickle.load(fileObject)
```

Для проверки, насколько корректно будет работать загруженная из файла сеть, в текст программы обучения нашей сети добавим следующий программный код (листинг 6.7).

Листинг 6.7

```
# Это промежуточный код, он не является рабочим
y = net.activate([1, 1])
print('Y1=', y)
```

```
fileObject = open('MyNet.txt', 'wb')
pickle.dump(net, fileObject)
fileObject.close()

fileObject = open('MyNet.txt', 'rb')
net2 = pickle.load(fileObject)

y = net2.activate([1, 1])
print('Y2=', y)
```

Здесь значение `Y1` получено из сети `net` сразу после ее обучения, а значение `Y2` — из сети `net2`, загруженной из копии сети `net`, сохраненной в файле MyNet.txt. Пример работы этого фрагмента программы приведен на рис. 6.11.

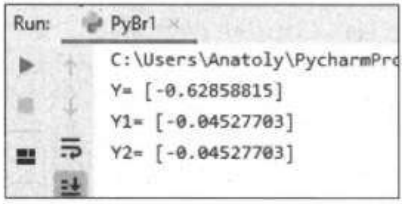


Рис. 6.11. Сопоставление результатов обученной сети `net` с работой ее сохраненной копии `net2`

Здесь можно видеть, что результаты работы сети `net2`, загруженной из файла, полностью идентичны работе сети первоисточника (`net`).

Для сохранения обученной сети можно также использовать модули, встроенные в библиотеку PyBrain. Обученную сеть вы можете сохранить в XML-файле с помощью метода `NetworkWriter.writeToFile()`, а вызвать из файла методом `NetworkReader.readFrom()`. Для выполнения этих действий используется следующий программный код (листинг 6.8).

Листинг 6.8

```
# Это промежуточный код, он не является рабочим
from pybrain3.tools.xml.networkwriter import NetworkWriter
from pybrain3.tools.xml.networkreader import NetworkReader
NetworkWriter.writeToFile(net, 'filename.xml')
net = NetworkReader.readFrom('filename.xml')
```

Здесь в первых двух строках подключаются соответствующие библиотеки. Далее применяются метод `NetworkWriter.writeToFile()` (записать структуру и параметры сети в XML-файл) и метод `NetworkReader.readFrom()` (загрузить сеть из XML-файла).

В листинге 6.9 приведен полный текст программного кода, относящегося к этому разделу.

Листинг 6.9

```
# Модуль PyBr_Trener_Ful
import pickle
import matplotlib.pylab as plt
from pybrain3.tools.shortcuts import buildNetwork
from pybrain3.datasets import SupervisedDataSet
from pybrain3.supervised.trainers import BackpropTrainer

net = buildNetwork(2, 3, 1)
y = net.activate([2, 1])
print('Y=', y)

ds = SupervisedDataSet(2, 1)
ds.addSample((0, 0), (0,))
ds.addSample((0, 1), (1,))
ds.addSample((1, 0), (1,))
ds.addSample((1, 1), (0,))
print(ds)

trainer = BackpropTrainer(net)
# trnerr,valerr = trainer.trainUntilConvergence(maxEpochs=100)
trnerr, valerr = trainer.trainUntilConvergence(dataset=ds, maxEpochs=100)
plt.plot(trnerr, 'b', valerr, 'r')
plt.show()

# Проверка работы сети после обучения
y = net.activate([1, 1])
print('Y1=', y)

# запись сети в файл txt
fileObject = open('MyNet.txt', 'wb')
pickle.dump(net, fileObject)
fileObject.close()

# чтение сети из файла txt
fileObject = open('MyNet.txt', 'rb')
net2 = pickle.load(fileObject)

# Проверка работы загруженной из файла сети
y = net2.activate([1, 1])
print('Y2=', y)
```

Многократно запуская этот программный модуль, можно получить набор весовых коэффициентов для достаточно уверенной работы нейронной сети и сохранить обученную сеть в виде текстового файла.

В этом разделе мы познакомились с методикой создания набора данных и способом передачи этого набора в модуль тренировки нейронной сети. Обучение нейронной

сети применению функции XOR имеет достаточно слабое практическое значение. Приведенный в этом разделе пример был показан исключительно в учебных целях, т. к. он позволяет понять основные принципы обучения сети на простых наборах данных. В следующем разделе мы попробуем реализовать нейронную сеть хотя и на упрощенном примере, но более приближенном к практике.

6.2.6. Пример создания нейронной сети с библиотекой PyBrain

К этому моменту у нас есть все, чтобы спроектировать и реализовать свою нейронную сеть на основе модулей библиотеки PyBrain.

Вернемся к упрощенному примеру прогноза клева при рыбной ловле, который мы рассматривали в *главе 3*, сформулируем и оцифруем условия нашей задачи. Конечной целью будет являться рекомендация: идти на рыбалку (ожидается хороший клев) или остаться дома (клева не будет). Оценим активность рыбы в баллах (табл. 6.2).

***Таблица 6.2.** Оценка уровня активности рыбы в баллах*

Активность рыбы	Оценка в баллах
Очень хороший клев	5
Хороший клев	4
Средний клев	3
Плохой клев	2
Очень плохой клев	1

Теперь определимся с параметрами, которые влияют на активность рыбы. Для упрощения задачи выберем всего четыре параметра: силу ветра (м/с), суточный перепад давления воздуха (мм рт. ст.), облачность или наличие на небе облаков (%), суточный перепад температуры воды (°С). Известно, что активность рыбы повышается при слабом ветре, отсутствии резких перепадов атмосферного давления и температуры воды, наличии на небе облаков (неяркое солнце). Составим таблицу значений этих параметров (табл. 6.3).

***Таблица 6.3.** Значения параметров, влияющих на активность рыбы*

Скорость ветра, м/с	Перепад давления воздуха, мм рт. ст.	Облачность, %	Перепад температуры воды, °С
2	3	80	1
5	5	50	2
10	7	40	3
15	9	20	4
20	11	10	5

Теперь на основе данных из табл. 6.2 и 6.3 сформируем обучающую выборку данных (табл. 6.4).

Таблица 6.4. *Обучающая выборка для прогноза активности рыбы*

Скорость ветра, м/с	Перепад давления воздуха, мм рт. ст.	Облачность, %	Перепад температуры воды, °С	Активность рыбы, баллы
x_1	x_2	x_3	x_4	Y
2	3	80	1	5
5	5	50	2	4
10	7	40	3	3
15	9	20	4	2
20	11	10	5	1

В результате для нашей нейронной сети мы получили: четыре входных параметра (x_1, x_2, x_3, x_4) и один выходной параметр (Y). В завершение, с использованием модулей библиотеки PyBrain, напишем программный код, в котором реализована нейронная сеть, предсказывающая успешность рыбной ловли при тех или иных погодных условиях (листинг 6.10).

Листинг 6.10

```
# Модуль Net_Fishing
import pickle
import matplotlib.pylab as plt
from pybrain3.tools.shortcuts import buildNetwork
from pybrain3.datasets import SupervisedDataSet
from pybrain3.supervised.trainers import BackpropTrainer

# Формирование обучающего набора данных
ds = SupervisedDataSet(4, 1)
ds.addSample([2, 3, 80, 1], [5])
ds.addSample([5, 5, 50, 2], [4])
ds.addSample([10, 7, 40, 3], [3])
ds.addSample([15, 9, 20, 4], [2])
ds.addSample([20, 11, 10, 5], [1])

# Формирование структуры нейронной сети
net = buildNetwork(4, 3, 1, bias=True)

# Тренировка (обучение) нейронной сети с визуализацией этапов тренировки
trainer = BackpropTrainer(net, dataset=ds, momentum=0.1, learningrate=0.01,
verbose=True, weightdecay=0.01)
trnerr, valerr = trainer.trainUntilConvergence()
```

```
plt.plot(trnerr, 'b', valerr, 'r')
plt.show()

# Запись обученной сети в файл MyNet_Fish.txt
fileObject = open('MyNet_Fish.txt', 'wb')
pickle.dump(net, fileObject)
fileObject.close()
```

Вот, собственно, и все. Для решения нашей задачи потребовался всего десяток строк программного кода (пояснения сделаны в тексте программы). После запуска этой программы на выполнение мы получим график зависимости ошибок сети от циклов обучения (рис. 6.12), на котором одна линия показывает ошибки обучения, а другая — ошибки проверки.

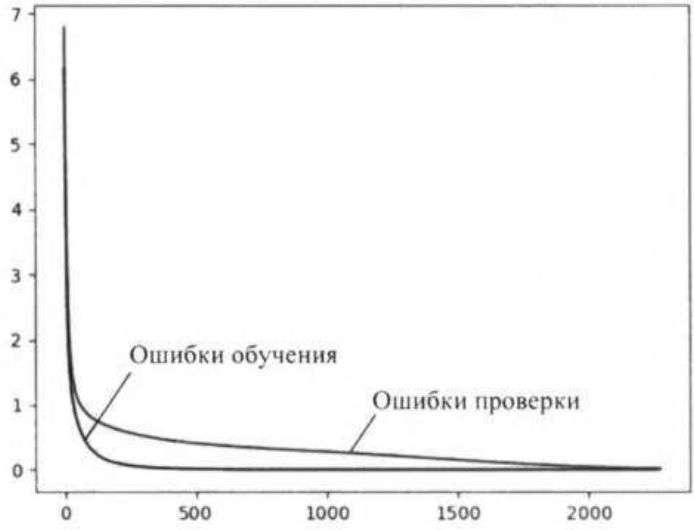


Рис. 6.12. График зависимости ошибок сети от циклов обучения

Из приведенного графика видно, что всего было выполнено порядка 2500 циклов (эпох) обучения. Однако уже после 500 циклов ошибки обучения стабилизировались, а после 2000 циклов ошибки обучения и проверки практически сравнялись (произошла сходимость сети). Теперь эту сеть можно использовать для получения совета, стóит ли идти на рыбалку.

Следует в очередной раз напомнить, что процессы обучения нейронных сетей носят вероятностный характер. В связи с этим при каждом запуске программы листинга 6.10 вы будете получать и разное количество циклов обучения, и разные результаты. Процесс обучения сети нужно остановить тогда, когда программа выдаст график зависимости ошибок сети от циклов обучения, аналогичный тому графику, который приведен на рис. 6.12.

Напишем следующий программный код для получения прогноза хорошего клева при разных погодных условиях (листинг 6.11).

Листинг 6.11

```
# Модуль Fishing_Test
import pickle

fileObject = open('MyNet_Fish.txt', 'rb')
net2 = pickle.load(fileObject)
fileObject.close()

# Хорошие погодные условия
y = net2.activate([2, 3, 80, 1])
print('Y1=', y)

# Средние погодные условия
y = net2.activate([10, 7, 40, 3])
print('Y2=', y)

# Плохие погодные условия
y = net2.activate([20, 11, 10, 5])
print('Y3=', y)
```

После запуска этой программы получим следующие результаты:

```
Y1= [4.94150737]
Y2= [2.95785519]
Y3= [1.0565008]
```

Поясним, что мы задали на вход и что получили на выходе:

1. Заданы достаточно благоприятные погодные условия:
   - скорость ветра — 2 м/с;
   - суточный перепад атмосферного давления — не более 3 мм рт. ст.;
   - достаточно высокая облачность — 80%;
   - суточный перепад температур воды — в пределах 1°С.

   Какой мы получили совет? Активность рыбы при таких погодных условиях достаточно высокая (`Y=4.94`) — практически 5 баллов. Значит, можно смело идти на рыбалку.
2. Погодные условия ухудшились:
   - скорость ветра — 10 м/с;
   - суточный перепад атмосферного давления — порядка 7 мм рт. ст.;
   - небо прояснилось, облачность — 40%;
   - суточный перепад температур воды — порядка 3°С.

   Какой мы получили совет? Активность рыбы при таких погодных условиях не очень высокая (`Y=2.95`) — практически 3 балла. Значит, клев может быть, а мо-

жет и не быть. Нужно подумать, стоит ли терять драгоценное время, и остаться дома, а может, просто пойти и подышать свежим воздухом.

3. Погода резко изменилась:
   - ветер усилился (скорость ветра — 20 м/с);
   - атмосферное давление скачет (суточный перепад давлений — порядка 11 мм рт. ст.);
   - выглянуло солнце, облачность слабая — 10%;
   - суточный перепад температур воды — порядка 5°С.

   Какой мы получили совет? Активность рыбы при таких погодных условиях очень низкая (`Y=1.05`) — практически 1 балл. Значит, клева точно не будет. На рыбалку идти бесполезно — сидим дома.

В действительности, на активность рыбы влияет и множество других факторов: направление ветра, время года, время суток, фаза луны, тип водоема, фаза прилива/отлива и т. д.

* * *

В *разд. 6.2* в учебных целях мы рассмотрели упрощенные примеры, на которых была показана последовательность создания нейронной сети и ее обучения с учителем (от постановки задачи до практического использования). Как видно из примеров этого раздела, программирование нейронных сетей при использовании специализированной библиотеки PyBrain сильно упрощается — мы получаем хороший результат при минимуме программного кода. Итак, нас можно поздравить — мы выполнили полный цикл работ по созданию собственной нейронной сети: постановка задачи, формирование обучающего набора данных, формирование модели нейронной сети, обучение модели, практическое использование обученной модели.

Теперь перейдем к изучению других библиотек, позволяющих реализовывать нейронные сети.

6.3. Библиотека scikit-learn для создания и обучения нейронных сетей

scikit-learn — это еще одна известная библиотека машинного обучения, которая работает совместно с Python и обеспечивает широкий спектр алгоритмов кластеризации, регрессии и классификации. Эта библиотека поддерживает алгоритмы обучения как с учителем, так и без учителя.

scikit-learn — библиотека с открытым исходным кодом. Иными словами, ее можно свободно использовать и распространять, и любой человек может легко получить исходный код, чтобы увидеть, что происходит «за кулисами». Проект scikit-learn постоянно развивается и совершенствуется и имеет очень активное сообщество пользователей. Он поддерживает ряд современных алгоритмов машинного обучения и содержит полную документацию по каждому алгоритму. scikit-learn — очень популярный инструмент и достаточно известная библиотека машинного обучения,

которая используется совместно с Python. Она широко применяется в промышленности и науке, а в Интернете имеется богатый выбор обучающих материалов и примеров программного кода. Библиотека scikit-learn прекрасно работает с рядом других научных инструментов Python. В Интернете можно ознакомиться с руководством пользователя по scikit-learn и документацией по API для получения дополнительной информации о многочисленных возможностях этой библиотеки.

Применение scikit-learn требует наличия еще двух пакетов Python: NumPy и SciPy. Для построения графиков и интерактивной работы необходимо также установить библиотеку matplotlib.

В этом издании книги задействованы следующие версии библиотек:

- scikit-learn — версия 1.1.3;
- numpy — версия 1.23.4;
- scipy — версия 1.9.3;
- matplotlib — версия 3.6.7.

NumPy — это один из основных пакетов для научных вычислений в Python. Он содержит функциональные возможности для работы с многомерными массивами, высокоуровневыми математическими функциями (операции линейной алгебры, преобразование Фурье, генератор псевдослучайных чисел). В scikit-learn массив NumPy — это основная структура данных, т. е. scikit-learn принимает данные в виде массивов NumPy. Любые данные, которые вы используете, должны быть преобразованы в массив NumPy. Базовый функционал NumPy — это класс `np.array`, многомерный (*n*-мерный) массив. Все элементы массива должны быть одного и того же типа. Массив NumPy выглядит следующим образом:

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print('Массив X')
print(x)
```

Если запустить этот программный код на выполнение, то на выходе получим следующий результат:

```
Массив X
[[1 2 3]
 [4 5 6]]
```

Объекты класса `np.array` мы будем называть *массивами NumPy* или просто *массивами*.

SciPy — это набор функций для научных вычислений в Python. Помимо всего прочего, он предлагает продвинутые процедуры линейной алгебры, математическую оптимизацию функций, обработку сигналов, специальные математические функции и статистические функции. Библиотека scikit-learn использует набор функций SciPy для реализации своих алгоритмов.

В машинном обучении наиболее важной частью SciPy является пакет `scipy.sparse` — с его помощью мы получаем *разреженные матрицы* (sparse matrices). Они пред-

ставляют собой еще один формат данных, который используется в scikit-learn. В листинге 6.12 приведен программный код для формирования различных типов матриц или массивов.

Листинг 6.12

```
import numpy as np
from scipy import sparse

# Создаем 2D-массив NumPy с единицами по главной диагонали
# и нулями в остальных ячейках
eye = np.eye(4)
print("массив NumPy:\n{}".format(eye))

# Преобразовываем массив NumPy в разреженную матрицу SciPy в формате CSR
sparse_matrix = sparse.csr_matrix(eye)
print("\nразреженная матрица SciPy в формате CSR:\n{}".format(sparse_matrix))

# Разреженная матрица формата COO
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("формат COO:\n{}".format(eye_coo))
```

Если выполнить эту программу, то на выходе из нее получим следующие результаты:

```
массив NumPy:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

разреженная матрица SciPy в формате CSR:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
формат COO:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
```

Основная библиотека для построения научных графиков, которая используется в Python, — это matplotlib. Она включает функции для создания высококачественной визуализации типа линейных диаграмм, гистограмм, диаграмм разброса и т. д.

Визуализация данных и различных аспектов анализа может дать важную информацию. В листинге 6.13 приведен пример того, как всего в несколько строк можно построить и вывести график зависимости двух величин.

Листинг 6.13

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 100)
y = np.sin(x)
plt.plot(x, y, marker=".")
plt.show()
```

Здесь мы сгенерировали массив значений по оси x (100 точек: от -10 до $+10$) и сформировали массив значений по оси y. В следующих двух строчках построили график и вывели его на экран. Результат работы этой программы представлен на рис. 6.13.

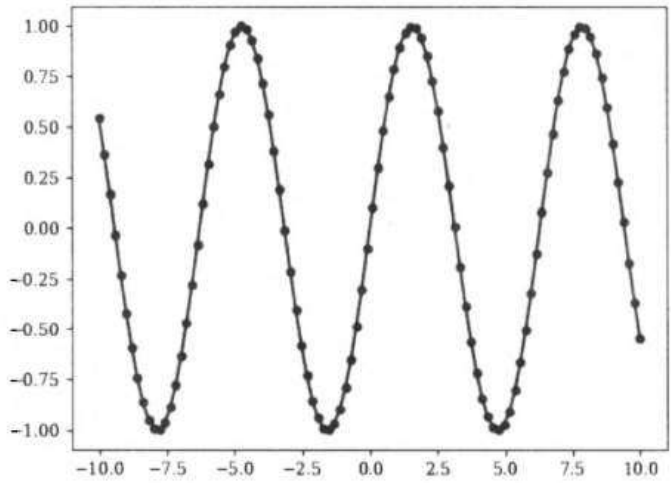


Рис. 6.13. График зависимости двух величин, построенный с использованием библиотеки matplotlib

Еще одна очень важная библиотека Python для обработки и анализа данных — это pandas. В этом издании книги использовалась библиотека pandas версии 1.5.1. Она построена на основе структуры данных, называемой DataFrame и смоделированной по принципу дата-фреймов среды статистического программирования R. Проще говоря, DataFrame библиотеки pandas представляет собой таблицу, похожую на электронную таблицу Excel. Библиотека pandas предлагает большой спектр методов по работе с этой таблицей — в частности, она позволяет выполнять SQL-подобные запросы и присоединение таблиц. В отличие от библиотеки NumPy, которая требует, чтобы все записи в массиве были одного и того же типа, в pandas каждый столбец может иметь отдельный тип (например, целые числа, даты, числа с плавающей точкой и строки). Еще одним преимуществом библиотеки pandas

является ее способность работать с различными форматами файлов и баз данных — например, с файлами SQL, Excel и CSV. Далее приводится небольшой пример создания DataFrame с использованием этой библиотеки:

```
import pandas as pd
data = {'Имя': ["Дима", "Анна", "Петр", "Вика"],
        'Город': ["Москва", "Курск", "Псков", "Воронеж"],
        'Возраст': [24, 13, 53, 33]}
data_pandas = pd.DataFrame(data)
print(data_pandas)
```

При работе такого программного кода будет получен следующий массив данных:

```
    Имя    Город  Возраст
0  Дима   Москва       24
1  Анна    Курск       13
2  Петр    Псков       53
3  Вика  Воронеж       33
```

Итак, мы получили общие представления о библиотеке scikit-learn. Теперь можно приступить к ее практическому использованию.

6.3.1. Наборы данных в библиотеке scikit-learn

Мы уже рассматривали задачу классификации объектов с использованием персептрона на примере цветов ириса (см. *разд. 4.10*). Теперь попробуем решить ту же задачу, но с применением библиотеки scikit-learn. Здесь мы будем ориентироваться на подход к обучению, несколько отличный от обучения персептрона, и построим классификационную модель на основе метода k ближайших соседей, обучим эту модель, а затем используем ее для практических целей.

Итак, наша задача — создать и обучить классификационную модель. Для работы программных модулей, приведенных в этом разделе, использовалась библиотека scikit-learn версии 1.1.3.

У нас уже есть характеристики ирисов, которые ранее позволили опытному эксперту отнести их к разным видам: setosa, versicolor и virginica. Относительно этих ирисов ботаник-любитель уверенно может сказать, к какому виду принадлежит каждый ирис. Наша цель заключается в построении классификационной модели, которая сможет обучиться на основе характеристик ирисов, уже классифицированных по видам, и затем предскажет вид любого нового ириса.

Поскольку у нас есть информация, определяющая правильные виды ирисов, решаемая задача является задачей обучения с учителем. В ней нам нужно спрогнозировать один из сортов ириса. Это пример *задачи классификации* (classification). Возможные ответы (различные сорта ириса) называются *классами* (classes). Каждый ирис в наборе данных принадлежит к одному из трех классов. Таким образом, решаемая задача является задачей трехклассовой классификации. Ответом для отдельной точки данных (ириса) является тот или иной вид этого цветка. Вид, к которому принадлежит цветок (конкретная точка данных), называется *меткой* (label).

Для этого примера мы воспользуемся классическим набором данных, применяемым в машинном обучении и статистике. Его достаточно часто выбирают в различных книжных изданиях и в интернет-ресурсах для демонстрации принципов машинного обучения. Выберем этот готовый набор данных и мы, тем более что он уже включен в модуль `datasets` библиотеки scikit-learn. Загрузить этот набор обучающих данных можно, вызвав функцию `load_iris()`:

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

Созданный нами набор данных `iris_dataset` является *объектом Bunch* (что в переводе означает пучок, связка, пачка, куча). Этот набор данных содержит ключи (наименования элементов массива данных) и значения (массив данных). Получить наименования ключей и другие сведения о структуре этого набора данных можно с помощью команд из листинга 6.14.

Листинг 6.14

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
print("Ключи iris_dataset: \n{}".format(iris_dataset.keys()))
print("Тип массива data: {}".format(type(iris_dataset['data'])))
print("Форма массива data: {}".format(iris_dataset['data'].shape))
print("Цель: {}".format(iris_dataset['target']))
print("Названия ответов: {}".format(iris_dataset['target_names']))
print(iris_dataset['DESCR'][:193] + "\n...")
print("Названия признаков: \n{}".format(iris_dataset['feature_names']))
print("Расположение файла: \n{}".format(iris_dataset['filename']))
```

При выполнении этой программы получим следующий вывод:

```
Ключи iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
Тип массива data: <class 'numpy.ndarray'>
Форма массива data: (150, 4)
Цель: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Названия ответов: ['setosa' 'versicolor' 'virginica']
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, pre
...
```

```
Названия признаков:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Расположение файла:
iris.csv
```

Ключи в этом массиве данных имеют следующий смысл:

- `data` — массив NumPy, который содержит количественные измерения длины чашелистиков, ширины чашелистиков, длины лепестков и ширины лепестков;
- `target` — массив, содержащий цель обучения (оцифрованные виды ирисов: 0, 1, 2);
- `target_names` — массив строк, содержащий виды цветов, которые мы хотим предсказать;
- `DESCR` — краткое описание набора данных;
- `feature_names` — список строк с описанием каждого признака;
- `filename` — наименование файла, в котором содержится этот набор данных.

Каждая строка в массиве `data` соответствует одному цветку ириса, а столбцы представляют собой признаки, которые характеризуют каждый цветок. Мы видим, что массив содержит 150 различных цветов (в массиве 150 строк), каждый цветок характеризуется четырьмя признаками (4 столбца). Вспомним, что в машинном обучении отдельные элементы называются *примерами* (samples), а их свойства — *характеристиками*, или признаками (feature). *Форма* (shape) массива данных определяется количеством примеров, умноженным на количество признаков. Это общепринятое соглашение в scikit-learn, и данные всегда будут представлены в такой форме. Далее приведена строка программы, которая выведет на печать значения всех признаков для первых пяти примеров:

```
print("Первые пять строк массива data:\n{}".format(iris_dataset['data'][:5]))
```

А вот как выглядит итог работы этой строки программного кода:

```
Первые пять строк массива data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

Массив `target` содержит виды уже измеренных цветов, тоже записанные в формате массива NumPy. Каждый вид цветка закодирован целыми числами: 0, 1, 2. Это одномерный массив — по одному элементу для каждого цветка. Получить значения этого массива можно следующей командой:

```
print("Правильные ответы:\n{}".format(iris_dataset['target']))
```

Массив содержит следующие значения:

```
Правильные ответы:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Здесь цифры соответствуют виду цветков:

- 0 — setosa;
- 1 — versicolor;
- 2 — virginica.

6.3.2. Обучающие и тестовые наборы данных в библиотеке scikit-learn

На основе сформированных в предыдущем разделе данных нам нужно построить модель машинного обучения, которая предскажет виды ириса для нового набора измерений. Но прежде чем мы применим нашу модель к новому набору, мы должны убедиться в том, что модель на самом деле работает и ее прогнозам можно доверять. К сожалению, для оценки качества модели мы не можем использовать данные, которые были взяты для обучения модели. Это обусловлено тем, что наша модель просто запомнит весь обучающий набор и поэтому всегда будет предсказывать правильную метку для любой точки данных в обучающем наборе. Такое «запоминание» ничего не говорит нам об обобщающей способности модели (другими словами, мы не знаем, будет ли эта модель так же хорошо работать на новых данных).

Для оценки эффективности модели мы предъявляем ей новые данные (данные, которые она не видела раньше). Обычно это делается путем разбиения собранных данных (в нашем случае сведений о 150 цветах) на две части. Одна часть этих данных используется для обучения нашей нейронной сети и называется *обучающими данными* (training data), или *обучающим набором* (training set). Остальные данные используются для оценки качества обучения — их называют *тестовыми данными* (test data), *тестовым набором* (test set) или *контрольным набором* (hold-out set). В библиотеке scikit-learn есть функция `train_test_split()`, которая перемешивает набор данных и разбивает его на две части. Она отбирает в обучающий набор 75% строк данных с соответствующими метками. Оставшиеся данные (25%) объявляются тестовым набором. Вопрос о том, сколько данных отбирать в обучающий набор данных, а сколько — в тестовый набор, является дискуссионным, однако использование тестового набора, содержащего 25% данных, считается хорошим правилом. В scikit-learn данные, как правило, обозначаются заглавной X, тогда как метки (правильные ответы) — строчной буквой y. Это навеяно стандартной математической формулой:

$$f(X) = y,$$

где X является аргументом функции, а y — выводом (значением функции). В соответствии с принятыми соглашениями мы используем заглавную X, потому что эти данные представляют собой двумерный массив (матрицу), и строчную y, потому

что целевая переменная является одномерным массивом (вектором). Как уже было сказано, для разбиения выборки обучающих данных на две части служит функция train_test_split():

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =  train_test_split(iris_dataset['data'],
                                                     iris_dataset['target'],
                                                     random_state=0)
```

Перед разбиением обучающего набора данных на две части функция train_test_split() перемешивает его с помощью генератора псевдослучайных чисел. Если просто брать последние 25% наблюдений в качестве тестового набора, все точки данных будут иметь вид цветка, равный 2. Учитывая это, функция перемешивает отсортированную информацию, чтобы тестовые данные содержали все три класса цветков.

Для того чтобы в точности повторно воспроизвести полученный результат, мы воспользуемся генератором псевдослучайных чисел с фиксированным стартовым значением, которое задается с помощью параметра random_state. Это позволит сделать результат воспроизводимым, поскольку приведенный программный код функции train_test_split() будет генерировать один и тот же результат. Выводом этой функции являются четыре массива: X_train, X_test, y_train и y_test, которые являются массивами (матрицами и векторами) библиотеки Numpy. При этом массив X_train содержит 75% строк исходного набора данных, а массив X_test — 25% строк. Размерность сформированных массивов можно получить с помощью команд:

```
print("Размерность массива X_train: {}".format(X_train.shape))
print("Размерность массива y_train: {}".format(y_train.shape))
print("Размерность массива X_test: {}".format(X_test.shape))
print("Размерность массива y_test: {}".format(y_test.shape))
```

В результате работы этого программного кода для нашего набора данных будет получен следующий результат:

```
Размерность массива X_train: (112, 4)
Размерность массива y_train: (112,)
Размерность массива X_test: (38, 4)
Размерность массива y_test: (38,)
```

6.3.3. Предварительный анализ наборов данных

Перед тем как строить и обучать нашу модель, неплохо было бы исследовать данные, чтобы понять, можно ли вообще решить поставленную задачу и содержится ли нужная информация в данных. Кроме того, исследование данных — это хороший способ обнаружить аномалии и ошибки. Например, вполне возможно, что некоторые из ирисов измерены в дюймах, а не в сантиметрах. В реальном мире нестыковки в данных и прочие подобные неожиданности очень распространены.

Один из лучших способов исследовать данные — визуализировать их. Это можно сделать, используя *диаграмму рассеяния* (scatter plot). В ней один признак откладывается по оси *x*, а другой признак — по оси *y*, и каждому наблюдению соответствует точка. К сожалению, экран компьютера имеет только два измерения, что позволяет разместить на графике лишь два (или, возможно, три) признака одновременно. То есть трудно разместить на графике наборы данных с более чем тремя признаками. Один из способов решения этой проблемы — построить *матрицу диаграмм рассеяния* (scatterplot matrix) или *парные диаграммы рассеяния* (pair plots), на которых будут изображены все возможные пары признаков. Если имеется небольшое число признаков — например, четыре, как у нас, то использование матрицы диаграмм рассеяния будет вполне разумным. Однако необходимо помнить, что матрица диаграмм рассеяния не показывает взаимодействие между всеми признаками сразу, поэтому некоторые интересные аспекты данных не будут выявлены с помощью этих графиков.

Достаточно просто построить матрицу диаграмм рассеяния, задействовав библиотеку seaborn. Эта библиотека работает совместно с библиотеками pandas и matplotlib. В этом издании книги использовалась библиотека seaborn версии 0.12.1. Достаточно импортировать указанные библиотеки, а дальше матрица диаграмм рассеяния создается всего несколькими программными строками (листинг 6.15).

Листинг 6.15

```
import seaborn as sb
from matplotlib import pyplot as plt

df = sb.load_dataset('iris')
sb.set_style("ticks")
sb.pairplot(df, hue='species', diag_kind="kde", kind="scatter", palette="husl")
plt.show()
```

В листинге 6.16 приведен полный код программы подготовки и анализа набора данных об ирисах.

Листинг 6.16

```
# Модуль SciLearn
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris_dataset = load_iris()
print("Ключи iris_dataset: \n{}".format(iris_dataset.keys()))
```

```
print("Тип массива data: {}".format(type(iris_dataset['data'])))
print("Форма массива data: {}".format(iris_dataset['data'].shape))
print("Цель: {}".format(iris_dataset['target']))
print("Названия ответов: {}".format(iris_dataset['target_names']))
print(iris_dataset['DESCR'][:193] + "\n...")
print("Названия признаков: \n{}".format(iris_dataset['feature_names']))
print("Расположение файла: \n{}".format(iris_dataset['filename']))
print("Первые пять строк массива data:\n{}".format(iris_dataset['data'][:5]))
print("Правильные ответы:\n{}".format(iris_dataset['target']))

X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                    iris_dataset['target'], random_state=0)

print("Размерность массива X_train: {}".format(X_train.shape))
print("Размерность массива y_train: {}".format(y_train.shape))
print("Размерность массива X_test: {}".format(X_test.shape))
print("Размерность массива y_test: {}".format(y_test.shape))

# Создание и обучение классификатора
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
# Практическое использование классификатора
X_new = np.array([[5, 2.9, 1, 0.2]])
pr = knn.predict(X_new)
# print("Метка вида цветка: {}".format(pr))
print("Вид цветка: {}".format(iris_dataset['target_names'][pr]))

pr = knn.predict(X_test)
print("Прогноз вида на тестовом наборе:\n {}".format(pr))
print("Точность прогноза на тестовом наборе: {:.2f}".format(np.mean(pr == y_test)))

# Матрица рассеяния с библиотекой seaborn
df = sb.load_dataset('iris')
sb.set_style("ticks")
sb.pairplot(df, hue='species', diag_kind="kde", kind="scatter", palette="husl")
plt.show()
```

В результате работы этого программного кода мы получим следующую картину (рис. 6.14).

Здесь видно, что наши три класса цветков можно разделить по размерам чашелистиков и лепестков. Значит, на основе имеющихся у нас данных нейронную сеть можно научить решать задачу классификации этих растений.

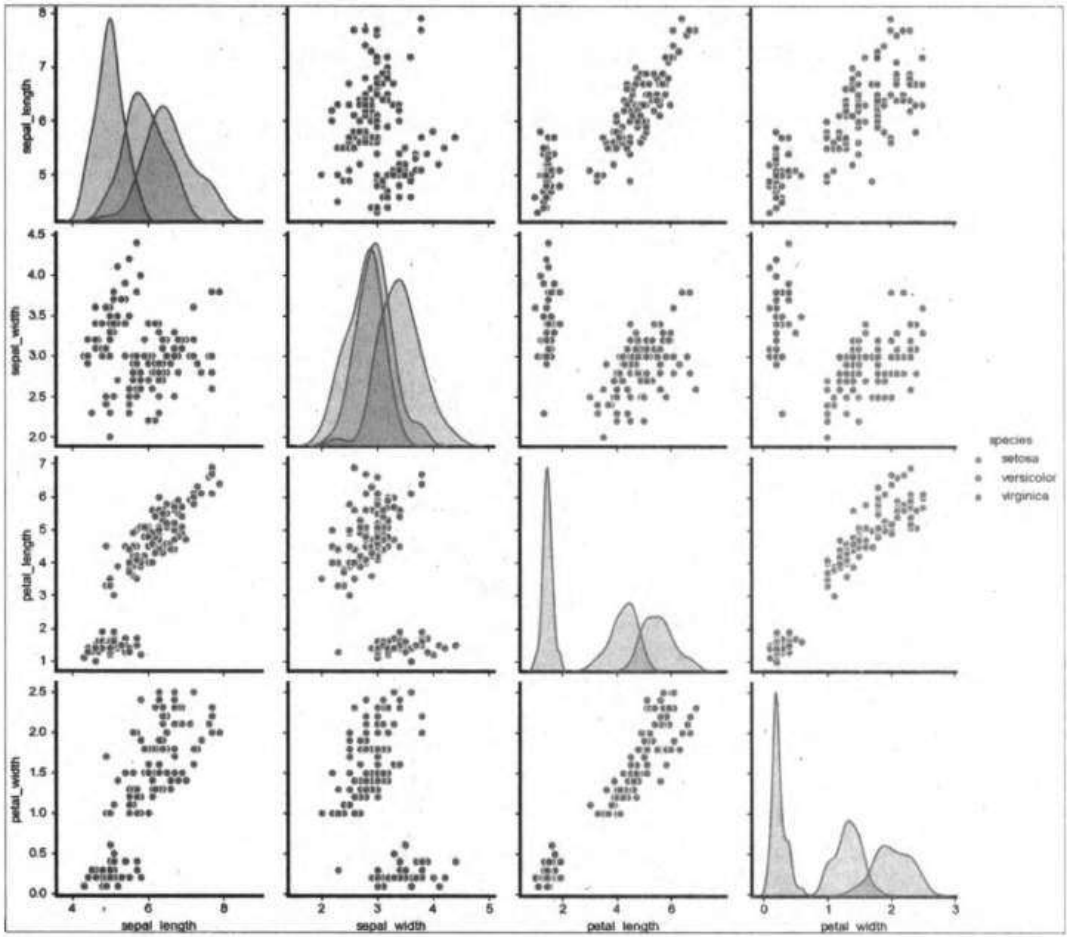


Рис. 6.14. Матрица диаграмм рассеяния параметров цветков ириса

6.3.4. Обучение нейронной сети с библиотекой scikit-learn

В библиотеке scikit-learn имеется довольно много алгоритмов классификации, которые можно задействовать для обучения моделей. В нашем примере мы воспользуемся классификатором на основе метода *k* ближайших соседей. Это обучение заключается лишь в запоминании обучающего набора данных. Для того чтобы сделать прогноз для новой точки данных, алгоритм отыскивает точку в обучающем наборе, которая находится ближе всего к заданной точке. Затем он присваивает метку, принадлежащую этой точке обучающего набора, новой точке данных.

В этом методе используются не только ближайшие соседи новой точки данных — в ходе обучения можно рассмотреть любое фиксированное число (*k*) соседей (например, рассмотреть ближайшие три, пять или десять соседей). Тогда мы сможем сделать прогноз для точки данных, ориентируясь на класс, которому принадлежит большинство ее соседей. Однако сейчас мы будем использовать только одного соседа.

В scikit-learn все модели машинного обучения реализованы в собственных классах, называемых *классами Estimator* (оценщик). Алгоритм классификации на основе метода *k* ближайших соседей реализован в классификаторе `KNeighborsClassifier` модуля `neighbors`. Прежде чем использовать этот классификатор, нам нужно создать объект-экземпляр класса. Это произойдет, когда мы зададим нашему объекту необходимые параметры. Самым важным параметром `KNeighborsClassifier` является количество соседей, которое мы установим равным 1:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Объект `knn` содержит в себе и алгоритм, который будет использоваться для построения модели на обучающих данных, и алгоритм, который сгенерирует прогнозы для новых точек данных. Он также будет содержать информацию, которую алгоритм извлек из обучающей выборки в процессе обучения. В случае с `KNeighborsClassifier` он будет просто хранить обучающий набор. Для тренировки модели на обучающем наборе мы вызываем метод `fit()` объекта `knn`, который принимает в качестве аргументов массив NumPy `X_train`, содержащий обучающие данные, и массив NumPy `y_train`, соответствующий обучающим меткам:

```
z = knn.fit(X_train, y_train)
print(z)
```

Метод `fit()` возвращает сам объект `knn` (и изменяет его). Таким образом, мы получаем строковое представление нашего классификатора. Последней командой будут распечатаны те параметры, которые были использованы при создании модели:

```
KNeighborsClassifier(algorithm='auto',leaf_size=30,
                     metric='minkowski', metric_params=None,
                     n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

Все параметры имеют значения по умолчанию, но существует один параметр, который мы задали сами, — это `n_neighbor=1`. У большинства моделей в scikit-learn масса параметров, но бóльшая часть из них связана с оптимизацией скорости вычислений или предназначена для особых случаев использования.

Теперь мы можем получить прогнозы, применив эту модель к новым данным, по которым мы еще не знаем правильные метки. Представьте, что мы нашли в дикой природе ирис с длиной чашелистика 5 см, шириной чашелистика 2,9 см, длиной лепестка 1 см и шириной лепестка 0,2 см. К какому виду ириса нужно отнести этот цветок? Мы можем поместить эти данные в массив NumPy, снова вычисляя форму массива, — т. е. количество примеров (1) умножая на количество признаков (4):

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))
```

На выходе из программы, где есть две такие строки, получим:

```
форма массива X_new: (1, 4)
```

Мы создали массив из одной строки и четырех столбцов, и в этой строке содержатся параметры нашего неизвестного цветка.

Для того чтобы сделать прогноз (определить, какой же это вид цветка), нужно вызвать метод `predict()` объекта `knn`:

```
pr = knn.predict(X_new)
print("Метка сорта цветка: {}".format(pr))
print("Сорт цветка: {}".format(iris_dataset['target_names'][pr]))
```

В листинге 6.17 приведен полный текст программы тренировки сети и ее использования для предсказания вида цветка.

Листинг 6.17

```
# Модуль SciLearn0
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris_dataset = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                   iris_dataset['target'], random_state=0)
knn = KNeighborsClassifier(n_neighbors=1)
z = knn.fit(X_train, y_train)
X_new = np.array([[5, 2.9, 1, 0.2]])
pr = knn.predict(X_new)
print("Метка вида цветка: {}".format(pr))
print("Вид цветка: {}".format(iris_dataset['target_names'][pr]))
```

В результате работы этого программного кода получим следующий результат:

```
Метка вида цветка: [0]
Вид цветка: ['setosa']
```

Наша обученная модель предсказала, что этот новый цветок ириса принадлежит к классу 0, что означает вид setosa.

Подведем итог наших усилий. С использованием библиотеки scikit-learn для создания и обучения интеллектуального классификатора нам понадобилось всего две строчки программного кода. И три строчки программного кода, чтобы воспользоваться результатами обучения (листинг 6.18).

Листинг 6.18

```
# Это промежуточный код, он не является рабочим
# создание и обучение классификатора
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

```
# Практическое использование классификатора
X_new = np.array([[5, 2.9, 1, 0.2]])
pr = knn.predict(X_new)
print("Вид цветка: {}".format(iris_dataset['target_names'][pr]))
```

Этот пример наглядно демонстрирует, насколько эффективно в Python применять готовые специализированные библиотеки для создания и обучения нейронных сетей. Но как узнать, можем ли мы доверять нашей обученной модели? Правильно ли наша модель определила вид цветка, размеры которого были поданы на ее вход? Ведь именно получение правильных прогнозов и является главной задачей построения модели. В следующем разделе будет показано, как можно оценить качество обучения модели и насколько можно доверять решениям обученной нейронной сети.

6.3.5. Оценка качества обучения моделей в библиотеке scikit-learn

Вот наступил тот самый момент, когда нам понадобится созданный ранее тестовый набор. Эти данные не использовались для обучения модели. Однако в тестовом наборе известны правильные виды каждого ириса. То есть мы можем сделать прогноз для каждого ириса в тестовом наборе и сравнить его с фактической меткой (уже известным сортом). Мы также можем оценить качество модели, вычислив точность (accuracy), т. е. подсчитать процент цветов, для которых модель правильно спрогнозировала вид. Это можно сделать, добавив к программе, приведенной в предыдущем разделе, следующие строки:

```
pr = knn.predict(X_test)
print("Прогноз вида на тестовом наборе:\n {}".format(pr))
print("Точность прогноза на тестовом наборе:{:.2f}".format(np.mean(pr == y_test)))
```

Код объединенного программного модуля приведен в листинге 6.18.1.

Листинг 6.18.1

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_splitfrom sklearn.neighbors import
KNeighborsClassifier

iris_dataset = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                   iris_dataset['target'], random_state=0)
knn = KNeighborsClassifier(n_neighbors=1)
z = knn.fit(X_train, y_train)
X_new = np.array([[5, 2.9, 1, 0.2]])
pr = knn.predict(X_new)
print("Метка вида цветка: {}".format(pr))
print("Вид цветка: {}".format(iris_dataset['target_names'][pr]))
```

```
pr = knn.predict(X_test)
print("Прогноз вида на тестовом наборе:\n {}".format(pr))
print("Точность прогноза на тестовом наборе:{:.2f}".format(np.mean(pr == y_test)))
```

После выполнения этого программного кода получим следующий результат:

```
Прогноз вида на тестовом наборе:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
Точность прогноза на тестовом наборе: 0.97
```

Точность этой модели для тестового набора данных составляет `0.97`, а это означает, что наша классификационная модель после обучения дала правильный прогноз для 97% ирисов в тестовом наборе. Следовательно, мы можем ожидать, что наша модель в 97% случаев даст правильный прогноз и для новых ирисов. То есть наша модель может быть достаточно надежной в использовании.

6.3.6. Персептрон и библиотека scikit-learn

В предыдущих главах мы знакомились с двумя связанными между собой алгоритмами обучения для решения задач классификации: правилом персептрона и алгоритмом адаптивных линейных нейронов (ADALINE). Для реализации этих алгоритмов мы использовали Python и библиотеки для работы с математическими функциями. Программный код при этом получился достаточно большим и сложным. Попробуем теперь реализовать те же функции на тех же примерах, но с использованием библиотеки scikit-learn, и посмотрим, насколько при этом упростится программный код.

Библиотека scikit-learn сочетает удобные для пользователя средства взаимодействия с высокооптимизированной реализацией нескольких алгоритмов классификации. Вместе с тем библиотека scikit-learn предлагает не только большое разнообразие алгоритмов обучения, но и значительное количество удобных функций для предобработки данных в целях тонкой настройки и оценки моделей.

Приступим к работе с библиотекой scikit-learn. Построим и натренируем модель на базе персептрона, аналогичную той, которую мы реализовали в *главе 4*. Для простоты мы воспользуемся уже знакомым нам набором данных ирисов (Iris). Как было отмечено в предыдущем разделе, он уже имеется в библиотеке scikit-learn, поскольку этот простой и одновременно популярный набор данных часто используется в обучении для тестирования алгоритмов и проведения с ним экспериментов. Кроме того, в целях визуализации мы воспользуемся из этого набора данных всего двумя признаками цветков: присвоим матрице признаков `X` длину и ширину лепестков из 150 образцов цветков, а вектору `y` — метки классов, которые соответствуют видам цветков.

Фрагмент соответствующего программного кода приведен в листинге 6.19.

Листинг 6.19

```
from sklearn import datasets
iris = datasets.load_iris()
```

```
X = iris.data[:, [2, 3]]
y = iris.target
print(y)
```

Последний оператор выводит метки классов, хранящихся в `iris.target`:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  2 2]
```

Здесь мы видим, что имена видов цветков ирис щетинистый (Iris setosa), ирис виргинский (Iris virginica) и ирис разноцветный (Iris versicolor) представлены в виде целых чисел (0, 1, 2). Так поступают во многих библиотеках машинного обучения в целях оптимизации производительности вычислений.

Разделим исходный набор данных на две части: на тренировочный и тестовый наборы. Это делается для возможности оценки, насколько хорошо натренированная модель будет работать на ранее не встречавшихся ей данных. Разбивку можно сделать следующим образом:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=0)
```

При помощи функции `train_test_split` из модуля `model_selection` библиотеки scikit-learn мы произвольным образом разделяем массивы `X` и `y` на тестовые данные в размере 30% от общего объема (45 образцов) и тренировочные данные в размере 70% (105 образцов). Как мы уже знаем, многие алгоритмы машинного обучения в целях улучшения качества обучения требуют масштабирования признаков. Здесь мы выполним стандартизацию признаков, воспользовавшись для этого классом `StandardScaler` из модуля `preprocessing` библиотеки scikit-learn (листинг 6.20).

Листинг 6.20

```
# Это промежуточный код, он не является рабочим
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

В приведенном коде мы загрузили класс `StandardScaler` из модуля предобработки и инициализировали новый объект `StandardScaler`, который мы присвоили переменной `sc`. Затем воспользовались методом `fit()` объекта `StandardScaler`, чтобы вычислить параметры μ (эмпирическое среднее) и σ (стандартное отклонение) для каждой размерности признаков из тренировочных данных. Вызвав метод `transform()`, мы затем стандартизировали тренировочные данные, используя для этого расчетные параметры μ и σ. Отметим, что для стандартизации тестового набора мы задейст-

вовали те же самые параметры масштабирования, благодаря чему значения в тренировочном и тестовом наборах между собой сопоставимы.

Стандартизировав тренировочные данные, теперь можно натренировать модель на базе персептрона. Большинство алгоритмов в библиотеке scikit-learn поддерживают многоклассовую классификацию уже по умолчанию. Это возможно благодаря методу «один против всех», который позволяет передать в персептрон сразу все три класса цветков. Исходный код для этого выглядит так, как представлено в листинге 6.21.

Листинг 6.21

```
# Это промежуточный код, он не является рабочим
from sklearn.linear_model import Perceptron
ppn = Perceptron(eta0=0.1, random_state=0, max_iter=40)
ppn.fit(X_train_std, y_train)
```

Интерфейс библиотеки scikit-learn выглядит аналогично нашей реализации персептрона в *главе 4*. После загрузки класса `Perceptron` из модуля `linear_model` мы инициализировали новый объект `Perceptron` и натренировали модель при помощи метода `fit()`. Параметр `eta0` здесь эквивалентен темпу обучения `eta`, который мы использовали в нашей собственной реализации персептрона, а параметр `max_iter` задает число эпох (проходов по тренировочному набору). Как мы помним из *главы 4*, для нахождения надлежащего темпа обучения требуется немного поэкспериментировать. Если темп обучения слишком большой, то алгоритм промахнется по глобальному минимуму стоимости. Если темп обучения слишком малый, то для достижения сходимости алгоритм потребует большего количества эпох, что может замедлить обучение, особенно для больших наборов данных. Кроме того, мы использовали параметр `random_state` для повторения исходного перемешивания тренировочного набора данных после каждой эпохи.

Натренировав модель в scikit-learn, мы можем приступить к выполнению прогнозов, используя для этого метод `predict()`, точно так же, как в нашей собственной реализации персептрона в *главе 4*. Соответствующий исходный код выглядит следующим образом:

```
y_pred = ppn.predict(X_test_std)
print('Число ошибочно классифицированных образцов: %d' % (y_test != y_pred).sum())
```

Вместо ошибки классификации для оценки точности прогноза используется также *верность предсказания* (accuracy). Это значение легко получить следующим образом:

```
from sklearn.metrics import accuracy_score
print('Верность: %.2f' % accuracy_score(y_test, y_pred))
```

Если объединить приведенные здесь четыре строки с листингами 6.19–6.21, то мы получим следующий код программы (листинг 6.22).

Листинг 6.22

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
print(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=0)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

ppn = Perceptron(eta0=0.1, random_state=0, max_iter=40)
ppn.fit(X_train_std, y_train)

y_pred = ppn.predict(X_test_std)
print('Число ошибочно классифицированных образцов: % d' % (y_test != y_pred).sum())
print('Верность: %.2f' % accuracy_score(y_test, y_pred))
```

Результат его выполнения будет следующий:

```
Число ошибочно классифицированных образцов: 5
Верность: 0.89
```

Здесь мы видим, что персептрон ошибочно классифицирует 5 из 45 образцов цветков. Таким образом, ошибка классификации на тестовом наборе данных составляет 0,111, или 11,1%.

Если не принимать во внимание фрагменты программного кода, связанного с подключением библиотек и подготовкой исходных данных, то с помощью библиотеки scikit-learn для создания и обучения персептрона потребовалось всего две строчки программного кода:

```
ppn = Perceptron(eta0=0.1, random_state=0, max_iter=40)
ppn.fit(X_train_std, y_train)
```

Этот код намного короче и проще чем тот, который был написан в *главе 4* с использованием только языка Python и библиотек работы с математическими функциями.

Можно дополнить наш программный код и посмотреть, как разделяются наши три вида цветов на классы (листинг 6.22.1).

Листинг 6.22.1

```
# Это промежуточный код, он не является рабочим
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # настроить генератор маркеров и палитру
    markers = ('s', 'x', 'o', '>', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # вывести поверхность решения
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                                    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # показать все образцы
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx),
        marker=markers[idx], label=cl)
        # выделить тестовые образцы
        if test_idx:
            X_test, y_test = X[test_idx, :], y[test_idx]
            plt.scatter(X_test[:, 0], X_test[:, 1], c='0', alpha=1.0,
                        linewidths=1, marker='.', s=55, label='тест набор')

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = p.hstack((y_train, y_test))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=ppn,
                      test_idx=range(105, 150))
plt.xlabel('Длина лепестка [стандартизованная]')
plt.ylabel('Ширина лепестка [стандартизованная]')
plt.legend(loc='upper left')
plt.show()
```

Объединим все фрагменты программ, представленные в этом разделе, в единый программный модуль (листинг 6.23).

Листинг 6.23

```
# Модуль Ski_Perseptron
from matplotlib.colors import ListedColormap
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
print(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=0)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

ppn = Perceptron(eta0=0.1, random_state=0, max_iter=40)
ppn.fit(X_train_std, y_train)

y_pred = ppn.predict(X_test_std)
print('Число ошибочно классифицированных образцов:'
      '% d' % (y_test != y_pred).sum())
print('Верность: %.2f' % accuracy_score(y_test, y_pred))

def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # настроить генератор маркеров и палитру
    markers = ('s', 'x', 'o', '>', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # вывести поверхность решения
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # показать все образцы
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)
        # выделить тестовые образцы
        if test_idx:
            X_test, y_test = X[test_idx, :], y[test_idx]
            plt.scatter(X_test[:, 0], X_test[:, 1], c='0', alpha=1.0,
                        linewidths=1, marker='.', s=55, label='тест-набор')
```

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=ppn,
                      test_idx=range(105, 150))
plt.xlabel('Длина лепестка [стандартизованная]')
plt.ylabel('Ширина лепестка [стандартизованная]')
plt.legend(loc='upper left')
plt.show()
```

Результаты работы объединенной программы приведены на рис. 6.15.

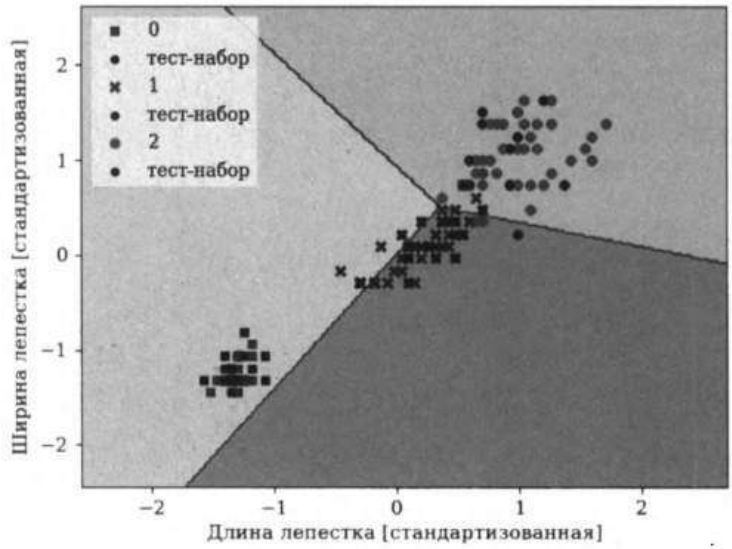


Рис. 6.15. График областей решений для представления разделения цветков ириса на классы

Как видно из полученного графика, полностью разделить три класса цветков линейной границей решения не получается. Известно, что алгоритм обучения персептрона никогда не сходится на наборах данных, которые не полностью линейно разделимы, и поэтому этот алгоритм обычно не рекомендуется применять на практике для таких данных. Известны более мощные линейные классификаторы, которые сходятся к минимуму стоимости потерь, даже если классы не полностью линейно разделимы. В следующем разделе мы познакомимся с одним из таких классификаторов.

6.3.7. Классификаторы на основе логистической регрессии в библиотеке scikit-learn

Несмотря на то что правило персептрона предлагает простое и удобное введение в алгоритмы машинного обучения для классификации данных, самый большой не-

достаток этого правила заключается в том, что оно не обеспечивает сходимости в случае, если классы не полностью линейно разделимы. Рассмотренная в предыдущем разделе задача классификации являлась примером такого сценария. Интуитивно мы можем аргументировать это тем, что веса постоянно обновляются из-за того, что в каждой эпохе всегда существует, по крайней мере, один ошибочно классифицированный образец. Разумеется, можно изменить темп обучения и увеличить число эпох, однако персептрон на таком наборе данных все равно никогда не сойдется. Взглянем на другой простой и одновременно более мощный алгоритм для задач линейной и бинарной классификации — *логистическую регрессию*. Отметим, что, несмотря на название этого метода, он решает задачи классификации, а не регрессии.

Логистическую регрессию как модель классификации очень просто реализовать, и одновременно она очень хорошо работает на линейно разделимых классах. Это один из наиболее широко используемых алгоритмов для классификации данных в машинном обучении. Подобно персептрону и ADALINE, логистическая регрессионная модель представляет собой линейную модель бинарной классификации, которую тоже можно расширить на многоклассовую классификацию.

В логистической регрессии роль функции активации играет сигмоидальная функция, как это показано на рис. 6.16.

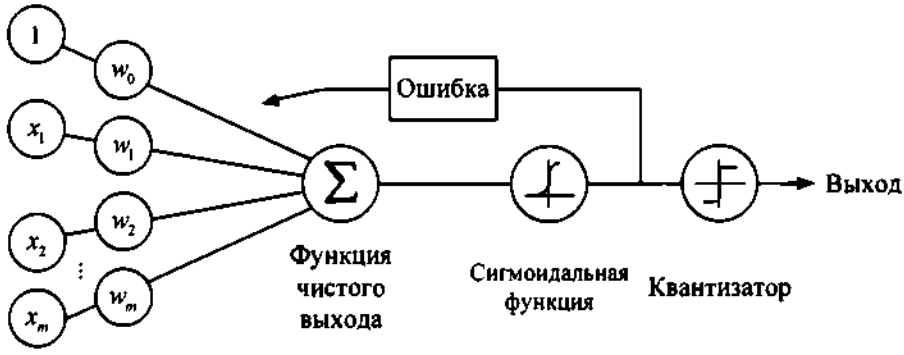


Рис. 6.16. Структура модели логистической регрессии

Выход из сигмоидальной функции интерпретируется как вероятность принадлежности отдельно взятого образца к объекту одного из классов (при наличии у этого объекта признаков x, параметризованных весами w). Например, если для отдельно взятого образца цветка мы на выходе сигмоидальной функции получаем значение $S = 0{,}8$, значит, этот цветок с вероятностью 80% относится к цветкам того же класса. Предсказанную вероятность затем можно просто конвертировать квантизатором (единичной ступенчатой функцией) в бинарный результат. Например, $Y = 1$, если $S \geq 0{,}5$, иначе $Y = 0$.

На деле во многих приложениях важна не только идентификация меток классов, но и оценка вероятности принадлежности к определенному классу. К примеру, логистическая регрессия применяется в погодном прогнозировании, чтобы не только

предсказывать, будет ли идти дождь в конкретный день, но и сообщать о шансе (вероятности) дождя. Подобным образом логистическая регрессия может использоваться для предсказания у пациента заболевания при наличии определенных симптомов. Вот почему логистическая регрессия имеет широкую популярность в сфере медицины.

Мы не будем углубляться в математические тонкости модели логистической регрессии. Просто отметим, что в библиотеке scikit-learn реализована высокооптимизированная версия логистической регрессии, которая к тому же поддерживает готовую многоклассовую конфигурацию. Мы воспользуемся готовым классом `sklearn.linear_model.LogisticRegression`, а также знакомым методом `fit()` для тренировки модели на стандартизированном тренировочном наборе данных цветков ириса. Допишем к программе, представленной в предыдущем разделе, следующие строки (листинг 6.24).

Листинг 6.24

```
# Это промежуточный код, он не является рабочим
from sklearn.linear_model import Perceptron, LogisticRegression
Lr = LogisticRegression(C=1000.0, random_state=0)
Lr.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std, y_combined, classifier=Lr, test_idx=range(105, 150))
plt.xlabel('Длина лепестка [стандартизированная]')
plt.ylabel('Ширина лепестка [стандартизированная]')
plt.legend(loc='upper left')
plt.show()
```

Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_24_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результаты работы этого программного кода представлены на рис. 6.17.

На нем видно, что с использованием логистической регрессии удалось разделить наши объекты на три класса. Однако два вида цветков разделяются не совсем четко и имеют область перекрещивания. В этой области отнести цветок к тому или иному виду можно только с определенной вероятностью. Так, предсказать вероятность принадлежности конкретного образца к тому или иному классу можно при помощи метода `predict_proba()`, как это сделано, например, для предсказания вероятности принадлежности к тому или иному виду трех образцов ириса (листинг 6.25).

Листинг 6.25

```
# Это промежуточный код, он не является рабочим
X1 = np.array([[1.5, 1.5]])
X2 = np.array([[0.0, 0.0]])
X3 = np.array([[-1, -1]])
p1 = Lr.predict_proba(X1)
p2 = Lr.predict_proba(X2)
p3 = Lr.predict_proba(X3)
```

```
print(X1)
print(p1)
print(X2)
print(p2)
print(X3)
print(p3)
```

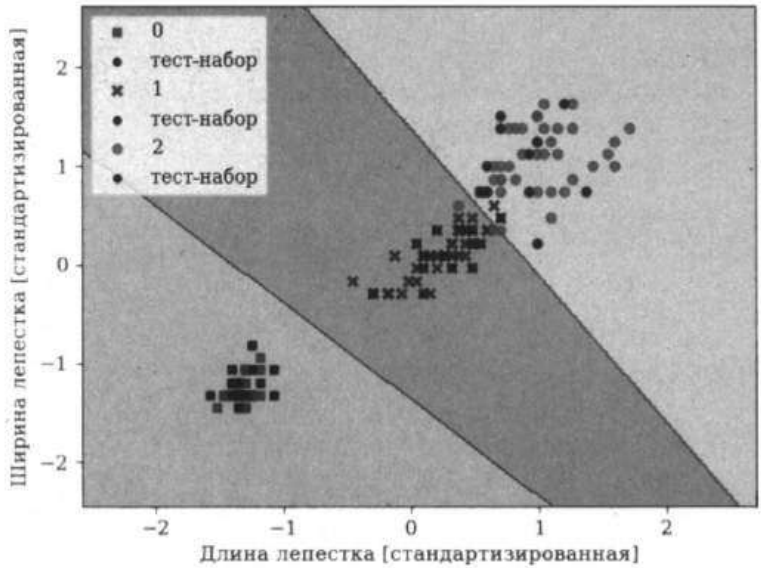


Рис. 6.17. Структура модели логистической регрессии

В листинге 6.26 представлен полный текст программы этого и предыдущего разделов.

Листинг 6.26

```
# Модуль SciPer
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=0)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

ppn = Perceptron(eta0=0.1, random_state=0, max_iter=40)
ppn.fit(X_train_std, y_train)
y_pred = ppn.predict(X_test_std)
print('Число ошибочно классифицированных образцов: %d' % (y_test != y_pred).sum())
print('Вероятность: %.2f' % accuracy_score(y_test, y_pred))

def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # настроить генератор маркеров и палитру
    markers = ('s', 'x', 'o', '>', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # вывести поверхность решения
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # показать все образцы
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)
        # выделить тестовые образцы
        if test_idx:
            X_test, y_test = X[test_idx, :], y[test_idx]
            plt.scatter(X_test[:, 0], X_test[:, 1], c='0', alpha=1.0,
                        linewidths=1, marker='.', s=55,
            label='тест-набор')

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=ppn,
                      test_idx=range(105, 150))

plt.xlabel('Длина лепестка [стандартизованная]')
plt.ylabel('Ширина лепестка [стандартизованная]')
plt.legend(loc='upper left')
plt.show()
```

```
Lr = LogisticRegression(C=1000.0, random_state=0)
Lr.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std, y_combined, classifier=Lr,
                      test_idx=range(105, 150))
plt.xlabel('Длина лепестка [стандартизированная]')
plt.ylabel('Ширина лепестка [стандартизированная]')
plt.legend(loc='upper left')
plt.show()

X1 = np.array([[1.5, 1.5]])
X2 = np.array([[0.0, 0.0]])
X3 = np.array([[-1, -1]])
p1 = Lr.predict_proba(X1)
p2 = Lr.predict_proba(X2)
p3 = Lr.predict_proba(X3)

print(X1)
print(p1)
print(X2)
print(p2)
print(X3)
print(p3)
```

Этот программный код выдаст следующие результаты:

```
[[1.5 1.5]]
[[9.49623993e-22  2.59528262e-07  9.99999740e-01]]
[[0. 0.]]
[[2.87730746e-05  9.99842381e-01  1.28846171e-04]]
[[-1 -1]]
[[9.91289946e-01  8.71005367e-03  1.16513783e-13]]
```

Оформим полученные результаты в виде табл. 6.5.

Таблица 6.5. *Результаты прогноза принадлежности цветка ириса к одному из видов*

№ образца	Входные параметры	Вероятность принадлежности к виду		
		Iris setosa	Iris virginica	Iris versicolor
1	[1.5, 1.5]	0.000	0.000	**0.999**
2	[0.0, 0.0]	0.000	**0.999**	0.000
3	[-1.0, -1.0]	**0.991**	0.008	0.000

Из данных табл. 6.5 видно, что с вероятностью 99% первый образец отнесен к виду Iris versicolor, второй образец — к виду Iris virginica, третий образец — к виду Iris setosa.

6.4. Библиотека Keras и сверточные нейронные сети

6.4.1. Общие сведения о библиотеке Keras

Keras — одна из основных библиотек Python с открытым исходным кодом, служащая для построения нейронных сетей и проектов машинного обучения. Keras может работать совместно с Deeplearning4j, MXNet, Microsoft Cognitive Toolkit (CNTK), Theano, TensorFlow. В этой библиотеке реализованы практически все автономные модули нейронной сети, включая оптимизаторы, нейронные слои, функции активации слоев, схемы инициализации, функции затрат и модели регуляризации. Это позволяет строить новые модули нейронных сетей, просто добавляя функции или классы. И поскольку модель уже определена в коде, разработчику не приходится создавать для нее отдельные конфигурационные файлы.

Разработка библиотеки Keras началась в самом начале 2015 года, и на сегодняшний день она эволюционировала в одну из самых популярных и широко используемых библиотек, построенных поверх библиотеки Theano, которая для ускорения тренировки нейронных сетей позволяет задействовать графический процессор. Одна из ее ярких черт — интуитивно понятный интерфейс, позволяющий реализовывать нейронные сети всего в нескольких строках исходного кода.

Приложения на основе Keras могут быть легко развернуты на большом количестве платформ:

- на iOS через Apple CoreML (поддержка Keras официально предоставляется Apple);
- на Android через среду выполнения TensorFlow Android;
- в браузере с помощью графического ускорителя JavaScript, такого как Keras.js и WebDNN;
- в Google Cloud через TensorFlow-Serving;
- в веб-приложении Python (например, в приложении Flask);
- на JVM через импорт модели DL4J, предоставляемый SkyMind;
- на Raspberry Pi.

Установить библиотеки Theano, TensorFlow и Keras можно из каталога пакетов PyPI при помощи следующих команд из командной строки в окне терминала:

```
pip install Keras
pip install Theano
pip install TensorFlow
```

Мы рассмотрим применение библиотеки Keras на примере анализа изображений с использованием сверточных нейронных сетей. В этой книге использовались следующие версии библиотек:

- Keras — версия 2.11.0;
- Theano — версия 1.0.5;
- ThensorFlow — версия 2.11.0.

Вместе с библиотекой ThensorFlow будет установлено более десятка дополнительных библиотек, которые она использует в своей работе.

6.4.2. Сверточные нейронные сети

Сверточная нейронная сеть (convolutional neural network, CNN) — основной инструмент для классификации и распознавания объектов, лиц на фотографиях, распознавания речи. Мы будем говорить в первую очередь о CNN для работы с изображениями, но свертка — универсальная операция. Ее можно применить для любого сигнала, будь то данные с датчиков, аудиосигнал или картинка. Сверточные сети базируются на достаточно сложном математическом аппарате, но мы не станем углубляться в сложные математические рассуждения, а рассмотрим использование этого вида нейронных сетей на распространенном примере.

Предположим, что у нас стоит задача: выделить на картинке какой-то объект — например, автомобиль. Человек легко понимает, что перед ним автомобиль, и распознает его по тысяче мелких признаков. А сможет ли компьютер понять, что именно этот набор точек на картинке является автомобилем? Ответ простой — да, сможет, если его этому научить, и сеть CNN справится с этой задачей значительно быстрее и эффективнее, чем любая другая.

Сверточная нейронная сеть за счет применения специальной операции, называемой *сверткой*, позволяет уменьшить количество обрабатываемой информации, вследствие чего лучше справляется с картинками высокого разрешения, а также способна выделить опорные признаки изображения, такие как ребра, контуры или грани. На следующем уровне обработки из этих ребер и граней можно распознать повторяющиеся фрагменты текстур, которые дальше могут сложиться во фрагменты изображения.

По сути, каждый слой нейронной сети использует собственное преобразование. Если на первых слоях сеть оперирует такими понятиями, как ребра, грани и т. п., то дальше используются понятия «текстура», «части объектов». В результате такой проработки мы можем правильно классифицировать картинку или выделить на конечном шаге искомый объект на изображении. Первой и, по сути, самой тривиальной задачей, которую научились решать с помощью сверточных нейронных сетей, стала классификация изображений. Но с их помощью можно классифицировать и любые другие сигналы.

Классификация с помощью CNN активно применяется в медицине: можно обучить нейронную сеть классификации болезней или симптомов — например, по анализу магнитно-резонансных снимков. В агробизнесе разрабатывается и внедряется методика анализа и распознавания изображений, при которой данные получают от спутников и используют для прогнозирования будущей урожайности конкретных земельных участков. Распознавание объектов на фото и видео с помощью нейронных сетей применяется в беспилотном транспорте, видеонаблюдении, системах контроля доступа, системах «умного дома» и т. п. Один из примеров изображения, обработанного сверточной нейронной сетью, представлен на рис. 6.18.

Рис. 6.18. Пример выделения объектов на изображении с видеокамеры

Мы рассмотрим применение сверточных нейронных сетей на примере распознавания рукописных цифр. И начнем с того, что разберемся с набором данных MNIST, на котором будем изучать сверточные сети.

Набор данных MNIST — это один из классических наборов данных, на котором принято пробовать все возможные подходы к классификации изображений. Набор содержит 60 тыс. изображений (тренировочная часть) и 10 тыс. изображений (тестовая часть) размером 28×28 пикселов (рукописные цифры от 0 до 9). В библиотеке TensorFlow есть стандартный скрипт для скачивания и загрузки этого набора данных, что весьма удобно. Образцы написания цифр из этого набора данных приведены на рис. 6.19.

Наша задача выглядит следующим образом: надо научить сеть на тренировочной части набора данных классифицировать черно-белое изображение размером

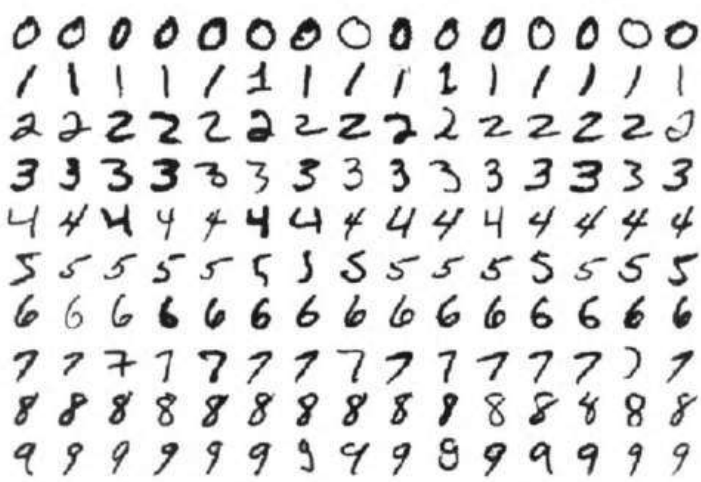


Рис. 6.19. Образцы написания цифр из набора данных MNIST

28×28 пикселов (точек) и определять цифру на этом изображении. Затем мы будем оценивать качество обучения, используя тестовую часть набора, т. е. считать, какой процент изображений из 10 тыс. сеть правильно классифицировала.

Можно построить совсем простую сеть, на входе в которую будет 784 нейрона (28×28 = 784). Каждый из нейронов подключен к одному из пикселов изображения. На выходе будет получен слой с десятью нейронами — по одному на каждую цифру (рис. 6.20).

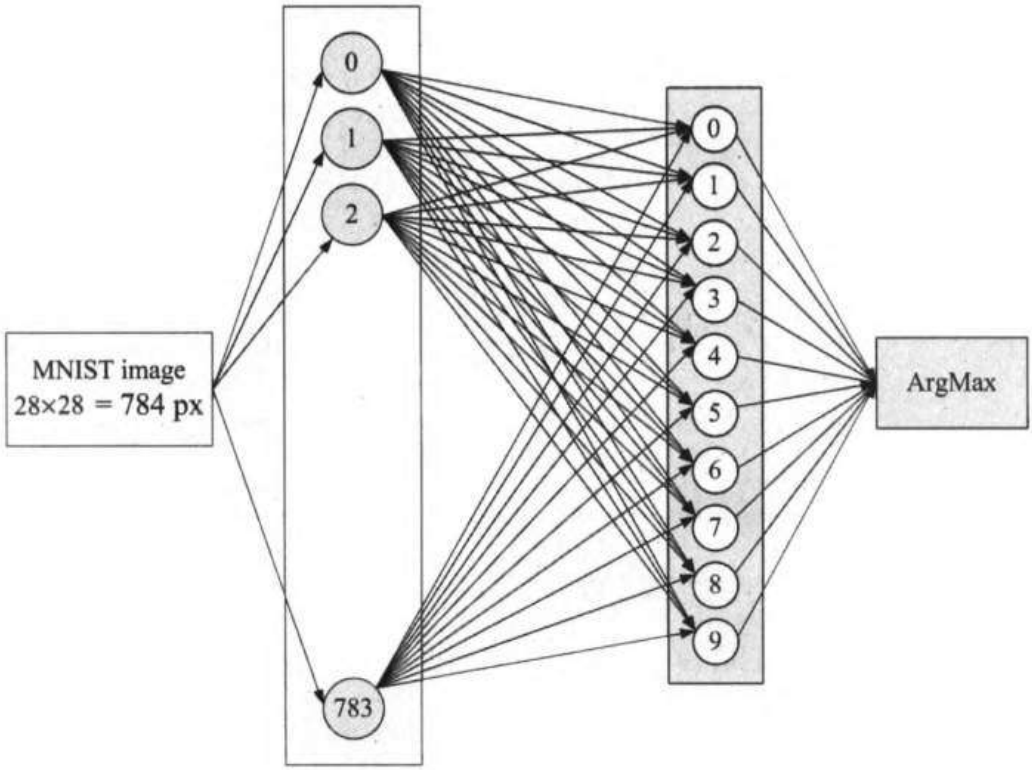


Рис. 6.20. Структура простой нейронной сети для распознавания цифр

При такой структуре сети каждый из 10 выходов будет получать сигналы от 784 точек изображения. То есть модель этой сети имеет 7850 параметров ($Np = 784 \cdot 10 + 10 = 7850$), которые надо натренировать. Изменим предыдущую модель сети, добавив один скрытый слой (рис. 6.21).

Количество параметров, которые нужно натренировать у такой сети, будет зависеть от числа нейронов в скрытом слое. Их количество можно подсчитать по формуле

$$Np = 784 \cdot H + H + H \cdot 10 + 10 = 795 \cdot H + 10.$$

Количество нейронов в этом скрытом слое можно менять. Сверточная нейронная сеть, как следует из названия, вместо обычных слоев использует слои свертки. Не вдаваясь в математические подробности, отметим, что слои свертки, получая на

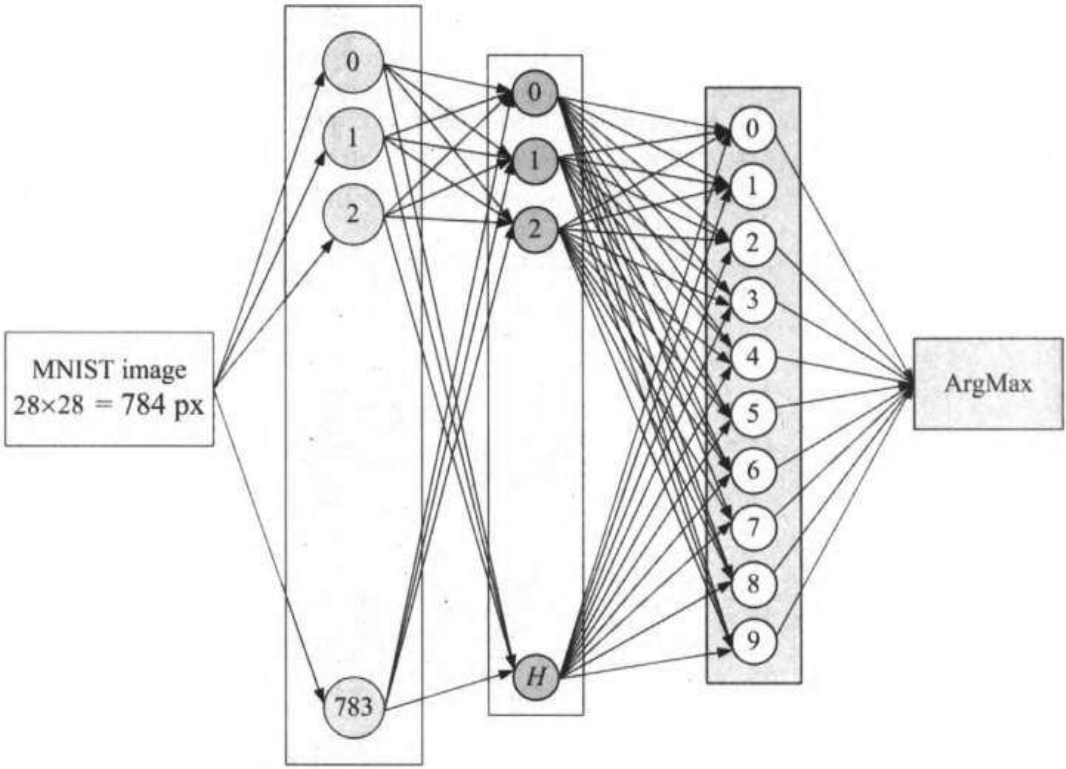


Рис. 6.21. Измененная структура простой нейронной сети для распознавания цифр

вход изображение — например, матрицу размером 28×28, на выходе формируют матрицу меньшего размера — например, матрицу размером 4×4.

Сверточные нейронные сети работают на основе фильтров, которые занимаются распознаванием определенных характеристик изображения (например, прямых линий, дуг и т. п.). *Фильтр* — это коллекция элементов, называемых *кернелами* (чаще всего в фильтре используется один кернел). Кернел (ядро) — это обычная матрица чисел, называемых *весами*, которые «обучаются» с целью поиска на изображениях определенных характеристик. Фильтр перемещается вдоль изображения и определяет, присутствует ли некоторая искомая характеристика в конкретной его части. Схематично работа свертки представлена на рис. 6.22.

Если некоторая искомая характеристика присутствует во фрагменте изображения, операция свертки на выходе будет выдавать число с относительно большим значением. Если же характеристика отсутствует, на выходе число будет маленьким (рис. 6.23).

Характер изменения свойств изображения в процессе свертки показан на рис. 6.24.

Более детально с характеристиками и возможностями сверточных нейронных сетей можно познакомиться в специальной литературе. А сейчас перейдем к реализации примера формирования и обучения сверточной нейронной сети с библиотекой Keras.

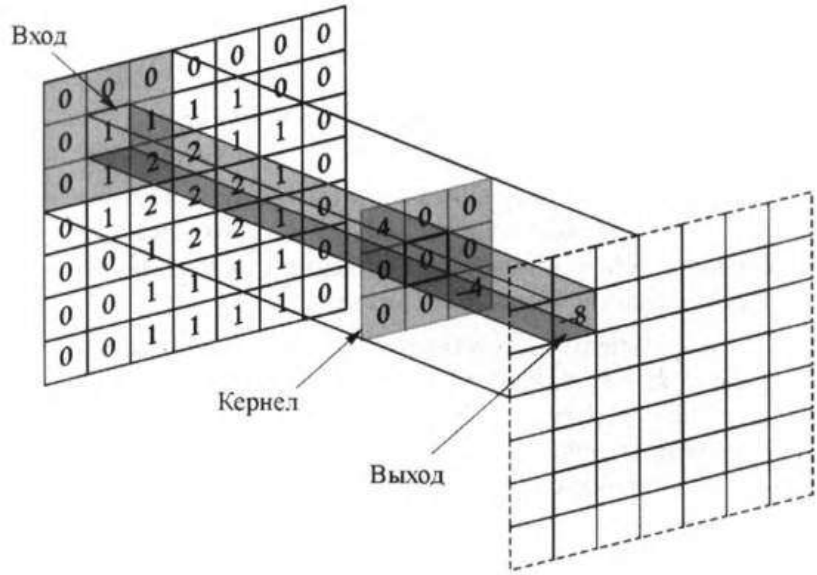


Рис. 6.22. Принцип работы свертки изображения

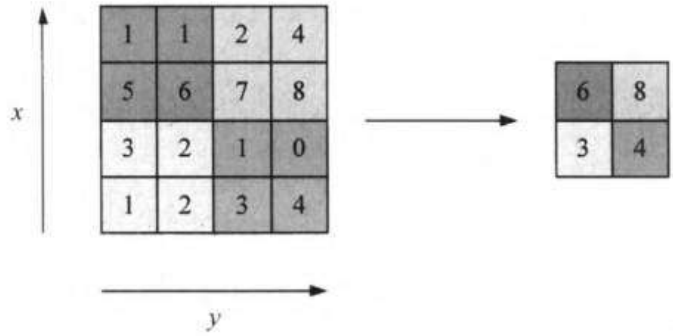


Рис. 6.23. Размеры матрицы до и после свертки

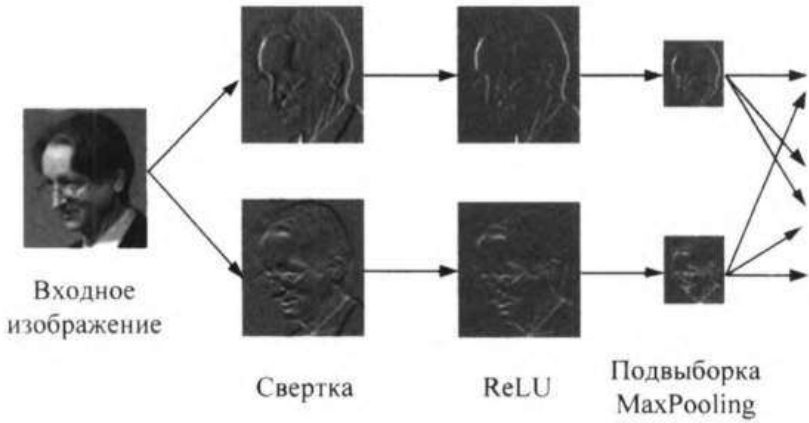


Рис. 6.24. Изменение характеристик изображения в процессе свертки

6.4.3. Строим сверточную нейронную сеть с библиотекой Keras

В этом разделе мы создадим сверточную нейронную сеть. Такие нейронные сети обычно используются для классификации изображений. Библиотека Keras, написанная на языке Python, позволяет создать CNN без каких-либо сложностей.

Компьютеры «видят» изображения как группы пикселов. Пикселы в изображениях, стоящие рядом и объединенные в группу, могут означать часть какого-то объекта или его определенной детали (линия, дуга). Сверточные нейронные сети используют этот принцип для распознавания изображений.

Сверточными сети называются из-за операции свертки, которая является основой всей сети. В этих сетях нет привычных связей и весовых коэффициентов. Вместо них используется ядро свертки размером от 3×3 до 7×7 пикселов. Ядро (kernel) перемещается по изображению и выделяет какой-то признак в картинке — например, переход от светлого пиксела к темному. Какой признак будет выделен, зависит от параметров ядра свертки. Например, в базе MNIST рукописная цифра — это картинка размером 28×28 пикселов (каждый пиксел имеет значения яркости от 0 до 255). По этой матрице мы проходим ядром и производим операцию свертки, после чего получаем слой свертки, или карту признака. Обычно слой свертки имеет тот же размер, но может быть большего или меньшего размера (рис. 6.25).

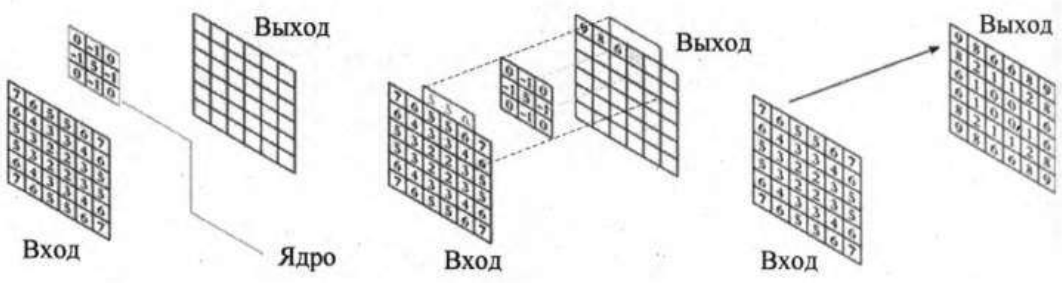


Рис. 6.25. Формирование слоя свертки ядром сверточной сети

Следующая операция — *пулинг*. Это как бы сжатие картинки или слоя свертки по максимуму или среднему значению, при этом группа пикселов (обычно размером 2×2) уплотняется до одного пиксела (рис. 6.26). Фактически мы увеличиваем область, которую захватывает ядро свертки, в два раза, переходя от мелких деталей изображения к более крупным. Также пулингом мы объединяем карты признаков (полученные сверткой) в более абстрактные признаки — уже не пикселы, а черточки, и т. д.

Для того чтобы увидеть, как выглядит тренировка нейронной сети при помощи библиотеки Keras, реализуем многослойную нейронную сеть для классификации рукописных цифр из набора данных MNIST. Набор данных MNIST можно скачать по адресу: **http://yann.lecun.com/exdb/mnist/**, но можно использовать и набор данных, который входит в состав библиотеки Keras. Его можно загрузить и посмотреть структуру данных с помощью следующего программного кода (листинг 6.27).

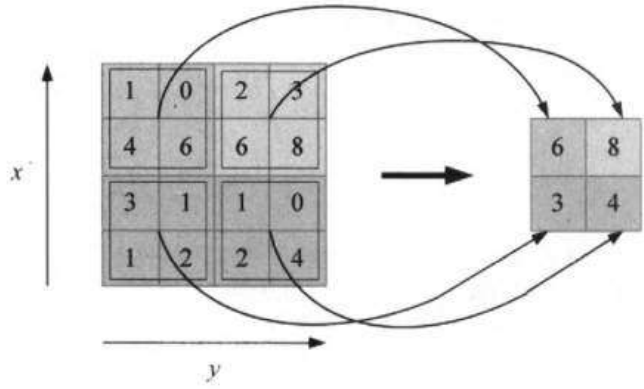


Рис. 6.26. Уплотнение изображения с помощью пуллинга

Листинг 6.27

```
from keras.datasets import mnist
import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print('Тренировочный набор данных', X_train.shape)
print('Метки тренировочного набора данных', y_train.shape)
print('Тестовый набор данных', X_test.shape)
print('Метки тестового набора данных', y_test.shape)
plt.imshow(X_train[1])
plt.show()
```

При работе этого программного кода на печать будет выведена следующая информация:

```
Тренировочный набор данных (60000, 28, 28)
Метки тренировочного набора данных (60000,)
Тестовый набор данных (10000, 28, 28)
Метки тестового набора данных (10000,)
```

Как можно видеть, в нашем обучающем наборе 60 тыс. изображений, и размер каждого из них составляет 28×28 пикселов. В тестовом наборе 10 тыс. аналогичных образцов. Можно просмотреть изображения из этого массива данных, указав индекс соответствующего элемента:

```
plt.imshow(X_train[1])
plt.show()
```

Результат работы этого фрагмента программного кода представлен на рис. 6.27.

Кроме самого изображения, можно посмотреть, в каком диапазоне располагаются значения пикселов в этом изображении (листинг 6.28).

Листинг 6.28

```
from keras.datasets import mnist
import matplotlib.pyplot as plt
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
plt.figure()
plt.imshow(X_train[1])
plt.colorbar()
plt.grid(False)
plt.show()
```

Результат, который мы увидим на экране, показан на рис. 6.28. Каждый пиксел в этом изображении может принимать значения в диапазоне от 0 до 250. В целом при работе с компьютерным зрением полезно визуально отобразить данные, прежде чем выполнять какую-либо работу алгоритма. Это поможет предотвратить ошибки.

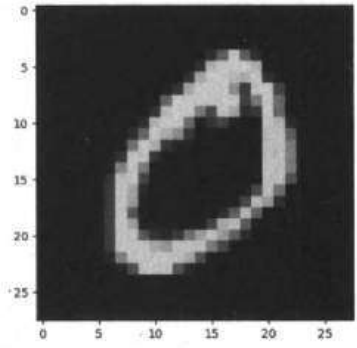


Рис. 6.27. Вид элемента `X_train[1]` из обучающего набора данных

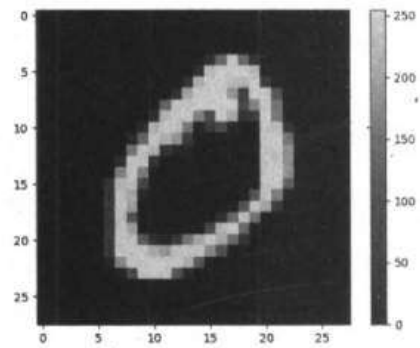


Рис. 6.28. Вид элемента `X_train[1]` из обучающего набора данных и шкала значений пикселов

Для того чтобы убедится, что мы работаем с нужными изображениями, выведем на экран первые 25 изображений цифр из тестового набора данных (листинг 6.29).

Листинг 6.29

```
from keras.datasets import mnist
import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = mnist.load_data()
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i], cmap=plt.cm.binary)
plt.show()
```

Результат работы этого программного модуля представлен на рис. 6.29.

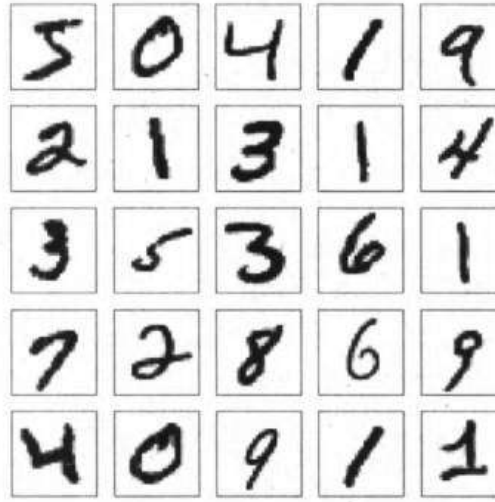


Рис. 6.29. Первые 15 изображений цифр из обучающего набора данных

Теперь нам нужно явно определить глубину цвета (или цветность). Например, полноцветное изображение с тремя каналами RGB будет иметь глубину 3. Наши изображения MNIST должны иметь глубину 1, и нам следует явно объявить это, для чего переформатировать исходные параметры базы данных (`X_train` и `X_test`) в ту форму, которая требуется для обучения модели. Выполним следующие команды:

```
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)
```

Здесь первое число — это количество изображений (`60000` для `X_train` и `10000` для `X_test`). Следом идет размер каждого изображения (28×28). Последнее число — `1` (глубина цвета), оно обозначает, что изображения черно-белые (градация серого).

Нам нужно также преобразовать метки в тренировочном и тестовом наборах данных. Вернемся к рис. 6.29 и посмотрим на изображения первых трех цифр. Это 5, 0, 4. Значит, эти же цифры присутствуют в первых трех элементах массива меток. Соответственно нам надо преобразовать значение каждой метки в массив из десяти элементов, состоящий из нулей и единиц. Положение единицы в этом массиве будет соответствовать значению метки. Такое преобразование приведено в табл. 6.6.

***Таблица 6.6.** Кодировка значений меток*

Значение метки	Кодировка метки	Значение метки	Кодировка метки
0	1000000000	5	0000010000
1	0100000000	6	0000001000
2	0010000000	7	0000000100
3	0001000000	8	0000000010
4	0000100000	9	0000000001

Выполнить это преобразование можно с помощью следующих команд:

```
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

До такого преобразования метки классов представляли собой одномерный массив 60 000 и 10 000 строк:

- метки тренировочного набора данных (60 000);
- метки тестового набора данных (10 000).

После преобразования метки классов будут представлены двумерными массивами, содержащими 10 столбцов нулей и единиц. Проверим это — распечатаем новую структуру массива меток и метки первых трех цифр (5, 0 и 4) с помощью команд из листинга 6.30.

Листинг 6.30

```
from keras.datasets import mnist
import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = mnist.load_data()
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print('Метки тренировочного набора данных после преобразования', y_train.shape)
print('Метки тестового набора данных после преобразования', y_test.shape)
y0 = y_train[0]
y1 = y_train[1]
y2 = y_train[2]
print(y0, y1, y2)
from keras.datasets import mnist
from keras.utils import to_categorical
```

В результате работы этого программного кода получим следующий вывод:

```
Метки тренировочного набора данных после преобразования (60000, 10)
Метки тестового набора данных после преобразования (10000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

Действительно, после преобразования метки стали двумерными массивами с десятью столбцами. Если обратиться к табл. 6.6, то мы увидим, что в первых трех элементах массива меток на самом деле закодированы цифры 5, 0 и 4.

Итак, мы загрузили и преобразовали тренировочный и тестовый наборы данных, а также провели некоторое исследование их структуры. Это исследование в большей степени преследует учебные цели и в меньшей степени касается построения модели нейронной сети. Теперь мы готовы определить архитектуру модели нашей нейронной сети. При этом мы не будем останавливаться на изучении различных вариантов

моделей обучения в библиотеке Keras, не станем вникать в теорию или математические основы. Мы просто воспроизведем уже проверенную на практике архитектуру для решения такого класса задач — последовательную модель Sequential. Такие модели в Keras строить проще всего. Этот метод позволяет выстраивать модель послойно. Для добавления слоев используем функцию `add()`. Итак, напишем следующий программный код (листинг 6.31).

Листинг 6.31

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D
from keras.utils import to_categorical
np.random.seed(123)

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
# Создание модели
model = Sequential()
# Первый сверточный слой
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
# Второй сверточный слой
model.add(Conv2D(32, kernel_size=3, activation='relu'))
# Создаем вектор для полносвязной сети
model.add(Flatten())
# Создадим однослойный персептрон
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=1)
print(hist.history)
```

Что мы задали в этом программном коде?

1. Первые два слоя — `Conv2D`. Эти сверточные слои будут работать с входными изображениями, которые рассматриваются как двумерные матрицы.
2. `64` и `32` — количество узлов в первом и втором слоях соответственно. Это количество можно увеличить или уменьшить в зависимости от размера базы данных. В нашем случае отлично подходят 64 и 32, поэтому эти параметры оставляем.

3. `kernel_size` (размер ядра) — размер матрицы фильтра для нейронной сети. Размер ядра `3` означает матрицу фильтров 3×3.
4. `activation` — функция активации для слоя. Для первых двух слоев мы будем использовать функцию активации ReLU (rectified linear activation). Она хорошо работает в нейронных сетях.
5. Первый слой принимает входную форму. Это форма каждого входного изображения: 28, 28, 1, как указывалось ранее. Число 1 (глубина цвета) здесь обозначает, что изображения черно-белые.
6. Между сверточными слоями `Conv2D` и слоем пулинга (`Dense`) находится слой выравнивания (`Flatten`). Он служит соединительным узлом между слоями. По сути, он преобразует двумерные массивы данных в одномерные.
7. Слой пулинга (`Dense`) мы будем использовать для выходных данных. Это стандартный тип слоя, который часто задействуется в нейронных сетях. По сути дела, это однослойный персептрон. В нашем выходном слое будет 10 узлов — по одному для каждого вероятного исхода (0–9).
8. `softmax` — это функция активации. Она сводит получившуюся сумму к 1, чтобы результат мог интерпретироваться как ряд возможных исходов. Тогда модель будет делать прогноз на основании того, какой вариант наиболее вероятен.
9. Далее нам нужно скомпилировать нашу модель: `model.compile()`.

   Для компиляции модели используются три параметра: оптимизатор (`optimizer`), потери (`loss`) и показатели (`metrics`):

   - оптимизатор контролирует скорость обучения. В качестве оптимизатора мы применим `'adam'`. Скорость обучения определяет, как быстро рассчитываются оптимальные веса для модели — при меньшей скорости весá будут определяться более точно (до определенного предела), но времени при этом будет затрачено больше;
   - для функции потерь мы будем использовать `'categorical_crossentropy'`. Это наиболее распространенный выбор для классификации. Более низкая оценка показывает, что модель работает лучше;
   - мы будем использовать метрику «точность» (`'accuracy'`), чтобы увидеть оценку точности при тестировании модели во время обучения.

Итак, все готово для обучения нашей модели нейронной сети. Для этого мы вызываем метод `fit()`, а историю обучения сохраним в переменной `hist`. В последних строках нашей программы присутствует следующий код:

```
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=1)
print(hist.history)
```

Здесь для обучения мы используем метод `fit()` со следующими параметрами:

- ❒ обучающая выборка данных или массив входных изображений цифр (`X_train`);
- ❒ целевые данные или правильные метки для входных изображений цифр (`y_train`);

- тестовая выборка данных или массив проверочных изображений цифр (`X_test`);
- целевые данные или правильные метки для тестовых изображений цифр (`y_test`);
- количество эпох или циклов обучения (`epochs`).

В процессе обучения модели будет проводиться и проверка ее успешности. Для проверки качества обучения мы используем набор тестовых изображений, который уже есть в базе данных (массивы `X_test` и `y_test`). Количество эпох — это количество циклов обработки моделью обучающих данных. Чем больше эпох мы задаем, тем тщательнее модель будет улучшаться. Но улучшение будет происходить до определенного уровня, а затем остановится. Для первого испытания нашей модели установим число эпох, равное 1. Итак, запускаем процесс обучения нашей нейронной сети. В процессе обучения будут выводиться промежуточные результаты обучения для каждой эпохи:

```
1875/1875 [==============================]
- 87s 46ms/step - loss: 0.2226
- accuracy: 0.9538
- val_loss: 0.0846
- val_accuracy: 0.9747
{'loss': [0.2225542515516281],
'accuracy': [0.9537666440010071],
'val_loss': [0.08457929641008377],
'val_accuracy': [0.9746999740600586]}
```

Даже после одной эпохи обучения при проверке на тестовых данных мы получаем показатель точности 97,47%, и для начала это хороший результат: мы построили и обучили первую сверточную нейронную сеть!

Результаты процесса обучения сети можно визуализировать, т. е. представить их в графическом виде. Для этого можно использовать следующий программный код (листинг 6.32).

Листинг 6.32

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D
from keras.utils import to_categorical
import matplotlib.pyplot as plt
np.random.seed(123)

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)
print(history.history)

# Построение графика точности предсказания
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Точность модели')
plt.ylabel('Точность')
plt.xlabel('Эпохи')
plt.legend(['Учебные', 'Тестовые'], loc='upper left')
plt.show()
# Построение графика потерь (ошибок)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Потери модели')
plt.ylabel('Потери')
plt.xlabel('Эпохи')
plt.legend(['Учебные', 'Тестовые'], loc='upper left')
plt.show()
```

Здесь мы запускаем процесс обучения сети (три эпохи), а потом строим графики зависимости точности предсказания и стоимости потерь (ошибок) от циклов обучения (эпох). Промежуточные результаты работы этой программы представлены на рис. 6.30.

Рис. 6.30. Вывод промежуточных результатов в процессе обучения сети

На построенных по итоговым результатам работы программы графиках показаны значения по двум массивам: по обучающей выборке (рис. 6.31) и по тестовой выборке данных (рис. 6.32).

После трех эпох обучения модель достигла следующей точности предсказания: на обучающей выборке — 98%, на тестовой — 97%. Это выше, чем при одном цикле обучения.

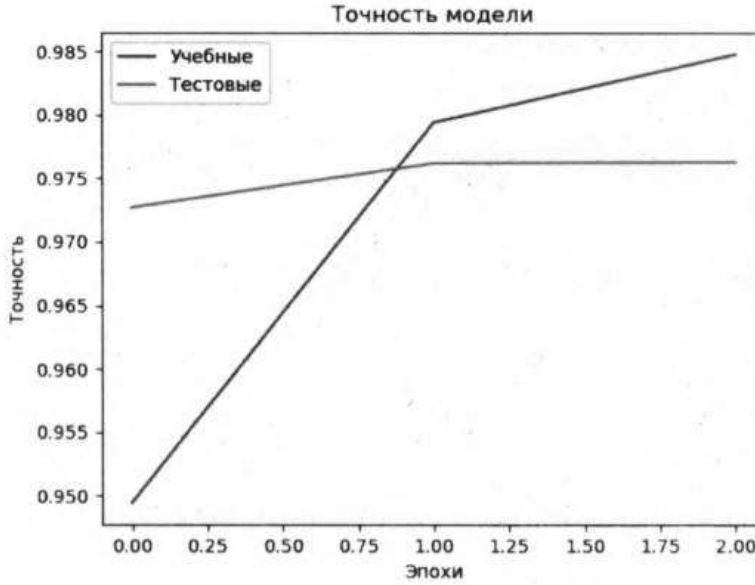


Рис. 6.31. График изменения точности в процессе обучения сети

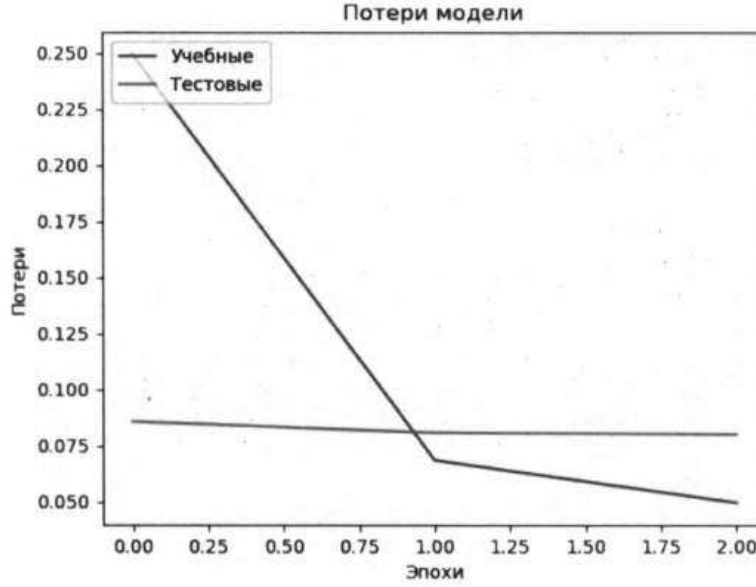


Рис. 6.32. График изменения потерь в процессе обучения сети

Однако еще не обеспечена сходимость модели. Точности прогноза по обучающей и тестовой выборке данных различаются.

Попробуем модернизировать нашу модель нейронной сети (листинг 6.33).

Листинг 6.33

```
# Это промежуточный код, он не является рабочим
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D())
model.add(Conv2D(128, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])
```

Здесь мы добавили дополнительный слой `MaxPooling2D`. Он нужен, чтобы уменьшить количество параметров в нашей модели, переместив фильтр пула 2×2 по предыдущему слою и взяв максимум 4 значения в фильтре 2×2. Количество узлов во втором слое увеличили до 128. Запустим процедуру обучения для пяти эпох и отобразим графики точности и потерь. Программный код теперь будет выглядеть следующим образом (листинг 6.34).

Листинг 6.34

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.utils import to_categorical
import matplotlib.pyplot as plt
np.random.seed(123)

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D())
model.add(Conv2D(128, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='sgd', loss='mean_squared_error',  metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5)
# Построение графика точности предсказания
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Точность модели')
plt.ylabel('Точность')
plt.xlabel('Эпохи')
plt.legend(['Учебные', 'Тестовые'], loc='upper left')
plt.show()
# Построение графика потерь (ошибок)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Потери модели')
plt.ylabel('Потери')
plt.xlabel('Эпохи')
plt.legend(['Учебные', 'Тестовые'], loc='upper left')
plt.show()
```

Результаты обучения представлены на рис. 6.33 и 6.34.

Как можно видеть на этих графиках, сходимость модели обеспечивается уже после четвертого цикла обучения. А на пятом цикле обучения точность модели по обучающей и тестовой выборкам практически одинакова. Такая сеть уже может использоваться на практике. Для того чтобы больше не заниматься тренировкой и тестированием сети, нужно как-то сохранить конфигурацию и параметры обученной сети. Это можно сделать с помощью следующего кода (листинг 6.35). При этом можно указать любой путь к папке, куда будет записана обученная модель.

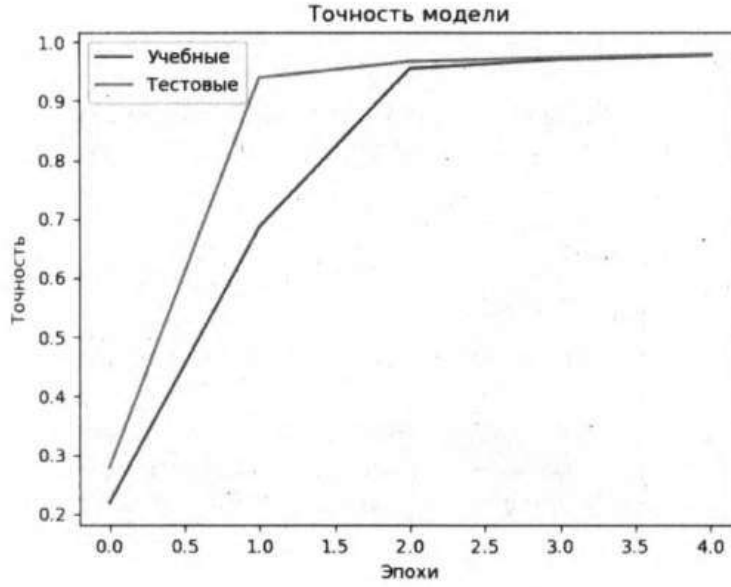


Рис. 6.33. График изменения точности в процессе обучения сети

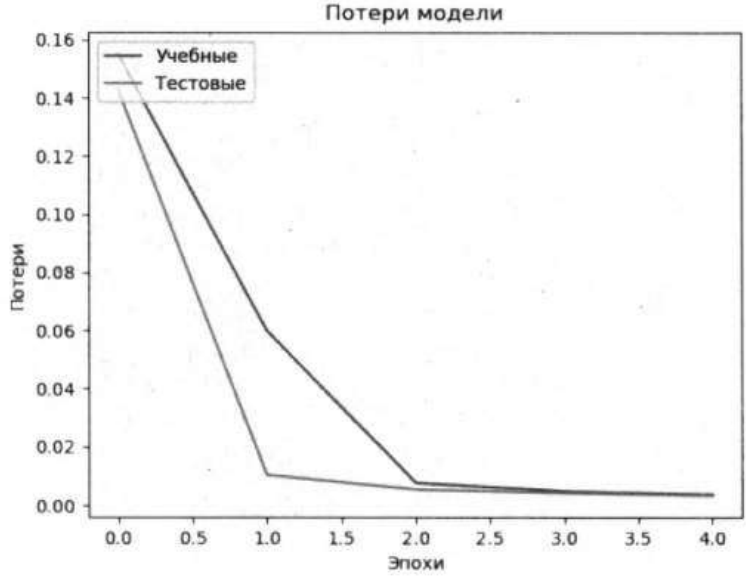


Рис. 6.34. График изменения потерь в процессе обучения сети

Листинг 6.35

```
# Это промежуточный код, он не является рабочим
from keras.models import load_model
# Запись обученной модели сети в файл 'my_model.h5
model.save('C:\mnist\my_model.h5')
# Удаление модели
del model
# Загрузка обученной модели сети из файла
model = load_model('C:\mnist\my_model.h5')
```

Для сохранения модели нейронной сети в файл применяют метод `model.save('Путь и имя файла')`. При этом используются файлы формата HDF5. Такой файл будет содержать:

- архитектуру модели нейронной сети;
- весовые коэффициенты модели, рассчитанные в процессе обучения;
- конфигурацию обучения (потеря, оптимизатор);
- состояние оптимизатора, позволяющее возобновить тренировку модели с того момента, где она была остановлена.

Чтобы предсказать метки классов, мы можем использовать метод `predict_classes()` — для возврата меток классов непосредственно в виде целых чисел. Теперь проверим нашу натренированную модель в деле — как она распознает цифры. Для этого возьмем первые три символа из обучающей выборки: 5, 0 и 4 (рис. 6.35), и напишем программный код, представленный в листинге 6.36 (с учетом того, что мы сохранили обученную модель в файле C:\mnist\my_model.h5).

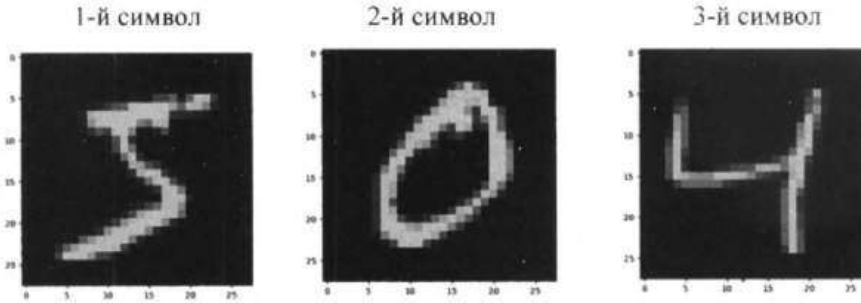


Рис. 6.35. Первые три символа из обучающего набора данных `X_train`

Листинг 6.36

```
# Это промежуточный код, он не является рабочим
# Загрузка обученной модели сети из файла
model_New = load_model('C:\mnist\my_model.h5')
y_train_pr = model_New.predict_classes(X_train[:3], verbose=0)
print('Первые 3 символа: ', y_train[:3])
print('Первые 3 предсказания: ', y_train_pr[:3])
```

В листинге 6.37 приведен полный текст программы этого раздела.

Листинг 6.37

```
import numpy as np
import theano
from keras.datasets import mnist
from matplotlib import pyplot as plt
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import SGD
from keras.utils import to_categorical
from keras.models import load_model
np.random.seed(123)

(X_train, y_train), (X_test, y_test) = mnist.load_data()
print('Тренировочный набор данных', X_train.shape)
print('Метки тренировочного набора данных', y_train.shape)
print('Тестовый набор данных', X_test.shape)
print('Метки тестового набора данных', y_test.shape)

plt.imshow(X_train[1])
plt.show()
```

```
plt.figure()
plt.imshow(X_train[1])
plt.colorbar()
plt.grid(False)
plt.show()

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i], cmap=plt.cm.binary)
plt.show()

X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print('Метки тренировочного набора данных после преобразования', y_train.shape)
print('Метки тестового набора данных после преобразования', y_test.shape)
y0 = y_train[0]
y1 = y_train[1]
y2 = y_train[2]
print(y0, y1, y2)

# Создание модели
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D())
model.add(Conv2D(128, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5)

# Построение графика точности предсказания
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Точность модели')
plt.ylabel('Точность')
plt.xlabel('Эпохи')
plt.legend(['Учебные', 'Тестовые'], loc='upper left')
plt.show()
```

```
# Построение графика потерь (ошибок)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Потери модели')
plt.ylabel('Потери')
plt.xlabel('Эпохи')
plt.legend(['Учебные', 'Тестовые'], loc='upper left')
plt.show()

# Запись обученной модели сети в файл my_model.h5
model.save('C:\mnist\my_model.h5')

# Удаление модели
del model

# Загрузка обученной модели сети из файла
model_New = load_model('./my_model.h5')y_predict =
np.argmax(model_New.predict(X_train[:3]), axis=-1)print('Первые 3 символа: ',
y_train[:3])print('Первые 3 предсказания: ', y_predict)
```

В результате работы этого программного кода получим следующий результат:

```
Первые 3 символа:
 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
Первые 3 предсказания: [5 0 4]
```

Как можно видеть, модель после обучения верно распознала все три цифры: 5, 0 и 4.

Отметим, что это всего лишь очень простая нейронная сеть без оптимизированных настроечных параметров. С моделью, построенной на основе Keras, можно экспериментировать: менять темп обучения, число скрытых узлов и т. п.

6.5. Нейронные сети с библиотекой TensorFlow

Если вы не пробовали TensorFlow в прежние времена, когда вам приходилось писать множество строк программного кода на чистом языке Python, то стоит попробовать поработать с этой библиотекой! Достаточно подключить TensorFlow к вашему проекту, и дальше можно взаимодействовать с ней из среды Python, сократив до минимума количество строк программного кода.

Библиотека TensorFlow работает в команде с другими полезными библиотеками — в частности, с Keras. Тандем TensorFlow + Keras значительно упрощает конструирование нейронных сетей. В Keras реализованы удобство и простота создания прототипов, а TensorFlow успешно использует эти качества. Если вы придерживаетесь объектно-ориентированного подхода в программировании и вам нравится строить нейронные сети слой за слоем, то вам понравится связка `tensorflow.keras`.

С помощью TensorFlow можно построить глубокие нейронные сети для распознавания образов и рукописного текста и рекуррентные нейронные сети для NLP (обработки естественных языков). В TensorFlow также включены модули для векторизации слов (embedding) и решения дифференциальных уравнений в частных производных (PDE). Этот фреймворк имеет отличную архитектурную поддержку, позволяющую с легкостью производить вычисления на самых разных платформах, в том числе на десктопах, серверах и мобильных устройствах.

Основной козырь TensorFlow — это абстракции. Они позволяют разработчикам сфокусироваться на общей логике приложения, а не на мелких деталях реализации тех или иных алгоритмов. С помощью этой библиотеки разработчики Python могут легко использовать искусственный интеллект и машинное обучение для создания уникальных адаптивных приложений, гибко реагирующих на пользовательские данные — например, на выражение лица или интонацию голоса. В этом издании книги использовалась библиотека TensorFlow версии 2.11.0.

6.5.1. Строим простую нейронную сеть с библиотекой TensorFlow

Рассмотрим простейшую нейронную сеть и научим ее выполнять функцию XOR. Разумеется, вычисление XOR с помощью нейронной сети не имеет практического смысла. Но именно оно поможет нам понять базовые принципы обучения и использования нейронной сети и проследить по шагам ее работу. С сетями большей размерности это было бы слишком сложно и громоздко. Мы уже рассматривали этот пример в *разд. 6.2.5*, но там это было сделано на PyBrain, а здесь мы применим связку TensorFlow + Keras.

Вспомним, что делает функция XOR. Таблица истинности для функции XOR была представлена ранее (см. табл. 6.1).

Итак, строим простейшую нейронную сеть. Сначала нужно подключить необходимые библиотеки: в нашем случае это TensorFlow, а также для работы с массивами нам понадобится библиотека NumPy (листинг 6.38).

Листинг 6.38

```
# Это промежуточный код, он не является рабочим
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
Activation, BatchNormalization, AveragePooling2D
from tensorflow.keras.optimizers import SGD, RMSprop, Adam
import tensorflow as tf
import logging
import numpy as np
```

Что ж, все библиотеки подключены, теперь мы готовы создать нейронную сеть. Благодаря Tensorflow на это понадобится всего лишь четыре строчки кода (листинг 6.39). Здесь пятой строкой просто выводится на печать структура нашей сети.

Листинг 6.39

```
# Это промежуточный код, он не является рабочим
model = Sequential()
model.add(Dense(2, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))
print(model.summary())
```

Здесь мы создали модель нейронной сети — класс `Sequential`, и добавили в нее два слоя: входной и выходной. Такая сеть называется *многослойным персептроном* (multilayer perceptron). В общем виде структура такой сети представлена на рис. 6.36.

В рассматриваемом случае для нашей сети были заданы следующие параметры: два входа (внешний слой), два нейрона во внутреннем слое и один выход. Это та часть нейронов, которые находятся выше пунктирной линии на рис. 6.36.

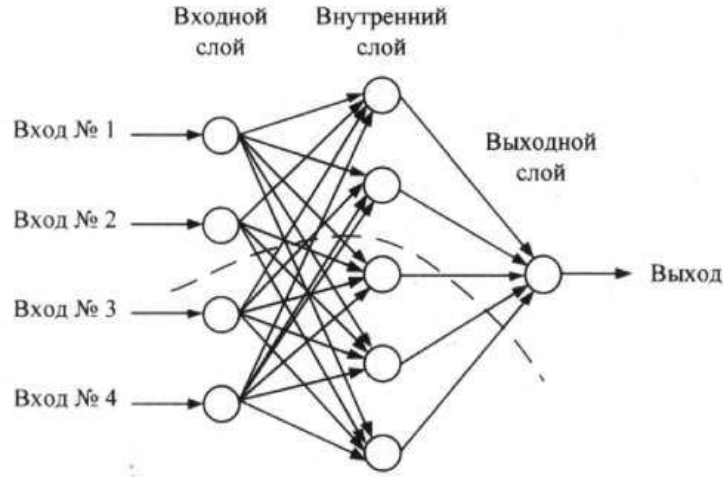


Рис. 6.36. Принципиальная структура нейронной сети — класс `Sequential`

Программа формирования модели нейронной сети приведена в листинге 6.40.

Листинг 6.40

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,\
    Activation, BatchNormalization, AveragePooling2D
from tensorflow.keras.optimizers import SGD, RMSprop, Adam
model = Sequential()
model.add(Dense(2, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.1))
print(model.summary())
```

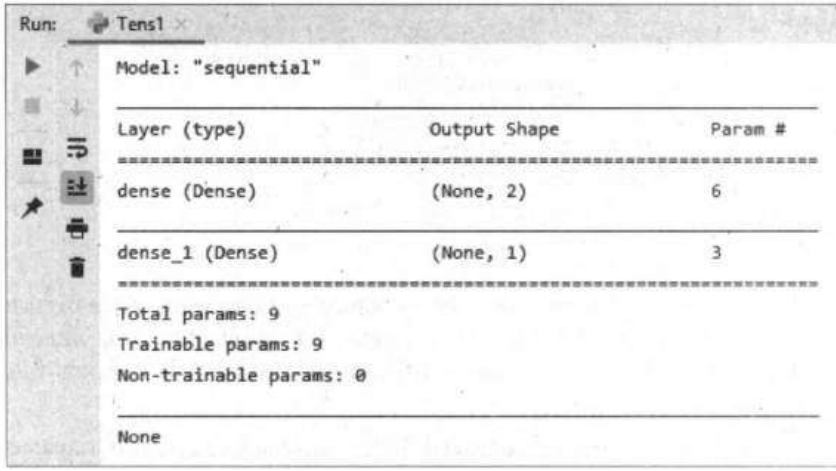


Рис. 6.37. Вывод структуры скомпонованной нейронной сети — класс `Sequential`

При запуске этой программы на экран будет выведена структура сети в виде, показанном на рис. 6.37.

Обучение будет заключаться в нахождении значений параметров этой сети. Как видно из рис. 6.37, наша сеть имеет девять параметров. Для того чтобы обучить ее, нам понадобится исходный набор данных, — в нашем случае это входные и выходные данные функции XOR, которые мы можем взять из табл. 6.1. Добавим в программу следующий код формирования обучающей выборки и запуска процесса обучения:

```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])
model.fit(X, y, batch_size=1, epochs=1000, verbose=0)
```

Здесь функция `fit()` запускает алгоритм обучения, которое у нас будет выполняться тысячу раз, причем на каждой итерации параметры сети будут корректироваться. Наша сеть небольшая, так что обучение пройдет быстро. После обучения сетью уже можно будет пользоваться. Проверим корректность выдаваемых сетью результатов с помощью следующего программного кода (листинг 6.41).

Листинг 6.41

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
import numpy as np
model = Sequential()
model.add(Dense(2, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.1))
print(model.summary())
```

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
model.fit(X, y, batch_size=1, epochs=1000, verbose=0)

print("Проверка работы обученной сети:")
print("XOR(0,0):", model.predict(np.array([[0, 0]])))
print("XOR(0,1):", model.predict(np.array([[0, 1]])))
print("XOR(1,0):", model.predict(np.array([[1, 0]])))
print("XOR(1,1):", model.predict(np.array([[1, 1]])))
```

В результате его работы получим:

```
Проверка работы обученной сети:
XOR(0,0): [[0.00708604]]
XOR(0,1): [[0.99853015]]
XOR(1,0): [[0.998531]]
XOR(1,1): [[0.00708604]]
```

Как можно видеть, обучение сети прошло вполне успешно — мы получили значения функции, достаточно близкие к требуемым значениям (0, 1, 1, 0). Можем также вывести значения найденных параметров сети (веса и смещения) для каждого уровня (листинг 6.42).

Листинг 6.42

```
# Это промежуточный код, он не является рабочим
# Параметры уровня 1
W1 = model.get_weights()[0]
b1 = model.get_weights()[1]
# Параметры уровня 2
W2 = model.get_weights()[2]
b2 = model.get_weights()[3]

print("W1:", W1)
print("b1:", b1)
print("W2:", W2)
print("b2:", b2)
```

Следует отметить, что алгоритм обучения, который используется в библиотеке Tensorflow, не идеален. Нейронные сети не всегда удается обучить за 1000 итераций с первого раза. Связка `tensorflow.keras` инициализирует начальные значения параметров сети случайными величинами, и при каждом запуске результаты могут различаться. Наша сеть с двумя нейронами успешно обучалась лишь в 20% случаев. Но это не страшно. Если видно, что нейронная сеть во время обучения не выдает правильных результатов, алгоритм обучения можно запустить еще раз. Правильно обученную сеть потом можно использовать без ограничений.

Можно сформировать структуру сети по-другому: использовать четыре нейрона вместо двух — для этого достаточно в листинге 6.42 заменить строку кода:

```
model.add(Dense(2, input_dim=2, activation='relu'))
```

на следующую:

```
model.add(Dense(4, input_dim=2, activation='relu'))
```

Такая сеть обучается уже в 60% случаев, а сеть из шести нейронов обучается с первого раза с вероятностью 90%. Все параметры нейронной сети полностью определяются коэффициентами. Обучив сеть, можно записать параметры сети на диск, а потом использовать уже готовую обученную сеть.

Полный текст программы формирования и обучения нейронной сети из шести нейронов представлен в листинге 6.43.

Листинг 6.43

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
import numpy as np

model = Sequential()
model.add(Dense(6, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.1))
print(model.summary())

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
model.fit(X, y, batch_size=1, epochs=1000, verbose=0)

print("Проверка работы обученной сети:")
print("XOR(0,0):", model.predict(np.array([[0, 0]])))
print("XOR(0,1):", model.predict(np.array([[0, 1]])))
print("XOR(1,0):", model.predict(np.array([[1, 0]])))
print("XOR(1,1):", model.predict(np.array([[1, 1]])))

# Параметры уровня 1
W1 = model.get_weights()[0]
b1 = model.get_weights()[1]
# Параметры уровня 2
W2 = model.get_weights()[2]
b2 = model.get_weights()[3]

print("W1:", W1)
print("b1:", b1)
print("W2:", W2)
print("b2:", b2)
```

В итоге мы получим достаточно уверенное предсказание:

```
Проверка работы обученной сети:
XOR(0,0): [[0.00294086]]
XOR(0,1): [[0.9986287]]
XOR(1,0): [[0.9983849]]
XOR(1,1): [[0.00135384]]
```

а также распечатку параметров двух уровней сети.

6.5.2. Строим нейронную сеть для классификации изображений с библиотекой TensorFlow

А теперь построим более продвинутую нейронную сеть, которая будет классифицировать изображения одежды и обуви (кроссовки, рубашки, брюки, кофты и т. д.). Для этой цели мы воспользуемся связкой двух библиотек: TensorFlow и Keras, а визуализацию результатов нам обеспечит библиотека matplotlib.

Обучение нашей сети мы выполним с помощью достаточно известного набора обучающих данных Fashion MNIST, который содержит 70 тыс. изображений в 10 категориях одежды (рис. 6.38). Каждое такое изображение представляет собой один предмет одежды в низком разрешении (28×28 пикселов).

В предыдущем разделе мы уже задействовали набор данных MNIST, который содержит изображения рукописных цифр (0, 1, 2, ..., 9). Набор данных Fashion MNIST также часто используют в примерах программ машинного обучения для компьютерного зрения. Форматы картинок в этих двух наборах данных идентичны. Оба набора данных относительно малы, поэтому их удобно применять в различных библиотеках для проверки корректности работы алгоритма. Это хорошие отправные точки для тестирования и отладки программного кода.

Набор данных Fashion MNIST содержит 60 тыс. изображений для обучения нейронной сети и 10 тыс. изображений для тестирования (чтобы проверить, насколько правильно сеть обучилась их классифицировать). Этот набор данных можно загрузить прямо из библиотеки TensorFlow, поскольку он входит в ее состав. Для этого используем следующие команды (листинг 6.44).

Листинг 6.44

```
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Процесс загрузки данных будет сопровождаться выводом следующих строк:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
```

```
32768/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/
                                        tf-keras-datasets/train-images-idx3-
ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
                                                  datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
                                                  datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step .
```

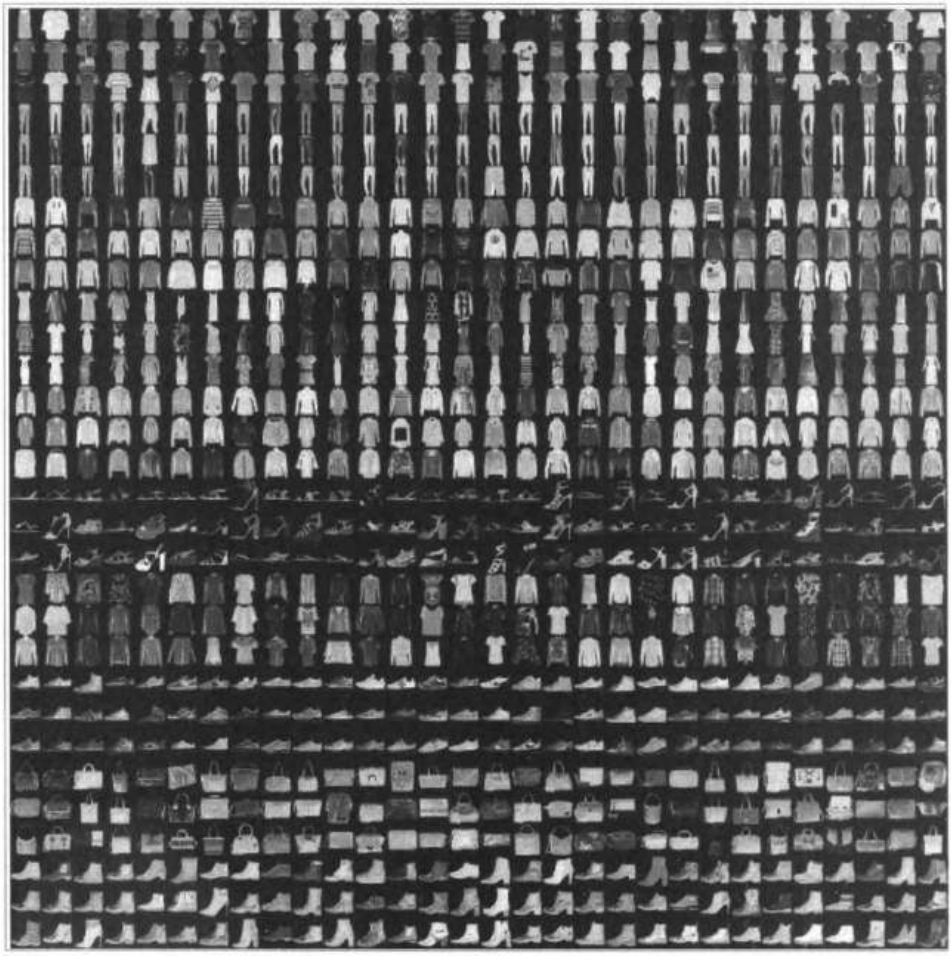


Рис. 6.38. Образцы изображений одежды из набора данных Fashion MNIST

При загрузке набора данных будет сформировано четыре массива NumPy:

- `train_images` — массив изображений тренировочного набора данных;
- `train_labels` — массив меток тренировочного набора данных;

- `test_images` — массив изображений тестового набора данных;
- `test_labels` — массив меток тестового набора данных.

Каждое изображение представляет собой матрицу пикселов размером 28×28 (массивы NumPy), где значение пикселов (цветность) варьирует от 0 до 255. Метки (labels) — это массив целых чисел от 0 до 9. Каждая метка соответствует одному из классов одежды, изображенной на картинках. Классы одежды приведены в табл. 6.7.

Таблица 6.7. *Классы одежды в наборе данных Fashion MNIST*

Label	Class	Класс одежды
0	T-shirt/top	Футболка/топ
1	Trouser	Брюки
2	Pullover	Пуловер
3	Dress	Платье
4	Coat	Пальто
5	Sandal	Сандаль
6	Shirt	Рубашка
7	Sneaker	Кроссовок
8	Bag	Сумка
9	Ankle boot	Полусапожки

Каждому изображению соответствует единственная метка. Так как названия классов не включены в набор данных, то сохраним их в отдельном массиве для дальнейшего использования при работе с изображениями с помощью следующей команды:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
               'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Перед обучением нашей модели можно посмотреть структуру массивов, содержащихся в обучающем и тренировочном наборах данных. Это можно сделать с помощью следующих команд (листинг 6.45).

Листинг 6.45

```
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Tr_Im = train_images.shape
Tr_label = len(train_labels)
```

```
Labels = train_labels
print('Тренировочный массив изображений', Tr_Im)
print('Тренировочный массив меток', Tr_label)
print('Метки изображений', Labels)

Test_Im = test_images.shape
Test_label = len(test_labels)
print('Тестовый массив изображений', Test_Im)
print('Тестовый массив меток', Test_label)
```

На экран будет выведена следующая информация:

```
Тренировочный массив изображений (60000, 28, 28)
Тренировочный массив меток 60000
Метки изображений [9 0 0 ... 3 0 5]
Тестовый массив изображений (10000, 28, 28)
Тестовый массив меток 10000
```

Из полученных данных видно, что в тренировочном наборе данных 60 тыс. изображений, каждое размером 28×28 пикселов. Соответственно в тренировочном наборе 60 тыс. меток. Каждая метка — это целое число от 0 до 9. Тестовый набор данных содержит 10 тыс. изображений, каждое размером 28×28 пикселов. В тестовом наборе данных 10 тыс. меток.

Можно посмотреть и на сами изображения, содержащиеся в наборе данных. Самую первую картинку из набора данных можно получить следующим образом (листинг 6.46).

Листинг 6.46

```
import matplotlib.pyplot as plt
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Tr_Im = train_images.shape
Tr_label = len(train_labels)
Labels = train_labels
print('Тренировочный массив изображений', Tr_Im)
print('Тренировочный массив меток', Tr_label)
print('Метки изображений', Labels)

Test_Im = test_images.shape
Test_label = len(test_labels)
print('Тестовый массив изображений', Test_Im)
print('Тестовый массив меток', Test_label)
```

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

На экран будет выведено следующее изображение (рис. 6.39).

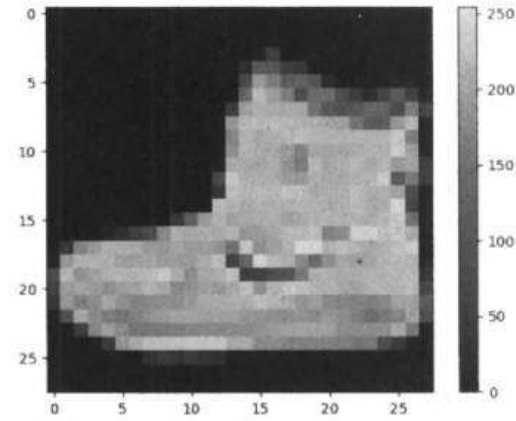


Рис. 6.39. Образец одежды `train_images[0]` из набора данных Fashion MNIST

Здесь видно, что значения пикселов (цветность) находятся в диапазоне от 0 до 255. Для упрощения вычислений мы масштабируем эти значения к диапазону от 0 до 1, перед тем как передать их в нейронную сеть. Для этого мы поделим значения всех пикселов на 255. Важно, чтобы обучающий и тестовый наборы данных были преобразованы одинаково. Такое преобразование для обучающей и тестовой выборок можно сделать с помощью следующих команд:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Можно снова посмотреть, как изменились значения пикселов после преобразования, для чего повторим выполнение команд из листинга 6.47.

Листинг 6.47

```
import matplotlib.pyplot as plt
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Tr_Im = train_images.shape
Tr_label = len(train_labels)
```

```
Labels = train_labels
print('Тренировочный массив изображений', Tr_Im)
print('Тренировочный массив меток', Tr_label)
print('Метки изображений', Labels)

Test_Im = test_images.shape
Test_label = len(test_labels)
print('Тестовый массив изображений', Test_Im)
print('Тестовый массив меток', Test_label)

train_images = train_images / 255.0
test_images = test_images / 255.0
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

Результат работы этого программного кода представлен на рис. 6.40.

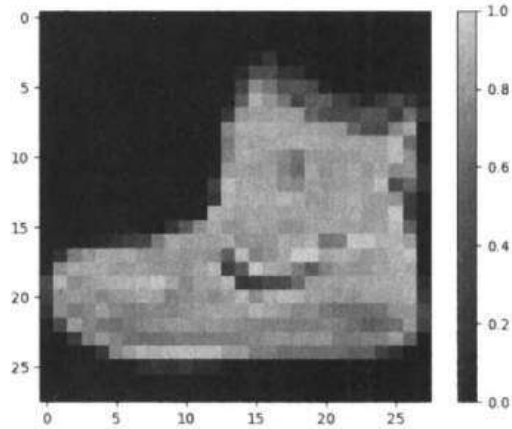


Рис. 6.40. Образец одежды `train_images[0]` из набора данных Fashion MNIST после преобразования значений шкалы пикселов

Убедимся, что все данные в правильном формате и мы готовы построить и обучить нейронную сеть, для чего выведем на экран первые 25 изображений из тестового набора данных и отобразим под ними наименования их классов (листинг 6.48).

Листинг 6.48

```
import matplotlib.pyplot as plt
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Tr_Im = train_images.shape
Tr_label = len(train_labels)
Labels = train_labels
print('Тренировочный массив изображений', Tr_Im)
print('Тренировочный массив меток', Tr_label)
print('Метки изображений', Labels)

Test_Im = test_images.shape
Test_label = len(test_labels)
print('Тестовый массив изображений', Test_Im)
print('Тестовый массив меток', Test_label)
train_images = train_images / 255.0
test_images = test_images / 255.0
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Результат работы этого программного кода представлен на рис. 6.41.

В принципе, мы получили то, что хотели, — монохромные изображения. Теперь можно перейти к конфигурированию нейронной сети. Построение модели нейронной сети требует правильной конфигурации каждого слоя и последующей компиляции модели.

Базовым строительным блоком нейронной сети является *слой*. Слои извлекают образы из данных, которые в них подаются. Большая часть глубокого обучения состоит из соединения в последовательность простых слоев. Большинство слоев, таких как `tf.keras.layers.Dense`, имеют параметры, которые настраиваются во время обучения. Создадим в нашей модели нейронной сети три слоя с помощью следующих команд (листинг 6.49).

Листинг 6.49

```
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Tr_Im = train_images.shape
Tr_label = len(train_labels)
```

Рис. 6.41. Первые 25 изображений из тестового набора данных Fashion MNIST

```
Labels = train_labels
print('Тренировочный массив изображений', Tr_Im)
print('Тренировочный массив меток', Tr_label)
print('Метки изображений', Labels)

Test_Im = test_images.shape
Test_label = len(test_labels)
print('Тестовый массив изображений', Test_Im)
print('Тестовый массив меток', Test_label)
train_images = train_images / 255.0
test_images = test_images / 255.0
# Модель нейронной сети
model = keras.Sequential([
  keras.layers.Flatten(input_shape=(28, 28)),
  keras.layers.Dense(128, activation='relu'),
  keras.layers.Dense(10, activation='softmax')
])
```

```
# Компиляция модели нейронной сети
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Тренировка (обучение) модели
model.fit(train_images, train_labels, epochs=10)
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nТочность на проверочных данных:', test_acc)
```

Первый слой этой сети: `tf.keras.layers.Flatten` — преобразует формат изображения из двумерного массива (28×28 пикселов) в одномерный массив (размером $28 \times 28 = 784$ пиксела). Слой извлекает строки пикселов из изображения и выстраивает их в один ряд. Этот слой не имеет параметров для обучения — он только переформатирует входные данные.

После разложения пикселов в один ряд нейронная сеть передает их в следующие два слоя `tf.keras.layers.Dense`. Это полносвязные нейронные слои. Первый Dense-слой состоит из 128 узлов (или нейронов). Второй (и последний) 10-узловой softmax-слой возвращает массив из 10 вероятностных оценок, дающих в сумме 1. Каждый узел содержит оценку, указывающую вероятность принадлежности изображения к одному из 10 классов.

Прежде чем модель будет готова для обучения, нам нужно указать еще несколько параметров. Они добавляются на шаге компиляции модели (compile). Здесь указываются:

- функция потерь (loss function) — измеряет точность модели во время обучения. Мы хотим минимизировать эту функцию, чтобы «направить» модель в верном направлении;
- оптимизатор (optimizer) — показывает, каким образом обновляется модель на основе входных данных и функции потерь;
- метрики (metrics) — используются для мониторинга тренировки и тестирования модели.

Наш пример использует метрику `accuracy`, равную доле правильно классифицированных изображений. Все эти параметры задаются в такой строке программного кода:

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Обучение модели нейронной сети требует выполнения следующих шагов:

1. Нужно подать тренировочные данные в модель. В этом примере тренировочные данные — это массивы `train_images` и `train_labels`.
2. Далее модель учится ассоциировать изображения с правильными классами.
3. Модель тестируется на проверочных данных. В этом примере проверочные данные — это массив `test_images`. Мы проверяем, соответствуют ли предсказанные классы меткам из массива `test_labels`.

Для начала обучения вызывается метод `model.fit()`, который называется так, поскольку «тренирует» (fits) модель на тренировочных данных. Это делается следующей командой:

```
model.fit(train_images, train_labels, epochs=10)
```

Можно посмотреть, какую точность модель покажет на тренировочных данных. В листинге 6.49 это последние две команды:

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nТочность на проверочных данных:', test_acc
```

Запустим программу листинга 6.49. В процессе обучения модели на экран выводятся промежуточные результаты тренировки (рис. 6.42).

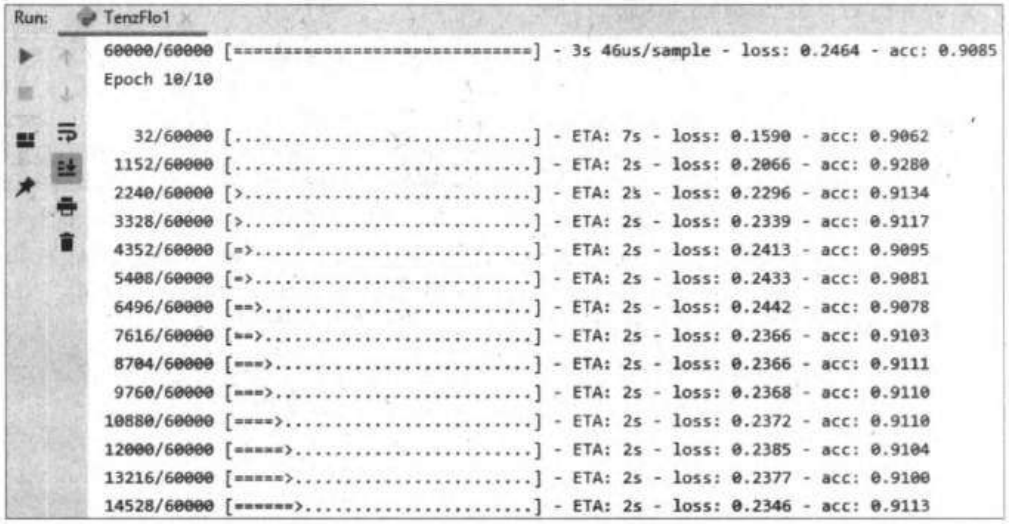


Рис. 6.42. Промежуточные результаты тренировки нейронной сети

В процессе обучения модели отображаются метрики потери (loss) и точности (accuracy). Эта модель достигает на тренировочных данных точности, равной приблизительно 0,88 (88%).

Полученная на тестовых данных точность оказалась немного ниже, чем на тренировочном наборе данных. Этот разрыв между точностью на тренировке и тесте является примером *переобучения* (overfitting). Переобучение возникает тогда, когда модель машинного обучения показывает на новых данных худший результат, чем на тех, на которых она обучалась.

Теперь, когда модель прошла курс обучения, мы можем использовать ее для того, чтобы сделать предсказания по поводу нескольких изображений. Дадим команду модели предсказать класс одежды для каждого изображения в тестовом наборе данных, а потом выведем на печать предсказание для первого изображения (оно имеет нулевой индекс в массиве данных). Это можно сделать следующими командами (листинг 6.50).

Листинг 6.50

```
# Это промежуточный код, он не является рабочим
import numpy as np
predictions = model.predict(test_images)
ver1 = predictions[0]
im1 = np.argmax(predictions[0])
lab1 = test_labels[0]
print('Вероятность предсказаний для первого рисунка', ver1)
print('Первое изображение (после обучения)', im1)
print('Метка первого изображения', lab1)
```

Объединим программу из листинга 6.49 с этими строками из листинга 6.51. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_50_1) в сопровождающем книгу файловом архиве (см. *приложение*). В результате работы этого объединенного программного кода получим следующий результат:

```
Вероятность предсказаний для первого рисунка
[0.000 0.000 0.000 0.000 0.000 0.002 0.000 0.007 0.000 0.990]
Первое изображение (после обучения) 9
Метка первого изображения 9
```

Здесь прогноз представлен в виде массива из 10 чисел. Каждое число характеризует вероятность того, что поданное на вход изображение соответствует каждому из 10 разных видов одежды. Из полученных данных видно, что первое изображение в массиве обучающих данных с наибольшей вероятностью (99%) соответствует десятому виду одежды (с меткой 9). А это у нас в соответствии с табл. 6.9 — Ankle boot (полусапожки).

Мы можем написать две функции, которые будут визуально отображать результаты предсказания для любого элемента из массива предсказаний. Программный код этих функций представлен в листинге 6.51.

Листинг 6.51

```
# Это промежуточный код, он не является рабочим
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
```

```
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                100 * np.max(predictions_array),
                class_names[true_label]),
                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

С использованием этих функций верные предсказания будут отображаться синим цветом, ошибочные — красным. Теперь применим указанные функции для визуализации предсказания для первого изображения (оно имеет нулевой индекс в массиве данных). Это можно сделать следующими командами (листинг 6.52).

Листинг 6.52

```
# Это промежуточный код, он не является рабочим
i = 0
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1, 2, 2)
plot_value_array(i, predictions, test_labels)
plt.show()
```

Отметим, что модель может и ошибаться, причем ошибаться очень уверенно. В этом можно убедиться, если визуализировать предсказание для 12-го изображения в массиве входных данных (листинг 6.53).

Листинг 6.53

```
# Это промежуточный код, он не является рабочим
i = 12
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plot_image(i, predictions, test_labels, test_images)
```

```
plt.subplot(1, 2, 2)
plot_value_array(i, predictions, test_labels)
plt.show()
```

Теперь объединим несколько программ в один модуль: к листингу 6.49 добавим строки из листингов 6.51–6.53. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_53_1) в сопровождающем книгу файловом архиве (см. *приложение*). В результате работы этого программного кода получим результат, представленный на рис. 6.43 и 6.44. Напомним, что наши функции визуализируют верные прогнозы синим цветом, а ошибочные прогнозы — красным.

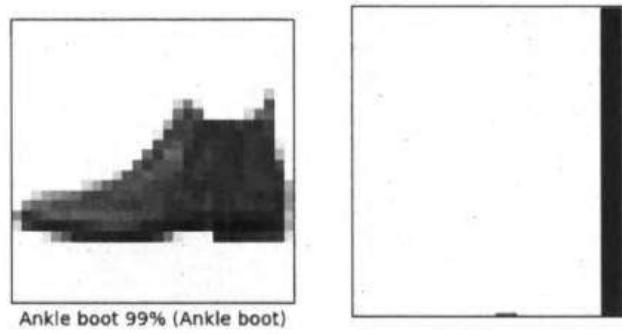


Рис. 6.43. Визуализация прогноза нейронной сети на изображение с индексом `i=0` в массиве данных

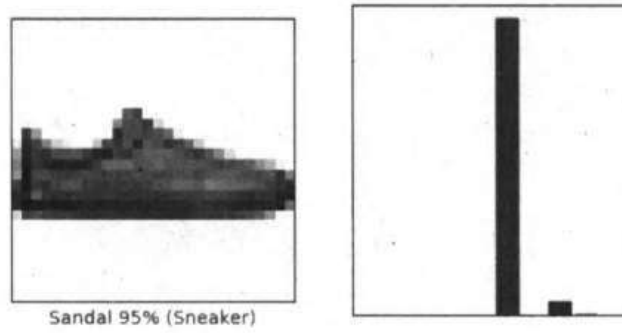


Рис. 6.44. Визуализация прогноза нейронной сети на изображение с индексом `i=12` в массиве данных

Как можно видеть, для изображения, показанного на рис. 6.43, *слева*, был сделан уверенный прогноз (вероятность правильного решения 99%). Это подтверждено и на правой части рисунка, где вероятность правильного решения характеризуется высотой столба синего цвета для 9-й метки одежды. Высота столба — это процент уверенности от 0 до 100.

А вот рис. 6.44 показывает, что после обучения наша сеть для изображения с индексом `i=12` с вероятностью 95% приняла неправильное решение. В этом случае требуется либо изменить конфигурацию сети, либо провести дополнительное обучение созданной модели нейронной сети.

Можно визуализировать прогноз нейронной сети для группы изображений. Следующий программный код позволяет вывести информацию о вероятности правильного предсказания для первых 15 изображений из нашего обучающего набора данных (листинг 6.54).

Листинг 6.54

```
# Это промежуточный код, он не является рабочим
# Отображаем первые X тестовых изображений, их предсказанную и настоящую метки.
# Корректные предсказания окрашиваем в синий цвет, ошибочные – в красный.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
plt.show()
```

Этот программный код нужно добавить к листингу 6_53_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_54_1) в сопровождающем книгу файловом архиве (см. *приложение*). Результаты работы этого объединенного программного кода представлены на рис. 6.45. После дополнительного обучения сеть начала давать правильный прогноз и для 12-го изображения из обучающего набора данных.

Наконец, используем нашу обученную модель нейронной сети для предсказания класса на одном изображении из тестового набора данных (листинг 6.55).

Листинг 6.55

```
# Это промежуточный код, он не является рабочим
# Берем одну картинку из тестового набора данных
img = test_images[0]
print(img.shape)

# Добавляем изображение в пакет данных, состоящий только из одного элемента
img = (np.expand_dims(img, 0))
print(img.shape)
predictions_single = model.predict(img)
print('Проверка на изображении из тестового набора данных')
print(predictions_single)
met = np.argmax(predictions_single[0])
print('Метка класса одежды', met)
plot_value_array(0, predictions_single, test_labels)
```

```
plt.xticks(range(10), class_names, rotation=45)
plt.show()
```

Этот программный код нужно добавать к листингу 6_54_1. Вы можете сделать это самостоятельно, а можете найти объединенный модуль (листинг 6_55_1) в сопровождающем книгу файловом архиве (см. *приложение*). После работы этого фрагмента программного кода получим следующий результат:

```
(28, 28)
(1, 28, 28)
Проверка на изображении из тестового набора данных
[[2.3437217e-06 3.7535163e-08 2.0439505e-07 2.6200818e-08 1.9223181e-07
 1.3369562e-03 8.9684036e-06 2.0042850e-02 1.2525952e-05 9.7859585e-01]]
Метка класса одежды 9
```

Рис. 6.45. Визуализация прогноза нейронной сети на первые 15 изображений из обучающего набора данных

Приведем полученные итоги вероятностей из экспоненциального вида к десятичному:

```
[[0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.002 0.000 0.978]]
```

Как видно из этих результатов, с вероятностью 97% выбранное изображение относится к классу одежды с меткой 9.

Эти результаты также визуализируются с помощью следующего фрагмента программного кода:

```
plot_value_array(0, predictions_single, test_labels)
plt.xticks(range(10), class_names, rotation=45)
plt.show()
```

Результаты работы этого фрагмента кода представлены на рис. 6.46.

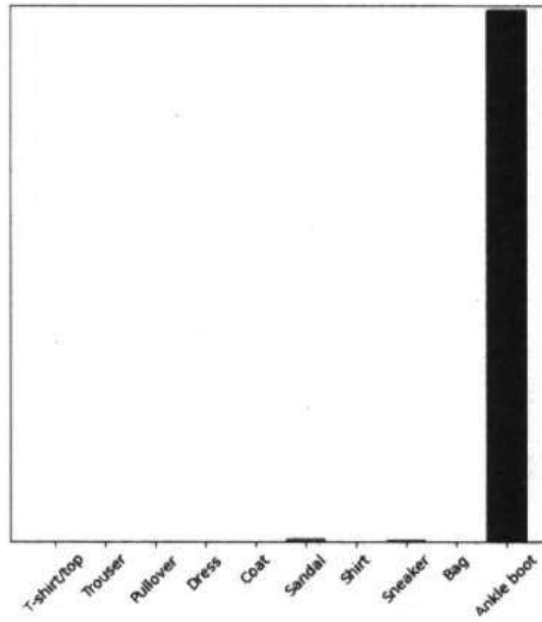


Рис. 6.46. Визуализация прогноза нейронной сети на первое изображение из тестового набора данных

Полный текст программы, о которой шла речь в этом разделе, приведен в листинге 6.56.

Листинг 6.56

```
# Модуль Tens3
import tensorflow as tf
from tensorflow import keras
# Вспомогательные библиотеки
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
Tr_Im = train_images.shape
Tr_label = len(train_labels)
Labels = train_labels
print('Тренировочный массив изображений', Tr_Im)
print('Тренировочный массив меток', Tr_label)
print('Метки изображений', Labels)

Test_Im = test_images.shape
Test_label = len(test_labels)
print('Тестовый массив изображений', Test_Im)
print('Тестовый массив меток', Test_label)

plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

train_images = train_images / 255.0
test_images = test_images / 255.0

plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

model = keras.Sequential([
  keras.layers.Flatten(input_shape=(28, 28)),
  keras.layers.Dense(128, activation='relu'),
  keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nТочность на проверочных данных:', test_acc)

predictions = model.predict(test_images)
ver1 = predictions[0]
im1 = np.argmax(predictions[0])
lab1 = test_labels[0]
print('Вероятность предсказаний для первого рисунка', ver1)
print('Первое изображение (после обучения)', im1)
print('Метка первого изображения', lab1)

def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
       color = 'blue'
    else:
       color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                100 * np.max(predictions_array),
                class_names[true_label]),
                color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')

i = 0
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1, 2, 2)
```

```
plot_value_array(i, predictions, test_labels)
plt.show()

i = 12
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1, 2, 2)
plot_value_array(i, predictions, test_labels)
plt.show()

# Отображаем первые X тестовых изображений, их предсказанную и настоящую метки.
# Корректные предсказания окрашиваем в синий цвет, ошибочные — в красный.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
plt.show()

# Берем одну картинку из тестового набора данных.
img = test_images[0]
print(img.shape)

# Добавляем изображение в пакет данных, состоящий только из одного элемента.
img = (np.expand_dims(img, 0))
print(img.shape)
predictions_single = model.predict(img)
print('Проверка на изображении из тестового набора данных')
print(predictions_single)
met = np.argmax(predictions_single[0])
print('Метка класса одежды', met)

plot_value_array(0, predictions_single, test_labels)
plt.xticks(range(10), class_names, rotation=45)
plt.show()
```

6.6. Краткие итоги главы

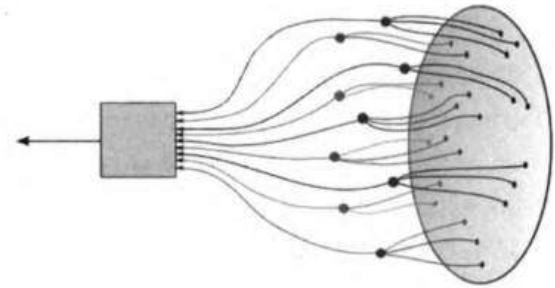
В этой главе мы познакомились с некоторыми работающими вместе с Python популярными библиотеками, которые позволяют решать достаточно большой класс практических задач и значительно облегчают и упрощают труд программистов.

При этом для обучения нейронных сетей были использованы уже готовые наборы обучающих данных. Это позволило в большей степени сконцентрироваться на алгоритмах обучения, не теряя время и силы на формирование самих наборов данных.

Однако можно еще больше упростить создание программных средств, связанных с искусственным интеллектом, — использовать не только готовые наборы данных, но и уже готовые, обученные модели нейронных сетей. Кроме того, преимущество нейронных сетей состоит в том, что одна и та же модель сети может решать совершенно разные задачи. Это будет зависеть от того, чему ее обучали. Поэтому важно не только научиться проектировать нейронные сети, но и готовить для них обучающие наборы данных.

Следующая глава посвящена как раз практике использования такой библиотеки — ImageAI. С одной стороны, она содержит модели сетей, уже обученных на распознавание в изображениях достаточно широкого класса объектов, а с другой — эти модели можно научить распознавать любые объекты пользователя.

ГЛАВА 7



Создание нейронных сетей обработки изображений: библиотека ImageAI

ImageAI — это пользующаяся большой популярностью библиотека Python, способная дать разработчикам, исследователям и студентам возможность создавать приложения и системы в области глубокого обучения (deep learning) и компьютерного зрения (computer vision), используя при этом небольшое количество строк простого программного кода. В этой главе мы познакомимся с большинством классов, доступных в ImageAI, и приведем ряд примеров их использования.

Для работы с ImageAI требуется установить Python версии 3.5.1 или выше, а также некоторые другие библиотеки. Первое важное замечание: для работы с ImageAI нужно создать отдельный проект со своей виртуальной средой. Если вы попытаетесь загрузить ImageAI в проект с другими модулями, то велика вероятность, что старые программы перестанут работать, т. к. будут заменены версии полутора десятков библиотек. Второе важное замечание: нужно строго соблюдать, в какой последовательности и какие версии библиотек вы загружаете. В частности, загрузку библиотек для ImageAI требуется выполнять в следующей последовательности:

1. Обновить модуль pip: `easy_install -U pip`;
2. Установить библиотеку tensorflow: `pip install tensorflow==2.4.0`;
3. Установить библиотеку tensorflow-gpu: `pip install tensorflow-gpu==2.4.0`;
4. Установить другие зависимости:

```
pip install keras==2.4.3 numpy==1.19.3 pillow==7.0.0 scipy==1.4.1 h5py==2.10.0
matplotlib==3.3.2 opencv-python keras-resnet==0.2.0
```

5. И на последнем этапе установить ImageAI:

```
pip install imageai --upgrade
```

На момент подготовки этого издания книги с указанным пакетом корректно работала библиотека ImageAI версии 2.1.6.

С помощью ImageAI можно с минимальным количеством программного кода создавать серьезные приложения из области компьютерного зрения. В частности:

- обнаруживать 80 наиболее распространенных предметов быта в изображениях;
- обнаруживать 80 наиболее распространенных предметов быта в видеофайлах;
- получать информацию о времени обнаружения объектов в видеопотоках;
- обучать модели нейронных сетей распознавать пользовательские объекты;
- обучать новые модели YOLOv3 для обнаружения пользовательских объектов.

7.1. Классы распознавания и обнаружения объектов на изображениях

Библиотека ImageAI предоставляет очень мощные, но простые в использовании классы для выполнения задач распознавания изображений и поиска в них ряда объектов. Вы сможете решать наиболее распространенные задачи компьютерного зрения, написав всего от 5 до 12 строк программного кода Python. Когда вы установите Python, набор дополнительных библиотек и ImageAI, у вас появится возможность создавать самые невероятные приложения. В этом разделе приведено описание основных классов и соответствующих методов (функции внутри классов), которые позволяют быстро и просто создавать эффективные приложения. Эти классы могут быть интегрированы в любую разрабатываемую программу на Python, будь то вебсайт, приложение для Windows, Linux или macOS.

7.1.1. Распознавание объектов в изображениях: класс *ImageClassification*

Класс `ImageClassification` предоставляет набор функций для использования в современных моделях распознавания изображений — таких как MobileNetV2, ResNet50, InceptionV3 и DenseNet121, которые были *предварительно обучены* на наборе данных ImageNet-1000. Это означает, что вы можете использовать этот класс для прогнозирования или распознавания 1000 различных объектов в любом изображении или в любом количестве изображений. Указанные модели можно скачать с официального сайта по следующей ссылке: **https://imageai.readthedocs.io/en/latest/prediction/index.html**.

Имена файлов, содержащих соответствующие модели, и их размеры приведены в табл. 7.1.

***Таблица 7.1.** Имена файлов, содержащих соответствующие модели обученных нейронных сетей*

Наименование модели	Наименование файла	Размер, Кбайт
MobileNetV2	mobilenet_v2.h5	14 196
ResNet50	resnet50_imagenet_tf.2.0.h5	100 555
InceptionV3	inception_v3_weights_tf_dim_ordering_tf_kernels.h5	93 860
DenseNet121	DenseNet-BC-121-32.h5	32 411

Рассмотрим характеристики упомянутых здесь моделей нейронных сетей.

- **MobileNetV2**

  Нейронная сеть MobileNetV2 представляет собой весьма эффективную модель, ориентированную на мобильные устройства, которую можно использовать в качестве основы для многих задач визуального распознавания. Появление MobileNet уже само по себе совершило революцию в компьютерном зрении на мобильных платформах, однако MobileNetV2 — следующее поколение нейросетей этого семейства — позволило достигать примерно такой же точности распознавания при еще большей скорости работы. Например, при обнаружении объектов новая модель работает примерно на 35% быстрее, чем MobileNetV1 при той же точности.

  С MobileNetV2 мобильные разработчики получили почти неограниченный инструментарий в области компьютерного зрения. Помимо относительно простых моделей для классификации изображений, теперь можно использовать алгоритмы детектирования объектов и семантической сегментации на мобильных устройствах. Это очень эффективное средство для извлечения признаков в процессе обнаружения и сегментации объектов. При этом использовать MobileNet с помощью Keras и TensorFlow настолько просто, что разработчики могут делать это, даже не углубляясь во внутреннее устройство алгоритмов.

- **ResNet50**

  Глубокие сверточные нейронные сети (CNN) показали достаточно высокий уровень классификации изображений. Что же произойдет с нейронной сетью, когда мы существенно увеличим число слоев? Скорее всего, более глубокая нейронная сеть покажет даже худшие результаты как при обучении, так и при тестировании. Когда более глубокая сеть начинает сворачиваться, возникает проблема: с увеличением глубины сети точность сначала увеличивается, а затем быстро ухудшается.

  Создатели ResNet предположили, что проблема здесь кроется в оптимизации — более глубокие модели гораздо хуже поддаются настройке. Тогда они решили не накладывать слои друг на друга для изучения отображения нужной функции напрямую, а использовать остаточные блоки, которые пытаются «подогнать» это отображение. Так ResNet стала первой *остаточной* нейронной сетью. ResNet — это сокращенное название от Residual Network (дословно «остаточная сеть»), а residual learning — это остаточное обучение. Говоря простыми словами, такая сеть в процессе работы при определенных ситуациях «перепрыгивает» через некоторые слои. За счет этого сеть ResNet сходится быстрее, чем ее простой аналог.

  ResNet50 — это CNN, которая имеет 50 основных слоев (сверточные + полносвязные). Она была разработана в Microsoft в 2015 году для решения задачи распознавания изображений. Эта модель Machine Learning обучена на более чем миллионе изображений из базы данных ImageNet. Как и предыдущие модели, ResNet50 может классифицировать до 1000 объектов.

- **Inceptionv3**

  Это сверточная нейронная сеть для анализа изображений и обнаружения в них различных объектов. Она представляет собой развитие модели нейронной сети GoogleNet — по сути, это третья версия модели Google Inception Convolutional Neural Network. Модель Inceptionv3 используется для классификации объектов в системах компьютерного зрения. В архитектуру этой модели заложены следующие основные идеи:

  - максимизация потока информации в сети за счет аккуратного баланса между ее глубиной и шириной — перед каждым процессом pooling увеличиваются карты свойств;
  - с увеличением глубины также систематически увеличивается количество свойств или ширина слоя;
  - ширина каждого слоя увеличивается ради роста комбинации свойств перед следующим слоем;
  - по мере возможности используются только свертки 3×3 (учитывая, что фильтры 5×5 и 7×7 можно декомпозировать с помощью нескольких фильтров 3×3).

- **DenseNet121**

  Это плотная сверточная сеть, в которой выполнено соединение предшествующих слоев с последующими слоями в режиме прямой связи. Сети DenseNet имеют ряд неоспоримых преимуществ: они облегчают проблему исчезающего градиента, улучшают распространение признаков, стимулируют повторное использование признаков и существенно сокращают количество параметров. Поскольку DenseNets требует меньше параметров и допускает повторное использование функций, они приводят к более компактным моделям и достигают самых современных характеристик и лучших результатов в конкурирующих наборах данных по сравнению со своими стандартными аналогами CNN или ResNet.

Как уже упоминалось ранее, все упомянутые модели были предварительно обучены на наборе данных ImageNet-1000, что дает вам возможность использовать класс `ImageClassification` для прогнозирования (распознавания, поиска) 1000 различных объектов в любом изображении или в любом количестве изображений.

Чтобы инициировать класс в вашем программном модуле, необходимо создать новый экземпляр класса:

```
from imageai.Classification import ImageClassification
prediction = ImageClassification()
```

После создания нового экземпляра класса `ImageClassification` вы можете использовать описанные далее его функции (методы), чтобы установить свойство этого экземпляра класса и начать распознавать объекты в изображениях.

Функция .setModelTypeAsMobileNetV2()

Эта функция устанавливает тип модели созданного вами экземпляра распознавания изображений — MobileNetV2. То есть вы будете выполнять свои задачи прогнозирования изображений с использованием именно этой предварительно обученной модели, которую могли скачать по ссылке, приведенной ранее. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsMobileNetV2()
```

Функция .setModelTypeAsResNet50()

Эта функция устанавливает тип модели созданного вами экземпляра распознавания изображений — ResNet50. То есть вы будете выполнять свои задачи прогнозирования изображений с использованием именно этой предварительно обученной модели, которую могли скачать по ссылке, приведенной ранее. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsResNet50()
```

Функция .setModelTypeAsInceptionV3()

Эта функция устанавливает тип модели созданного вами экземпляра распознавания изображений — InceptionV3. То есть вы будете выполнять задачи прогнозирования изображения с использованием именно этой предварительно обученной модели, которую могли скачать по ссылке, приведенной ранее. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsInceptionV3()
```

Функция .setModelTypeAsDenseNet121()

Эта функция устанавливает тип модели созданного вами экземпляра распознавания изображений — DenseNet121. То есть вы будете выполнять свои задачи прогнозирования изображений с использованием именно этой предварительно обученной модели, которую могли скачать по ссылке, приведенной ранее. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsDenseNet121()
```

Функция .setModelPath()

Эта функция принимает строку, которая указывает путь к загружаемому файлу обученной модели. Загружаемый файл должен соответствовать типу модели, заданному при создании экземпляра класса прогнозирования изображения.

Синтаксис функции:

```
prediction.setModelPath(model_path)
```

Пример ее использования:

```
prediction.setModelPath("resnet50_imagenet_tf.2.0.h5")
```

Параметр *model_path* в этой функции является обязательным. Соответственно сам файл обученной модели (файл с расширением h5) должен быть предварительно скачан и расположен в указанной в этом параметре папке компьютера.

Функция .loadModel()

Эта функция загружает обученную модель из пути, который был указан при вызове функции `setModelPath()`. Загрузить обученную модель можно с помощью следующего программного кода:

```
prediction.loadModel()
prediction.loadModel(self, prediction_speed=значение)
```

В этой функции можно указать необязательный строковый параметр `prediction_speed` (скорость предсказания), который позволяет до 80% сократить время, необходимое для прогнозирования изображения. Однако это приводит к некоторому снижению точности прогноза. Доступны следующие значения: `normal` (нормальный), `fast` (быстрый), `faster` (очень быстрый) и `fastest` (самый быстрый). Значения по умолчанию: `normal` (нормальный).

Функция .classifyImage()

Эта функция запускает процесс фактического предсказания (распознавания) изображения. Ее можно вызывать многократно для различных изображений (когда модель уже загружена в экземпляр класса прогнозирования). Запустить модель на распознавание изображения можно с помощью следующего программного кода:

```
predictions, probabilities = prediction.classifyImage("image1.jpg", result_count=10)
```

Функция имеет несколько параметров:

- `image_input` (обязательный) — задает путь к файлу изображения, к Numpy-массиву изображения или потоку видеоизображения (в зависимости от указанного типа ввода);
- `result_count` (необязательный) — указывает число возможных предсказаний, которые должны быть возвращены (по умолчанию для этого параметра установлено значение 5);
- `input_type` (необязательный) — относится к типу ввода, который указан в параметре `image_input`. Этот символьный параметр имеет значение `file` (файл) по умолчанию, а также может принимать значения: `array` (массив) и `stream` (поток);
- результат `prediction_results` (список Python). Первое значение — список, который содержит все возможные результаты прогнозирования (результаты расположены в порядке убывания вероятности в процентах);
- результат `prediction_probabilities` (список Python). Второе значение — список, который содержит процент вероятности всех возможных предсказаний, указанных в списке `prediction_results`.

Напишем программный код (листинг 7.1) с использованием класса `ImageClassification` и модели ResNet.

Листинг 7.1

```
# Listing 7_1
from imageai.Classification import ImageClassification
import os

# Текущий каталог
execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\resnet50_imagenet_tf.2.0.h5'
# Путь к файлу с изображением
img_path = execution_path + '\\Images\\image1.jpg'

prediction = ImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(model_path)
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(img_path, result_count=5)
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction, " : ", eachProbability)
```

Здесь в первых двух строках мы подключили необходимые библиотеки. Затем в переменной `execution_path` сохранили путь к папке, в которой находится наш проект. В переменной `model_path` указали путь к папке с файлом модели, в переменной `im_path` — путь к папке с файлом изображения. После чего создали экземпляр класса `Prediction`, определили его тип и загрузили модель обученной нейронной сети (файл **resnet50_imagenet_tf.2.0.h5**). Следует отметить, что этот файл мы загрузили из папки **Model**, которая является дочерней папкой нашего проекта. Далее мы передали модели изображение **image1.jpg** (в качестве примера использовано изображение, представленное на рис. 7.1) которое находится в папке **Images**, также являющейся дочер-

Рис. 7.1. Изображение, переданное в модель нейронной сети

ней папкой проекта. При этом мы просим модель распознать на этом изображении пять наиболее вероятных предметов.

В результате работы программы были распознаны пять наиболее вероятных предметов, имеющихся на предъявленном ей изображении (табл. 7.2). Как можно видеть, модель выдала вполне разумные ответы, хотя и не очень точные. В частности, версия того, что это авиалайнер, находится только на третьем месте с вероятностью около 17%.

Таблица 7.2. *Наиболее вероятные предметы, распознанные с помощью модели ResNet*

Наименование предмета	Вероятность соответствия предмету, %
Space_shuttle (космический челнок)	27.59304642677307
Airship (дирижабль)	9.719205647706985
Airliner (авиалайнер)	7.796771824359894
Recreational_vehicle (транспортное средство)	5.614634230732918
Web_site (веб-сайт)	4.503747448325157

Посмотрим, как будет работать с этим же изображением другая модель распознавания объектов — InceptionV3:

```
inception_v3_weights_tf_dim_ordering_tf_kernels.h5
```

Для этого напишем следующий программный код (листинг 7.2).

Листинг 7.2

```
# Listing 7_2
from imageai.Classification import ImageClassification
import os

# Текущий каталог
execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path +
'\\Model\\inception_v3_weights_tf_dim_ordering_tf_kernels.h5'
# Путь к файлу с изображением
img_path = execution_path + '\\Images\\image1.jpg'
# img_path = execution_path + '\\Images\\image4.jpg'

prediction = ImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath(os.path.join(execution_path, model_path))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(
    os.path.join(execution_path, img_path), result_count=5)
```

```
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction, " : ", eachProbability)
```

Результат работы этого программного кода представлен в табл. 7.3.

***Таблица 7.3.** Наиболее вероятные предметы на изображении, распознанные с помощью модели InceptionV3*

Наименование предмета	Вероятность соответствия предмету, %
Airliner (авиалайнер)	98.84645342826843
Wing (крыло)	1.1153464205563068
Space shuttle (космический челнок)	0.023960326507221907
Warplane (военный самолет)	0.005111367863719352
Aircraft carrier (авианосец)	0.0022653195628663525

Эта модель с достаточно большой точностью — 99% — распознала, что на предъявленном ей изображении основным объектом является авиалайнер.

Проверим работу этой программы еще на одном изображении (рис. 7.2). Для этого в строке программы:

```
img_path = execution_path + '\\Images\\image4.jpg'
```

изменим имя файла (в нашем примере это рисунок image4.jpg).

Рис. 7.2. Изображение автомобиля, переданное в модель нейронной сети

Программа выдаст результат, приведенный в табл. 7.4. Как можно видеть, эта модель нейронной сети вполне корректно определила элементы, имеющиеся в переданном ей изображении.

А теперь загрузим для обработки в модель нейронной сети сразу несколько изображений (листинг 7.3). Напомним также, что файл с моделью нейронной сети resnet50_imagenet_tf.2.0.h5 находится в папке Model, которая является дочерней папкой проекта.

Таблица 7.4. *Наиболее вероятные предметы на изображении (модель InceptionV3)*

Наименование предмета	Вероятность соответствия предмету, %
Convertible (кабриолет)	73.95122647285461
Sports car (спортивный автомобиль)	26.011955738067627
Grille (решетка радиатора)	0.01659276895225048
Amphibian (амфибия)	0.01606700971024111
Car wheel (колесо автомобиля)	0.0022917540263733827

Листинг 7.3

```
# Listing 7_3
from imageai.Classification import ImageClassification
import os

# Текущий каталог
execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path.= execution_path + '\\Model\\DenseNet-BC-121-32.h5'

prediction = ImageClassification()
prediction.setModelTypeAsDenseNet121()
prediction.setModelPath(os.path.join(execution_path, model_path))
prediction.loadModel()

# массив с файлами рисунков
all_images_array = []
all_files = os.listdir('.\\Images\\')  # все файлы
for each_file in all_files:            # выборка только рисунков
    if each_file.endswith(".jpg") or each_file.endswith(".png"):
        all_images_array.append(execution_path + "\\Images\\" + each_file)

for img_path in all_images_array:
    predictions, probabilities = prediction.classifyImage(
        os.path.join(execution_path, img_path), result_count=5)
    for eachPrediction, eachProbability in zip(predictions, probabilities):
        print(eachPrediction, " : ", eachProbability)
    print('-----------------------')
```

Здесь мы в переменной `inm_path` указали путь к файлам с изображениями, а в папку с изображениями добавили еще два файла: image2.jpg и image3.jpg (рис. 7.3).

После чего сконфигурировали модель нейронной сети и загрузили ее в объект `prediction`. С помощью следующих команд создали пустой массив, загрузили в него имена всех файлов, которые находятся в папке с нашим проектом, а затем выбрали

из них только те файлы, что являются изображениями. Массив с файлами изображений передали нашей модели на обработку с указанием найти на них пять объектов. В последних строках программы вывели параметры найденных объектов.

Рис. 7.3. Обработка двух изображений, переданных в модель нейронной сети

Программа выдала следующий результат для изображений image2.jpg и image3.jpg (табл. 7.5). Как можно видеть, первые наиболее вероятные предметы найдены и распознаны достаточно корректно.

Таблица 7.5. *Наиболее вероятные предметы на массиве изображений (модель ResNet)*

Image2.jpg		image3.jpg	
Наименование предмета	**Вероятность соответствия предмету, %**	**Наименование предмета**	**Вероятность соответствия предмету, %**
Cowboy_boot (ковбойские сапоги)	18.920007348060608	Tractor (трактор)	83.64035487174988
Potter's_wheel (гончарный круг)	13.304844498634338	Plow (плуг)	9.00954157114029
Mortar (ступа)	7.538526505231857	Harvester (жнец)	5.929012596607208
Breastplate (нагрудный знак)	6.47355243563652	Thresher (молотилка)	1.0732742957770824
Shieldr (щит)	4.627934843301773	Snowplow (снегоочиститель)	0.0970609195064753

7.1.2. Обнаружение и извлечение объектов из изображений: класс *ObjectDetection*

Класс `ObjectDetection` предоставляет функции (методы) для обнаружения объектов в любом изображении или в наборе изображений с помощью моделей, предвари-

тельно обученных на наборе данных COCO. Класс поддерживает следующие модели: RetinaNet, YOLOv3 и TinyYOLOv3. С их использованием вы можете обнаружить и распознать 80 различных повседневных предметов. Для начала работы необходимо загрузить любую из этих предварительно обученных моделей, скачав ее по ссылке: **https://imageai.readthedocs.io/en/latest/detection/index.html**.

Имена файлов, содержащих соответствующие модели, и их размеры приведены в табл. 7.6.

Таблица 7.6. Имена файлов с соответствующими моделями обученных нейронных сетей

Наименование модели	Наименование файла	Размер, Кбайт
RetinaNet	resnet50_coco_best_v2.0.1.h5	149 084
YOLOv3	yolo.h5	242 859
TinyYOLOv3	yolo-tiny.h5	34 723

Чтобы начать использовать выбранную модель, нужно создать новый экземпляр класса `ObjectDetection`. Это можно сделать так:

```
from imageai.Detection import ObjectDetection
detector = ObjectDetection()
```

Создав экземпляр класса, вы сможете использовать описанные далее его функции (методы), чтобы установить свойство этого экземпляра класса и начать обнаружение объектов на изображениях.

Функция .setModelTypeAsRetinaNet()

Эта функция устанавливает тип модели созданного вами экземпляра обнаруживаемого объекта — RetinaNet. То есть вы будете выполнять задачи обнаружения объекта с использованием именно этой предварительно обученной модели. Задать указанный тип модели можно с помощью следующего программного кода:

```
detector.setModelTypeAsRetinaNet()
```

Функция .setModelTypeAsYOLOv3()

Эта функция устанавливает тип модели созданного вами экземпляра обнаруживаемого объекта — YOLOv3. То есть вы будете выполнять задачи обнаружения объекта с использованием именно этой предварительно обученной модели. Задать указанный тип модели можно с помощью следующего программного кода:

```
detector.setModelTypeAsYOLOv3()
```

Функция .setModelTypeAsTinyYOLOv3()

Эта функция устанавливает тип модели созданного вами экземпляра обнаруживаемого объекта — TinyYOLOv3. То есть вы будете выполнять задачи обнаружения

объекта с использованием именно этой предварительно обученной модели. Задать указанный тип модели можно с помощью следующего программного кода:

```
detector.setModelTypeAsTinyYOLOv3()
```

Функция .setModelPath()

Эта функция принимает строку, которая указывает путь к загружаемому файлу обученной модели. Задать этот путь можно так:

```
detector.setModelPath(model_path)
```

Пример:

```
detector.setModelPath("yolo.h5")
```

Параметр *model_path* является обязательным. В указанную папку нужно поместить файл с загруженной из Интернета моделью нейронной сети.

Функция .loadModel()

Эта функция загружает модель из папки, указанной в функции `setModelpath()`, в ваш экземпляр класса обнаружения объектов. Инициировать загрузку можно с помощью следующего программного кода:

```
detector.loadModel(detection_speed=значение)
```

В этой функции можно указать необязательный строковый параметр `detection_speed` (скорость обнаружения), который позволяет до 80% сократить время, необходимое для обнаружения объектов на изображении. Однако это приводит к некоторому снижению точности прогноза. Доступны следующие значения: `normal` (нормальный), `fast` (быстрый), `faster` (очень быстрый) и `fastest` (самый быстрый). Значения по умолчанию: `normal` (нормальный).

Функция .detectObjectsFromImage()

Эта функция выполняет задачу обнаружения объекта после загрузки модели. Ее можно вызывать многократно для обнаружения объектов на любом количестве изображений. Инициировать процесс обнаружения объектов можно с помощью следующего программного кода:

```
detections = detector.detectObjectsFromImage(
                    input_image="image.jpg",
                    output_image_path="imagenew.jpg",
                    minimum_percentage_probability=30)
```

Функция имеет несколько параметров:

- `input_image` (обязательный) — путь и имя файла с изображением, которое вы собираетесь обрабатывать. Этот символьный параметр имеет значение `file` (файл) по умолчанию, а также может принимать значения: `array` (массив) и `stream` (поток). Возможно обрабатывать не один файл, а группу файлов. Для этого нужно

имена обрабатываемых файлов записать в массив Numpy (если для параметра `input_type` указано значение `array` или `stream`);

- `output_image_path` (требуется только в том случае, если `input_type="file"`) — путь к файлу, в котором будет сохранено обработанное изображение;
- `minimum_percentage_probability` (необязательный) — параметр для задания вероятности результатов обнаружения объектов (%). Понижение этого значения приведет к показу большего числа объектов, в то время как его увеличение обеспечивает обнаружение меньшего числа объектов с более высокой точностью предсказания. Значение по умолчанию равно 50;
- `output_type` (необязательный) — параметр для установки формата, в котором будет получено обработанное изображение. Доступны значения: `file` (файл, значение по умолчанию) и `array` (массив). Если для этого параметра установлено значение `array`, функция вернет массив Numpy обработанных изображений. Пример задания указанных параметров:

```
returned_image, detections =
   detector.detectObjectsFromImage(input_image="image.jpg",
                                   output_type="array",
                                   minimum_percentage_probability=30)
```

- `display_percentage_probability` (необязательный) — параметр для скрытия объектов с малой вероятностью верности распознавания, обнаруженных в обработанном изображении (если установлено значение `False`). Значение по умолчанию: `True`;
- `display_object_name` (необязательный) — параметр для скрытия имени каждого объекта, обнаруженного в обработанном изображении (если установлено значение `False`). Значения по умолчанию: `True`;
- `extract_detected_objects` (необязательный) — параметр для извлечения и сохранения каждого объекта, обнаруженного в обработанном изображении, в виде отдельного изображения (файла). Значение по умолчанию: `False`;
- `thread_safe` (необязательный) — параметр, гарантирующий, что загруженная модель обнаружения работает во всех потоках, если установлено значение `True`;
- `return` — набор возвращаемых значений, которые будут зависеть от параметров, заданных в функции `deteObjectsFromImage()`.

Возможны следующие варианты возвращаемых значений:

- если установлены все обязательные параметры, а для параметра `output_image_path` задан путь к файлу, в котором вы хотите сохранить обнаруженные на обработанном изображении объекты, то функция вернет новое (обработанное) изображение и массив, в котором каждый элемент массива будет соответствовать обнаруженному на исходном изображении объекту. Для каждого обнаруженного объекта будут выведены следующие параметры:

- имя обнаруженного объекта (строка);
- вероятность (в процентах) соответствия обнаруженному объекту (число с плавающей точкой);
- box_points (кортеж из координат x1, y1, x2 и y2), т. е. цветная рамка, которая будет выделять найденный объект на изображении.

Далее приведен пример строк программного кода для запуска процесса обнаружения объектов на изображении с такими параметрами:

```
detections = detector.detectObjectsFromImage(input_image="image.jpg",
                                             output_image_path="imagenew.jpg",
minimum_percentage_probability=30)
```

Здесь image.jpg — имя поданного для обработки изображения, imagenew.jpg — имя обработанного изображения, 30 — выделять на изображении только те объекты, для которых точность предсказания выше 30%;

- если все необходимые параметры установлены и output_type="array", функция вернет:
  - пустой массив обнаруженного изображения;
  - массив словарей, каждый из которых соответствует объектам, обнаруженным на изображении. Каждый словарь содержит:
    - имя обнаруженного объекта (строка);
    - вероятность (в процентах) соответствия обнаруженному объекту (число с плавающей точкой);
    - box_points (кортеж из координат x1, y1, x2 и y2), т. е. цветную рамку, которая будет выделять найденный объект на изображении.

Далее приведен пример строк программного кода для запуска процесса обнаружения объектов на изображении с такими параметрами:

```
returned_image, detections = detector.detectObjectsFromImage(
    input_image="image.jpg",
    output_type="array",
    minimum_percentage_probability=30
```

- если задано extract_detected_objects=True и указан желаемый путь к файлу output_image_path, то обнаруженное изображение будет сохранено, а функция вернет:
  - массив словарей, каждый из которых соответствует объектам, обнаруженным на изображении. Каждый словарь содержит:
    - имя обнаруженного объекта (строка);
    - вероятность (в процентах) соответствия обнаруженному объекту (число с плавающей точкой);
    - box_points (кортеж из координат x1, y1, x2 и y2), т. е. цветную рамку, которая будет выделять найденный объект на изображении;

- массив строковых путей к изображению каждого объекта, извлеченный из изображения.

Далее приведен пример строк программного кода для запуска процесса обнаружения объектов на изображении с такими параметрами:

```
detections, extracted_objects = detector.detectObjectsFromImage(
    input_image="image.jpg",
    output_image_path="imagenew.jpg",
    extract_detected_objects=True,
    minimum_percentage_probability=30
```

- Если заданы `extract_detected_objects=True` и `output_type="array"`, то функция вернет:
  - пустой массив обнаруженного изображения;
  - массив словарей, каждый из которых соответствует объектам, обнаруженным на изображении. Каждый словарь содержит:
    - имя обнаруженного объекта (строка);
    - вероятность (в процентах) соответствия обнаруженному объекту (число с плавающей точкой);
    - `box_points` (кортеж из координат `x1`, `y1`, `x2` и `y2`), т. е. цветную рамку, которая будет выделять найденный объект на изображении;
  - массив NumPy-массивов каждого объекта, обнаруженного на изображении.

Далее приведен пример строк программного кода для запуска процесса обнаружения объектов на изображении с такими параметрами:

```
returned_image, detections, extracted_objects = detector.detectObjectsFromImage(
    input_image="image.jpg",
    output_type="array",
    extract_detected_objects=True,
    minimum_percentage_probability=30)
```

Функция *.CustomObjects()*

Эта функция используется, когда необходимо задать обнаружение только определенных объектов. Функция возвращает словарь объектов и их значения `True` или `False`. Для того чтобы обнаружить выбранные объекты на изображении, следует использовать словарь, возвращаемый функцией `this()`, с функцией:

```
detectCustomObjectsFromImage()
```

Словарь возможных наименований объектов приведен в табл. 7.7.

Модели можно указать, что на изображении надо обнаружить только некоторые из перечисленных в табл. 7.7. объектов. Для этого необходимо вызвать функцию `CustomObjects()` и задать ей в качестве параметра имя объекта или список имен желаемых объектов. Имена этих объектов получат признак `true`, а всем остальным

именам объектов будет присвоен признак `false` по умолчанию. В следующем примере модели дано задание обнаруживать на изображении только людей и собак:

```
custom = detector.CustomObjects(person=True, dog=True)
```

Таблица 7.7. *Имена объектов, которые можно задать для поиска на изображениях*

№ п/п	Объект	Перевод	№ п/п	Объект	Перевод
1	airplane	самолет	30	donot	выигрыш
2	apple	яблоко	31	elephant	слон
3	backpack	рюкзак	32	fire hydrant	пожарный гидрант
4	banana	банан	33	fork	вилка
5	baseball bat	бейсбольная бита	34	frisbee	фрисби
6	baseball glove	бейсбольная перчатка	35	giraffe	жираф
7	bear	медведь	36	hair dryer	фен
8	bed	кровать	37	handbag	сумка
9	bench	скамейка	38	horse	лошадь
10	bicycle	велосипед	39	hot dog	хот-дог
11	bird	птица	40	keyboard	клавиатура
12	boat	лодка	41	kite	воздушный змей
13	book	книга	42	knife	нож
14	bottle	бутылка	43	laptop	ноутбук
15	bowl	чаша	44	microwave	микроволновая печь
16	broccoli	брокколи	45	motorcycle	мотоцикл
17	bus	автобус	46	mouse	мышь
18	cake	торт	47	orange	апельсин
19	car	машина	48	oven	духовка
20	carrot	морковь	49	parking meter	парковочный счетчик
21	cat	кошка	50	person	человек
22	cell phone	сотовый телефон	51	pizza	пицца
23	chair	стул	52	potted plant	комнатные растения
24	clock	часы	53	refrigerator	холодильник
25	couch	диван	54	remote	удаленный
26	cow	корова	55	sandwich	сэндвич
27	cup	чашка	56	scissors	ножницы
28	dining table	обеденный стол	57	sheep	овца
29	dog	собака	58	sink	раковина

Таблица 7.7 (окончание)

№ п/п	Объект	Перевод	№ п/п	Объект	Перевод
59	skateboard	скейтборд	70	toaster	тостер
60	skis	лыжи	71	toilet	туалет
61	snowboard	сноуборд	72	toothbrush	зубная щетка
62	spoon	ложка	73	traffic light	светофор
63	sports ball	спортивные мячи	74	train	поезд
64	stop_sign	знак остановки	75	truck	грузовик
65	suitcase	чемодан	76	tv	телевизор
66	surfboard	доска для серфинга	77	umbrella	зонтик
67	teddy bear	плюшевый мишка	78	vase	ваза
68	tennis racket	теннисная ракетка	79	wine glass	бокал для вина
69	tie	галстук	80	zebra	зебра

Функция *.detectCustomObjectsFromImage()*

Эта функция имеет те же параметры и возвращает те же значения, что и функция `detectObjectsFromImage()`, но у нее есть небольшое отличие: она позволяет обнаруживать на изображении только те объекты, которые задал пользователь. В отличие от обычной функции `detectObjectsFromImage()`, для этого требуется явно задать дополнительный параметр `custom_object`, который принимает словарь, возвращаемый функцией `CustomObjects()`. В следующем примере модели дано задание обнаруживать на изображении только людей и собак:

```
custom = detector.CustomObjects(person=True, dog=True)
detections = detector.detectCustomObjectsFromImage(
    custom_objects=custom,
    input_image=os.path.join(execution_path, "image3.jpg"),
    output_image_path=os.path.join(execution_path, "image3new-custom.jpg"),
    minimum_percentage_probability=30)
```

Здесь мы первым оператором двум объектам присвоили значение `true`: `person=True`, `dog=True`. Затем вызвали функцию `detector.detectCustomObjectsFromImage()`, в качестве параметров передали ей список объектов пользователя (`custom_objects`), указали пути и наименования входного и выходного файлов изображения. А также дали команду выдавать сведения только о тех объектах, для которых вероятность правильного решения превышает 30%.

Напишем небольшую программу и посмотрим, насколько эффективно работает эта модель нейронной сети (листинг 7.4).

Листинг 7.4

```
# Listing 7_4
from imageai.Detection import ObjectDetection
```

```
import os

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo.h5'
# Путь к файлу с изображением
img_path_in = execution_path + '\\Images\\image5.jpg'
img_path_out = execution_path + '\\Images\\image_out5.jpg'

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(model_path)
detector.loadModel()
detections = detector.detectObjectsFromImage(
    input_image=img_path_in,
    output_image_path=img_path_out,
    minimum_percentage_probability=30)

for eachObject in detections:
    print(eachObject["name"], " : ", eachObject["percentage_probability"],
          " : ", eachObject["box_points"])
    print("--------------------------------")
```

Здесь мы подключили две библиотеки, сконфигурировали модель нейронной сети, подали на вход сети одно изображение (image5.jpg), а также дали модели предписание, что после обработки поданного на вход изображения необходимо сформировать файл обработанного изображения с именем image_out5.jpg. В последнем фрагменте программы вывели итоговые результаты.

Обратите внимание, что файлы изображений не обязательно располагать в той же папке, что и весь проект, — их можно хранить в любом месте.

Итак, подадим на вход нашей нейронной сети изображение image5.jpg, представленное на рис. 7.4.

В результате работы программы мы получили обработанное изображение (рис. 7.5). Как можно видеть, нейронная сеть уверенно распознала и человека (100%), и собаку (100%), и кошку (91%). Кроме обработанного рисунка, программа выдала и текстовую информацию, которая представлена в табл. 7.8.

***Таблица 7.8.** Таблица результатов обработки изображения image.jpg*

Распознанный на изображении объект	Точность прогноза, %	Координаты обрамления объекта на рисунке
person (человек)	99.93067979812622	[185, 103, 322, 457]
cat (кошка)	90.51359295845032	[110, 348, 169, 412]
dog (собака)	99.59741830825806	[324, 329, 420, 463]

Рис. 7.4. Изображение image5.jpg, поданное на вход нейронной сети

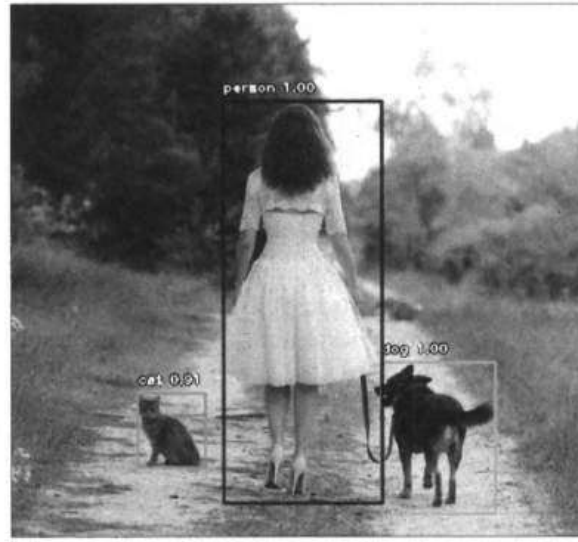


Рис. 7.5. Изображение image_out5.jpg, полученное на выходе из нейронной сети

В листинге 7.5 приведен код той же программы, но с использованием модели нейронной сети RetinaNet .

Листинг 7.5

```
# Listing 7_5
from imageai.Detection import ObjectDetectionimport os
```

```
execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\resnet50_coco_best_v2.1.0.h5'
# Путь к файлу с изображением
img_path_in = execution_path + '\\Images\\image5.jpg'
img_path_out = execution_path + '\\Images\\image_out5.jpg'

detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(model_path)
detector.loadModel()
detections = detector.detectObjectsFromImage(
    input_image=img_path_in,
    output_image_path=img_path_out,
    minimum_percentage_probability=30)

for eachObject in detections:
    print(eachObject["name"], " : ", eachObject["percentage_probability"],
          " : ", eachObject["box_points"])
    print("--------------------------------")
```

Теперь немного изменим программный код листинга 7.4 и ограничим количество распознаваемых на изображении объектов, — дадим команду определить на том же изображении лишь человека и собаку (листинг 7.6).

Листинг 7.6

```
# Listing 7_6
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo.h5'
# Путь к файлу с изображением
img_path_in = execution_path + '\\Images\\image5.jpg'
img_path_out = execution_path + '\\Images\\image_out5.jpg'

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(model_path)
detector.loadModel()
custom = detector.CustomObjects(person=True, dog=True)
detections = detector.detectObjectsFromImage(
    custom_objects=custom,
    input_image=img_path_in,
```

```
    output_image_path=img_path_out,
    minimum_percentage_probability=30)

for eachObject in detections:
    print(eachObject["name"], " : ", eachObject["percentage_probability"],
          " : ", eachObject["box_points"])
    print("--------------------------------")
```

В результате обработки той же картинки мы получили иное итоговое изображение (рис. 7.6). Как можно видеть, нейронная сеть распознала именно два заданных ей изображения.

Рис. 7.6. Изображение, полученное на выходе из нейронной сети при параметрах: `person=True, dog=True`

А теперь дадим на вход модели более насыщенную картинку — например, изображение оживленной улицы (рис. 7.7), и посмотрим, насколько успешно нейронная сеть распознает на этом изображении множество объектов (листинг 7.7).

Листинг 7.7

```
# Listing 7_7
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo.h5'
# Путь к файлу с изображением
img_path_in = execution_path + '\\Images\\image_str.jpg'
img_path_out = execution_path + '\\Images\\image_str_out.jpg'
```

```
detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(model_path)
detector.loadModel()
detections = detector.detectObjectsFromImage(
    input_image=img_path_in,
    output_image_path=img_path_out,
    minimum_percentage_probability=30)

for eachObject in detections:
    print(eachObject["name"], " : ", eachObject["percentage_probability"],
          " : ", eachObject["box_points"])
    print("--------------------------------")
```

Рис. 7.7. Изображение оживленной улицы, поданное на вход нейронной сети

Результаты работы программы представлены на рис. 7.8. Как можно видеть, сеть смогла выделить и распознать на изображении много объектов. Вот их полный список:

```
traffic light  :  33.30996036529541  :  [283, 274, 297, 296]
traffic light  :  37.60130703449249  :  [479, 238, 492, 266]
traffic light  :  95.56607007980347  :  [1, 31, 29, 96]
truck  :  57.94483423233032  :  [360, 307, 419, 360]
car  :  32.715195417404175  :  [16, 340, 81, 370]
car  :  50.835227966308594  :  [112, 334, 152, 372]
car  :  66.2396252155304  :  [35, 341, 90, 419]
car  :  72.23671078681946  :  [222, 340, 251, 364]
car  :  96.21666669845581  :  [167, 328, 216, 369]
car  :  96.4270830154419  :  [2, 330, 43, 438]
car  :  99.07432794570923  :  [276, 322, 356, 392]
```

```
person  :  53.254568576812744  :  [688, 329, 704, 384]
person  :  68.03338527679443  :  [667, 324, 687, 383]
person  :  74.1044819355011  :  [673, 329, 697, 382]
person  :  98.06962013244629  :  [376, 317, 449, 438]
person  :  99.00452494621277  :  [527, 303, 585, 488]
person  :  99.34167861938477  :  [599, 305, 662, 507]
person  :  99.34455752372742  :  [208, 322, 302, 483]
person  :  99.51716661453247  :  [67, 328, 127, 477]
```

Всего распознано 20 объектов (светофоры, легковые и грузовые автомобили, люди). При этом вероятность правильной идентификации объектов довольно высока — в пределах 50–99%.

Рис. 7.8. Изображение оживленной улицы, полученное на выходе из нейронной сети

7.2. Классы распознавания объектов в видеофайлах и видеопотоках

С помощью библиотеки ImageAI можно решать задачи, связанные с обнаружением объектов в режиме реального времени на потоковом видео с видео- и IP-камер.

7.2.1. Обнаружение объектов в видеофайлах и видеопотоках с видеокамер: класс *VideoObjectDetection*

Класс `VideoObjectDetection` предоставляет функции (методы) для обнаружения объектов в видеофайлах, в потоках прямой трансляции с видеокамер и с IP-камер, используя модели, предварительно обученные на наборе данных COCO. Класс поддерживает следующие модели: RetinaNet, YOLOv3 и TinyYOLOv3. С их помощью вы можете обнаружить и распознать 80 различных повседневных предметов. Для

начала работы необходимо загрузить любую из этих предварительно обученных моделей, скачав ее по ссылке: **https://imageai.readthedocs.io/en/latest/video/index.html**.

Имена файлов, содержащих соответствующие модели, и их размеры приведены в табл. 7.9.

***Таблица 7.9.** Имена файлов с соответствующими моделями обученных нейронных сетей*

Наименование модели	Наименование файла	Размер, Кбайт
RetinaNet	resnet50_coco_best_v2.0.1.h5	149 084
YOLOv3	yolo.h5	242 859
TinyYOLOv3	yolo-tiny.h5	34 723

Чтобы начать использовать выбранную модель, нужно создать новый экземпляр класса `VideoObjectDetection`. Это можно сделать так:

```
from imageai.Detection import VideoObjectDetection
detector = VideoObjectDetection()
```

Создав экземпляр класса, вы сможете использовать описанные далее его функции (методы), чтобы установить свойство этого экземпляра класса и начать обнаружение объектов на видео.

Функция .setModelTypeAsRetinaNet()

Эта функция устанавливает тип модели созданного вами экземпляра обнаруживаемого объекта — RetinaNet. То есть вы будете выполнять задачи обнаружения объекта с использованием именно этой предварительно обученной модели. Задать указанный тип модели можно с помощью следующего программного кода:

```
detector.setModelTypeAsRetinaNet()
```

Функция .setModelTypeAsYOLOv3()

Эта функция устанавливает тип модели созданного вами экземпляра обнаруживаемого объекта — YOLOv3. То есть вы будете выполнять задачи обнаружения объекта с использованием именно этой предварительно обученной модели. Задать указанный тип модели можно с помощью следующего программного кода:

```
detector.setModelTypeAsYOLOv3()
```

Функция .setModelTypeAsTinyYOLOv3()

Эта функция устанавливает тип модели созданного вами экземпляра обнаруживаемого объекта — TinyYOLOv3. То есть вы будете выполнять задачи обнаружения объекта с использованием именно этой предварительно обученной модели. Задать указанный тип модели можно с помощью следующего программного кода:

```
detector.setModelTypeAsTinyYOLOv3()
```

Функция .setModelPath()

Эта функция принимает строку, которая указывает путь к загружаемому файлу обученной модели. Задать этот путь можно так:

```
detector.setModelPath(model_path)
```

Пример:

```
detector.setModelPath("yolo.h5")
```

Параметр *model_path* является обязательным. В указанную папку нужно поместить файл с загруженной из Интернета моделью нейронной сети.

Функция .loadModel()

Эта функция загружает модель из папки, указанной в функции `setModelpath()`, в ваш экземпляр класса обнаружения объектов. Инициировать загрузку можно с помощью следующего программного кода:

```
detector.loadModel(Detection_speed=значение)
```

В этой функции можно указать необязательный строковый параметр `detection_speed` (скорость обнаружения), который позволяет до 80% сократить время, необходимое для обнаружения объектов на изображении. Однако это приводит к некоторому снижению точности прогноза. Доступны следующие значения: `normal` (нормальный), `fast` (быстрый), `faster` (очень быстрый) и `fastest` (самый быстрый). Значения по умолчанию: `normal` (нормальный).

Функция .detectObjectsFromVideo()

Функция выполняет обнаружение объектов в видеофайле или видеопотоке с видеокамеры в режиме реального времени после загрузки модели в созданный вами экземпляр класса. Инициировать процесс обнаружения объектов можно с помощью следующего программного кода:

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "traffic.mp4"),
    output_file_path=os.path.join(execution_path, "traffic_detected"),
    frames_per_second=20,
    log_progress=True)
print(video_path)
```

В этом программном коде используются следующие параметры функции:

- `input_file_path` — путь к видеофайлу, в котором требуется обнаруживать объекты (указывается в том случае, если вы не установили входящий поток изображений с видеокамеры `camera_input`);
- `output_file_path` — путь к папке, в которой будет сохранен обработанный выходной видеофайл. По умолчанию эта функция сохраняет видео в формате AVI (указывается, если не установлен параметр отказа от записи выходного файла `save_detected_video = False`);
- `frames_per_second` (необязательный, но рекомендуемый) — желаемая частота кадров в секунду для обработанного выходного видеофайла (который будет сохранен). По умолчанию равно 20, но имеется возможность установить то значение, которое больше подойдет для выходного видеофайла;
- `log_progress` (необязательный) — если для этого параметра установлено значение `True`, то будет отображаться ход обработки видеофайла (в виде прямой трансляции). Значение по умолчанию: `False`;
- `return_detected_frame` (необязательный) — параметр, позволяющий возвращать обнаруженный объект в кадре в виде массива Numpy (для каждого кадра), секунды и минуты обнаружения объекта на видео. Далее полученный массив Numpy будет проанализирован в соответствующих функциях: `per_frame_function()`, `per_second_function()` и `per_minute_function()` (описание этих функций приведено далее);
- `camera_input` (необязательный) — параметр, который можно установить вместо параметра `input_file_path` (если требуется обнаруживать объекты в прямом эфире с видеокамеры, а не из видеофайла). Для этого необходимо активировать видеокамеру с помощью функции `VideoCapture()` из библиотеки OpenCV.

Далее приведен программный код обработки видеофайлов, получаемых с видеокамеры:

```
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(
    camera_input=camera,
    output_file_path=os.path.join(execution_path, "camera_detected_video"),
    frames_per_second=20,
```

```
    log_progress=True,
    minimum_percentage_probability=30)

print(video_path)
```

В этом программном коде используются следующие параметры функции:

- `minimum_percentage_probability` (необязательный, минимальная процентная вероятность) — параметр для определения целостности (достоверности) результатов обнаружения объектов. При понижении значения этой вероятности будет обнаружено и показано больше объектов, в то время как увеличение ее значения уменьшит количество найденных объектов, но обеспечит более высокую точность обнаружения. Значение по умолчанию составляет 50;
- `display_percentage_probability` (необязательный) — параметр для скрытия (отображения) вероятности соответствия объекта образцам, содержащимся в обученной модели (если значение параметра равно `False`, то вероятность, выраженная в процентах, отображаться не будет). Значение по умолчанию: `True`;
- `display_object_name` (необязательный) — параметр для скрытия имени каждого объекта, обнаруженного в видео, если установлено значение `False`. Значение по умолчанию: `True`;
- `save_detected_video` (необязательный) — параметр, предписывающий программе, сохранять или не сохранять обнаруженные объекты в обработанном видеофайле. По умолчанию установлено значение `True`;
- `per_frame_function` (необязательный) — параметр для имени функции, которую вы задаете самостоятельно. Затем для каждого обнаруженного в кадре объекта эта видеофункция будет анализировать заданные параметры. Полученные данные можно визуализировать или сохранить в базе данных NoSQL для дальнейшей обработки и визуализации.

Рассмотрим далее несколько примеров использования описанных функций для поиска объектов в видеофайлах и потоковых видео с видеокамер.

7.2.2. Примеры программы распознавания объектов в видеофайлах

Сначала применим для обработки видеофайла следующий программный код (листинг 7.8).

Листинг 7.8

```
# Listing 7_8
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo.h5'
```

```
# Путь к файлу с видео
vide_path_in = execution_path + '\\Video\\traffic.mp4'
vide_path_out = execution_path + '\\Video\\traffic_detected'

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(model_path)
detector.loadModel()

video_path = detector.detectObjectsFromVideo(
    input_file_path=vide_path_in,
    output_file_path=vide_path_out,
    frames_per_second=20,
    log_progress=True)
print(video_path)
```

В этой программе к проекту подключены две библиотеки, и в переменной `execution_path` сохранен путь к папке с проектом. Затем мы создали объект для работы с видеофайлом `detector` и загрузили в этот объект файл обученной модели нейронной сети yolo.h5. После чего запустили процесс поиска объектов в видеофайле traffic.mp4 и указали, что обработанный файл с именем traffic_detected нужно сохранить в той же папке. Для выходного файла задали частоту 20 кадров в секунду и указали, чтобы выводился журнал процесса обработки кадров.

Чтобы поверить работу этой программы, годится любой видеофайл. Для приведенного примера из Интернета был скачан видеофайл с автомобилями, движущимися по одной из ночных улиц, и с именем traffic.mp4 помещен в дочернюю папку Video нашего проекта. Фрагмент изображения из этого видеофайла приведен на рис. 7.9.

Рис. 7.9. Фрагмент изображения из видеофайла, поданного на вход нейронной сети

После запуска программы началась покадровая обработка изображений, и мониторинг этого процесса выводился на экран (рис. 7.10). По завершении процесса обработки кадров программа выдала сообщение о том, что итоговый файл с найденными объектами записан с папку с проектом под именем traffic_detected.avi. Фрагмент из этого итогового видеофайла приведен на рис. 7.11.

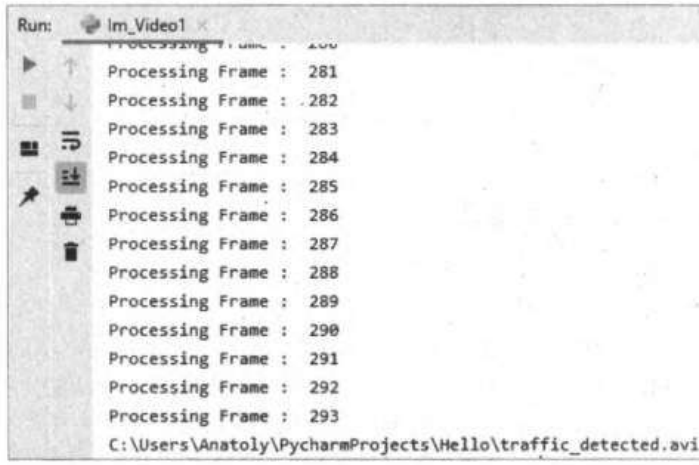


Рис. 7.10. Мониторинг процесса обработки видеофайла

Рис. 7.11. Фрагмент изображения из видеофайла, полученного на выходе из нейронной сети

Как можно видеть, обученная нейронная сеть с вероятностью до 99% верно распознала в видеопотоке легковые и грузовые автомобили.

Видеофайлы не обязательно хранить в папке с проектом — их можно размещать в любой папке на компьютере и обрабатывать с использованием любой из трех упомянутых ранее моделей сети.

В следующем примере программы (листинг 7.9) видеофайл, размещенный в дочерней папке текущего проекта Video, обрабатывался моделью сети из файла yolo-tiny.h5.

Листинг 7.9

```
# Listing 7_9
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo-tiny.h5'
# Путь к файлу с видео
vide_path_in = execution_path + '\\Video\\traffic.mp4'
vide_path_out = execution_path + '\\Video\\traffic_detected'

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath(model_path)
detector.loadModel()

video_path = detector.detectObjectsFromVideo(
    input_file_path=vide_path_in,
    output_file_path=vide_path_out,
    frames_per_second=20,
    log_progress=True)
print(video_path)
```

7.2.3. Пример программы распознавания объектов в видеопотоках с видеокамер

Теперь рассмотрим пример использования описанных ранее функций для поиска объектов в потоковых видео с видеокамер. Применим для обработки изображений в процессе прямой трансляции с видеокамеры следующий программный код (листинг 7.10).

Листинг 7.10

```
# Listing 7_10
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo.h5'
```

```
# Путь для записи обработанного видеофайла
video_path_out = execution_path + '\\Video\\camera_detected_video'

# для встроенной камеры ноутбука
camera = cv2.VideoCapture(0)

# для подключенной внешней веб-камеры
# camera = cv2.VideoCapture(0, cv2.CAP_DSHOW)

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(model_path)
detector.loadModel()

video_path = detector.detectObjectsFromVideo(camera_input=camera,
    output_file_path=video_path_out,
    frames_per_second=20,
    log_progress=True,
    minimum_percentage_probability=30)

print(video_path)
```

Здесь к проекту подключены три библиотеки, и в переменной `execution_path` сохранен путь к папке с этим проектом. Для работы с веб-камерой вам может потребоваться еще один программный модуль — `opencv-contrib-python`. Установить его можно с помощью следующей команды:

```
pip install opencv-contrib-python.
```

В этой книге использовалась библиотека OpenCV версии 4.6.0.66.

Далее мы активировали видеокамеру компьютера с помощью команды:

```
# для встроенной камеры ноутбука
camera = cv2.VideoCapture(0)
```

и внешнюю веб-камеру с помощью команды:

```
# для подключенной внешней веб-камеры
# camera = cv2.VideoCapture(0, cv2.CAP_DSHOW)
```

Обратите внимание, что команды активирования встроенной камеры ноутбука и подключенной внешней веб-камеры несколько различаются.

В следующих строках мы создали объект для работы с потоковым видео `detector` и загрузили в этот объект файл `yolo.h5` обученной модели нейронной сети. Затем запустили процесс поиска объектов в потоковом видео и указали, что обработанный файл с именем `camera_detected_video` нужно сохранить в папке `Video`. Для выходного файла задали частоту 20 кадров в секунду, указали, чтобы выводился журнал процесса обработки кадров, а также определили, чтобы в выходном файле фикси-

ровались все объекты, вероятность верного предсказания для которых будет превышать 30%.

Чтобы проверить работу этой программы, понадобится компьютер с видеокамерой. С веб-камерами работают также практически все современные компьютеры. Для рассматриваемого примера с видеокамеры ноутбука была запущена прямая трансляция прогулки собаки вместе с ее владельцем. В процессе работы программы формировался выходной файл с именем camera_detected_video, который записывался в дочернюю папку проекта — Video. Фрагмент изображения из этого обработанного потокового видео приведен на рис. 7.12.

Рис. 7.12. Фрагмент изображения из обработанного потокового видео

В процессе прямой трансляции потокового видео программа корректно (с вероятностью правильного прогноза 97%) распознала человека и с вероятностью правильного прогноза 93% определила собаку.

7.2.4. Пример программы с пользовательскими функциями для распознавания объектов в видеофайлах

В этом разделе мы рассмотрим пример программы, в которой реализовано применение пользовательской функции. Такая функция будет вызываться на исполнение каждый раз, когда в видеокадре будет обнаружен новый объект.

Создадим пользовательскую функцию, которая для кадров, в которых обнаружен новый объект, будет выдавать следующую информацию:

- номер кадра (фрейма);
- массив параметров по каждому объекту, который был обнаружен в кадре. Каждый элемент массива будет содержать следующие данные:

- name — имя обнаруженного в кадре объекта;
- percentage_probability — уровень достоверности (уверенности в правильности идентификации объекта) в процентах;
- box_points — координаты рамки, которая обрамляет найденный объект;

❐ массив с именами объектов и их общим количеством, обнаруженным в кадре.

В листинге 7.11 приведен программный код, в котором эта функция реализована и подключена к объекту класса, обрабатывающему видеофайл traffic.mp4.

Листинг 7.11

```
# Listing 7_11
from imageai.Detection import VideoObjectDetection
import os

def forFrame(frame_number, output_array, output_count):
    print("НОМЕР ФРЕЙМА ", frame_number)
    print("Массив параметров найденных объектов: ", output_array)
    print("Количество найденных объектов: ", output_count)
    print("------------КОНЕЦ ФРЕЙМА --------------")

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + '\\Model\\yolo.h5'
# Путь к файлам с видео
video_path_in = execution_path + '\\Video\\traffic.mp4'
video_path_out = execution_path + '\\Video\\video_frame_analysis'

execution_path = os.getcwd()
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(model_path)
video_detector.loadModel()

video_detector.detectObjectsFromVideo(
    input_file_path=video_path_in,
    output_file_path=video_path_out,
    frames_per_second=20,
    per_frame_function=forFrame,
    minimum_percentage_probability=30)
```

В этой программе на вход объекта video_detector передан видеофайл traffic.mp4. После его обработки получен новый видеофайл — video_frame_analysis, в котором рамками выделены обнаруженные объекты. Фрагмент изображения с этого видеофайла приведен на рис. 7.13.

Рис. 7.13. Фрагмент изображения из обработанного видеофайла video_frame_analysis

Далее приведен фрагмент распечатки, которая получена в результате работы пользовательской функции, переданной в объект `video_detector`:

```
НОМЕР ФРЕЙМА  80
Массив параметров найденных объектов:
[{'name': 'truck', 'percentage_probability': 32.31269121170044, 'box_points': [495, 972, 646, 1080]},
{'name': 'truck', 'percentage_probability': 49.131572246551514, 'box_points': [900, 576, 997, 690]},
{'name': 'truck', 'percentage_probability': 91.70636534690857, 'box_points': [759, 527, 895, 713]},
{'name': 'car', 'percentage_probability': 38.186877965927124, 'box_points': [984, 530, 1040, 591]},
{'name': 'car', 'percentage_probability': 47.61531352996826, 'box_points': [900, 576, 997, 690]},
{'name': 'car', 'percentage_probability': 54.76024150848389, 'box_points': [611, 802, 748, 902]},
{'name': 'car', 'percentage_probability': 64.411860704422, 'box_points': [575, 875, 709, 976]},
{'name': 'car', 'percentage_probability': 94.55206394195557, 'box_points': [610, 765, 752, 864]},
{'name': 'car', 'percentage_probability': 96.78226709365845, 'box_points': [1126, 502, 1192, 555]},
{'name': 'car', 'percentage_probability': 97.52147197723389, 'box_points': [792, 910, 954, 1017]},
{'name': 'car', 'percentage_probability': 97.7426528930664, 'box_points': [884, 684, 983, 789]},
{'name': 'car', 'percentage_probability': 98.61152768135071, 'box_points': [1496, 951, 1711, 1077]},
{'name': 'car', 'percentage_probability': 98.71078133583069, 'box_points': [1139, 586, 1231, 664]},
{'name': 'car', 'percentage_probability': 99.33229088783264, 'box_points': [839, 778, 975, 915]},
{'name': 'person', 'percentage_probability': 30.167457461357117, 'box_points': [266, 460, 300, 528]}]
Количество найденных объектов:  {'truck': 3, 'car': 11, 'person': 1}
------------КОНЕЦ ФРЕЙМА --------------
НОМЕР ФРЕЙМА  81
Массив параметров найденных объектов:
[{'name': 'truck', 'percentage_probability': 31.430470943450928, 'box_points': [897, 582, 1000,
695]},
{'name': 'truck', 'percentage_probability': 60.182762145996094, 'box_points': [492, 970, 652, 1080]},
{'name': 'truck', 'percentage_probability': 94.44217681884766, 'box_points': [756, 526, 893, 715]},
{'name': 'car', 'percentage_probability': 43.05436909198761, 'box_points': [982, 530, 1040, 593]},
{'name': 'car', 'percentage_probability': 68.7540054321289, 'box_points': [574, 873, 709, 978]},
```

```
{'name': 'car', 'percentage_probability': 69.32180523872375, 'box_points': [900, 585, 998, 693]},
{'name': 'car', 'percentage_probability': 94.2551851272583, 'box_points': [1137, 581, 1230, 651]},
{'name': 'car', 'percentage_probability': 95.05894184112549, 'box_points': [795, 920, 947, 1021]},
{'name': 'car', 'percentage_probability': 96.02027535438538, 'box_points': [609, 764, 753, 868]},
{'name': 'car', 'percentage_probability': 96.3109016418457, 'box_points': [1125, 496, 1190, 546]},
{'name': 'car', 'percentage_probability': 97.38959074020386, 'box_points': [883, 686, 988, 789]},
{'name': 'car', 'percentage_probability': 99.18128848075867, 'box_points': [839, 785, 972, 917]},
{'name': 'car', 'percentage_probability': 99.72811937332153, 'box_points': [1465, 914, 1670, 1075]},
 {'name': 'person', 'percentage_probability': 32.75151252746582, 'box_points': [266, 458, 299, 528]}]
Количество найденных объектов:  {'truck': 3, 'car': 10, 'person': 1}
------------КОНЕЦ ФРЕЙМА --------------
```

Теперь создадим пользовательскую функцию, которая обрабатывает видеофайл, находит в нем различные объекты и в режиме реального времени визуализирует результаты этой обработки. Пример такого программного кода приведен в листинге 7.12.

Листинг 7.12

```
# Listing 7_12
from imageai.Detection import VideoObjectDetection
import os
from matplotlib import pyplot as plt

color_index = {'bus': 'red', 'handbag': 'steelblue', 'giraffe': 'orange',
               'spoon': 'gray', 'cup': 'yellow', 'chair': 'green',
               'elephant': 'pink', 'truck': 'indigo',
               'motorcycle': 'azure','refrigerator': 'gold',
               'keyboard': 'violet', 'cow': 'magenta',
               'mouse': 'crimson', 'sports ball': 'raspberry',
               'horse': 'maroon', 'cat': 'orchid', 'boat': 'slateblue',
               'hot dog': 'navy', 'apple': 'cobalt',
               'parking meter': 'aliceblue', 'sandwich': 'skyblue',
               'skis': 'deepskyblue', 'microwave': 'peacock',
               'knife': 'cadetblue', 'baseball bat': 'cyan',
               'oven': 'lightcyan', 'carrot': 'coldgrey',
               'scissors': 'seagreen', 'sheep': 'deepgreen',
               'toothbrush': 'cobaltgreen', 'fire hydrant': 'limegreen',
               'remote': 'forestgreen', 'bicycle': 'olivedrab',
               'toilet': 'ivory', 'tv': 'khaki', 'skateboard':
               'palegoldenrod', 'train': 'cornsilk', 'zebra': 'wheat',
               'tie': 'burlywood', 'orange': 'melon', 'bird': 'bisque',
               'dining table': 'chocolate','hair drier': 'sandybrown',
               'cell phone': 'sienna', 'sink': 'coral', 'bench':
               'salmon', 'bottle': 'brown', 'car': 'silver', 'bowl':
               'maroon', 'tennis racket': 'palevilotered', 'airplane':
               'lavenderblush', 'pizza': 'hotpink', 'umbrella':
               'deeppink', 'bear': 'plum', 'fork': 'purple', 'laptop':
               'indigo', 'vase': 'mediumpurple', 'baseball glove':
```

```
                'slateblue', 'traffic light': 'mediumblue','bed': 'navy',
                'broccoli': 'royalblue', 'backpack': 'slategray',
                'snowboard': 'skyblue', 'kite': 'cadetblue', 'teddy bear':
                'peacock', 'clock': 'lightcyan', 'wine glass': 'teal',
                'frisbee': 'aquamarine', 'donut': 'mincream', 'suitcase':
                'seagreen', 'dog': 'springgreen', 'banana':
                'emeraldgreen', 'person': 'honeydew','surfboard':
                'palegreen', 'cake': 'sapgreen', 'book': 'lawngreen',
                'potted plant': 'greenyellow', 'toaster': 'ivory',
                'stop sign': 'beige', 'couch': 'khaki'}

resized = False

def forFrame(frame_number, output_array, output_count, returned_frame):
    plt.clf()
    this_colors = []
    labels = []
    sizes = []
    counter = 0

    for eachItem in output_count:
        counter += 1
        labels.append(eachItem + " = " + str(output_count[eachItem]))
        sizes.append(output_count[eachItem])
        this_colors.append(color_index[eachItem])

    global resized

    if (resized == False):
        manager = plt.get_current_fig_manager()
        manager.resize(width=1000, height=500)
        resized = True

    plt.subplot(1, 2, 1)
    plt.title("Frame : " + str(frame_number))
    plt.axis("off")
    plt.imshow(returned_frame, interpolation="none")

    plt.subplot(1, 2, 2)
    plt.title("Analysis: " + str(frame_number))
    plt.pie(sizes, labels=labels, colors=this_colors,
            shadow=True, startangle=140, autopct="%1.1f%%")
    plt.pause(0.01)

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + "\\Model\\yolo.h5"
```

```
# Путь к файлам с видео
video_path_in = execution_path + "\\Video\\traffic.mp4"
video_path_out = execution_path + "\\Video\\video_frame_analysis"

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(model_path)
video_detector.loadModel()

plt.show()

video_detector.detectObjectsFromVideo(
    input_file_path=video_path_in,
    output_file_path=video_path_out,
    frames_per_second=20,
    per_frame_function=forFrame,
    minimum_percentage_probability=30,
    return_detected_frame=True)
```

Здесь мы подключили еще одну библиотеку — matplotlib. Используя возможности этой библиотеки, мы получим круговую диаграмму, в которой будет покадрово отображена статистика присутствия различных найденных объектов. На вход в нейронную сеть дадим тот же файл с потоком автомобилей на вечерней улице (traffic.mp4). Обработанные данные запишутся в файл video_frame_analysis.

Фрагмент визуализации процесса обработки данных представлен на рис. 7.14. Как можно видеть, в 14-м обработанном кадре видеоизображения уличного трафика нейронная сеть распознала 3 грузовика, 2 автобуса и 15 легковых автомобилей.

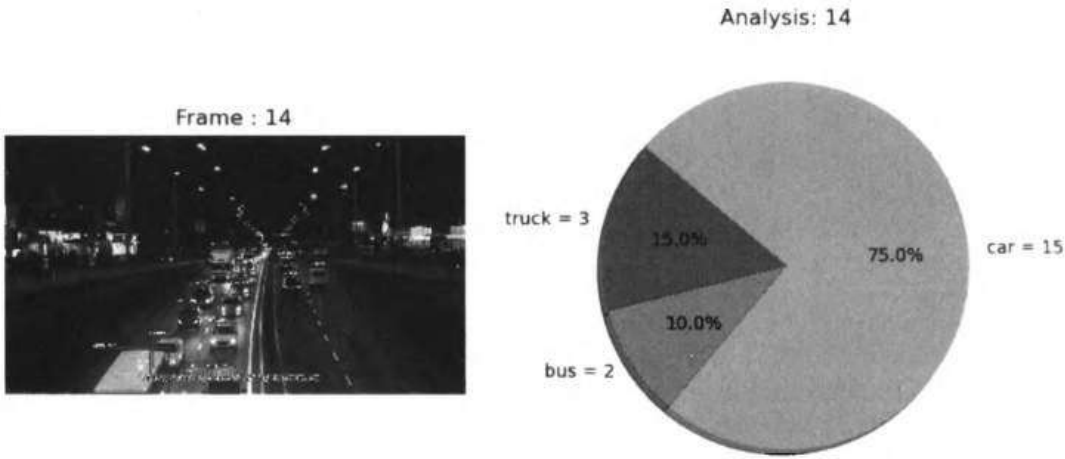


Рис. 7.14. Фрагмент изображения из обработанного видеофайла video_frame_analysis

У функции `detectObjectsFromVideo()` класса объектов `VideoObjectDetection` есть еще один, необязательный, параметр, на котором следует остановиться, — `per_second_function`. Он позволяет анализировать имя функции, которую определяет пользова-

тель. При задании этого параметра для каждой секунды процесса обработки видеопотока параметры всех обнаруженных объектов будут передаваться в эту функцию. Возвращенные данные могут быть визуализированы в процессе обработки видеоизображения или сохранены в базе данных NoSQL для дальнейшей обработки и визуализации.

В листинге 7.13 приведен пример программного кода использования указанного параметра при обработке видеоизображений.

Листинг 7.13

```
# Listing 7_13
from imageai.Detection import VideoObjectDetection
import os

def forSeconds(second_number, output_arrays, count_arrays, average_output_count):
    print("Секунда : ", second_number)
    print("Массив выходных данных каждого кадра ", output_arrays)
    print("Массив подсчета уникальных объектов в каждом кадре: ", count_arrays)
    print("Среднее количество уникальных объектов в последнюю секунду: ",
          average_output_count)
    print("------------КОНЕЦ ДАННЫХ В ЭТОЙ СЕКУНДЕ --------------")

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + "\\Model\\yolo.h5"
# Путь к файлам с видео
video_path_in = execution_path + "\\Video\\traffic.mp4"
video_path_out = execution_path + "\\Video\\video_second_analysis"

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(model_path)
video_detector.loadModel()
video_detector.detectObjectsFromVideo(
    input_file_path=video_path_in,
    output_file_path=video_path_out,
    frames_per_second=20,
    per_second_function=forSeconds,
    minimum_percentage_probability=90)
```

Примененный здесь необязательный параметр `per_second_function` дает возможность выполнить анализ данных, полученных в пользовательской функции, которая будет вызвана на исполнение после каждой секунды обработки видеоизображения. Приведенный программный код позволяет вывести следующие данные по обнаруженным объектам после каждой секунды обработки видео:

- номер текущей секунды;
- массив параметров по каждому объекту, который был обнаружен в изображении в течение этой секунды. Каждый элемент массива будет содержать следующие данные:
  - `name` — имя обнаруженного в кадре объекта;
  - `percentage_probability` — уровень достоверности (уверенности в правильности идентификации объекта) в процентах;
  - `box_points` — координаты рамки, которая обрамляет найденный объект;
- массив с именами объектов и их общим количеством, обнаруженным в изображении на этой секунде;
- среднее количество уникальных объектов, обнаруженных в изображении в течение этой секунды.

В результате работы этого программного кода будет выдан большой массив данных, фрагмент которых приведен далее:

```
Секунда :  1
Массив выходных данных каждого кадра
[[{'name': 'bus', 'percentage_probability': 97.82456159591675, 'box_points': [433, 823, 738, 1079]},
{'name': 'car', 'percentage_probability': 93.02968978881836, 'box_points': [1135, 517, 1215, 575]},
{'name': 'car', 'percentage_probability': 93.20203065872192, 'box_points': [919, 638, 1015, 703]},
{'name': 'car', 'percentage_probability': 95.3839361667633, 'box_points': [1203, 821, 1365, 970]},
{'name': 'car', 'percentage_probability': 95.71582078933716, 'box_points': [689, 766, 797, 878]},
{'name': 'car', 'percentage_probability': 97.48069643974304, 'box_points': [875, 725, 981, 833]},
{'name': 'car', 'percentage_probability': 97.79983162879944, 'box_points': [765, 656, 856, 747]},
{'name': 'car', 'percentage_probability': 98.20508360862732, 'box_points': [862, 861, 1001, 1001]},
{'name': 'car', 'percentage_probability': 98.77014756202698, 'box_points': [1149, 604, 1238, 684]}],

[{'name': 'bus', 'percentage_probability': 96.07585668563843, 'box_points': [425, 826, 741, 1073]},
{'name': 'car', 'percentage_probability': 93.98402571678162, 'box_points': [678, 778, 788, 880]},
{'name': 'car', 'percentage_probability': 96.85880541801453, 'box_points': [1185, 786, 1342, 927]},
{'name': 'car', 'percentage_probability': 97.17276096343994, 'box_points': [1145, 594, 1235, 660]},
{'name': 'car', 'percentage_probability': 97.5315809249878, 'box_points': [859, 872, 1002, 1007]},
{'name': 'car', 'percentage_probability': 97.71096110343933, 'box_points': [873, 726, 981, 836]},
{'name': 'car', 'percentage_probability': 98.77245426177979, 'box_points': [763, 664, 855, 750]}],

Массив подсчета уникальных объектов в каждом кадре:
[{'bus': 1, 'car': 8},
 {'bus': 1, 'car': 6},
 {'bus': 1, 'car': 8},
 {'bus': 1, 'car': 5},
 {'car': 5}, {'car': 6},
 {'car': 6}, {'car': 5},

Среднее количество уникальных объектов в последнюю секунду:
{'bus': 0, 'car': 5}
------------КОНЕЦ ДАННЫХ В ЭТОЙ СЕКУНДЕ --------------
```

Теперь создадим пользовательскую функцию, которая обрабатывает видеофайл, находит в нем различные объекты, в режиме реального времени визуализирует результаты этой обработки и показывает их после каждой секунды обработанного видеоизображения. Пример такого программного кода приведен в листинге 7.14.

Листинг 7.14

```
# Listing 7_14
from imageai.Detection import VideoObjectDetection
import os
from matplotlib import pyplot as plt

color_index = {'bus': 'red', 'handbag': 'steelblue', 'giraffe': 'orange',
               'spoon': 'gray', 'cup': 'yellow', 'chair': 'green',
               'elephant': 'pink', 'truck': 'indigo',
               'motorcycle': 'azure','refrigerator': 'gold',
               'keyboard': 'violet', 'cow': 'magenta',
               'mouse': 'crimson', 'sports ball': 'raspberry',
               'horse': 'maroon', 'cat': 'orchid', 'boat': 'slateblue',
               'hot dog': 'navy', 'apple': 'cobalt',
               'parking meter': 'aliceblue', 'sandwich': 'skyblue',
               'skis': 'deepskyblue', 'microwave': 'peacock',
               'knife': 'cadetblue', 'baseball bat': 'cyan',
               'oven': 'lightcyan', 'carrot': 'coldgrey',
               'scissors': 'seagreen', 'sheep': 'deepgreen',
               'toothbrush': 'cobaltgreen', 'fire hydrant': 'limegreen',
               'remote': 'forestgreen', 'bicycle': 'olivedrab',
               'toilet': 'ivory', 'tv': 'khaki', 'skateboard':
               'palegoldenrod', 'train': 'cornsilk', 'zebra': 'wheat',
               'tie': 'burlywood', 'orange': 'melon', 'bird': 'bisque',
               'dining table': 'chocolate','hair drier': 'sandybrown',
               'cell phone': 'sienna', 'sink': 'coral', 'bench':
               'salmon', 'bottle': 'brown', 'car': 'silver', 'bowl':
               'maroon', 'tennis racket': 'palevilotered', 'airplane':
               'lavenderblush', 'pizza': 'hotpink', 'umbrella':
               'deeppink', 'bear': 'plum', 'fork': 'purple', 'laptop':
               'indigo', 'vase': 'mediumpurple', 'baseball glove':
               'slateblue', 'traffic light': 'mediumblue','bed': 'navy',
               'broccoli': 'royalblue', 'backpack': 'slategray',
               'snowboard': 'skyblue', 'kite': 'cadetblue', 'teddy bear':
               'peacock', 'clock': 'lightcyan', 'wine glass': 'teal',
               'frisbee': 'aquamarine', 'donut': 'mincream', 'suitcase':
               'seagreen', 'dog': 'springgreen', 'banana':
               'emeraldgreen', 'person': 'honeydew','surfboard':
               'palegreen', 'cake': 'sapgreen', 'book': 'lawngreen',
               'potted plant': 'greenyellow', 'toaster': 'ivory',
               'stop sign': 'beige', 'couch': 'khaki'}
```

```
resized = False

def forSecond(frame_number, output_arrays, count_arrays, average_count,
              returned_frame):
    plt.clf()
    this_colors = []
    labels = []
    sizes = []
    counter = 0

    for eachItem in average_count:
        counter += 1
        labels.append(eachItem + " = " + str(average_count[eachItem]))
        sizes.append(average_count[eachItem])
        this_colors.append(color_index[eachItem])

    global resized

    if (resized == False):
        manager = plt.get_current_fig_manager()
        manager.resize(width=1000, height=500)
        resized = True

    plt.subplot(1, 2, 1)
    plt.title("Second : " + str(frame_number))
    plt.axis("off")
    plt.imshow(returned_frame, interpolation="none")

    plt.subplot(1, 2, 2)
    plt.title("Analysis: " + str(frame_number))
    plt.pie(sizes, labels=labels, colors=this_colors, shadow=True,
            startangle=140, autopct="%1.1f%%")
    plt.pause(0.01)

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path + "\\Model\\yolo.h5"
# Путь к файлам с видео
video_path_in = execution_path + "\\Video\\traffic.mp4"
video_path_out = execution_path + "\\Video\\video_second_analysis"

video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(model_path)
video_detector.loadModel()
plt.show()

video_detector.detectObjectsFromVideo(
    input_file_path=video_path_in,
```

```
    output_file_path=video_path_out,
    frames_per_second=20,
    per_second_function=forSecond,
    minimum_percentage_probability=30,
    return_detected_frame=True,
    log_progress=True)
```

Здесь мы снова подключили библиотеку matplotlib. Используя возможности этой библиотеки, мы получим круговую диаграмму, на которой будет отображена статистика присутствия различных найденных объектов на каждой секунде. На вход нейронной сети подадим тот же файл с потоком автомобилей на вечерней улице (traffic.mp4). Обработанные данные запишутся в файл video_frame_analysis. Фрагмент визуализации процесса обработки данных представлен на рис. 7.15.

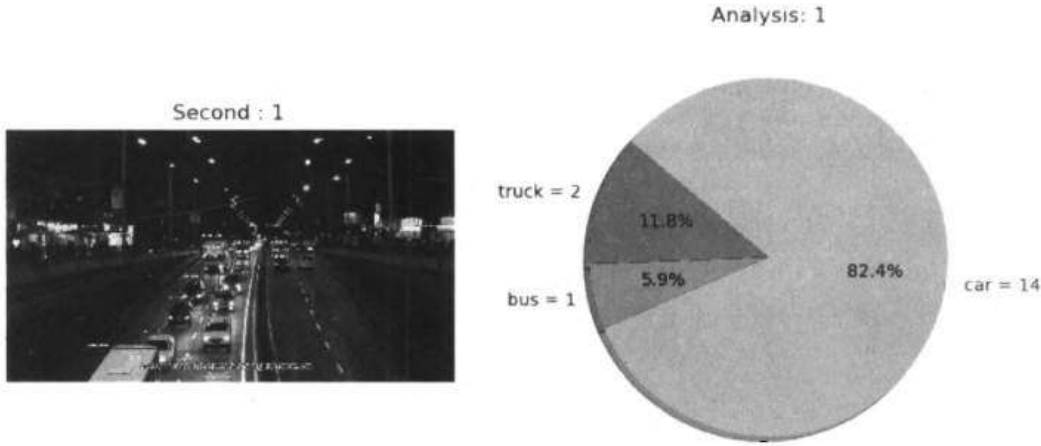


Рис. 7.15. Фрагмент изображения из обработанного видеофайла video_frame_analysis

Как можно видеть, в течение первой секунды в обработанном видеоизображении уличного трафика нейронная сеть распознала 2 грузовика, 1 автобус и 14 легковых автомобилей.

У функции `detectObjectsFromVideo()` класса объектов `VideoObjectDetection` есть необязательный параметр `per_minute_function`. Он позволяет анализировать имя функции, которую определяет пользователь. При задании этого параметра для каждой минуты процесса обработки видеопотока параметры всех обнаруженных объектов будут передаваться в эту функцию. В листинге 7.15 приведен пример программного кода использования указанного параметра при обработке видеоизображений.

Листинг 7.15

```
# Это фрагмент программного модуля, он не является рабочим
def forMinute(minute_number, output_arrays, count_arrays, average_output_count):
    print("МИНУТА : ", minute_number)
```

```
    print("Массив выходных данных каждого кадра ", output_arrays)
    print("Массив количества уникальных объектов в каждом кадре: ", count_arrays)
    print("Среднее количество объектов за минуту: ", average_output_count)
    print("------------КОНЕЦ МИНУТЫ --------------")
```

Обращение к этой функции будет выглядеть следующим образом:

```
video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "traffic.mp4"),
    output_file_path=os.path.join(execution_path, "video_second_analysis"),
    frames_per_second=20,
    per_second_function= forMinute,
    minimum_percentage_probability=30,
    return_detected_frame=True,
    log_progress=True)
```

И наконец, у функции `detectObjectsFromVideo()` класса объектов `VideoObjectDetection` есть еще один необязательный параметр — `response_timeout`. С его помощью можно указать продолжительность (количество секунд) обработки изображения с видеокамеры. По истечении указанного времени нужно прекратить обработку видеопотока. Пример использования этого параметра приведен в листинге 7.16.

Листинг 7.16

```
# Listing 7_16
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
# для встроенной камеры ноутбука
camera = cv2.VideoCapture(0)
# для подключенной внешней веб-камеры
# camera = cv2.VideoCapture(0, cv2.CAP_DSHOW)

# Путь к файлу с моделью сети
model_path = execution_path + "\\Model\\resnet50_coco_best_v2.1.0.h5"
# Путь для записи обработанного видеофайла
video_path_out = execution_path + "\\Video\\camera_detected_video"

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(model_path)
detector.loadModel()

video_path = detector.detectObjectsFromVideo(
    camera_input=camera,
    output_file_path=video_path_out,
```

```
frames_per_second=20,
log_progress=True,
minimum_percentage_probability=40,
detection_timeout=2)
```

В этом программном коде с видеокамеры будет формироваться выходной файл с именем `camera_detected_video`. Продолжительность видео в этом файле действительно 2 секунды. Однако запись этого файла в абсолютном времени длилась порядка 10 минут, поскольку через каждые 20 секунд в файл записывался 1 кадр. И так продолжалось до тех пор, пока продолжительность итогового видеоролика не стала равной 2 секундам. Всего было сформировано 39 кадров. Если задать параметр `frames_per_second=10`, то в итоговый файл запишется 19 кадров, при этом продолжительность сформированного итогового ролика останется прежней — 2 секунды.

Таким образом, два параметра (`frames_per_second` и `detection_timeout`) позволяют фиксировать в итоговом файле не сплошной видеопоток, а отдельные кадры через заданный промежуток времени. То есть мы как бы сжимаем (уплотняем) видеоизображение. В частности, в приведенном примере мы сохраняли изображение с частотой 1 кадр в 20 секунд, но при этом в каждом кадре осуществлялись поиск и распознавание объектов.

7.3. Обучение нейронных сетей на пользовательских наборах данных

Библиотека ImageAI предоставляет очень мощные, но простые в использовании классы для обучения нейронных сетей на основе собственных (пользовательских) наборов данных (изображений), используя алгоритмы глубокого обучения, такие как MobileNetV2, ResNet50, InceptionV3 и DenseNet121, При этом достаточно написать порядка пяти строк программного кода для создания собственных пользовательских моделей нейронных сетей. Обучив собственную модель на своих изображениях, вы сможете использовать класс `CustomImagePrediction` для распознавания (прогнозирования) любого изображения или набора любых изображений, которым вы обучили сеть.

7.3.1. Обучение нейронной сети на пользовательском наборе данных: класс *ClassificationModelTraining*

Класс `ClassificationModelTraining` позволяет натренировать любой из четырех поддерживаемых библиотекой ImageAI алгоритмов глубокого обучения: MobileNetV2, ResNet50, InceptionV3 и DenseNet123) —, на вашем наборе изображений. То есть у вас имеется возможность создать собственную (пользовательскую) модель нейронной сети, которая будет распознавать на изображениях заданные пользователем объекты. При этом пользовательский набор данных (изображений) должен содержать не менее двух различных классов (типов изображений) — например, кошку и

собаку. Чтобы достичь максимальной точности прогноза, для обучающей выборки данных необходимо собрать не менее 500 изображений каждого класса.

В процессе обучения создается файл формата JSON, содержащий все типы ваших объектов, распознаваемых в изображениях, и все необходимые параметры обученной модели нейронной сети. Используя этот файл, вы с достаточно высокой точностью сможете распознавать на изображениях собственные объекты.

Поскольку обучение модели является сложной задачей, настоятельно рекомендуется выполнить этот процесс, используя компьютер с графическим процессором NVIDIA и установленной версией графического процессора Tensorflow. В обычных условиях обучение модели может занять несколько часов или даже дней, но с компьютерной системой NVIDIA GPU у вас на это уйдет лишь несколько часов. Вы также можете использовать для обучения модели сервис Google Colaboratory, поскольку этот сервис работает с использованием графического ускорителя NVIDIA K80.

Сервис Google Colaboratory, или просто Google Colab — это облачный сервис, направленный на облегчение исследований в области машинного и глубокого обучения. Посредством Colab можно получить удаленный доступ к машине с подключенной видеокартой, причем совершенно бесплатно, что сильно упрощает жизнь, когда приходится обучать глубокие нейронные сети. В Colaboratory предустановлены Tensorflow и практически все необходимые для работы Python-библиотеки. Если какой-то пакет отсутствует, он с легкостью устанавливается на ходу через pip.

Для обучения модели пользовательского прогнозирования необходимо подготовить изображения, которые вы хотите использовать для обучения модели. Чтобы сформировать собственный набор обучающих изображений, надо сделать следующие шаги:

1. Создайте папку для набора данных с именем, которым будет называться ваш набор данных (например, Домашние_животные или pets).
2. В папке набора данных создайте папку с именем train.
3. В папке набора данных создайте папку с именем test.
4. В папке train создайте папку для каждого объекта, который вы хотите распознавать в изображениях, и дайте папке имя, которое будет ассоциироваться с именем распознаваемого объекта (например, собака, кошка, белка, змея).
5. В папке test также создайте папку для каждого объекта, который вы хотите распознавать в изображениях, и дайте папке имя, которое будет ассоциироваться с именем распознаваемого объекта (например, собака, кошка, белка, змея).
6. В каждую папку (собака, кошка, белка, змея), созданную в папке train, поместите изображения распознаваемых объектов. Эти изображения будут использоваться для обучения модели.

   Чтобы создать модель, которая будет хорошо работать в практических приложениях, рекомендуется разместить в папках не менее 500 изображений на каждый распознаваемый объект. А если для каждого объекта вы загрузите порядка 1000 изображений, это будет просто великолепно.

7. В каждую папку (собака, кошка, белка, змея), присутствующую в папке test, поместите от 100 до 200 изображений каждого объекта. Эти изображения будут использоваться для проверки модели во время тренировки.

После выполнения всех этих шагов структура папок с набором изображений должна выглядеть следующим образом:

- pets/train/dog/dog-train-images;
- pets/train/cat/cat-train-images;
- pets/train/squirrel/squirrel-train-images;
- pets/train/snake/snake-train-images;
- pets/test/dog/dog-test-images;
- pets/test/cat/cat-test-images;
- pets/test/squirrel/squirrel-test-images;
- pets/test/snake/snake-test-images.

Как только ваш набор данных будет готов, можно приступить к созданию экземпляра класса `ModelTraining`. Программный код для создания экземпляра этого класса выглядит следующим образом:

```
from imageai.Classification.Custom import ClassificationModelTrainer
model_trainer = ClassificationModelTrainer()
```

Для созданного экземпляра класса `ModelTraining` можно использовать следующие функции (методы).

Функция .setModelTypeAsMobileNetV2()

Эта функция устанавливает тип модели обучающего экземпляра класса — MobileNetV2. То есть обучение на вашем наборе данных будет выполняться с использованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
model_trainer.setModelTypeAsMobileNetV2()
```

Функция .setModelTypeAsResNet50()

Эта функция устанавливает тип модели обучающего экземпляра класса — ResNet50. То есть обучение на вашем наборе данных будет выполняться с использованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
model_trainer.setModelTypeAsResNet()
```

Функция .setModelTypeAsInceptionV3()

Эта функция устанавливает тип модели обучающего экземпляра класса — InceptionV3. То есть обучение на вашем наборе данных будет выполняться с ис-

пользованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
model_trainer.setModelTypeAsInceptionV3()
```

Функция .setModelTypeAsDenseNet121()

Эта функция устанавливает тип модели обучающего экземпляра класса — DenseNet121. То есть обучение на вашем наборе данных будет выполняться с использованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
model_trainer.setModelTypeAsDenseNet121()
```

Функция .setDataDirectory()

Эта функция принимает строку, которая указывает путь к папке с вашим набором изображений (в этой папке должны находиться дочерние папки test и train). Пример программного кода для задания этого пути:

```
prediction.setDataDirectory(
    r"C:/Users/Moses/Documents/Moses/AI/Custom Datasets/pets")
```

Функция может иметь следующие параметры:

- `data_directory` (обязательный) — путь к папке, содержащей ваш набор данных;
- `train_subdirectory` (необязательный) — путь к папке train вашего набора данных;
- `test_subdirectory` (необязательный) — путь к папке test вашего набора данных;
- `model_subdirectory` (необязательный) — путь к папке, в которой будут сохранены ваши обученные модели;
- `json_subdirectory` (необязательный) — путь к папке, в которой сохранен файл JSON для ваших обученных моделей.

Функция .trainModel()

Эта функция запускает тренировочный процесс (процесс обучения). После его завершения в папке с набором данных будет создан файл формата JSON (например, pets/json), содержащий итоги обучения (сопоставление классов объектов для вашего набора данных). Этот файл будет в дальнейшем использоваться в пользовательских программах для поиска ваших объектов в изображениях. Вот пример программного кода для запуска процесса обучения:

```
model_trainer.trainModel(num_objects=4, num_experiments=100,
                         enhance_data=True, batch_size=32,
                         show_network_summary=True)
```

Функция может иметь следующие параметры:

- `num_objects` (обязательный) — количество различных классов в наборе изображений;

- параметр `num_experiment` (обязательный) — количество циклов (эпох) обучения на вашем наборе данных изображений. Точность вашей натренированной модели будет расти с увеличением количества циклов обучения. Тем не менее эта точность достигает пика после определенного количества тренировок. Кроме того, этот момент зависит от размера и характера набора данных в обучающей выборке;
- `enhance_data` (необязательный) — параметр для преобразования набора изображений (для генерирования большего количества образцов для обучения). По умолчанию установлено значение `False`. Тем не менее полезно установить значение `True`, если ваш набор изображений содержит менее 1000 изображений на класс;
- `batch_size` (необязательный) — количество параллельных обработок изображений во время обучения алгоритма. Значение по умолчанию установлено равным 32. Можно увеличить или уменьшить это значение, учитывая производительность компьютерной системы, которую вы используете для обучения. Параметр должен быть кратным 8 (16, 32, 64, 128 и т. д.);
- `show_network_summary` (необязательный) — параметр, предписывающий программе отображать структуру алгоритма, который вы тренируете на своем наборе изображений, если значение параметра равно `True`. По умолчанию установлено значение `False`;
- `initial_learning_rate` (необязательный) — параметр, который определяет и контролирует поведение процесса обучения, и это имеет решающее значение для достижимой точности. Изменяйте значение этого параметра, только если вы полностью понимаете его функциональность;
- `training_image_size` (необязательный) — размер, при котором изображения в вашем наборе изображений будут обучаться (независимо от их первоначальных размеров). Значение по умолчанию равно 224, и его нельзя устанавливать меньше 100. Увеличение этого значения повышает точность, но увеличивает время обучения, и наоборот;
- `continue_from_model` (необязательный) — параметр для установки пути к файлу модели, обученному на том же наборе данных. Это связано с продолжением обучения модели, если оно было в какой-то момент остановлено;
- `transfer_from_model` (необязательный) — параметр для установки пути к файлу модели, обученному на другом наборе данных;
- `transfer_with_full_training` (необязательный) — параметр для установки предварительно обученной модели для повторного обучения на всех слоях или только на верхних слоях;
- `save_full_model` (необязательный) — параметр для сохранения обученных моделей с их типами сетей. Любая модель, сохраненная по этой спецификации, может быть загружена без указания типа сети.

Далее приведен фрагмент программного кода обучения пользовательских моделей на собственных наборах данных:

```
from imageai.Classification.Custom import ClassificationModelTrainer

model_trainer = ClassificationModelTrainer()
model_trainer.setModelTypeAsResNet50()
model_trainer.setDataDirectory(
    r"C:/Users/Moses/Documents/Moses/AI/Custom Datasets/pets")
model_trainer.trainModel(num_objects=10,
    num_experiments=100,
    enhance_data=True,
    batch_size=32,
    show_network_summary=True)
```

Рассмотрим применение этого класса на конкретном примере.

7.3.2. Пример программы обучения нейронной сети на пользовательском наборе данных

Перед обучением нейронной сети нужно собрать и подготовить соответствующий набор данных. Для нашего примера были взяты изображения с дорожными знаками двух классов: знаки «Stop» (рис. 7.16), обозначающие запрет проезда без остановки, и знаки «Zebra» (рис. 7.17), указывающие наличие пешеходного перехода. Как было сказано в предыдущем разделе, необходимо иметь от 500 до 1000 таких изображений. Здесь для демонстрации работы программы мы ограничимся 20 изображениями для объектов каждого класса.

Рис. 7.16. Изображения дорожных знаков «Stop»

Создадим в папке с нашим проектом следующую структуру папок с обучающим и тестовым наборами данных (рис. 7.18).

В папки с тренировочным (обучающим) набором данных поместим по 16 файлов изображений соответствующего класса, в папки с тестовым набором данных — по 4 файла изображения. Таким образом, мы сформировали обучающий набор данных с нашими объектами. Теперь можно приступить к формированию структуры ней-

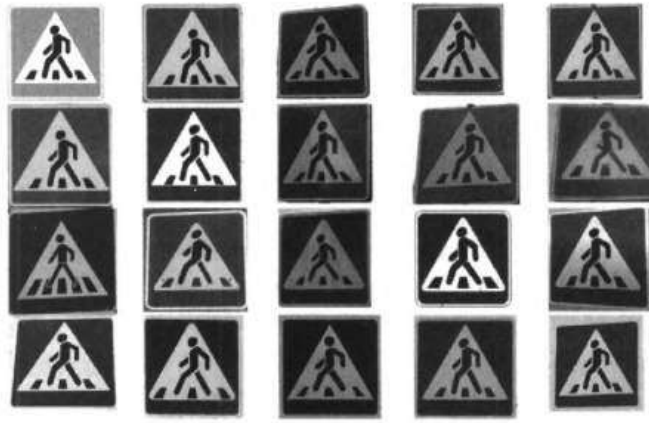


Рис. 7.17. Изображения дорожных знаков «Zebra»

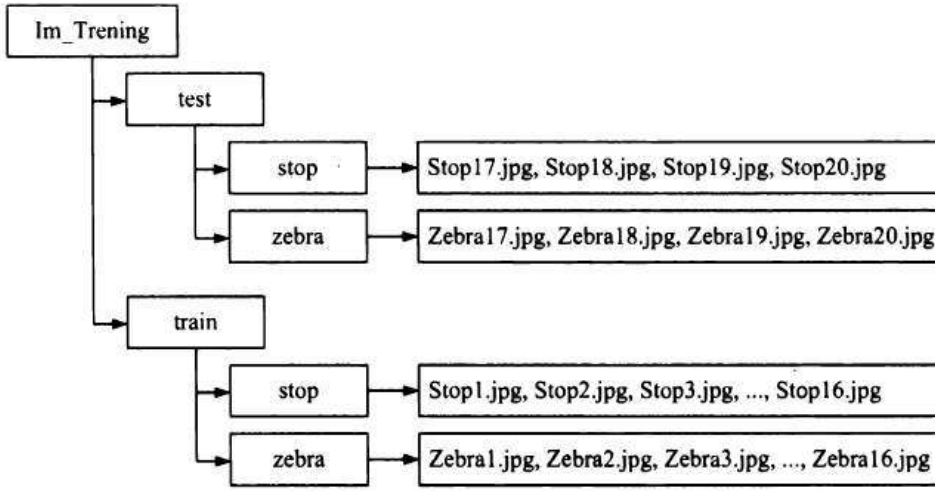


Рис. 7.18. Структура папок с обучающим и тестовым наборами данных

ронной сети и ее обучению. Для этого напишем следующий программный код (листинг 7.17).

Листинг 7.17

```
# Listing 7_17
from imageai.Classification.Custom import ClassificationModelTrainerimport os

execution_path = os.getcwd()
# Путь к обучающей выборке
data_set = execution_path + "\\Im_Trening\\"

model_trainer = ClassificationModelTrainer()
model_trainer.setModelTypeAsMobileNetV2()
```

```
model_trainer.setDataDirectory(data_set)
model_trainer.trainModel(num_objects=2,
                         num_experiments=15,
                         enhance_data=True,
                         show_network_summary=True)
```

Здесь мы создали экземпляр класса — `model_trainer`, определили тип модели нейронной сети — `MobileNetV2()`, задали путь к обучающему набору данных — `Im_Trening` и запустили процесс обучения нейронной сети — `model_trainer.trainModel`. В последней строке задали следующие параметры обучения:

- `num_objects=2` — количество классов распознаваемых объектов (у нас два класса: `Stop` и `Zebra`);
- `num_experiments=15` — количество эпох (циклов) обучения;
- `enhance_data=True` — создание дополнительных объектов в обучающей выборке;
- `show_network_summary=True` — отображение структуры алгоритма, который мы тренируем.

Запускаем процесс обучения сети. Далее приведены фрагменты результатов работы программы после начала процесса обучения:

```
Found 32 images belonging to 2 classes.
Found 8 images belonging to 2 classes.
JSON Mapping for the model classes saved to
C:\Users\Anatoly\PycharmProjects\Glava_7p8\Im_Trening\json\model_class.json
Number of experiments (Epochs) :  15
Epoch 1/15
1/1 [==============================] - 11s 11s/step - loss: 0.6873 - accuracy: 0.5000

Epoch 00001: accuracy improved from -inf to 0.50000, saving model to
C:\Users\Anatoly\PycharmProjects\Glava_7p8\Im_Trening\models\model_ex-001_acc-0.500000.h5
Epoch 2/15
1/1 [==============================] - 6s 6s/step - loss: 0.9640 - accuracy: 0.6562

Epoch 00002: accuracy improved from 0.50000 to 0.65625, saving model to
C:\Users\Anatoly\PycharmProjects\Glava_7p8\Im_Trening\models\model_ex-002_acc-0.656250.h5
Epoch 3/15
1/1 [==============================] - 6s 6s/step - loss: 0.4717 - accuracy: 0.8125

Epoch 00003: accuracy improved from 0.65625 to 0.81250, saving model to
C:\Users\Anatoly\PycharmProjects\Glava_7p8\Im_Trening\models\model_ex-003_acc-0.812500.h5
Epoch 4/15
1/1 [==============================] - 7s 7s/step - loss: 0.1752 - accuracy: 1.0000

Epoch 00004: accuracy improved from 0.81250 to 1.00000, saving model to
C:\Users\Anatoly\PycharmProjects\Glava_7p8\Im_Trening\models\model_ex-004_acc-1.000000.h5
Epoch 5/15
1/1 [==============================] - 7s 7s/step - loss: 0.1351 - accuracy: 0.9062
```

```
Epoch 00005: accuracy did not improve from 1.00000
 . . . . . .
Epoch 6/15
Epoch 15/15
1/1 [==============================] - 7s 7s/step - loss: 0.0051 - accuracy: 1.0000

Epoch 00015: accuracy did not improve from 1.00000
```

Поясним смысл сообщений, которые выводятся в процессе обучения модели:

1. Сначала была продемонстрирована структура модели нейронной сети (в процессе обучения эту структуру выводить не обязательно, и здесь мы ее показали в учебных целях);
2. Затем была отображена структура обучающей выборки:

   ```
   Using Enhanced Data Generation
   Found 32 images belonging to 2 classes.
   Found 8 images belonging to 2 classes.
   JSON Mapping for the model classes saved to  Im_Trening\json\model_class.json
   Number of experiments (Epochs):  15
   ```

   Здесь показано, что на вход в сеть при обучении подано 32 изображения в обучающих данных (двух классов) и 8 изображений тестовых данных (двух классов). Создан файл с информацией о классах изображений (файл model_class.json), количество циклов обучения — 15.
3. Затем в окно терминала выводятся данные о процессе обучения в разрезе каждой эпохи. Все эти данные помогают следить за процессом обучения, а после его завершения оценить, насколько успешно прошла тренировку модель нейронной сети и можно ли ее использовать для практических целей. По завершении обучения в папке Im_Traning будут созданы три новые папки:
   - json — для хранения классов распознаваемых объектов;
   - logs — для хранения журнала обучения;
   - models — для хранения обученных моделей.

В папке json можно посмотреть структуру файла model_class.json. В нашем случае там будут находиться всего две строки с указанием номера класса изображения и имени класса:

```
{
    "0" : "Stop",
    "1" : "Zebra"
}
```

В папка models сохранились файлы с обученной моделью нейронной сети. В нашем случае это файлы размером 92567 Кбайт с именами:

- model_ex-001_acc-0.500000.h5;
- model_ex-002_acc-0.625000.h5;

- model_ex-003_acc-0.812500.h5;
- model_ex-004_acc-1.000000.h5.

Здесь первые цифры (001) означают номер эпохи, после которой была сохранена модель, вторые цифры (0.500000–1.000000) — вероятность достоверности предсказаний обученной модели. После каждой эпохи обучения программа проверяет, улучшились ли характеристики обученной модели. Если они улучшились, то программа сохраняет улучшенный вариант обученной сети в файле .h5. В нашем примере параметры обученной модели были сохранены после 1, 2, 3 и 4-й эпох обучения.

Теперь можно испытать работу нашей обученной нейронной сети, которую мы научили распознавать дорожные знаки «Stop» и «Zebra».

7.4. Применение пользовательских нейронных сетей с библиотекой ImageAI

Как только завершится процесс обучения пользовательской модели нейронной сети, можно применять класс `CustomImagePrediction`, чтобы решать практические задачи с вашим вариантом обученной сети.

7.4.1. Поиск пользовательских объектов в изображениях: класс *CustomImageClassification*

Этот класс можно считать точной копией класса `ImageClassification` (см. *разд. 7.1.1*), поскольку он имеет все те же функции, параметры и результаты. Единственное различие состоит в том, что этот класс работает с вашей собственной обученной моделью нейронной сети. Для этого нужно будет указать путь к файлу формата JSON, сгенерированному во время обучения, и задать количество классов в наборе изображений при загрузке модели. Далее приведен пример программного кода создания экземпляра этого класса:

```
from imageai.Classification.Custom import CustomImageClassification
prediction = CustomImageClassification()
```

Создав новый экземпляр, вы можете использовать указанные далее функции (методы), чтобы установить свойство экземпляра и начать распознавать объекты на изображениях.

Функция *.setModelTypeAsResNet50()*

Эта функция устанавливает тип модели экземпляра класса для распознавания изображений — ResNet50. То есть обученная на вашем наборе данных модель будет искать объекты на изображениях с использованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsResNet50()
```

Функция .setModelTypeAsInceptionV3()

Эта функция устанавливает тип модели экземпляра класса для распознавания изображений — InecptionV3. То есть обученная на вашем наборе данных модель будет искать объекты на изображениях с использованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsInceptionV3()
```

Функция .setModelTypeAsDenseNet121()

Эта функция устанавливает тип модели экземпляра класса для распознавания изображений — DenseNet121. То есть обученная на вашем наборе данных модель будет искать объекты на изображениях с использованием именно этого алгоритма. Задать указанный тип модели можно с помощью следующего программного кода:

```
prediction.setModelTypeAsDenseNet121()
```

Функция .setModelPath()

Эта функция принимает строку, которая указывает путь к файлу модели, сгенерированному во время пользовательского обучения и соответствующему типу модели, установленному вами для экземпляра класса прогнозирования изображения. Задать этот путь можно так:

```
prediction.setModelPath("resnet_model_ex-020_acc-0.651714.h5")
```

Указание пути к загружаемому файлу обученной модели является обязательным.

Функция .setJsonPath()

Эта функция принимает строку, которая указывает путь к файлу формата JSON, сгенерированному во время обучения пользовательской модели. Задать этот путь можно так:

```
prediction.setJsonPath("model_class.json")
```

Указание пути к загружаемому файлу JSON (model_class.json) является обязательным.

Функция .loadModel()

Эта функция загружает модель из файла, который вы указали в вызове функции `setModelPath()`, в ваш экземпляр класса поиска объектов в изображениях. Инициировать загрузку можно с помощью следующего программного кода:

```
prediction.loadModel(num_objects=4)
```

Функция может иметь следующие параметры:

- `num_objects` (обязательный) — количество объектов, которые модель обучена распознавать;
- `prediction_speed` (необязательный) — строковый параметр, позволяющий до 80% сократить время, необходимое для обнаружения объектов на изображении. Од-

нако это приводит к некоторому снижению точности прогноза. Доступны следующие значения: `normal` (нормальный), `fast` (быстрый), `faster` (очень быстрый) и `fastest` (самый быстрый). Значения по умолчанию: `normal` (нормальный)

Функция *.classifyImage()*

Эта функция выполняет поиск объектов в изображении. Ее можно вызывать многократно для различных изображений, если модель уже загружена в ваш экземпляр класса прогнозирования. Программный код для выполнения этой операции:

```
predictions, probabilities = prediction.classifyImage("image1.jpg", result_count=2)
```

Функция может иметь следующие параметры:

- `image_input` (обязательный) — путь к файлу с изображением, или к массиву Numpy ваших изображений, или к потоковому видео с видеокамеры (в зависимости от указанного типа ввода);
- `result_count` (необязательный) — число возможных предсказаний, которые должны быть возвращены. Значение по умолчанию установлено равным 5;
- `input_type` (необязательный) — тип вводимых данных (это значение будет использовано в параметре `image_input`), указывает тип входного изображения. Это текстовый параметр. По умолчанию имеет значение `file` (файл), может иметь значения `array` (массив) и `stream` (поток);
- `thread_safe` (необязательный) — параметр, гарантирующий, что загруженная модель будет работать во всех потоках, если установлено значение `True`;
- `prediction_results` — возвращаемый функцией список, в котором содержатся все возможные результаты прогнозирования (найденные объекты). Результаты расположены в порядке убывания вероятности в процентах;
- `prediction_probabilities` — возвращаемый функцией список, который содержит уровень вероятности (в процентах) верности предсказаний по каждому найденному объекту.

Далее приведен пример кода для пользовательского прогноза:

```
from imageai.Classification.Custom import CustomImageClassification
import os
execution_path = os.getcwd()

prediction = CustomImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(
    os.path.join(execution_path,
    "resnet_model_ex-020_acc-0.651714.h5"))
prediction.setJsonPath(os.path.join(execution_path, "model_class.json"))
prediction.loadModel(num_objects=4)

predictions, probabilities = prediction.classifyImage(
    os.path.join(execution_path, "4.jpg"), result_count=5)
```

```
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

Рассмотрим применение этого класса на конкретном примере.

7.4.2. Пример программы поиска пользовательских объектов в изображениях

В листинге 7.18 приведен образец программного кода использования класса `imageai.Classification.Custom`.

Листинг 7.18

```
# Listing 7_18
from imageai.Classification.Custom import CustomImageClassification
import os

execution_path = os.getcwd()
# Путь к файлу с моделью сети
model_path = execution_path +\
             "\\Im_Trening\\models\\model_ex-004_acc-1.000000.h5"
# Путь к файлу json
json_path = execution_path + "\\Im_Trening\\json\\model_class.json"
# Путь к файлу с изображением
img_path = execution_path + "\\Images\\Znak_Stop.jpg"

print(model_path)
print(json_path)
print(img_path)

prediction = CustomImageClassification()
prediction.setModelTypeAsMobileNetV2()
prediction.setModelPath(model_path)
prediction.setJsonPath(json_path)
prediction.loadModel(num_objects=2)

predictions, probabilities = prediction.classifyImage(img_path, result_count=2)

for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction, " : ", eachProbability)
```

В этой программе заданы следующие параметры:

- `model_ex-004_acc-1.000000.h5` — имя файла, в котором мы сохранили обученную модель;
- `model_class.json` — имя файла, в котором сохранены наименования классов нашей обученной модели (`Stop` и `Zebra`);

- `num_objects=2` — количество классов, которые модель обучена распознавать;
- `Znak_Stop.jpg` — имя файла с изображением, которое мы будем подавать на вход в нейронную сеть;
- `result_count=2` — количество ответов о классах найденных объектов.

Для испытания работы программы будем последовательно подавать на вход нашей нейронной сети два изображения знаков: «Stop» и «Zebra» (рис. 7.19).

По завершении работы программы были получены следующие результаты:

- для знака «Stop» — `Stop: 50.61366558074951`;
- для знака пешеходного перехода «Zebra» — `Zebra: 49.38632845878601`.

То есть даже для модели, которая прошла обучение всего на 32 рисунках с пятнадцатью эпохами, мы получили достаточно высокую точность распознавания: для первого рисунка — 51,6%, для второго — 49,4%.

Теперь можно проверить работу программы на изображении, где имеются оба дорожных знака, а также другие объекты. Дадим на вход в сеть изображение, представленное на рис. 7.20.

Рис. 7.19. Изображения дорожных знаков «Stop» и «Zebra» (пешеходный переход), подаваемых на вход нейронной сети

Рис. 7.20. Изображения нескольких дорожных знаков, подаваемых на вход нейронной сети

После обработки этого изображения программа выдала следующий результат:

```
Zebra:  50.61366558074951
Stop:  49.38632845878601
```

Конечно, точность распознавания здесь не очень высокая, но тем не менее сеть нашла на изображении оба объекта, поиску которых она была натренирована. Для повышения точности нужно продолжить обучение сети с бóльшим количеством рисунков в обучающей выборке и бóльшим количеством циклов обучения.

7.5. Нейронные сети с архитектурой YOLOv3 для обнаружения объектов в изображениях

Библиотека ImageAI предоставляет простой и мощный способ обучения пользовательских моделей нейронных сетей обнаружению объектов на изображениях, основанный на использовании архитектуры YOLOv3. Это позволяет тренировать собственную модель на любом наборе изображений, который соответствует любому типу интересующего вас объекта.

Алгоритм YOLOv3 использует для обнаружения объектов сверточные нейронные сети. Это один из самых быстрых алгоритмов обнаружения объектов (хотя и не самый точный). К нему имеет смысл прибегать, когда требуется обнаружение объектов в режиме реального времени при незначительной потере точности.

По сравнению с алгоритмами распознавания алгоритм обнаружения не только предсказывает метки классов, но и определяет местоположение объектов. Таким образом, он не только классифицирует изображение в категорию, но также может обнаружить на изображении несколько объектов. Этот алгоритм применяет к полному изображению одну нейронную сеть — эта сеть делит изображение на области и показывает ограничивающие рамки, а также указывает в обработанном изображении точность обнаружения объектов для каждой выделенной области.

Работа алгоритмов в YOLO не зависит от размера входного изображения. Однако на практике желательно придерживаться постоянного размера ввода. Это связано с тем, что обработка изображений идет в пакетном режиме, где изображения обрабатываются графическим процессором параллельно (несколько изображений одновременно), что приводит к увеличению скорости обработки данных. В связи с этим объединенные в пакет изображения должны иметь фиксированную высоту и ширину. Сеть сама сокращает изображение с помощью фактора, называемого *шагом сети*. Например, если шаг сети равен 32, тогда входное изображение размером 416×416 пикселов даст на выходе изображение размером 13×13 пикселов. Естественно, что изображения меньшего размера будут обрабатываться значительно быстрее.

Появилась и новая интересная версия YOLO — YOLOv3. Это система обнаружения объектов в режиме реального времени (Real-Time Object Detection), написанная на C. Для работы с этой версией вам не понадобится большой набор картинок, где имеются фотографии объекта в разных условиях и ракурсах, — YOLOv3 тренируется на одной картинке и пытается определить этот объект везде, где только возможно.

7.5.1. Обучение пользовательской модели: класс *Custom.DetectionModelTrainer*

Класс `Custom.DetectionModelTrainer` содержит набор объектов и функций, которые позволяют обучать модели для обнаружения объектов на изображениях в формате Pascal VOC с использованием YOLOv3. В процессе обучения создается файл фор-

мата JSON, в котором сопоставлены имена объектов из набора данных с их изображениями, а также все параметры моделей.

Pascal VOC предоставляет стандартизированные наборы данных изображений для обнаружения объектов. Этот формат файлов имеет следующие особенности:

- файлы Pascal VOC — это файл XML (в отличие от COCO, который имеет файл формата JSON);
- в Pascal VOC создается отдельный файл для каждого изображения из набора данных;
- ограничительная рамка Pascal VOC имеет следующие координаты: `xmin` и `ymin` — левый верхний, `xmax` и `ymax` — правый нижний.

Для начала вам нужно подготовить свой набор данных в формате Pascal VOC и упорядочить его следующим образом:

1. Определить тип объекта (объектов), которые вы хотите обнаружить на изображениях, и собрать около 200 (минимальная рекомендация) или более изображений каждого объекта.
2. Собрав изображения, вам нужно аннотировать объект (объекты) на изображениях. Чтобы создавать аннотации для своих изображений, вы можете использовать такой инструмент, как LabelIMG).
3. Подготовив аннотации для всех своих изображений, создайте папку для набора данных .(например, headsets), а в ней — дочерние папки для обучения (train) и проверки (validation).
4. В папке обучения (train) создайте подпапки изображений (images) и аннотаций (annotations). Поместите около 70–80% вашего набора данных изображений каждого объекта в папку изображений и соответствующие аннотации для этих изображений в папку аннотаций.
5. В папке проверки (validation) создайте подпапки изображений и аннотаций. Поместите остальные изображения из набора данных в папку изображений, а соответствующие аннотации для этих изображений — в папку аннотаций.

Когда вы выполните указанные шаги, структура ваших папок с набором изображений и аннотаций к изображениям должна выглядеть так, как показано на рис. 7.21.

Вы можете обучить свою пользовательскую модель обнаружения объектов полностью с нуля или использовать трансферное обучение из предварительно обученной модели YOLOv3 (этот способ рекомендуется для получения большей точности). В этом разделе для тестирования наших собственных программ мы воспользуемся образцом аннотированного набора данных с изображениями очков виртуальной реальности Hololens. Загрузить предварительно обученную модель YOLOv3 и образцы наборов данных можно по ссылке: **https://github.com/OlafenwaMoses/ImageAI/releases/tag/essential-v4**.

Имена содержащихся там файлов, их назначение и размер приведены в табл. 7.10.

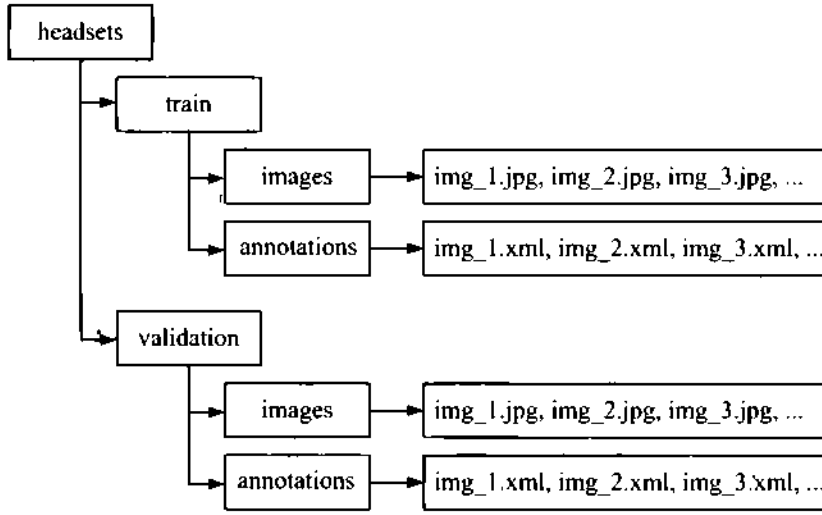


Рис. 7.21. Структура папок с набором обучающих данных

***Таблица 7.10.** Имена и назначение файлов для апробации моделей нейронной сети YOLOv3*

Наименование модели	Наименование файла	Размер, Мбайт
Предварительно обученная модель YOLOv3 для трансферного обучения новых моделей обнаружения	pretrained-yolov3.h5	237
Набор данных для обнаружения образцов гарнитур Hololens с аннотацией Pascal VOC	hololens.zip	2,24
Набор данных для обнаружения образцов гарнитур Hololens и Oculus с аннотацией Pascal VOC	headset.zip	4,37
Модель YOLOv3, обученная с ImageAI для набора данных гарнитур Hololens	hololens-ex-60-loss-2.76.h5	236
Файл конфигурации JSON для обнаружения изображений и видео с использованием обученной модели YOLOv3 для гарнитур Hololens	detection_config.json	175 Bytes

Для нашего тестового примера был скачан обучающий набор данных hololens.zip, содержащий 299 изображений. Из них 240 аннотированных изображений помещены в папку train, а тестовый набор данных из 59 аннотированных изображений — в папку validation.

В целях обучения пользовательской модели обнаружения объектов рекомендуется установить библиотеку Tensorflow-GPU v1.13.1. Это можно сделать с помощью следующей команды:

```
pip3 install tensorflow-gpu==1.13.1
```

В листинге 7.19 приведен программный код для обучения пользовательских моделей нейронных сетей обнаружению объектов в изображениях на наборе данных, созданных пользователем.

Листинг 7.19

```
# Listing 7_19
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="hololens")
trainer.setTrainConfig(object_names_array=["hololens"],
                       batch_size=4,
                       num_experiments=200,
                       train_from_pretrained_model="pretrained-yolov3.h5")
trainer.trainModel()
```

Здесь мы сначала импортировали класс `DetectionModelTrainer` и создали его экземпляр:

```
from imageai.Detection.Custom import DetectionModelTrainer
trainer = DetectionModelTrainer()
```

Затем вызвали методы этого класса, описание которых приведено далее.

Метод *.setModelTypeAsYOLOv3()*

Этот метод устанавливает тип модели обучения YOLOv3:

```
trainer.setModelTypeAsYOLOv3()
```

Метод *.trainer.setDataDirectory()*

Этот метод устанавливает путь к папке с вашим набором данных:

```
trainer.setDataDirectory()
```

Метод является обязательным.

Метод *.trainer.setTrainConfig()*

Метод устанавливает свойства для обучающего экземпляра класса и имеет следующие параметры:

- `object_names_array` (обязательный) — список имен всех различных объектов в вашем наборе данных;
- `batch_size` (необязательный) — размер пакета для обучающего экземпляра;
- `num_experiment` (обязательный) — параметр, также известный как *эпохи* (количество циклов обучения);
- `train_from_pretrained_model` (необязательный) — параметр, используемый для обучения путем указания пути к предварительно обученной модели YOLOv3.

Когда вы запустите обучающий процесс, программа выполнит следующие действия:

1. Сгенерирует файл detection_config.json в папке dataset_folder/json. Обратите внимание, что файл формата JSON, созданный во время сеанса обучения, может использоваться только с теми моделями обнаружения, которые сохранены в сеансе обучения.
2. Сохранит отчет Tensorboard о процессе обучения в папке dataset_folder/logs.
3. Сохранит — по мере того, как потери (ошибки) в процессе обучения будут уменьшаться — новые обученные модели нейронной сети в папке dataset_folder/models.

Во время обучения в окне терминала будет отображаться ход процесса обучения. Вывод этого процесса выглядит следующим образом:

```
Generating anchor boxes for training images and annotation...
Average IOU for 9 anchors: 0.78
Anchor Boxes generated.
Detection configuration saved in  hololens\json\detection_config.json
Evaluating over 59 samples taken from hololens\validation
Training over 240 samples  given at hololens\train
Training on:    ['hololens']
Training with Batch Size:  4
Number of Training Samples:  240
Number of Validation Samples:  59
Number of Experiments:  200

Epoch 1/200
480/480 [==============================] - 395s 823ms/step - loss: 36.9000 -
yolo_layer_1_loss: 3.2970 - yolo_layer_2_loss: 9.4923 - yolo_layer_3_loss: 24.1107 -
val_loss: 15.6321 - val_yolo_layer_1_loss: 2.0275 - val_yolo_layer_2_loss: 6.4191 -
val_yolo_layer_3_loss: 7.1856
Epoch 2/200
480/480 [==============================] - 293s 610ms/step - loss: 11.9330 -
yolo_layer_1_loss: 1.3968 - yolo_layer_2_loss: 4.2894 - yolo_layer_3_loss: 6.2468 -
val_loss: 7.9868 - val_yolo_layer_1_loss: 1.7054 - val_yolo_layer_2_loss: 2.9156 -
val_yolo_layer_3_loss: 3.3657
Epoch 3/200
480/480 [==============================] - 293s 610ms/step - loss: 7.1228 -
yolo_layer_1_loss: 1.0583 - yolo_layer_2_loss: 2.2863 - yolo_layer_3_loss: 3.7782 -
val_loss: 6.4964 - val_yolo_layer_1_loss: 1.1391 - val_yolo_layer_2_loss: 2.2058 -
val_yolo_layer_3_loss: 3.1514
Epoch 4/200
480/480 [==============================] - 297s 618ms/step - loss: 5.5802 -
yolo_layer_1_loss: 0.9742 - yolo_layer_2_loss: 1.8916 - yolo_layer_3_loss: 2.7144 -
val_loss: 6.4275 - val_yolo_layer_1_loss: 1.6153 - val_yolo_layer_2_loss: 2.1203 -
val_yolo_layer_3_loss: 2.6919
Epoch 5/200
480/480 [==============================] - 295s 615ms/step - loss: 4.8717 -
yolo_layer_1_loss: 0.7568 - yolo_layer_2_loss: 1.6641 - yolo_layer_3_loss: 2.4508 -
val_loss: 6.3723 - val_yolo_layer_1_loss: 1.6434 - val_yolo_layer_2_loss: 2.1188 -
val_yolo_layer_3_loss: 2.6101
```

```
Epoch 6/200
480/480 [==============================] - 300s 624ms/step - loss: 4.7989 -
yolo_layer_1_loss: 0.8708 - yolo_layer_2_loss: 1.6683 - yolo_layer_3_loss: 2.2598 -
val_loss: 5.8672 - val_yolo_layer_1_loss: 1.2349 - val_yolo_layer_2_loss: 2.0504 -
val_yolo_layer_3_loss: 2.5820
```

В последних трех строках заголовка (строки `Number of`) говорится, что для обучения будет использован набор данных из 240 тренировочных изображений, 59 тестовых изображений, количество циклов обучения — 200. Далее идут строки с параметрами обучения. Запаситесь терпением. Процесс обучения на предложенном разработчиками библиотеки ImageAI наборе данных, которые вы скачали в виде файла hololens.zip, может занять несколько часов (до 10 часов и более, в зависимости от производительности вашего компьютера). После завершения процесса обучения в папку hololens\models\ будут записаны сохраненные в процессе обучения модели обученной сети, а в папке hololens\json сохранится файл конфигурации обученной модели нейронной сети — detection_config.json.

Поскольку процесс обучения долгий, то можно не ждать его завершения и прервать выполнение программы. А для дальнейшей работы скачать уже обученную модель hololens-ex-60-loss-2.76.h5 по ссылке, приведенной ранее.

После завершения обучения вы можете оценить балл mAP (максимальная апостериорная оценка вероятности) сохраненных моделей, чтобы выбрать модель с наиболее точными результатами. Для этого просто запустите программный код, представленный в листинге 7.20.

Листинг 7.20

```
# Listing 7_20
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="hololens")
metrics = trainer.evaluateModel(model_path="hololens/models",
                                json_path="hololens/json/detection_config.json",
                                iou_threshold=0.5,
                                object_threshold=0.3,
                                nms_threshold=0.5)
print(metrics)
```

Приведенный здесь код аналогичен нашему обучающему коду, за исключением строки, в которой мы вызвали функцию `trainer.evaluateModel()`.

Функция *.trainer.evaluateModel()*

Эта функция позволяет вычислять и получать оценку mAP сохраненных моделей на основе таких критериев, как IoU и доверительная оценка:

```
trainer.setTrainConfig()
```

Функция имеет следующие параметры:

- `model_path` (обязательный) — путь к отдельной модели или к папке, содержащей сохраненные модели;
- `json_path` (обязательный) — определение пути к сгенерированному во время обучения файлу detection_config.json, в котором сохранена обученная модель;
- `iou_threshold` (необязательный) — параметр для установки желаемого пересечения по объединению для оценки mAP;
- `object_threshold` (необязательный) — параметр для установки минимального доверительного показателя для оценки mAP;
- `nms_threshold` (необязательный) — параметр для установки минимального значения подавления для оценки mAP.

Для проверки работы программы из листинга 7.20 загрузите файл hololens-ex-60--loss-2.76.h5 с моделью обученной сети в папку hololens\models\.

Запустив приведенный в листинге 7.20 код, вы получите следующий результат:

```
Model File:  hololens/models\hololens-ex-60--loss-2.76.h5
Evaluation samples:  59
Using IoU:  0.5
Using Object Threshold:  0.3
Using Non-Maximum Suppression:  0.5
hololens: 0.9930
mAP: 0.9930
================================
[{'model_file': 'hololens/models\\hololens-ex-60--loss-2.76.h5', 'using_iou':
0.5, 'using_object_threshold': 0.3, 'using_non_maximum_suppression': 0.5,
'average_precision': {'hololens': 0.9930278253807667}, 'evaluation_samples':
59, 'map': 0.9930278253807667}]
```

Как можно видеть, средняя точность предсказания *обученной модели* (average precision) — порядка 99%.

7.5.2. Обнаружение и извлечение пользовательских объектов из изображений: класс *CustomObjectDetection*

А теперь посмотрим, как будет на практике работать модель из предыдущего раздела, прошедшая обучение.

Класс `CustomObjectDetection` предоставляет очень удобные и мощные методы для обнаружения объектов на изображениях и извлечения каждого объекта из изображения с использованием собственной пользовательской модели YOLOv3 и соответствующего файла конфигурации этой модели — detection_config.json, созданного во время обучения.

Для получения навыков работы с этим классом объектов можно скачать с сайта разработчиков библиотеки ImageAI следующие образцы:

- образец пользовательской модели нейронной сети, которая обучена для обнаружения гарнитуры виртуальной реальности Hololens (файл hololens-ex-60--loss-2.76.h5);
- файл конфигурации для этой модели нейронной сети (detection_config.json);
- образец файла с изображением, на котором можно протестировать работу этой модели нейронной сети (holo1.jpg).

Соответствующие ссылки для скачивания приведены в табл. 7.11.

Таблица 7.11. *Ссылки на интернет-ресурсы с моделью нейронной сети обнаружения гарнитуры Hololens на изображениях*

Образцы	Ссылка	Наименование файла	Размер
Модель обнаружения гарнитуры Hololens	**https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/hololens-ex-60--loss-2.76.h5**	hololens-ex-60--loss-2.76.h5	241 Мбайт
Файл конфигурации модели	**https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/detection_config.json**	detection_config.json	1 Кбайт
Образец файла изображения	**https://imageai.readthedocs.io/en/latest/customdetection/index.html**	holo1.jpg	33 Кбайт

Образец изображения, на котором можно испытать эту модель нейронной сети (holo1.jpg), приведен на рис. 7.22.

Рис. 7.22. Изображение, поданное на вход обученной сети

В качестве образца можно использовать любое найденное в Интернете изображение, на котором есть гарнитуры виртуальной реальности Hololens или любые другие очки виртуальной реальности. Для испытания работы обученной модели нейронной сети напишем следующий программный код (листинг 7.21).

Листинг 7.21

```
# Listing 7_21
from imageai.Detection.Custom import CustomObjectDetection
detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("./hololens/models/hololens-ex-60--loss-2.76.h5")
detector.setJsonPath("./hololens/json/detection_config.json")
detector.loadModel()
detections = detector.detectObjectsFromImage(
    input_image="./Images/holo1.jpg",
    output_image_path="./Images/holo1-detected.jpg")
for detection in detections:
    print(detection["name"], " : ", detection["percentage_probability"],
                             " : ", detection["box_points"])
```

Файл с моделью обученной нейронной сети (hololens-ex-60--loss-2.76.h5) загрузите в дочернюю папку проекта ./hololens/model/, а файл конфигурации (detection_config.json) — в дочернюю папку проекта ./hololens/json/. Скачайте с сайта разработчика библиотеки картинку (см. рис. 7.22) или любое другое изображение, на котором есть гарнитуры Hololens (очки виртуальной реальности). Поместите файл с этим изображением в папку с созданным проектом. Когда вы запустите приведенный в листинге 7.21 программный код, программа выдаст следующий результат:

```
hololens  :  89.59426879882812  :  [27, 50, 102, 88]
hololens  :  89.89483118057251  :  [218, 70, 278, 106]
hololens  :  66.92376136779785  :  [499, 86, 588, 147]
hololens  :  92.50108599662781  :  [436, 123, 485, 152]
```

Здесь указаны обнаруженные объекты, точность предсказания объектов тем классам, для которых проводилось обучение модели, и координаты рамок, обрамляю-

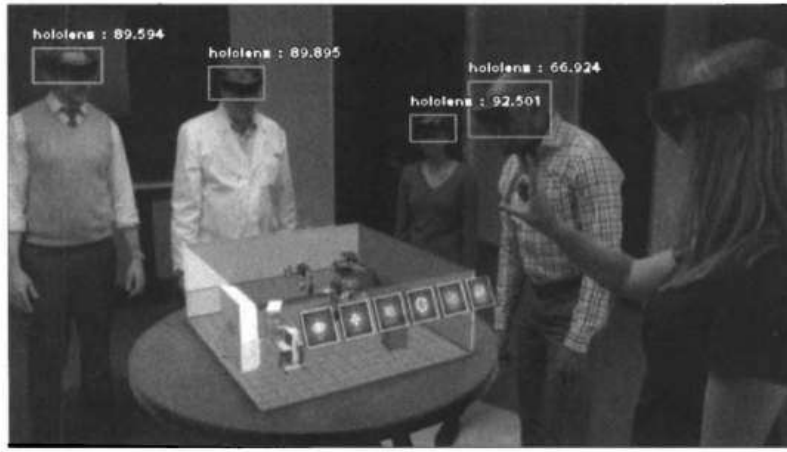


Рис. 7.23. Изображение, полученное на выходе из обученной сети

щих найденные объекты. Будет сохранено и обработанное изображение с найденными объектами (рис. 7.23). Можно также проверить работу этой модели нейронной сети на другом изображении (рис. 7.24).

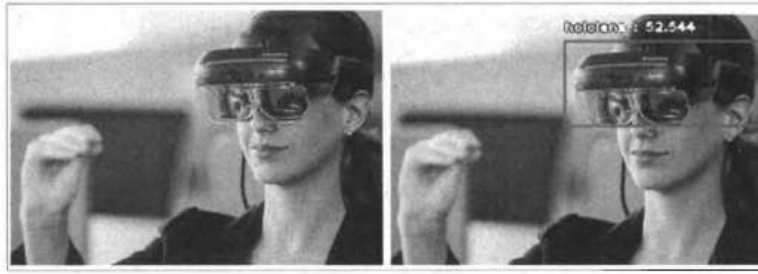


Рис. 7.24. Проверка работы нашей модели нейронной сети на другом изображении

Проанализируем приведенный в листинге 7.21 программный код и использованные в нем методы.

Метод *.setModelTypeAsYOLOv3()*

Этот метод указывает, что вы используете обученную модель YOLOv3:

```
detector.setModelTypeAsYOLOv3()
```

Метод *.setModelPath()*

Этот метод применяется для установки пути к файлу для вашей обученной модели нейронной сети:

```
detector.setModelPath()
```

Указание пути является обязательным.

Метод *.setJsonPath()*

Этот метод используется для установки пути к файлу конфигурации JSON:

```
detector.setJsonPath()
```

Указание пути является обязательным.

Метод *.loadModel()*

Этот метод позволяет загрузить модель обнаружения объектов на изображениях:

```
detector.loadModel()
```

Метод *.detectObjectsFromImage()*

Этот метод инициирует процесс обнаружения объектов на изображениях после загрузки модели. Его можно вызывать многократно для обнаружения объектов на любом количестве изображений:

```
detections = detector.detectObjectsFromImage(input_image="image.jpg",
                                             output_image_path="imagenew.jpg",
                                             minimum_percentage_probability=30)
```

Метод может иметь следующие параметры:

- `input_image` (обязательный) — путь к файлу изображения, которое вы хотите обработать. Значение по умолчанию: `file`. Вы можете установить этот параметр как массив Numpy для группы файлов любых изображений или как потоковое видео с видеокамеры. Для этого в параметре `input_type` нужно указать значение `array` или `stream`;
- `output_image_path` (определяется только в том случае, если `input_type="file"`) — путь к файлу, в котором будет сохранено обработанное изображение;
- `minimum_percentage_probability` (необязательный) — параметр для задания вероятности результатов обнаружения объектов (%). Понижение этого значения приведет к показу большего числа объектов, в то время как его увеличение обеспечивает обнаружение меньшего числа объектов с более высокой точностью предсказания. Значение по умолчанию составляет 50;
- `output_type` (необязательный) используется для установки формата, в котором будет получено обработанное изображение. Доступны значения `file` (файл) и `array` (массив). Значением по умолчанию является `file`. Если для этого параметра установлено значение `array`, функция вернет массив Numpy обнаруженного изображения. Вот пример использования этого параметра:

```
returned_image, detections = detector.detectObjectsFromImage(
                    input_image="image.jpg",
                    output_type="array",
                    minimum_percentage_probability=30)
```

- `display_percentage_probability` (необязательный) — параметр для скрытия показа точности (в процентах) обнаружения объекта (если установить его значение равным `False`). Значение по умолчанию: `True`;
- `display_object_name` (необязательный) — параметр для скрытия имени объекта, обнаруженного в изображении (если его значение установлено равным `False`). Значение по умолчанию: `True`;
- `extract_detected_objects` (необязательный) — параметр для извлечения и сохранения каждого объекта, обнаруженного в изображении (в качестве отдельного изображения). Значение по умолчанию: `False`;
- `thread_safe` (необязательный) — параметр, гарантирующий, что загруженная модель обнаружения работает во всех потоках, если установлено значение `True`;
- `return` (возвращаемые значения) — возвращаемые этим методом значения зависят от параметров, заданных в функции `deteObjectsFromImage()`.

Далее описаны различные варианты возможных значений, которые вернет эта функция.

- Если установлены все обязательные параметры, а для параметра `output_image_path` задан путь к файлу, в котором вы хотите сохранить обработанное изобра-

жение, то функция вернет массив словарей, каждый из которых соответствует найденным объектам. Каждый словарь будет содержать:

- имя объекта (строку);
- процент точности распознавания (число);
- box_points — координаты рамки, обрамляющей найденное изображение (x1, y1, x2 и y2).

Пример:

```
detections = detector.detectObjectsFromImage(input_image="image.jpg",
                                             output_image_path="imagenew.jpg",
                                             minimum_percentage_probability=30)
```

- Если все обязательные параметры установлены и output_type = 'array', функция вернет:
  - массив Numpy обработанных изображений;
  - массив словарей, каждый из которых соответствует объектам.

  Каждый словарь будет содержать:
  - имя объекта (строку);
  - процент точности распознавания (число);
  - box_points — координаты рамки, обрамляющей найденное изображение (x1, y1, x2 и y2).

  Пример:

```
detector.detectObjectsFromImage(input_image="image.jpg",
                                output_type="array",
                                minimum_percentage_probability=30)
```

- Если extract_detected_objects = True и в параметре output_image_path задан желаемый путь к итоговому файлу, то обработанное изображение будет сохранено, а функция вернет:
  - массив словарей, каждый из которых соответствует объектам, найденным на обработанном изображении. Каждый словарь будет содержать:
    - имя объекта (строку);
    - процент точности распознавания (число);
    - box_points — координаты рамки, обрамляющей найденное изображение (x1, y1, x2 и y2);
  - массив строковых путей к изображению каждого объекта, обнаруженного на обработанном изображении.

  Пример:

```
detections, extracted_objects = detector.detectObjectsFromImage(
    input_image="image.jpg",
```

```
    output_image_path="imagenew.jpg",
    extract_detected_objects=True,
    minimum_percentage_probability=30)
```

- Если `extract_detected_objects = True` и `output_type = 'array'`, то функция вернет:
  - массив Numpy обработанных изображений;
  - массив словарей, каждый из которых соответствует объектам и будет содержать:
    - имя объекта (строку);
    - процент точности распознавания (число);
    - `box_points` — координаты рамки, обрамляющей найденное изображение (`x1`, `y1`, `x2` и `y2`).

Пример:

```
returned_image, detections, extracted_objects = detector.detectObjectsFromImage(
    input_image="image.jpg",
    output_type="array",
    extract_detected_objects=True,
    minimum_percentage_probability=30)
```

В этом разделе мы познакомились с объектами, которые позволяют обрабатывать изображения. Перейдем теперь к программированию приложений, обеспечивающих обработку изображений с видеокамер.

7.5.3. Обнаружение и извлечение пользовательских объектов из видеопотоков с видеокамер: класс *CustomVideoObjectDetection*

Класс `CustomVideoObjectDetection` предоставляет очень удобные и мощные методы для обнаружения объектов на видео и получения аналитических данных из видео с использованием собственной пользовательской модели YOLOv3 и соответствующего файла конфигурации этой модели detection_config.json, созданного во время обучения.

Для получения навыков работы с этим классом объектов можно скачать с сайта разработчиков библиотеки ImageAI следующие образцы:

- образец пользовательской модели нейронной сети, которая обучена для обнаружения гарнитуры Hololens;
- файл конфигурации для этой модели нейронной сети detection_config.json;
- образец видеофайла holo1.mp4, на котором можно протестировать работу этой модели нейронной сети.

Скачать эти файлы можно с официального сайта с документацией на ImageAI по ссылке: **https://imageai.readthedocs.io/en/latest/customdetection/index.html** или по ссылкам, приведенным в табл. 7.12.

Для испытания модели напишем следующий программный код (листинг 7.22).

Таблица 7.12. *Ссылки на интернет-ресурсы с моделью нейронной сети обнаружения гарнитуры*

Образцы	Ссылка	Файл	Размер
Модель обнаружения гарнитуры Hololens	**https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/hololens-ex-60--loss-2.76.h5**	hololens-ex-60--loss-2.76.h5	241 Мбайт
Файл конфигурации модели	**https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/detection_config.json**	detection_config.json	1 Кбайт
Образец видеофайла	**https://github.com/OlafenwaMoses/ImageAI/blob/master/data-videos/holo1.mp4**	holo1.mp4	20 Мбайт

Листинг 7.22

```
# Listing 7_22
from imageai.Detection.Custom import CustomVideoObjectDetection

video_detector = CustomVideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath("./hololens/models/hololens-ex-60--loss-2.76.h5")
video_detector.setJsonPath("./hololens/json/detection_config.json")
video_detector.loadModel()

video_detector.detectObjectsFromVideo(
    input_file_path="./Video/holo.mp4",
    output_file_path="./Video/holo1-detected",
    frames_per_second=20,
    minimum_percentage_probability=40, log_progress=True)
```

Рис. 7.25. Один из кадров видеофайла, поданного на вход нейронной сети

Поместим файлы, упомянутые в табл. 7.12, в соответствующие папки проекта и запустим программу на исполнение. Один из кадров видеофайла, который будет подан на вход нейронной сети, представлен на рис. 7.25.

В процессе работы программы в окно терминала будет выводиться информация об обработанных кадрах видеофайла (рис. 7.26).

После завершения обработки видеофайла на выходе из нейронной сети будет создан новый видеофайл с именем holo1-detected.avi, на кадрах которого окажутся выделенными найденные изображения объектов. Один из кадров полученного видеофайла показан на рис. 7.27.

Далее приведено описание доступных методов в этой модели нейронной сети.

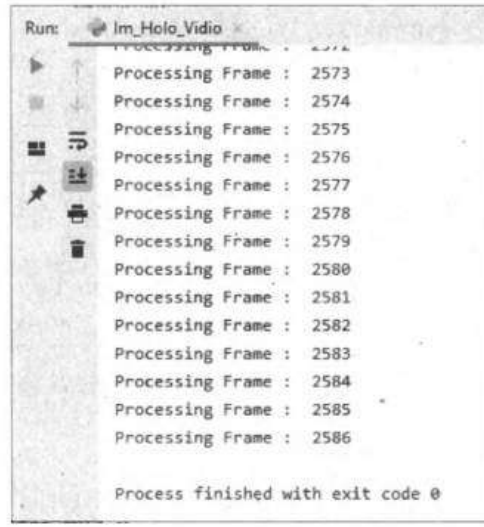


Рис. 7.26. Мониторинг процесса обработки видеофайла

Рис. 7.27. Один из кадров видеофайла, полученного на выходе из нейронной сети

Метод .setModelTypeAsYOLOv3()

Указывает, что будет использоваться обученная модель YOLOv3:

```
video_detector.setModelTypeAsYOLOv3()
```

Метод .setModelPath()

Задает путь к файлу для обученной модели пользователя (обязательный для указания в программе):

```
video_detector.setModelPath()
```

Метод .setJsonPath()

Задает путь к файлу конфигурации JSON — detection_config.json (обязательный для указания в программе):

```
video_detector.setJsonPath()
```

Метод .loadModel()

Загружает обученную модель нейронной сети пользователя:

```
video_detector.loadModel()
```

Метод .detectObjectsFromVideo()

Выполняет обнаружение объектов в видеофайле или видеопотоке в режиме реального времени.

Этот метод имеет несколько параметров:

- `input_file_path` (обязательный, если не установлен ввод с видеокамеры `camera_input`) — путь к видеофайлу, который вы хотите обрабатывать;
- `output_file_path` (обязательный, если не установлен `save_detected_video = False`) — путь, по которому будет сохранено обработанное видео. По умолчанию видео сохраняется в формате AVI;
- `frames_per_second` (необязательный, но рекомендуется) — параметр, позволяющий установить желаемую частоту кадров в секунду для обработанного видео, которое будет сохранено. Значение по умолчанию: 20;
- `log_progress` (необязательный) — параметр, задающий отображение хода обработки видео в режиме прямой трансляции, если установлен равным `True`. Сообщается о каждом кадре по мере обработки видео. Значение по умолчанию: `False`;
- `return_detected_frame` (необязательный) — параметр, позволяющий возвращать обработанные кадры в виде массива Numpy с указанием секунды и минуты в обработанном видео. Полученный массив Numpy можно позднее проанализировать в функциях: `per_frame_function()`, `per_second_function()` и `per_minute_function()` (подробности см. далее);
- `camera_input` (необязательный) — параметр, который можно установить вместо `input_file_path`, если вы хотите обрабатывать видео с видеокамеры в режиме

прямого эфира. Для этого нужно активировать камеру с помощью функции `videoCapture()` из библиотеки OpenCV;

- `minimum_percentage_probability` (необязательный) — параметр для определения целостности результатов обнаружения. Понижение параметра приведет к показу большего числа обнаруженных объектов, в то время как его увеличение обеспечивает обнаружение меньшего числа объектов с более высокой точностью предсказания. Значение по умолчанию составляет 50;
- `display_percentage_probability` (необязательный) — параметр для скрытия точности распознавания каждого объекта, обнаруженного в обработанном видео (если установить значение `False`). Значение по умолчанию: `True`;
- `display_object_name` (необязательный) — параметр для скрытия имени каждого объекта, обнаруженного в обработанном видео (если установить значение `False`). Значение по умолчанию: `True`;
- `save_detected_video` (необязательный) — параметр для того, чтобы сохранить (`True`) или не сохранять (`False`) обработанное видео. Значение по умолчанию: `True`;
- `per_frame_function` (необязательный) — имя функции, которую определяет пользователь. Позднее каждый обработанный кадр видео можно передать этой функции. Данные, возвращенные из нее, могут быть визуализированы или сохранены в базе данных NoSQL для дальнейшей обработки и визуализации;
- `per_second_function` (необязательный) — имя функции, которую определит пользователь. Позднее каждую секунду обработанного видеофайла можно передать этой функции. Возвращенные данные могут быть визуализированы или сохранены в базе данных NoSQL для дальнейшей обработки и визуализации;
- `per_minute_function` (необязательный) — имя функции, которую определит пользователь. Позднее каждую минуту обработанного видеофайла можно передать этой функции. Возвращенные данные могут быть визуализированы или сохранены в базе данных NoSQL для дальнейшей обработки и визуализации. Возвращенные данные имеют ту же структуру, что и в `per_second_function`, — разница лишь в том, что будут охвачены все кадры за последнюю минуту видео;
- `video_complete_function` (необязательный) — имя функции, которую определит пользователь. Как только все кадры в видео будут полностью обработаны, они будут переданы этой функции. Возвращенные данные имеют ту же структуру, что и в `per_second_function` и `per_minute_function`, — разница лишь в том, что индексы массивов не будут возвращены, т. е. результат обработки будет охватывать все кадры во всем видео;
- `response_timeout` (необязательный) — количество секунд обрабатываемого видео, после которых функция обработки видео должна прекратить свою работу.

Пример с указанной пользовательской функцией в параметре `per_frame_function` (вывод информации о каждом обработанном кадре видеоизображения) приведен в листинге 7.23.

Листинг 7.23

```
Listing 7_23
# Это фрагмент программы, он не является рабочим
def forFrame(frame_number, output_array, output_count):
print("КАДР " , frame_number)
print("Массив уникальных объектов в кадре: ", output_array)
print("Количество найденных объектов: ", output_count)
print("———КОНЕЦ КАДРА ———-")

video_detector.detectObjectsFromVideo(
    input_file_path="holo1.mp4",
    output_file_path=os.path.join(execution_path, "holo1-detected"),
    frames_per_second=30,
    minimum_percentage_probability=40,
    per_frame_function(forFrame),
    return_detected_frame(True),
    log_progress=True)
```

В этой функции после каждого обработанного видеокадра будут выводиться:

- номер кадра;
- информация о найденных объектах в виде массива Numpy. Каждый элемент массива соответствует обнаруженному объекту и содержит:
  - имя найденного объекта;
  - процент точности соответствия объекта образцу объектов этого типа;
  - `box_points` — координаты рамки, обрамляющей объект;
- количество экземпляров каждого найденного объекта.

Пример с пользовательской функцией в параметре `per_second_function` (вывод информации о каждой обработанной секунде в видеоизображении) приведен в листинге 7.24.

Листинг 7.24

```
Listing 7_24
# Это фрагмент программы, он не является рабочим
def forSeconds(second_number, output_arrays, count_arrays, average_output_count):
    print("СЕКУНДА : ", second_number)
    print("Массив выходных данных об объектах каждого кадра ", output_arrays)
    print("Массив уникальных объектов в кадрах: ", count_arrays)
    print("Среднее количество объектов в течение секунды: ", average_output_count)
    print("———КОНЕЦ СЕКУНДЫ ———-")

video_detector.detectObjectsFromVideo(
    input_file_path="holo1.mp4",
    output_file_path=os.path.join(execution_path, "holo1-detected"),
```

```
    frames_per_second=30,
    minimum_percentage_probability=40,
    return_detected_frame(True),
    per_frame_function(forSeconds),
    log_progress=True)
```

После каждой обработанной секунды видеоизображения эта функция будет выводить:

- номер (значение) текущей секунды в обработанном видеоизображении;
- массив Numpy выходных данных о найденных объектах каждого кадра в текущей секунде. Каждый элемент массива соответствует кадрам в текущей секунде. Каждый кадр содержит следующую информацию о найденных объектах:
  - имена найденных объектов;
  - количество уникальных объектов, обнаруженных в каждом кадре;
- массив Numpy (словарь), ключами которого являются имена уникального объекта, обнаруженного в течение последней секунды, а значениями ключей — среднее число объектов, найденных во всех кадрах, содержащихся в последней секунде.

Пример с пользовательской функцией в параметре `per_minute_function` (вывод информации о каждой обработанной минуте в видеоизображении) приведен в листинге 7.25.

Листинг 7.25

```
# Listing 7_25
# Это фрагмент программы, он не является рабочим
def forMinute(minute_number, output_arrays, count_arrays, average_output_count):
    print("МИНУТА: ", minute_number)
    print("Массив выходных данных каждого кадра ", output_arrays)
    print("Массив количества уникальных объектов в каждом кадре: ", count_arrays)
    print("Среднее количество уникальных объектов за данную минуту: ",
          average_output_count)
    print("------------КОНЕЦ МИНУТЫ --------------")
```

Пример с пользовательской функцией в параметре `video_complete_function` (вывод информации о найденных объектах во всем видеоизображении) приведен в листинге 7.26.

Листинг 7.26

```
# Listing 7_26
# Это фрагмент программы, он не является рабочим
def forFull(output_arrays, count_arrays, average_output_count):
    print("Массив выходных данных для всего изображения ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
```

```
print("Вывод среднего количества уникальных объектов во всем видео:
      ", average_output_count)
print("------------КОНЕЦ ВИДЕО --------------")
```

Итак, мы познакомились с тем, как использовать класс `CustomVideoObjectDetection` на примере уже готовой обучающей выборки. Однако нам на практике нужно уметь создавать собственные наборы обучающих и тестовых данных и использовать их для тренировки своих нейронных сетей. Следующий раздел как раз будет посвящен этой теме.

7.5.4. Формирование пользовательского обучающего набора данных: приложение LabelImg

Теперь у нас есть все необходимое для того, чтобы создать нейронную сеть и обучить ее распознаванию объектов в изображениях на пользовательском наборе данных. Но для начала работы нужно сформировать собственный набор данных. Мы реализуем нейронную сеть, которая будет распознавать на изображениях два дорожных знака: «Stop» и «Zebra» (пешеходный переход).

На первом шаге нужно сформировать набор обучающих данных. Для нашего набора данных было взято 20 изображений дорожных знаков «Stop» и 20 изображений дорожных знаков «Zebra» (рис. 7.28).

Рис. 7.28. Изображения дорожных знаков «Stop» и «Zebra» (пешеходный переход)

Поскольку нейронная сеть будет обрабатывать изображения в пакетном режиме, то желательно, чтобы они имели одинаковый размер. С этой целью все рисунки были обработаны в графическом редакторе, и им был задан размер изображений 416×416 пикселов. Впрочем, это делать не обязательно: если посмотреть на обучающую выборку, скачанную с сайта официальной документации ImageAI, то там все рисунки имеют разные размеры.

Следующий важный шаг — это аннотирование изображений. Оно заключается в создании специального XML-файла, в котором описаны параметры исходного изображения и заданы координаты рамки, обрамляющей объект поиска (в нашем случае это дорожные знаки). Для аннотирования рисунков имеются специальные программные средства, и мы здесь воспользуемся инструментом для аннотирования графических изображений LabelImg. Он написан на Python и использует библиотеку Qt для своего графического интерфейса. Аннотации сохраняются в виде файлов XML в формате Pascal VOC. Кроме того, инструмент также поддерживает формат YOLO. Посмотреть информацию о приложении LabelImg можно по ссылке: **https://github.com/tzutalin/labelImg**.

Для аннотирования изображений создадим новый проект Label_Img и загрузим в него пакет LabelImg.

Чтобы установить приложение LabelImg, нужно выполнить следующие команды:

```
easy_install -U pip
pip install labelImg
```

Первая команда обновляет pip, вторая — загружает пакет LabelImg, при этом будет также установлен и ряд дополнительных библиотек. После этого в среде Python или PyCharm в окне терминала можно запустить это приложение командой:

```
labelImg
```

В каталоге с проектом создадим две папки: Images — для исходных изображений, и Annotations — для файлов с их аннотациями. В каталоге Images создадим еще две вложенные папки: Stop — для изображений со знаками «Stop», и Zebra — для изображений со знаками пешеходных переходов. В эти папки перенесем все наши файлы изображений дорожных знаков. Теперь можно приступить к аннотированию изображений.

После запуска программы будет открыто главное окно приложения (рис. 7.29).

Прежде всего надо задать каталог, в котором будут сохраняться файлы аннотаций обрабатываемых изображений, — нажмите на панели инструментов кнопку **Change Save Dir** и выберите папку для сохранения этих файлов (рис. 7.30).

В результате откроется окно, в котором можно выбрать имя папки для сохранения аннотаций изображений. Для нашего примера в качестве имени папки выберем Annotations (рис. 7.31). Затем нужно открыть каталог, в котором находятся файлы обрабатываемых изображений, нажав кнопку **Open Dir** на панели инструментов (рис. 7.32).

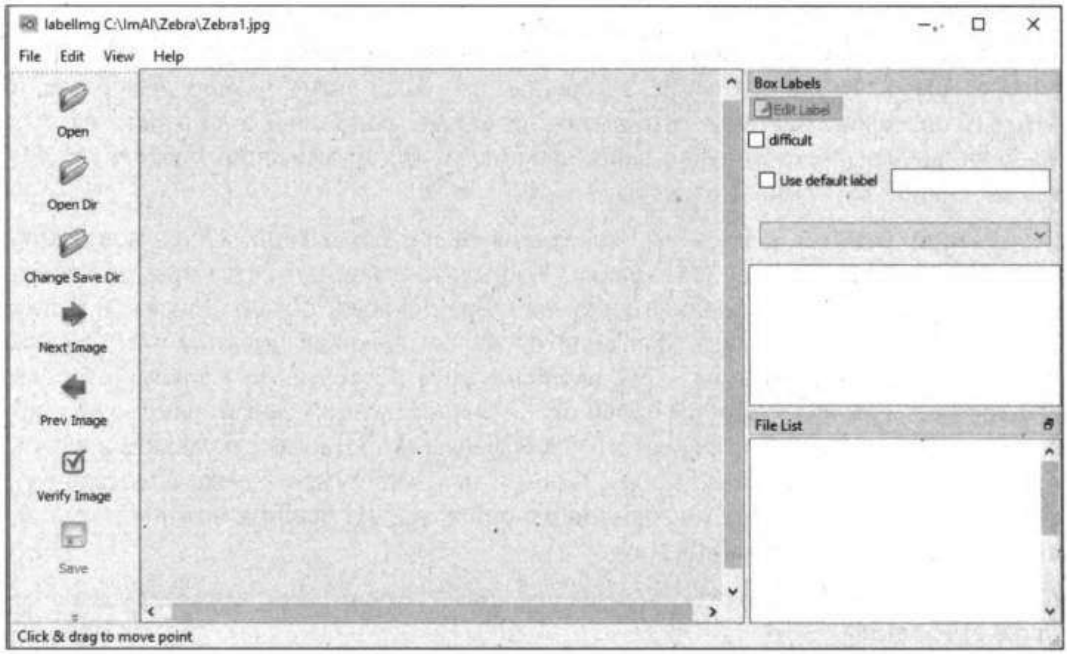


Рис. 7.29. Главное окно приложения LabelImg

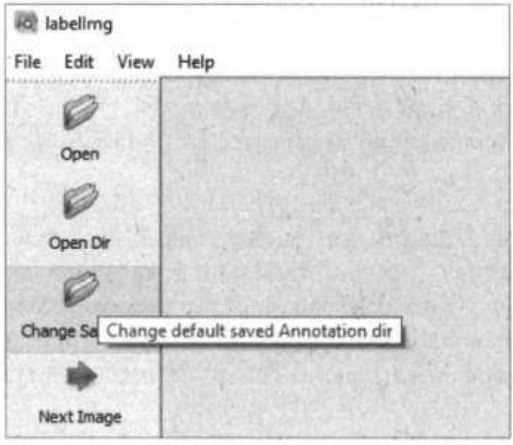


Рис. 7.30. Вызов функции для выбора папки, в которой будут сохраняться файлы аннотаций изображений

Когда вы укажете папку с рисунками и нажмете кнопку **Выбор папки**, имена всех файлов ваших изображений появятся в правой нижней части главного окна в панели **File List** (рис. 7.33).

Выполните в панели **File List** двойной щелчок на имени первого файла в списке — выбранное изображение появится в центральном окне главного экрана, в котором оно и будет обрабатываться (рис. 7.34).

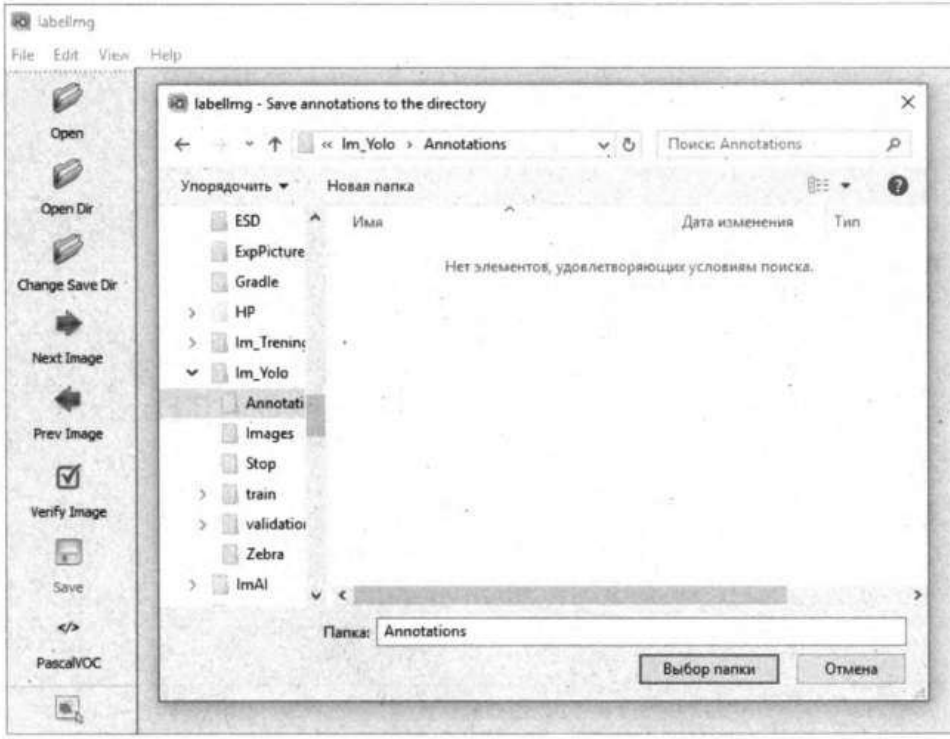


Рис. 7.31. Выбор папки, в которой будут сохраняться файлы аннотаций изображений

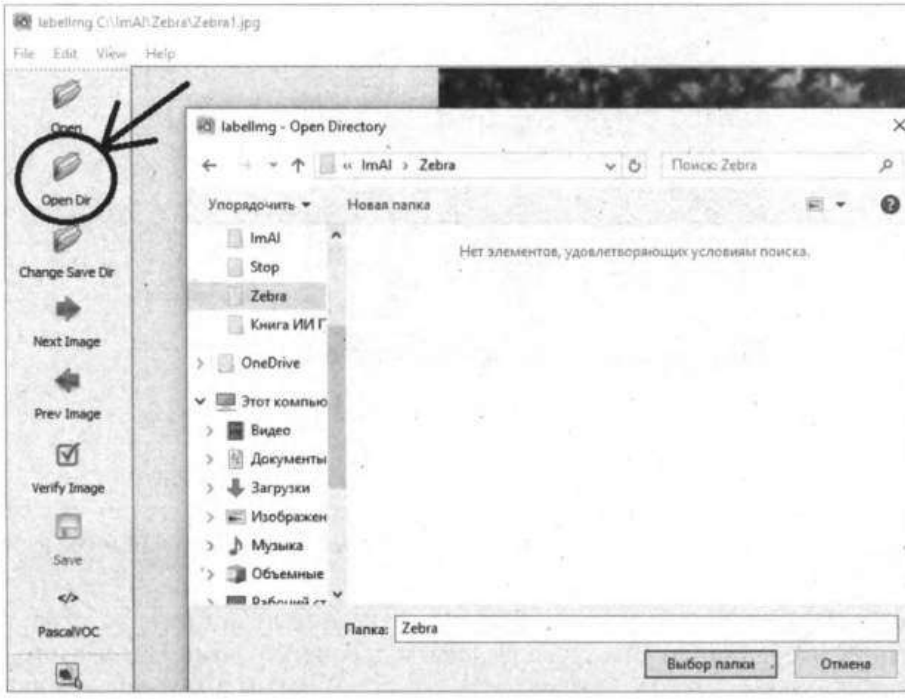


Рис. 7.32. Задание каталога с файлами обрабатываемых изображений

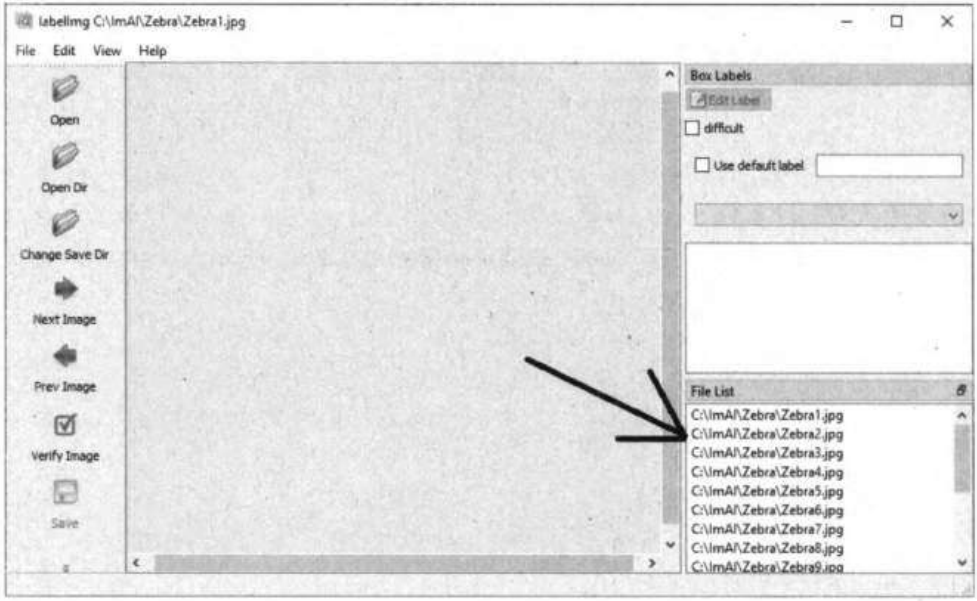


Рис. 7.33. Список файлов обрабатываемых изображений

Рис. 7.34. Рисунок, загруженный в окно обработки изображений

Нажмите теперь кнопку **Create\nRecBox** на панели инструментов (рис. 7.35) — в результате на обрабатываемом изображении активируется функция обрамления объекта, а на самом изображении появятся две линии: вертикальная и горизонтальная (рис. 7.36).

Подведите мышью эти две линии к левому верхнему углу объекта, который нужно выделить на изображении (мы здесь выделяем дорожный знак). После этого, удерживая левую кнопку мыши, переместите указатель мыши в правый нижний угол выделяемого объекта — вокруг объекта на изображении появится выделяющая

Рис. 7.35. Кнопка активации функции обрамления объекта на изображении

Рис. 7.36. Обрамление объекта на изображении

рамка (рис. 7.37). Когда вы отпустите кнопку мыши, появится всплывающее окно, в котором нужно указать имя выделенного объекта (рис. 7.38).

В нашем примере мы будем выделять на изображениях дорожные знаки и при этом сформируем на изображениях два типа объектов. Всем объектам, относящимся к дорожным знакам «Stop», зададим имя `stop`. А всем объектам, относящимся к дорожным знакам «пешеходный переход», зададим имя `zebra`.

После того как выделенному объекту будет присвоено имя, оно появится в среднем окне правой панели инструментов. На этом процесс обработки изображения закончен. Теперь нужно сохранить результаты этой работы. Для этого следует нажать кнопку **Save** на левой панели инструментов (рис. 7.39). В результате файл аннотации изображения будет сформирован, а затем сохранен в каталоге, который был указан в папке для сохранения аннотаций.

Рис. 7.37. Рамка выделения объекта на изображении

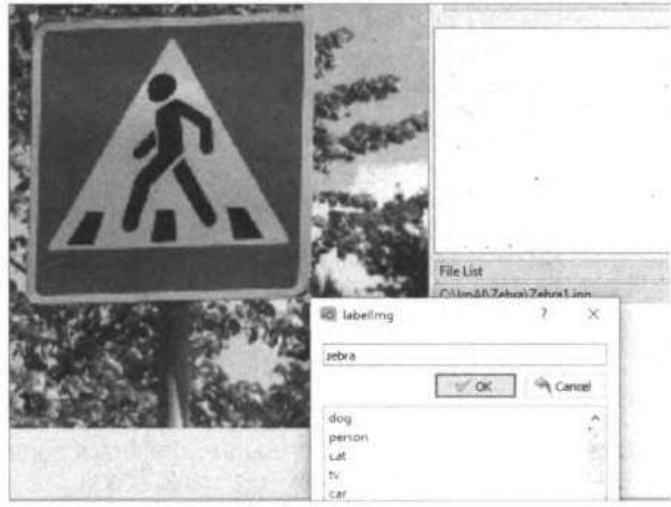


Рис. 7.38. Указание имени обрамленного объекта на изображении

Теперь можно перейти к обработке следующего изображения — для этого достаточно нажать кнопку **Next Image** на левой панели инструментов (рис. 7.40).

После обработки всех изображений обучающей выборки данных все XML-файлы с аннотациями появятся в папке, которая была задана в каталоге при нажатии кнопки **Change Save Dir** (рис. 7.41).

Структура файла аннотации изображения показана на рис. 7.42. В аннотации изображения указаны имя и размер файла с изображением, имя объекта, который выделен на изображении, координаты рамки, обрамляющей выделенный объект.

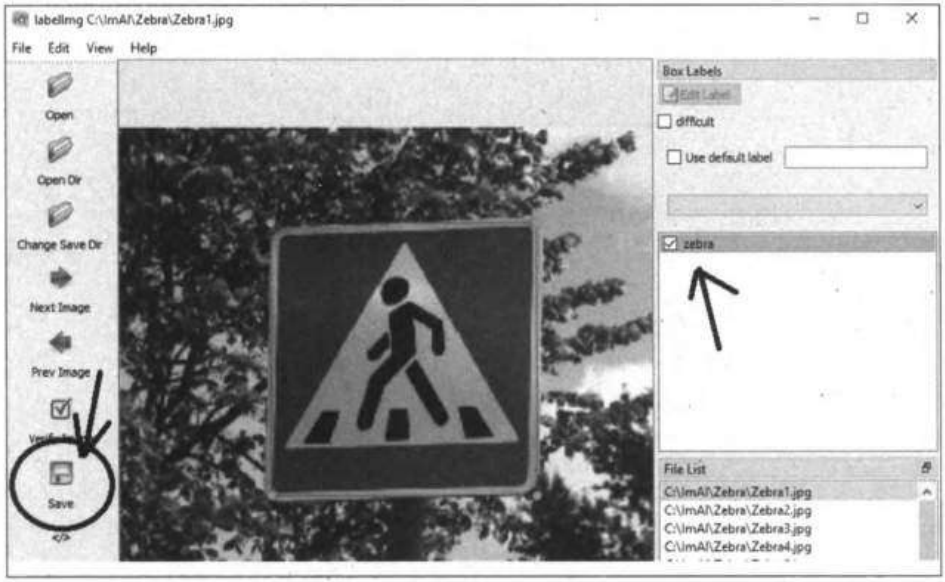


Рис. 7.39. Сохранение параметров объекта на изображении

Рис. 7.40. Переход к обработке следующего изображения

На этом мы завершили подготовку файлов с изображениями. Теперь сформируем обучающую выборку для обучения модели нашей нейронной сети. Для этого в проекте, где находится библиотека ImageAI, сформируем папки и поместим в них обучающие файлы с рисунками и аннотациями так, как показано на рис. 7.43.

В настоящее время появился достаточно мощный инструмент с открытым исходным кодом для аннотирования и маркировки обучающих данных — Label Studio.

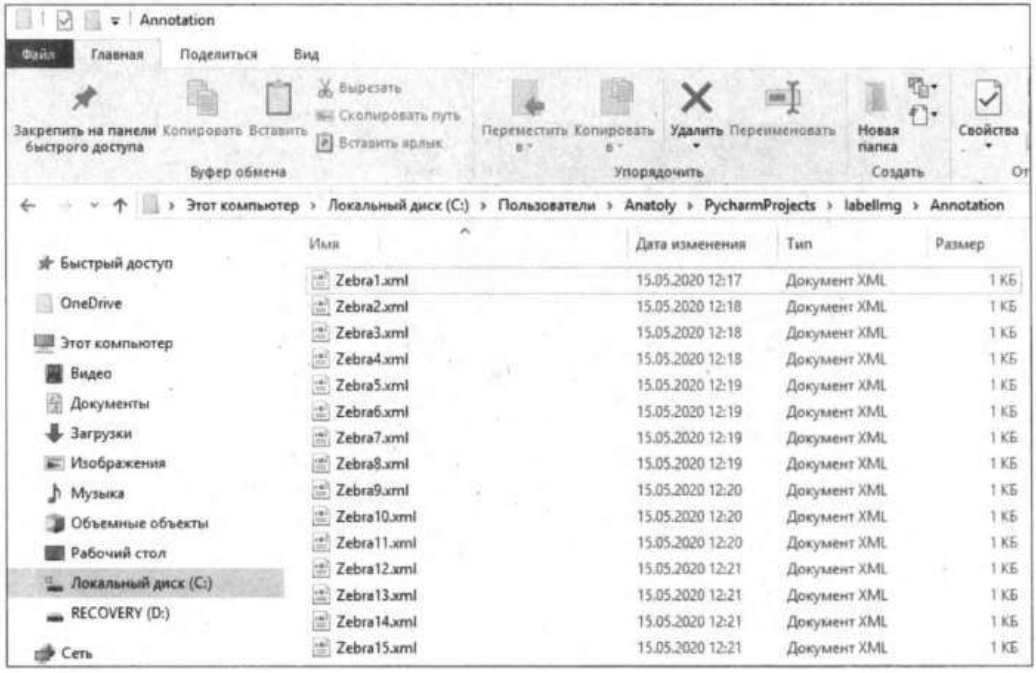


Рис. 7.41. Файлы с аннотацией изображений

labelImg.py Stop1.xml

```
<filename>Stop1.jpg</filename>
<path>C:\Im_Yolo\Stop\Stop1.jpg</path>
<source>
    <database>Unknown</database>
</source>
<size>
    <width>416</width>
    <height>416</height>
    <depth>3</depth>
</size>
<segmented>0</segmented>
<object>
    <name>stop</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>65</xmin>
        <ymin>69</ymin>
        <xmax>344</xmax>
        <ymax>348</ymax>
    </bndbox>
</object>
</annotation>
```

Рис. 7.42. Структура файла аннотации изображения

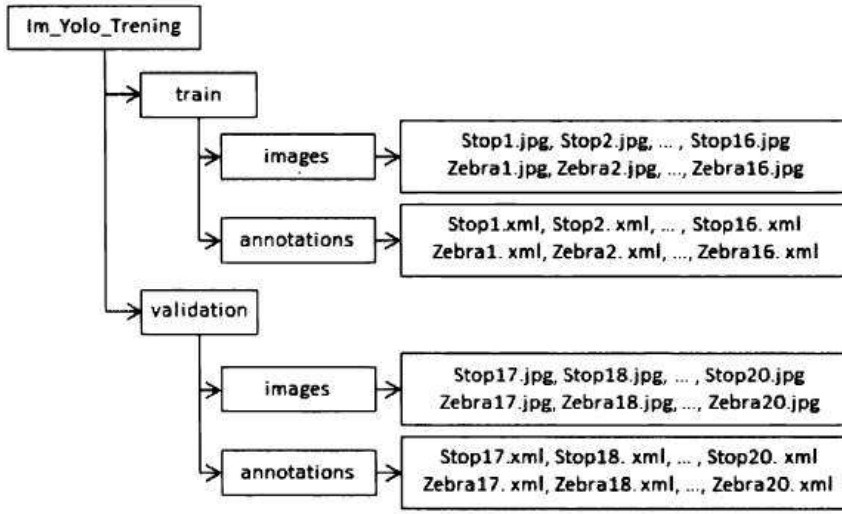


Рис. 7.43. Структура папок с изображениями и аннотациями

Он позволяет маркировать различные типы данных, такие как аудио, текст, изображения, видео и временные ряды, с помощью простого и понятного пользовательского интерфейса и экспортировать их в различные форматы моделей. Его можно использовать для подготовки необработанных данных или улучшения существующих обучающих данных с целью получения более точных моделей машинного обучения. Описание инструмента Label Studio — это, по сути, отдельная книга, но вы можете ознакомиться с ним по следующей ссылке: **https://github.com/heartexlabs/label-studio**.

7.5.5. Пример программы обучения модели YOLOv3 на пользовательском наборе данных

Для формирования и обучения пользовательской нейронной сети напишем следующий программный код (листинг 7.27).

Листинг 7.27

```
# Listing 7_27
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="Im_Yolo_Trening")
trainer.setTrainConfig(object_names_array=["stop", "zebra"],
                       batch_size=4,
                       num_experiments=2,
                       train_from_pretrained_model="./Model/pretrained-yolov3.h5")
trainer.trainModel()
```

В процессе работы программы в окно терминала будут выводиться сообщения о процессе обучения сети. Вот фрагмент этих сообщений:

```
Generating anchor boxes for training images and annotation...
Average IOU for 9 anchors: 0.93
Anchor Boxes generated.
Detection configuration saved in  Im_Yolo_Trening\json\detection_config.json
Training on:    ['stop', 'zebra']
Training with Batch Size:  4
Number of Experiments:  2
Training with transfer learning from pretrained Model
Epoch 1/2

 1/64 [..............................] - ETA: 42:20 - loss: 194.7140 -
yolo_layer_1_loss: 29.0981 - yolo_layer_2_loss: 64.0022 - yolo_layer_3_loss: 101.6136
 2/64 [..............................] - ETA: 31:27 - loss: 194.1556 -
yolo_layer_1_loss: 29.1622 - yolo_layer_2_loss: 63.8883 - yolo_layer_3_loss: 101.1051
 3/64 [>.............................] - ETA: 27:28 - loss: 193.4392 -
yolo_layer_1_loss: 29.0708 - yolo_layer_2_loss: 63.7071 - yolo_layer_3_loss: 100.6613
63/64 [============================>.] - ETA: 15s - loss: 39.2451 - yolo_layer_1_loss:
4.5166 - yolo_layer_2_loss: 12.9417 - yolo_layer_3_loss: 21.7867
64/64 [==============================] - 1075s 17s/step - loss: 39.0961 -
yolo_layer_1_loss: 4.4973 - yolo_layer_2_loss: 12.9079 - yolo_layer_3_loss: 21.6909 -
val_loss: 48.5721 - val_yolo_layer_1_loss: 7.1310 - val_yolo_layer_2_loss: 13.4748 -
val_yolo_layer_3_loss: 21.9659
```

По завершении процесса тренировки в папке json будет сформирован файл detection_config.json конфигурации обученной сети. Он имеет следующее содержание:

```
{
    "labels" : [ "stop", "zebra"],
    "anchors" : [
        [352, 354, 359, 357, 376, 266],
        [247, 237, 273, 281, 319, 316],
        [    0,     0, 104, 101, 191, 192]]
}
```

А в папке models будут сохранены файлы с расширением h5 — это модели обученных сетей на разных стадиях (эпохах) обучения. Например:

- detection_model-ex-001--loss-0108.459.h5;
- detection_model-ex-002--loss-0039.096.h5.

7.5.6. Пример программы распознавания с помощью пользовательской модели YOLOv3

Проверим работу нашей обученной нейронной сети, которую мы натренировали на распознавание дорожных знаков «Stop» и «Zebra» (пешеходный переход). Для этого напишем следующий программный код (листинг 7.28).

Листинг 7.28

```
# Listing 7_28
# Модуль Detect_Znaki
from imageai.Detection.Custom import CustomObjectDetection
from imageai.Detection.Custom import CustomObjectDetection

detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("detection_model-ex-002--loss-0041.077.h5")
detector.setJsonPath("detection_config.json")
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image="Znak_Zebra.jpg",
                    output_image_path="Znak_Zebra_New.jpg",)
for detection in detections:
    print(detection["name"], " : ", detection["percentage_probability"],
          " : ", detection["box_points"])
```

Программа отказалась работать с последними версиями библиотеки tensorflow. Пришлось открыть новый проект и в нем создать виртуальную среду выполнения программ библиотеками более ранних версий. Для этого были подключены следующие версии библиотек tensorflow:

- tensorflow 1.13.1;
- tensorflow-gpu 1.13.1.

После того как программа отработала с изображением знака «Zebra», был получен следующий ответ:

```
zebra  :  64.24534320831299  :  [106, 0, 143, 297]
```

Изображение, поданное на вход нейронной сети, и изображение, полученное после его обработки, приведены на рис. 7.44.

Конечно, уровень достоверности распознавания объекта на изображении, равный 64%, недостаточно высок. Однако нужно иметь в виду, что наш упрощенный обучающий набор данных содержал всего 20 изображений, а сеть прошла лишь две эпохи обучения.

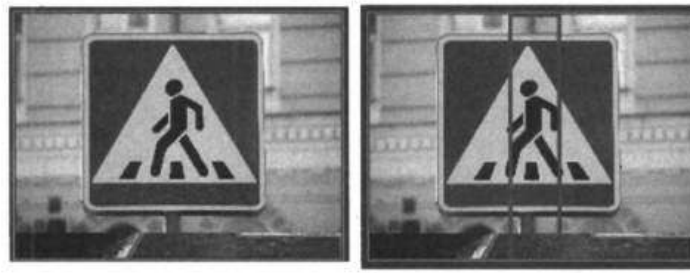
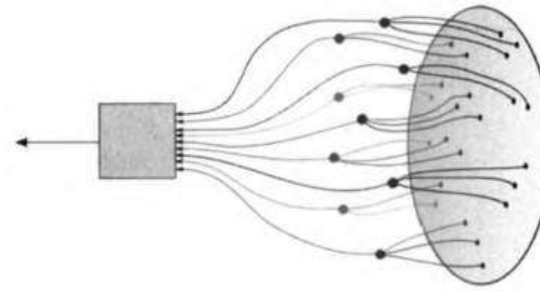


Рис. 7.44. Изображение «Zebra» до и после обработки пользовательской нейронной сетью

7.6. Краткие итоги главы

В этой главе мы познакомились с работающей вместе с Python популярной библиотекой ImageAI, которая позволяет решать большой класс практических задач, значительно облегчая и упрощая труд программистов. Однако это не единственная библиотека. Вызывает большой интерес и пользуется популярностью у специалистов другая библиотека — OpenCV. В ней имеется целый ряд уже готовых, обученных моделей для распознавания объектов в изображениях, основанных на примитивах Хаара. При этом существует возможность обучения этих моделей на распознавание не только просто объектов, а объектов с конкретными характеристиками. Например, можно научить модель распознавать лицо конкретного человека на фотографии или в кадрах с видеокамеры. Практике применения библиотеки Open CV и посвящена следующая глава.

ГЛАВА 8



Создание приложений для обработки изображений: библиотека OpenCV

Библиотека OpenCV (Open Source Computer Vision Library) — одна из самых популярных библиотек для приложений по компьютерному зрению. OpenCV-Python — это Python-версия интерфейса для OpenCV. Наличие кода на C/C++ в бэкенде гарантирует быструю работу приложений, а Python-обертка обеспечивает легкость настройки и развертывания. Благодаря этому OpenCV-Python является отличным решением для высоконагруженных вычислительных программ по компьютерному зрению. В эту библиотеку с открытым исходным кодом входят более 2500 алгоритмов, среди которых есть как классические, так и современные алгоритмы для компьютерного зрения и машинного обучения.

При работе с этой библиотекой есть несколько тонкостей. Общепринятое ее название — OpenCV. Но когда мы инсталлируем ее на свой компьютер для использования с Python, то должны указывать имя `opencv-python`:

```
pip install opencv-python
```

Когда же требуется подключить эту библиотеку к своей программе, то нужно указывать имя `cv2`:

```
import cv2
```

Это следует иметь в виду и всегда помнить при разработке своих программ.

На момент подготовки этого издания книги вышла OpenCV версии 4.6.0.66. Однако у нее есть один недостаток — при вводе программного кода в PyCharm для объектов библиотеки OpenCV этой версии после набранной точки не показываются методы и атрибуты объектов, что вызывает некоторые неудобства. Поэтому в этой книге использовалась предыдущая, но стабильно работающая версия 4.5.5.64, которая лишена указанного недостатка. Выполнять установку библиотеки нужно в следующей последовательности:

1. Обновите pip: `pip install --upgrade pip`.
2. Установите из пакетов:

- `pip install opencv-python` — пакет основных модулей;
- `pip install opencv-contrib-python` — полный пакет (содержит как основные модули, так и дополнительные).

Для установки версии 4.5.5.64 можно использовать следующую команду в окне терминала PyCharm:

```
pip install opencv-contrib-python==4.5.5.64
```

В этой главе будут представлены следующие материалы:

- сведения об обученных классификаторах Хаара для распознавания объектов в изображениях;
- пример программы поиска лиц на фотографиях;
- пример программы поиска лиц в видеопотоках с видеокамер;
- пример программы распознавания глаз на фотографиях;
- пример программы распознавания эмоций на изображениях;
- пример программы распознавания автомобильных номеров на изображениях;
- пример программы распознавания автомобильных номеров на видеоданных;
- пример программы распознавания движущихся автомобилей в транспортном потоке;
- пример программы распознавания различных объектов из одного программного кода;
- пример программы распознавания пешеходов на изображениях с использованием OpenCV и HOG-детекторов;
- пример программы распознавания пешеходов на видео с использованием OpenCV и HOG-детекторов;
- способ распознавания конкретных людей на фотографиях в OpenCV;
- пример программы для обучения модели распознавания лиц на фотографиях;
- пример программы распознавания лиц конкретных людей на фотографиях;
- все этапы создания пользовательской модели нейронной сети для распознавания людей в видеопотоке с видеокамеры в OpenCV.

8.1. Обученные классификаторы Хаара для распознавания объектов в изображениях

Для того чтобы нейронная сеть смогла распознавать объекты в изображениях, в видеофайлах и видеопотоках, ее нужно этому обучить. Однако для библиотеки OpenCV имеется набор уже обученных классификаторов, которые можно использовать в собственных приложениях. Это классификаторы на основе каскадов Хаара.

Каскад Хаара — способ обнаружения объектов на изображении, основанный на машинном обучении. Обученный каскад Хаара, принимая на вход изображение,

определяет, есть ли на нем искомый объект, т. е. выполняет задачу классификации, разделяя входные данные на два класса: есть искомый объект, нет искомого объекта. Собственно, сам алгоритм работы сводится к тому, что определяются признаки Хаара. Это, по сути, набор прямоугольных областей, которые проходят по изображению и находят в нем детали искомого объекта.

Скачать такой набор обученных классификаторов в виде XML-файлов можно, например, по следующим ссылкам:

- **https://github.com/opencv/opencv/tree/master/data/haarcascades;**
- **https://github.com/anaustinbeing/haar-cascade-files.**

В табл. 8.1 приведены некоторые из этих образцов. Их можно использовать в приложениях, разрабатываемых на Python совместно с библиотекой OpenCV.

***Таблица 8.1.** Набор обученных классификаторов Хаара для распознавания объектов в изображениях*

№ п/п	Наименование файла	Распознаваемые объекты
1	haarcascade_eye.xml	Глаза
2	haarcascade_eye_tree_eyeglasses.xml	Глаза в очках
3	haarcascade_frontalcatface.xml	Фронтальная кошачья морда
4	haarcascade_frontalcatface_extended.xml	Фронтальная кошачья морда
5	haarcascade_frontalface_alt.xml	Фронтальное лицо
6	haarcascade_frontalface_alt2.xml	Фронтальное лицо 2
7	haarcascade_frontalface_alt_tree.xml	Фронтальное лицо каскад
8	haarcascade_frontalface_default.xml	Фронтальное лицо
9	haarcascade_fullbody.xml	Фронтальное тело
10	haarcascade_lefteye_2splits.xml	Левый глаз
11	haarcascade_licence_plate_rus_16stages.xml	Российские номерные знаки
12	haarcascade_lowerbody.xml	Нижняя часть тела
13	haarcascade_profileface.xml	Профиль лица
14	haarcascade_righteye_2splits.xml	Правый глаз
15	haarcascade_russian_plate_number.xml	Российские номерные знаки
16	haarcascade_smile.xml	Улыбка
17	haarcascade_upperbody.xml	Верхняя часть тела

Подключить их можно с использованием следующей инструкции:

```
classifier = cv2.CascadeClassifier(
    'C:/XML/haarcascade_frontalface_default.xml')
```

Здесь `C:/XML/` — путь к папке с каскадами Хаара.

Если вы работаете с библиотекой OpenCV, то скачивать каскады Хаара не обязательно, т. к. они уже включены в состав библиотеки. Их можно просто подключать с использованием следующей инструкции:

```
classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

В этой инструкции `"haarcascade_frontalface_default.xml"` — имя соответствующего каскада Хаара. Однако если вы используете сторонние каскады Хаара, то нужно использовать следующую инструкцию:

```
cascade = cv2.CascadeClassifier('./XML/cars.xml')
```

Здесь `'./XML/cars.xml'` — имя подключаемого файла с указанием полного пути.

8.2. Пример программы поиска лиц на фотографиях

Вот мы с вами и подошли к самому интересному — к поиску лица (лиц) на фото. Для этого мы воспользуемся каскадом Хаара. Все, что нам нужно, уже есть в библиотеке OpenCV: и натренированные признаки, и набор функций, реализующих все алгоритмы. Программа распознавания лиц на фото приведена в листинге 8.1.

Листинг 8.1

```
# Listing 8_1
mport cv2

# загрузка изображения
img = cv2.imread('./Images/Test_Face.jpg')
# показать загруженное изображение
cv2.imshow('Input photo', img)
# загрузка каскада Хаара
classifier = cv2.CascadeClassifier(cv2.data.haarcascades
                                   + "haarcascade_frontalface_default.xml")
# выполнение распознавания лиц
bboxes = classifier.detectMultiScale(img)
# формирование прямоугольника вокруг каждого обнаруженного лица
for box in bboxes:
    # формирование координат
    x, y, width, height = box
    x2, y2 = x + width, y + height
    # рисование прямоугольников
    cv2.rectangle(img, (x, y), (x2, y2), (0, 0, 255), 1)

cv2.imshow('Window with face detection', img)  # показать обработанное изображение
cv2.imwrite('./Images/Test_Face_det.jpg', img)  # сохранить обработанное изображение

cv2.waitKey(0)  # держать окна с изображениями открытыми
cv2.destroyAllWindows()  # закрыть все окна
```

В первой строке подключена библиотека OpenCV. Далее в переменную `img` загружено тестовое изображение Test_Face.jpg, а в переменную `classifier` — обученный классификатор распознавания лиц. После этого инициирован процесс распознавания лиц, и его результат записан в переменную `bboxes`. На следующем шаге запущен процесс формирования прямоугольника вокруг каждого обнаруженного лица. Затем обработанное изображение показано в окне и сохранено в файле с именем Test_Face_det.jpg. Последние две строки программы закрывают все окна.

Для проверки работы программы было взято изображение, представленное на рис. 8.1.

В результате работы программы получен следующий результат (рис. 8.2). Как можно видеть, алгоритм на основе каскадов Хаара haarcascade_frontalface_default.xml вполне успешно справился с поставленной задачей.

Рис. 8.1. Изображение для тестирования работы программы распознавания лиц

Рис. 8.2. Изображение, обработанное программой распознавания лиц

8.3. Пример программы поиска лиц в видеопотоках с видеокамер

Теперь проверим, как работает эта модель с поиском лиц в видеопотоке с видеокамер. Для этого применим следующий программный код (листинг 8.2).

Листинг 8.2

```
# Listing 8_2
import cv2
# загрузка каскада Хаара
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
video_capture = cv2.VideoCapture(0, cv2.CAP_DSHOW) # Активация камеры
# video_capture = cv2.VideoCapture(0)  # Активация встроенной видеокамеры
while True:
    # Захват кадр за кадром
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(30, 30),
                                         flags=cv2.CASCADE_SCALE_IMAGE)
    # Рисование прямоугольников вокруг лиц
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    # Показ обработанного кадра
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Когда обработка закончена, закрыть все окна
cv2.destroyAllWindows()
```

В первой строке мы подключили библиотеку OpenCV. Далее идет строка:

```
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

Этой командой мы загружаем классификатор haarcascade_frontalface_default.xml, который обучен на распознавание лиц. Файл с ним уже включен в библиотеку OpenCV.

Далее идет команда активации видеокамеры:

```
video_capture = cv2.VideoCapture(0,cv2.CAP_DSHOW)
```

В следующих четырех строках программы:

- организован цикл последовательного захвата кадров;
- чтения очередного кадра;

- преобразования цветного изображения кадра в черно-белое изображение (в градациях серого);
- распознавание лица в текущем кадре.

Далее выполняется рисование цветных прямоугольных рамок вокруг найденных лиц и показ обработанного кадра. Этот процесс продолжается до тех пор, пока пользователь не нажмет клавишу <q> для завершения процесса обработки кадров и освобождения ресурсов Windows.

Запустив этот программный код на исполнение, получим следующий результат (рис. 8.3).

Рис. 8.3. Видеокадр, обработанный программой распознавания лиц

Конечно, бывают ложные срабатывания, или, наоборот, программе не удается обнаружить лицо. Все очень сильно зависит от освещения, угла поворота, наклона головы и т. д. Для более точного распознавания лиц прибегают к разнообразным методам нормализации изображения.

8.4. Пример программы распознавания глаз на фотографиях

На изображении можно распознать не только лица, но и различные элементы на лицах, — например, глаза. Для решения таких задач тоже есть уже готовые каскады Хаара. Напишем программу распознавания глаз на изображениях (листинг 8.3).

Листинг 8.3

```
# Listing 8_3
import cv2
```

```
# Загрузка изображения
img = cv2.imread('./Images/Test_Face_eye.jpg')
# показать загруженное изображение
cv2.imshow('Input photo', img)
# Загрузка каскадов Хаара
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_alt.xml")
eye_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_eye.xml")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Выполнение распознавания лиц
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x, y, w, h) in faces:
    img = cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)  # распознавание глаз
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh),
                      (0, 255, 0), 2

cv2.imshow('Out photo', img)  # показать обработанное изображение
cv2.imwrite('./Images/Test_Face_Eye_det.jpg', img)  # сохранить изображение

cv2.waitKey(0)  # держать окно с изображением открытым
cv2.destroyAllWindows()  # закрыть все окна
```

Здесь в первой строке подключена библиотека OpenCV. Далее в переменную `img` загружено тестовое изображение Test_Face_eye.jpg, и это изображение выведено пользователю. Потом в переменную `face_cascade` загружен классификатор распознавания лиц, а в переменную `eye_cascade` — классификатор распознавания глаз. Затем загруженное изображение конвертировано из цветного в черно-белое изображение (в градациях серого), и этот результат записан в переменную `gray`.

Далее инициирован процесс распознавания лиц, а на лице — распознавания глаз и формирования прямоугольника вокруг каждого обнаруженного элемента. Затем обработанное изображение показано в окне и сохранено в файле с именем Test_Face_Eye_det.jpg. Последние две строки программы закрывают все окна. Как видите, программный код получился достаточно коротким.

Для проверки работы программы было взято изображение, представленное на рис. 8.4.

После завершения работы программы был получен следующий результат (рис. 8.5). Как можно видеть, программа успешно справилась со своей задачей.

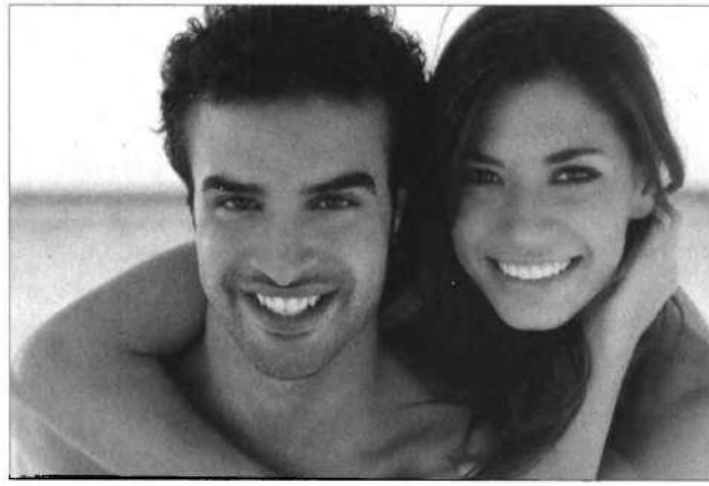


Рис. 8.4. Изображение для тестирования работы программы распознавания глаз

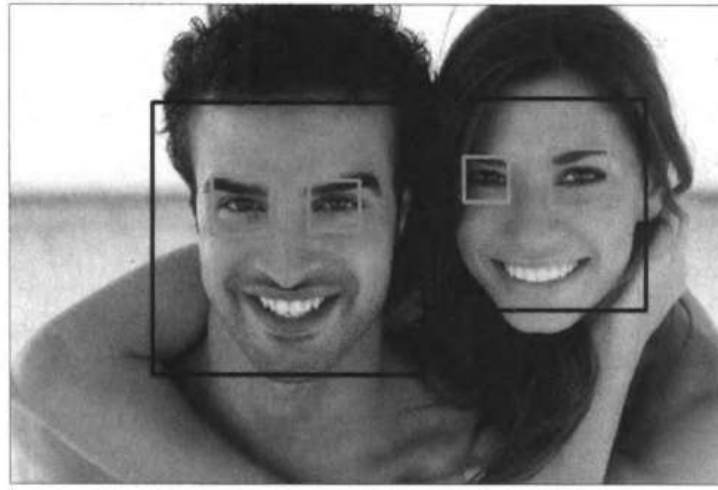


Рис. 8.5. Изображение, полученное на выходе из программы распознавания глаз

8.5. Пример программы распознавания эмоций на изображениях

На изображении можно распознать не только отдельные элементы, но и различные эмоции, — например, улыбку. Для решения таких задач есть уже готовые, натренированные модели. Напишем программу распознавания улыбки на лице. В ней мы используем каскады Хаара, в том числе каскад haarcascade_smile.xml. В листинге 8.4 приведен программный код, позволяющий выделить улыбку на лице при получении изображения с видеокамеры.

Листинг 8.4

```
# Listing 8_4
import cv2

# Загрузка каскадов Хаара
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
smileCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_smile.xml")

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)  # Активация видеокамеры
# cap = cv2.VideoCapture(0)
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray, scaleFactor=1.3,
                                         minNeighbors=5)

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]

        smile = smileCascade.detectMultiScale(roi_gray, scaleFactor=1.5,
                                              minNeighbors=15,
                                              minSize=(25, 25))
        for (xx, yy, ww, hh) in smile:
            cv2.rectangle(roi_color, (xx, yy), (xx + ww, yy + hh),
                          (0, 255, 0), 2)
        cv2.imshow('video', img)

    k = cv2.waitKey(30) & 0xff
    if k == 27:  # press 'ESC' to quit
        break

cap.release()
cv2.destroyAllWindows()
```

В этой программе используются два каскада Хаара: для распознавания лица и для распознавания улыбки на лице. Функциональное назначение остальных строк этой программы ясно из описания предыдущих листингов программ. Результат работы этой программы представлен на рис. 8.6.

Как можно видеть, видеокамера успешно обработала все поступающие кадры и выделила на них тот фрагмент изображения, где на лице имеется улыбка.

Рис. 8.6. Изображение, полученное на выходе из программы распознавания улыбки на лице

8.6. Пример программы распознавания автомобильных номеров на изображениях

Приложения, обеспечивающие распознавание автомобильных номеров, являются одними из самых востребованных программных продуктов, применяемых в транспортной отрасли. Они используются органами ГИБДД, на автоматизированных парковках, в системах обеспечения доступа на склады и в коттеджные поселки, организациями, обеспечивающими дорожный сервис, мониторинг дорожного трафика и т. п. В библиотеку OpenCV включены каскады Хаара для распознавания номеров автомобилей — как российских номеров, так и номеров различных европейских стран. В листинге 8.5 приведен пример программного кода программы, которая обеспечивает распознавание на изображениях российских номеров.

Листинг 8.5

```
# Listing 8_5
import cv2

# Загрузка изображения
img = cv2.imread('./Images/Test_Numer.jpg')
# показать загруженное изображение
cv2.imshow('Input photo', img)
# загрузка каскада Хаара
classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_russian_plate_number.xml")
# Выполнение распознавания объектов
bboxes = classifier.detectMultiScale(img)
```

```
# формирование прямоугольника вокруг каждого обнаруженного объекта
for box in bboxes:
    # формирование координат
    x, y, width, height = box
    x2, y2 = x + width, y + height
    # рисование прямоугольников
    cv2.rectangle(img, (x, y), (x2, y2), (0, 0, 255), 3)

cv2.imshow('Window with object detection', img)  # показать изображение
cv2.imwrite('./Images/Test_Numer_det.jpg', img)  # сохранить изображение

cv2.waitKey(0)  # держать окно с изображениями открытым
cv2.destroyAllWindows()  # закрыть все окна
```

В этой программе использован каскад Хаара haarcascade_russian_plate_number.xml. Функциональное назначение остальных ее строк ясно из описания предыдущих листингов программ. На вход программы подано изображение, представленное на рис. 8.7.

Рис. 8.7. Изображение, поданное на вход программы распознавания автомобильных номеров

Рис. 8.8. Изображение, обработанное программой распознавания автомобильных номеров

Итоговое изображение, обработанное программой, представлено на рис. 8.8. Как можно видеть, программа успешно справилась с поставленной перед ней задачей.

Следует иметь в виду, что в разных странах используются различные форматы автомобильных номеров. Поэтому если стоит задача определения автомобильного номера другой страны, то либо нужно использовать другие обученные каскады Хаара, либо проводить обучение модели нейронной сети распознавать автомобильные номера другого формата.

8.7. Пример программы распознавания автомобильных номеров в видеопотоке

На практике являются более востребованными решения, позволяющие распознавать автомобильные номера в реальном режиме времени с видеокамер (в транс-

портном потоке, на парковках). При этом решения могут быть разными: можно просто выделить обнаруженный номер автомобиля на видеокадре, а можно в видеокадре выделить номерной знак и сформировать на основе этого фрагмента новое изображение. В листинге 8.6 приведена программа, которая выделяет на видеокадрах обнаруженные номерные знаки.

Листинг 8.6

```
# Listing 8_6
import cv2
# загрузка каскада Хаара
classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_russian_plate_number.xml")
video_capture = cv2.VideoCapture(0, cv2.CAP_DSHOW)  # Активация камеры
# video_capture = cv2.VideoCapture(0)     # активация видеокамеры
# захват кадра за кадром
while True:
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = classifier.detectMultiScale(gray,
                                        scaleFactor=1.1,
                                        minNeighbors=5,
                                        minSize=(30, 30),
                                        flags=cv2.CASCADE_SCALE_IMAGE)
    # Рисование прямоугольников вокруг объектов
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    # Показ обработанного кадра
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Когда обработка закончена, закрыть все окна
cv2.destroyAllWindows()
```

В этой программе организован цикл, внутри которого происходит последовательная обработка кадров, поступающих с видеокамеры. Как только обнаруживается фрагмент изображения с номерным знаком, вокруг этого фрагмента прорисовывается цветная прямоугольная рамка. Пример работы представлен на рис. 8.9.

На практике важно не просто найти фрагмент изображения с номерным знаком автомобиля, но и получить его алфавитно-цифровое значение. Это процесс можно разбить на несколько этапов: найти фрагмент изображения с номерным знаком автомобиля, сформировать отдельное изображение с номерным знаком, передать последнее изображение на модуль распознавания символов. В листинге 8.7 приведен код программы, которая позволяет не только найти область видеокадра с номерным знаком, но и сохранить номерной знак в виде отдельного изображения.

Рис. 8.9. Изображение, обработанное программой распознавания автомобильных номеров с видеокамеры

Листинг 8.7

```
# Listing 8_7
import cv2

classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_russian_plate_number.xml")
video_capture = cv2.VideoCapture(0, cv2.CAP_DSHOW)  # Активация камеры
# video_capture = cv2.VideoCapture(0)  # Активация камеры

while True:
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # градация серого
    plaques = classifier.detectMultiScale(gray, 1.3, 5)
    for i, (x, y, w, h) in enumerate(plaques):
        roi_color = frame[y:y + h, x:x + w]
        cv2.putText(frame,
                    str(x) + " " + str(y) + " " + str(w) + " " + str(h),
                    (480, 220), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    (255, 255, 255))
        r = 400.0 / roi_color.shape[1]
        dim = (400, int(roi_color.shape[0] * r))
        resized = cv2.resize(roi_color, dim,
                             interpolation=cv2.INTER_AREA)
        w_resized = resized.shape[0]
        h_resized = resized.shape[1]
```

```
            # Собираем в основную картинку
            frame[100:100 + w_resized, 100:100 + h_resized] = resized
            cv2.rectangle(roi_color, (x, y), (x + w, y + h), (0, 255, 0), 2)
            if cv2.waitKey(1) & 0xFF == ord('c'):
                # сохранить только знак
                cv2.imwrite('Images/Video_Znak.jpg', resized)
        # Отображение результирующего кадра
        cv2.imshow('Video', frame)
        if cv2.waitKey(1) & 0xFF == ord('c'):
            # сохранить автомобиль и знак
            cv2.imwrite('Images/Video_Znak_det.jpg', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
           break

# деактивировать камеру, закрыть все окна
video_capture.release()
cv2.destroyAllWindows()
```

В этой программе сначала активируется видеокамера и подключается каскад Хаара для поиска российских автомобильных номеров — haarcascade_russian_plate_number.xml.

Затем организуется цикл покадровой обработки изображения. Если во всех предыдущих программах вокруг обнаруженного объекта рисовался цветной прямоугольник, то здесь выполняются другие действия. Когда в кадре обнаруживается автомобильный номер, то формируется новое изображение, содержащее только этот номер. Изображение запоминается в переменной `resized`. Далее кадр с транспортным средством и изображение автомобильного номера объединяются, и результат выводится в окно терминала. Образец такого совмещенного изображения, полученного этой программой, представлен на рис. 8.10.

Если в этот момент нажать клавишу <c>, это действие будет обработано следующими строками программы:

```
if cv2.waitKey(1) & 0xFF == ord('c'):
    cv2.imwrite('Video_Numer_Znak.jpg', resized) # сохранить только знак
```

Рис. 8.10. Изображение автомобиля и его номера, полученное программой распознавания автомобильных номеров с видеокамеры

В результате изображение номерного знака будет сохранено в виде файла Video_Znak.jpg. Содержимое этого файла, полученного приведенной в листинге 8.7 программой, представлено на рис. 8.11.

Рис. 8.11. Изображение номерного знака автомобиля, сохраненное в отдельном файле

В дальнейшем уже это изображение можно передать модулю программы преобразования номера в алфавитно-цифровой вид.

8.8. Пример программы распознавания движущихся автомобилей в транспортном потоке

В интеллектуальных транспортных системах применяются решения, позволяющие определять интенсивность транспортного потока в реальном режиме времени. Для этого используются детекторы, которые подвешиваются на специальной арке над каждой полосой движения. Однако для решения такой задачи можно использовать видеокамеры, которые не обязательно размещать над каждой полосой движения. При этом исчезает необходимость монтировать специальные арки над дорогами — камеру можно разместить на осветительном столбе на обочине трассы, что значительно дешевле. В листинге 8.8 приведена программа, которая позволяет обнаруживать автомобили в потоке транспортных средств с видеокамеры.

Листинг 8.8

```
# Listing 8_8
import cv2

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)  # активация камеры
car_cascade = cv2.CascadeClassifier('./XML/cars.xml')  # загрузка классификатора

# цикл обработки кадров
while True:
    ret, frames = cap.read()  # читает кадры из видео
    gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)    # оттенки серого
    # обнаруживает автомобили
    cars = car_cascade.detectMultiScale(gray, 1.1, 1)
    # Нарисовать прямоугольник в найденном авто
    for (x, y, w, h) in cars:
        cv2.rectangle(frames, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.imshow('video', frames)  # отображать обработанные кадры в окне
```

```
    if cv2.waitKey(33) == 27:     # завершить, если нажата клавиша Esc
        break

cv2.destroyAllWindows()  # закрыть все окна
```

Здесь сначала активируется видеокамера и подключается каскад Хаара для поиска автомобилей — cars.xml. Обратите внимание, что этого каскада Хаара нет в составе библиотеки OpenCV — его нужно скачивать и загружать в одну из папок компьютера самостоятельно. Далее организуется цикл покадровой обработки изображения. Затем во вложенном цикле происходит последовательная обработка кадров, поступающих с видеокамеры. Как только обнаруживается фрагмент изображения с автомобилем, вокруг этого фрагмента прорисовывается цветная прямоугольная рамка. Пример работы программы представлен на рис. 8.12. Как можно видеть, практически все автомобили, которые двигаются в транспортном потоке, были программой успешно распознаны.

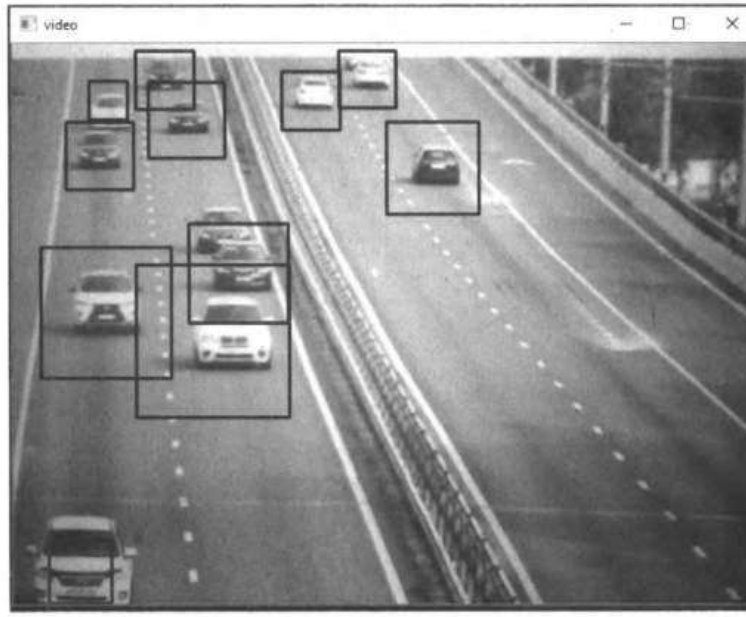


Рис. 8.12. Распознавание автомобилей в потоке транспортных средств на кадрах с видеокамеры

8.9. Пример программы распознавания различных объектов из одного программного кода

На самом деле для распознавания различных объектов на изображениях можно написать один программный код. В нем для распознавания разных объектов достаточно менять только одну строчку, в которой указывается имя XML-файла с тем или иным набором каскадов Хаара. В листинге 8.9 приведен пример такой программы, в которой для смены распознаваемого объекта достаточно снять комментарий с одной из строк программы с именем нужного XML-файла.

Листинг 8.9

```
# Listing 8_9
import cv2

# загрузка фотографии
img = cv2.imread('./Images/Test4.jpg')
cv2.imshow('Input photo', img)
# загрузка предварительно обученной модели
classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_fullbody.xml")
# classifier = cv2.CascadeClassifier(
    # cv2.data.haarcascades + "haarcascade_upperbody.xml")
# classifier = cv2.CascadeClassifier(
    # cv2.data.haarcascades + "haarcascade_lowerbody.xml")
# classifier = cv2.CascadeClassifier(
    # cv2.data.haarcascades + "haarcascade_righteye_2splits.xml")
# classifier = cv2.CascadeClassifier(
    # cv2.data.haarcascades + "haarcascade_lefteye_2splits.xml")

# выполнение распознавания объектов
bboxes = classifier.detectMultiScale(img)
# формирование прямоугольника вокруг каждого обнаруженного объекта
for box in bboxes:
    # формирование координат
    x, y, width, height = box
    x2, y2 = x + width, y + height
    # рисование прямоугольников
    cv2.rectangle(img, (x, y), (x2, y2), (0, 0, 255), 1)

cv2.imshow('Window with object detection', img)  # показать
cv2.imwrite('./Images/Test4_det.jpg', img)       # сохранить

cv2.waitKey(0)     # держать окно с изображением открытым
cv2.destroyAllWindows()  # закрыть все окна
```

В этой программе в качестве примера можно поочередно подключать следующие каскады Хаара:

- haarcascade_fullbody.xml — для распознавания всего тела человека;
- haarcascade_upperbody.xml — для распознавания верхней части тела человека;
- haarcascade_lowerbody.xml — для распознавания нижней части тела человека;
- haarcascade_righteye_2splits.xml — для распознавания правого глаза;
- haarcascade_lefteye_2splits.xml — для распознавания левого глаза.

Для проверки работы этой программы на вход было подано изображение, представленное на рис. 8.13. Результаты работы программы при разных подключенных XML-файлах приведены на рис. 8.14.

Рис. 8.13. Изображение, поданное на вход в программу распознавания различных объектов

Рис. 8.14. Изображения, обработанные разными каскадами Хаара

8.10. Пример программы распознавания пешеходов на изображениях с использованием OpenCV и HOG-детекторов

Для многих служб является актуальной задача распознавания людей. Это могут быть пешеходы на дороге, посетители торговых центров, пассажиры в транспортных средствах и т. п. Попробуем создать базовый детектор пешеходов на изображениях с использованием OpenCV. Для решения этой задачи необходимо извлечь такие особенности тела человека, как голова, две руки, две ноги и т. д., и передать эти элементы для обучения модели нейронной сети. После обучения модель можно использовать для обнаружения и отслеживания людей в изображениях и видеопотоках. Однако библиотека OpenCV уже имеет встроенный метод для обнаружения

пешеходов. Он основан на предварительно обученной HOG-гистограмме (гистограмме ориентированных градиентов) и линейной модели SVM для обнаружения пешеходов в изображениях и видеопотоках.

Гистограмма ориентированных градиентов — это специальный алгоритм, который проверяет окружающие пикселы каждого отдельного пиксела. Цель его работы состоит в том, чтобы проверить, насколько темнее текущий пиксел по сравнению с окружающими пикселами. Алгоритм рисует стрелки, показывающие направление темнеющего изображения, и повторяет процесс для каждого пиксела изображения. В завершение каждый пиксел заменяется такой стрелкой. Стрелки называются *градиентами* и показывают поток света от светлых частей изображений к темным частям. Сейчас мы не станем заострять внимание на принципах работы этого алгоритма, а воспользуемся уже готовыми решениями, встроенными в библиотеку OpenCV.

Кроме библиотеки OpenCV, для нашей программы понадобится еще один модуль — `imutils` версии 0.5.4. Установить его можно с помощью следующей команды в окне терминала:

```
pip install imutils
```

В листинге 8.10 приведена программа для обнаружения пешеходов в изображении.

Листинг 8.10

```
# Listing 8_10
import cv2
import imutils

# Инициализация детектора человека
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

# Чтение изображения
image = cv2.imread('./Images/image_str.jpg'

cv2.imshow('Input photo', image)
# Изменение размера изображения
image = imutils.resize(image, width=min(800, image.shape[1]))

# Обнаружение всех областей на изображении, где есть пешеходы
(regions, _) = hog.detectMultiScale(image, winStride=(4, 4),
                                    padding=(4, 4), scale=1.05)

# Рисование прямоугольников на изображении
for (x, y, w, h) in regions:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

cv2.imshow("Image", image)  # Отображение обработанного изображения
cv2.imwrite('./Images/image_str_det.jpg', image)  # сохранение
```

```
cv2.waitKey(0)  # Ожидание нажатия любой клавиши
cv2.destroyAllWindows()  # закрыть все окна
```

В этой программе после импорта необходимых библиотек активирован HOG-детектор, на основе которого создан экземпляр класса — `hog`. Затем с помощью метода `setSVMDetector` в созданный экземпляр класса загружен детектор обнаружения человека — `cv2.HOGDescriptor_getDefaultPeopleDetector`. Все дальнейшие строки программы выполняют уже известные процедуры: загрузка исходного рисунка и его отображение, предварительная обработка рисунка, поиск на рисунке пешеходов, рисование рамки вокруг найденных объектов, вывод обработанного изображения. Для проверки работы программы в качестве примера было взято изображение, приведенное на рис. 8.15.

Рис. 8.15. Изображение, поданное на вход программы распознавания пешеходов

Рис. 8.16. Изображение, полученное на выходе из программы распознавания пешеходов

Обработанное программой изображение показано на рис. 8.16. Как можно видеть, программа вполне корректно распознала людей на пешеходном переходе.

8.11. Пример программы распознавания пешеходов на видео с использованием OpenCV и HOG-детекторов

Для беспилотных автомобилей актуальной является задача распознавания пешеходов на дороге. Это очень важная задача, поскольку ее решение может улучшить функциональность системы защиты пешеходов от автомобилей с автономным управлением. Системы обнаружения людей на пешеходных переходах также позволяют реализовать «умные» светофоры, которые работают на пропуск автомобильного трафика и включаются автоматически на пропуск пешеходов при появлении их перед светофором. В торговых центрах такие системы дают возможность собирать статистику по посетителям.

Попробуем создать базовый детектор пешеходов для изображений с видеокамеры с использованием OpenCV. В листинге 8.11 приведен листинг такой программы.

Листинг 8.11

```
# Listing 8_11
import cv2
import imutils
# Инициализация детектора человека
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
cap = cv2.VideoCapture('./Video/Person.mp4')  # Загрузка видеофайла

while cap.isOpened():
    # Чтение видеопотока
    ret, image = cap.read()
    if ret:
        image = imutils.resize(image, width=min(600, image.shape[1]))

        # Обнаружение всех областей на изображении,
        #  в которых есть пешеходы
        (regions, _) = hog.detectMultiScale(image, winStride=(4, 4),
                                            padding=(4, 4), scale=1.05)

        # Рисование прямоугольников на изображении
        for (x, y, w, h) in regions:
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

        cv2.imshow("Video", image)              # показ видео
        if cv2.waitKey(25) & 0xFF == ord('q'):  # нажатие клавиши 'q'
            break
```

```
    else:
        break

cap.release()              # освободить видеопоток
cv2.destroyAllWindows()    # закрыть все окна
```

В этой программе после импорта необходимых библиотек активирован HOG-детектор, на основе которого создан экземпляр класса — `hog`. Затем с помощью метода `setSVMDetector()` в созданный экземпляр класса загружен детектор обнаружения человека — `cv2.HOGDescriptor_getDefaultPeopleDetector`. На следующем этапе активирован видеофайл, после чего организован цикл последовательной обработки каждого кадра видеопотока.

Все дальнейшие строки программы выполняют уже известные процедуры: предварительную обработку кадра, поиск на кадре пешеходов, рисование рамки вокруг найденных объектов, вывод обработанного кадра. Образец кадра из видеопотока, обработанного программой, представлен на рис. 8.17.

Рис. 8.17. Распознавание пешеходов на кадрах с видеокамеры

Как можно видеть, даже на затемненном кадре программа смогла успешно распознать пешехода, который переходит дорогу.

8.12. Распознавание конкретных людей на фотографиях в OpenCV

В предыдущих разделах мы рассмотрели примеры программ поиска любых лиц на фотографиях и в видеопотоке с видеокамер. Однако с помощью нейронных сетей можно не просто обнаружить фрагмент изображения с лицом человека, но и найти

на изображении лицо конкретного человека (идентифицировать личность). В этом разделе мы остановимся на алгоритмах распознавания лиц конкретных людей библиотекой OpenCV. С помощью основанных на них программ можно по известной фотографии человека найти его в базе данных изображений, или обнаружить человека в видеокадрах, получаемых с видеокамеры. Для этого нам понадобится Python с библиотеками NumPy, PIL и OpenCV, а также расширенная библиотека `opencv-contrib-python`. В частности, в приведенных далее примерах использовалась расширенная библиотека версии 4.5.5.64, которую можно подгрузить командой:

```
pip install opencv-contrib-python==4.5.5.64
```

Для начала давайте разберемся, как можно распознавать лица на изображении (фото или видео). Во-первых, нужно найти, где на фото расположено именно лицо человека, и не спутать его с другими частями тела или различными предметами. Это достаточно простая задача для человека, однако она не столь тривиальна для компьютера. Чтобы компьютер смог найти лицо, необходимо выделить на изображении его основные компоненты — такие как лоб, нос, глаза, губы, подбородок и т. д. Для этого используются шаблоны элементов лица (они же примитивы, или каскады Хаара). Образцы таких примитивов приведены на рис. 8.18.

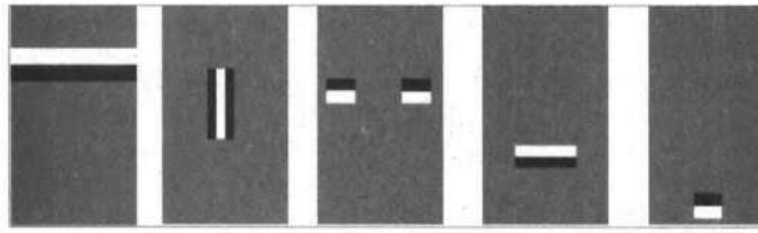


Рис. 8.18. Образцы шаблонов элементов лица (примитивы Хаара)

Если шаблоны будут соответствовать конкретным областям на изображении, то компьютер сочтет, что на изображении есть человеческое лицо. Для каждого из этих шаблонов определяется разность между яркостью белой и черной областей изображения. Это значение сравнивается с эталоном и принимается решение о том, есть ли здесь часть человеческого лица или нет. Аналог лица, составленного из таких примитивов, приведен на рис. 8.19.

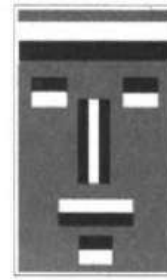


Рис. 8.19. Формирование образа лица на основе примитивов (каскадов Хаара)

На самом деле, подобных шаблонов гораздо больше, и не только для лица, но и для многих других объектов. Этот метод называется *методом Виолы — Джонса* (он также известен как и каскады Хаара).

Теперь представим, что на фотографии есть не одно большое лицо, а много мелких лиц разных людей. Если применить шаблоны к этому изображению в целом, то мы не найдем там лиц, поскольку лица на изображениях будут меньше самих шаблонов. Поэтому сначала нужно выделить фрагмент изображения из всей фотографии в отдельное окно, а затем именно внутри этого окна использовать примитивы Хаара (нужно иметь в виду, что лица на фото могут быть разных размеров). Для выделения фрагментов изображения используется *метод скользящего окна* небольшого размера. Скользящее окно перемещается по всему изображению, выделяя его фрагменты в виде отдельных окон. После этого каждый такой выделенный фрагмент изображения увеличивается, и в этом масштабированном окне выполняется распознавание лиц с использованием примитивов Хаара.

Итак, с помощью скользящего окна на фотографии найдено лицо. Но как определить, что это лицо именно того человека, которого мы ищем? Для решения этой задачи используется алгоритм Local Binary Patterns (локальные бинарные шаблоны). Суть его заключается в том, что все анализируемое изображение разбивается на части и в каждой такой части каждый пиксел сравнивается с соседними 8 пикселами. Если значение центрального пиксела больше соседнего, то значение соседнего пиксела заменяется на 0, в противном случае оно заменяется на 1 (рис. 8.20).

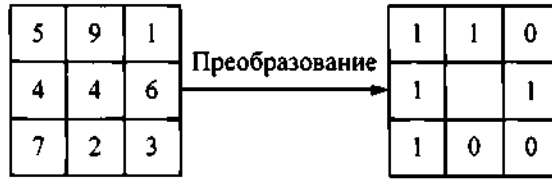


Рис. 8.20. Алгоритм формирования локальных бинарных шаблонов изображения

В итоге в каждый пиксел обработанного изображения будет записано число 0 или 1. Далее на основе этих чисел для всех частей, на которые была разбита фотография, формируется гистограмма. Все гистограммы со всех частей изображения объединяются в один вектор, характеризующий изображение в целом. Таким образом формируется оцифрованный аналог изображения конкретного человека (рис. 8.21).

Если теперь требуется узнать, насколько похожи два лица, нужно вычислить для каждого из них такие оцифрованные аналоги и сравнить их. Пояснение сути этого процесса представлено на рис. 8.22.

В этом разделе будут приведены два примера программ: программа для обучения распознавания лиц конкретных людей на фотографиях и программа поиска конкретных людей на изображениях по заранее обученной модели.

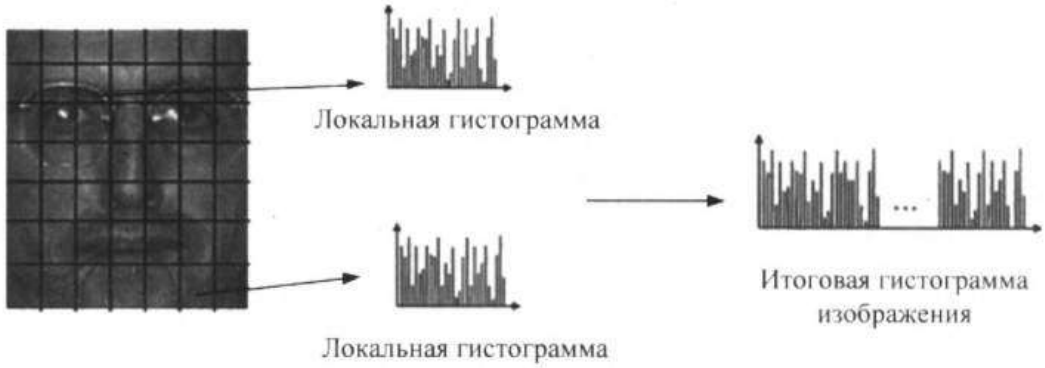


Рис. 8.21. Оцифровка фотографии с использованием локальных бинарных шаблонов изображения

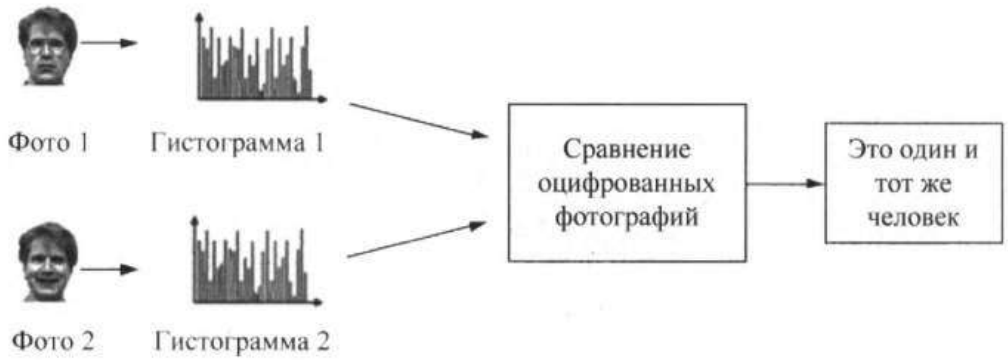


Рис. 8.22. Алгоритм сравнения похожести лиц на основе локальных бинарных шаблонов изображения

8.12.1. Пример программы для обучения модели распознавания лиц по фотографиям

Теперь перейдем к рассмотрению программного кода, в котором реализован алгоритм обучения модели поиску лиц конкретных людей (листинг 8.12). Программа будет формировать набор бинарных шаблонов изображений для тех людей, которые внесены в обучающую выборку.

Листинг 8.12

```
# Listing 8_12
# Это промежуточный модуль, он не является рабочим
import cv2
import os
import numpy as np
from PIL import Image

# Загрузка каскадов Хаара для поиска лиц
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

```
# Формирование локального бинарного шаблона
recognizer = cv2.face.LBPHFaceRecognizer_create(1, 8, 8, 8, 123)
```

В этом фрагменте программы в переменную `faseCascade` загружается файл с уже готовыми каскадами Хаара для поиска лиц — haarcascade_frontalface_default.xml. Далее создается объект для распознавания лиц `recognizer` на основе класса `LBPHFaceRecognizer`. На последнем объекте — точнее, на его параметрах — остановимся подробнее. Первые два значения: `1` и `8` — характеризуют окрестности пиксела, на основе которого строится локальная гистограмма. Суть этих значений показана на рис. 8.23.

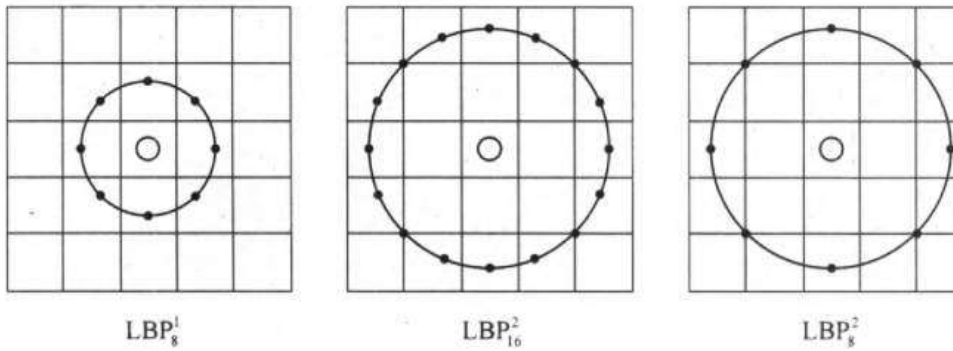


Рис. 8.23. Иллюстрация сущности параметров локальных бинарных шаблонов изображения

Здесь первое число — это радиус, в котором выбираются обрабатываемые пикселы, а второе число — количество этих пикселов. Чем больше пикселов в окрестности точки будет взято, тем точнее распознавание.

Следующие параметры: `8` и `8` — характеризуют размеры областей (окон), на которые будет разбито исходное изображение с лицом. Чем меньше значение этих параметров, тем больше будет таких областей, а значит, и выше качество распознавания.

И наконец, последнее значение: `123` — это параметр `confidence_threshold` (порог доверия), определяющий пороговое значение для распознавания лица. Чем меньше `confidence`, тем больше алгоритм уверен в том, что на фотографии изображено известное ему лицо. Когда этот порог превышен, алгоритм считает рассматриваемое лицо незнакомым.

Идем дальше и напишем функцию, которая находит по определенному пути на всех фотографиях лица людей и сохраняет их (листинг 8.13).

Листинг 8.13

```
# Listing 8_13
# Это промежуточный модуль, он не является рабочим
def get_images(path):
    # Ищем все фотографии и записываем их в image_paths
    image_paths = [os.path.join(path, f)
                   for f in os.listdir(path) if not f.endswith('.happy')]
```

```
    count = 0
    images = []
    labels = []

    for image_path in image_paths:
        # Переводим изображение в черно-белый формат и приводим его к формату массива
        gray = Image.open(image_path).convert('L')
        image = np.array(gray, 'uint8')
        # Из каждого имени файла извлекаем номер человека,
        # изображенного на фото
        subject_number = int(os.path.split(
            image_path)[1].split(".")[0].replace("subject", ""))

        # Определяем области, где есть лица
        faces = faceCascade.detectMultiScale(image, scaleFactor=1.1,
                                             minNeighbors=5,
                                             minSize=(30, 30))
        # Если лицо нашлось, добавляем его в список images,
        # а соответствующий ему номер — в список labels
        for (x, y, w, h) in faces:
            images.append(image[y: y + h, x: x + w])
            labels.append(subject_number)
            # В окне показываем изображение
            cv2.imshow("", image[y: y + h, x: x + w])
            cv2.waitKey(50)
            count += 1
            # Сохраняем лицо
            cv2.imwrite('./DataSet1/user.' + str(subject_number) +
                        '.' + str(count) + '.jpg',
                        image[y:y + h, x:x + w])
    return images, labels
```

Для тренировки нашей модели воспользуемся уже готовым набором обучающих данных (фотографии лиц) под названием Yale Faces, который можно скачать по следующей ссылке: **https://github.com/galvanom/FaceRecognition/blob/master/yalefaces.zip**.

В наборе данных Yale Faces собраны фотографии 15 человек с разными выражениями лиц на каждой фотографии (рис. 8.24).

Имя каждого файла в этой базе данных выглядит следующим образом: subject01.sad. Здесь сначала идет слово subject (субъект, в нашем случае человек), далее порядковый номер человека (01, 02, 03, ...) и в конце — характеристика выражения лица на фотографии. Например, характеристика sad означает грустное лицо, happy — веселое или счастливое и т. п.

Функция `get_images()` считывает каждую фотографию кроме тех, которые имеют окончание .happy, и выделяет ту область изображения, где находится лицо. Фотографии с веселым выражением лиц (.happy) будут использоваться в другой про-

Рис. 8.24. Образцы фотографий из базы данных Yale Faces

грамме распознавания лиц. Они станут тестовой выборкой — т. е. теми фотографиями, на которых будет проверяться качество распознавания.

В этой функции также из каждого названия файла извлекается номер человека на фотографии и сохраняется в списке `labels`. Каждая фотография в итоге будет сопоставлена с этим номером.

Метод `faceCascade.detectMultiScale()` определяет области на фотографиях, где есть человеческие лица. Он возвращает список с параметрами `[x, y, w, h]` для каждого найденного лица. Эти параметры описывают прямоугольную область в том месте, где было найдено лицо.

Теперь рассмотрим параметры этого метода:

- `image` — исходное изображение;
- `scaleFactor` — этот параметр определяет ту величину, на которую будет увеличиваться скользящее окно в процессе поиска лиц на изображении. Значение `1.1` означает, что найденный фрагмент будет увеличен на 10%, `1.05` — на 5% и т. д. Чем больше это значение, тем быстрее работает алгоритм;
- `minNeighbors` — число пикселов вокруг центрального пиксела. Оптимальное значение — от 3 до 6 (чем больше это значение, тем чаще алгоритм будет пропускать реальные лица, и пойдут ложные срабатывания);
- `minSize` — минимальный размер лица на фотографии (размера 30×30 пикселов обычно вполне достаточно).

Приведенный в листинге 8.13 программный код позволит создать набор лиц и соответствующих им меток. В процессе работы программы обрабатываемые изображения будут отображаться в окне терминала (команда `cv2.imshow("", image[y: y + h, x: x + w])`), а все обработанные изображения — записаны в файлы в папку ./DataSet1/ с помощью следующей команды:

```
cv2.imwrite ('./DataSet1/user.' + str(subject_number) +
             '.' + str(count) +  '.jpg', image[y:y + h, x:x + w])
```

Выполнение последней команды не является обязательным — здесь это сделано в учебных целях и для отслеживания процессов, происходящих при работе программы.

Теперь нужно реализовать программный код, который научит компьютер распознавать найденные лица (листинг 8.14).

Листинг 8.14

```
# Listing 8_14
# Это промежуточный модуль, он не является рабочим
# Путь к фотографиям
path = './YaleFace/yalefaces/'
# Получаем лица и соответствующие им номера
images, labels = get_images(path)
cv2.destroyAllWindows()

# Обучаем программу распознавать лица
recognizer.train(images, np.array(labels))
# Сохраняем результат тренировки
recognizer.write('./YaleFace/Yale_face2.yml')
print('Обучение закончено')
```

Здесь в первой строке указан путь к нашим фотографиям (в нашем примере обучающая выборка фотографий помещена в папку ./YaleFace/yalefaces/). Затем мы обращаемся к функции `get_images(path)`, которая формирует список (массив) с фотографиями из обучающей выборки и их метками. А дальше запускаем метод тренировки модели распознавания с помощью алгоритма LBP:

```
recognizer.train(images, np.array(labels))
```

Ничего сверхъестественного в этом процессе нет — просто в виде параметров мы передаем в этот метод значения, полученные после работы функции `get_images()`. Все остальное программа сделает сама.

В листинге 8.15 приведен полный текст этой программы.

Листинг 8.15

```
# Listing 8_15
import cv2
import os
import numpy as np
from PIL import Image

# Загрузка каскадов Хаара для поиска лиц
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

# Формирование локального бинарного шаблона
recognizer = cv2.face.LBPHFaceRecognizer_create(1, 8, 8, 8, 123)
```

```
def get_images(path):
    # Ищем все фотографии и записываем их в image_paths
    image_paths = [os.path.join(path, f)
                   for f in os.listdir(path) if not f.endswith('.happy')]
    count = 0
    images = []
    labels = []

    for image_path in image_paths:
        # Переводим изображение в черно-белый формат и приводим его
        # к формату массива
        gray = Image.open(image_path).convert('L')
        image = np.array(gray, 'uint8')
        # Из каждого имени файла извлекаем номер человека,
        # изображенного на фото
        subject_number = int(os.path.split(
            image_path)[1].split(".")[0].replace("subject", ""))

        # Определяем области, где есть лица
        faces = faceCascade.detectMultiScale(image, scaleFactor=1.1,
                                             minNeighbors=5,
                                             minSize=(30, 30))
        # Если лицо нашлось, добавляем его в список images,
        # а соответствующий ему номер - в список labels
        for (x, y, w, h) in faces:
            images.append(image[y: y + h, x: x + w])
            labels.append(subject_number)
            # В окне показываем изображение
            cv2.imshow("", image[y: y + h, x: x + w])
            cv2.waitKey(50)
            count += 1
            # Сохраняем лицо
            cv2.imwrite('./DataSet1/user.' + str(subject_number) +
                        '.' + str(count) + '.jpg',
                        image[y:y + h, x:x + w])
    return images, labels

# Путь к фотографиям
path = './YaleFace/yalefaces/'
# Получаем лица и соответствующие им номера
images, labels = get_images(path)
cv2.destroyAllWindows()

# Обучаем программу распознавать лица
recognizer.train(images, np.array(labels))
# Сохраняем результат тренировки
recognizer.write('./YaleFace/Yale_face2.yml')
print('Обучение закончено')
```

Запустим этот программный код на выполнение. В процессе работы программы на экране будут появляться окна с теми фотографиями, которые загружаются из набора обучающих данных. Некоторые из этих окон в качестве примера приведены на рис. 8.25.

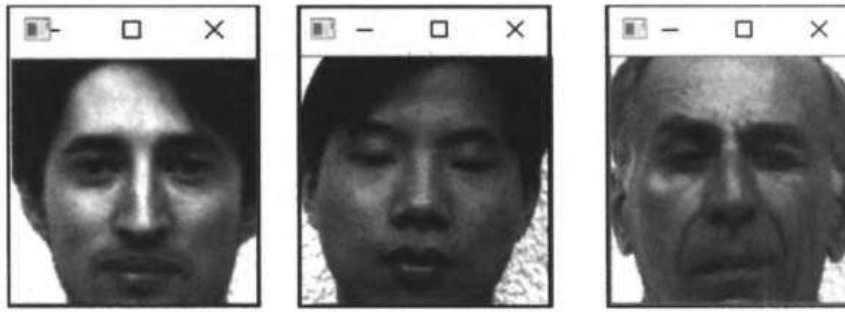


Рис. 8.25. Примеры загружаемых фотографий из набора обучающих данных

Когда процесс обучения будет завершен, все обработанные изображения будут сохранены в папке ./DataSet1/. Пример содержимого этой папки приведен на рис. 8.26.

Рис. 8.26. Папка с обработанными фотографиями из набора обучающих данных

Обученная модель будет сохранена в файле ./YaleFace/Yale_face2.yml, который можно найти в дочернем каталоге, где развернуто само приложение. Содержимое этого файла показано на рис. 8.27.

Файл обученной модели содержит бинарные гистограммы каждого изображения. которое присутствовало в обучающей выборке, и цифровые метки этих гистограмм.

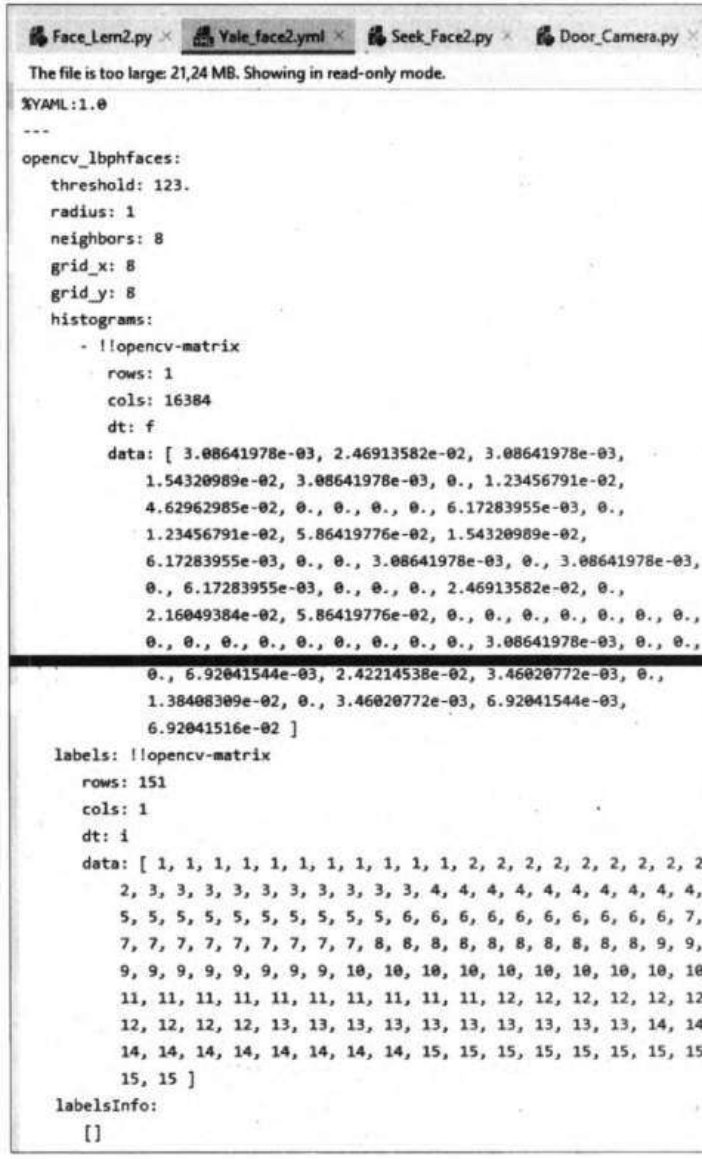


Рис. 8.27. Примеры содержимого файла бинарных гистограмм лиц конкретных людей

Теперь с использованием такой обученной модели можно найти фото именно этих людей на любых изображениях.

8.12.2. Пример программы распознавания лиц конкретных людей на фотографиях

Итак, у нас есть обученный «распознаватель» лиц и набор тестовых «счастливых» лиц. Программный код для распознавания всех этих людей (на примере тестовых фотографий) приведен в листинге 8.16.

Листинг 8.16

```
# Listing 8_16
# Это промежуточный модуль, он не является рабочим
# Формирование локального бинарного шаблона
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('./YaleFace/Yale_face2.yml')

path = './YaleFace/yalefaces/'
print(path)
# Создаем список фотографий для распознавания
image_paths = [os.path.join(path, f)
               for f in os.listdir(path) if f.endswith('.happy')]

for image_path in image_paths:  # Ищем лица на фотографиях
    gray = Image.open(image_path).convert('L')
    image = np.array(gray, 'uint8')
    faces = faceCascade.detectMultiScale(image,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(30, 30))

    for (x, y, w, h) in faces:
        # Если лица найдены, пытаемся распознать их.
        # Функция recognizer.predict в случае успешного распознавания
        # возвращает номер и параметр confidence.
        # Параметр confidence указывает на уверенность алгоритма,
        # что это именно тот человек. Чем он меньше, тем больше уверенность
        number_predicted, conf = recognizer.predict(
                 image[y: y + h, x: x + w])

        # Извлекаем настоящий номер человека на фото и сравниваем с тем,
        # что выдал алгоритм
        number_actual = int(os.path.split(image_path)[1].split(".")[0].
                            replace("subject", ""))

        if number_actual == number_predicted:
            print("{} is Correctly Recognized with confidence {}".
                  format(number_actual, conf))
        else:
            print("{} is Incorrect Recognized as {}".
                  format(number_actual, number_predicted))

        cv2.imshow("Recognizing Face", image[y: y + h, x: x + w])
        cv2.waitKey(1000)
```

В этой программе в цикле определяется расположение лица на каждом из тестовых фото (тех, что с окончанием .happy в имени фото — т. е. счастливых, улыбающихся лиц). Все параметры и процедуры такие же, что и на предыдущем этапе.

Для каждого найденного лица запускается метод `recognizer.predict()`, возвращающий номер-идентификатор субъекта, который, предположительно, находится на фото, а также параметр `confidence` (уверенность распознавания). Далее мы сравниваем значение, которое нам вернул метод распознавания, с реальным номером субъекта. Если они равны, то выводим на печать, — распознавание прошло успешно.

Далее в консоль выводятся результаты распознавания для каждой фотографии из контрольной выборки и окно с изображением лиц из контрольной выборки. В листинге 8.17 приведен полный текст этой программы.

Листинг 8.17

```
# Listing 8_17
import cv2
import os
import numpy as np
from PIL import Image

# Загрузка каскадов Хаара для поиска лиц
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

# Формирование локального бинарного шаблона
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('./YaleFace/Yale_face2.yml')

path = './YaleFace/yalefaces/'
print(path)
# Создаем список фотографий для распознавания
image_paths = [os.path.join(path, f)
               for f in os.listdir(path) if f.endswith('.happy')]

for image_path in image_paths:  # Ищем лица на фотографиях
    gray = Image.open(image_path).convert('L')
    image = np.array(gray, 'uint8')
    faces = faceCascade.detectMultiScale(image,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(30, 30))

    for (x, y, w, h) in faces:
        # Если лица найдены, пытаемся распознать их.
        # Функция recognizer.predict в случае успешного распознавания
        # возвращает номер и параметр confidence.
        # Параметр confidence указывает на уверенность алгоритма,
        # что это именно тот человек. Чем он меньше, тем больше уверенность
        number_predicted, conf = recognizer.predict(
            image[y: y + h, x: x + w])
```

```
        # Извлекаем настоящий номер человека на фото и сравниваем с тем,
        # что выдал алгоритм
        number_actual = int(os.path.split(image_path)[1].split(".")[0].
                            replace("subject", ""))

        if number_actual == number_predicted:
            print("{} is Correctly Recognized with confidence {}".
                  format(number_actual, conf))
        else:
            print("{} is Incorrect Recognized as {}".
                  format(number_actual, number_predicted))

        cv2.imshow("Recognizing Face", image[y: y + h, x: x + w])
        cv2.waitKey(1000)

cv2.destroyAllWindows()
```

Рис. 8.28. Примеры загружаемых фотографий из набора тестовых данных

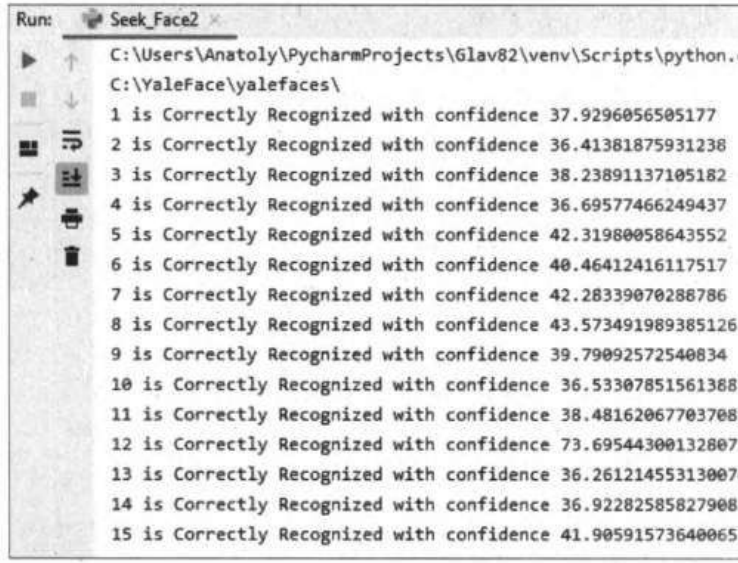


Рис. 8.29. Итоговая оценка успешности распознавания людей из тестовой выборки фотографий

Запустим программу на выполнение. В процессе ее работы будут выводиться окна с фотографиями тех людей, которые поданы на вход модуля распознавания лиц. Примеры с этими изображениями приведены на рис. 8.28.

По завершении работы программы в окне терминала будут показаны результаты распознавания каждого человека, чье фото присутствовало в тестовой выборке (рис. 8.29). Как можно видеть, программа успешно распознала каждого человека по предъявленному ей фото.

8.13. Создание пользовательской модели распознавания людей в видеопотоке с видеокамеры в OpenCV

В предыдущем разделе мы рассмотрели пример программы распознавания конкретных людей на фотографиях. При этом для обучения модели были использованы уже готовые наборы обучающей и тестовой выборок. На практике чаще приходится сталкиваться с потребностями пользователей обнаруживать конкретных людей не на фотографиях, а на изображениях с видеокамер. Это системы допуска персонала в помещения, поиск террористов по видеопотоку с видеокамер, установленных в общественных местах (транспорт, вокзалы, аэропорты, улицы и др.), различные охранные системы и т. п. В этом разделе мы как раз и рассмотрим примеры программ для решения такого класса задач. При этом не станем использовать готовую обучающую выборку изображений, а сформируем собственную. Реализацию подобного класса задач можно разделить на три этапа:

- создание набора данных для обучения модели поиска конкретного человека;
- обучение модели на основе пользовательского набора данных;
- распознавание (поиск) конкретного человека на основе обученной модели.

Итак, приступим к изучению каждого из этих этапов на конкретных примерах программного кода.

8.13.1. Пример программы формирования обучающей выборки пользователя для тренировки модели распознавания конкретных людей

Чтобы распознать (обнаружить) конкретного человека, нужно сначала сформировать базу данных с изображениями (фотографиями) этого человека — собрать несколько десятков фотографий индивидуума и обработать их в графическом редакторе: выделить лица на фото, сделать их одного размера, сохранить каждое лицо в виде отдельного файла. Для упрощения выполнения этой работы напишем программу, которая будет искать лицо человека в видеопотоке с веб-камеры, обрабатывать кадры и сохранять их в виде отдельных файлов в указанной папке компьютера. Для формирования нашей обучающей выборки ограничимся 30 фотографиями. В листинге 8.18 приведен программный код выполнения такой процедуры.

Листинг 8.18

```
# Listing 8_18
import cv2

cam = cv2.VideoCapture(0, cv2.CAP_DSHOW)
face_detector = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
# Вводим id лица, которое добавляется в имя и потом будет
# использоваться при  распознавании.
face_id = "Mark"
print("\n [INFO] Захват лица. Смотрите в камеру и ждите...")
count = 0

while True:
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        count += 1
        # Сохраняем лицо
        cv2.imwrite('./DataSet/user.' + str(face_id) + '.' +
                    str(count) + '.jpg',  gray[y:y+h, x:x+w])
    cv2.imshow('image', img)
    k = cv2.waitKey(100) & 0xff  # 'ESC'
    if k == 27:
        break
    elif count >= 30:  # Если сохранили 30 изображений, выход.
        Break

print("\n Программа завершена")
cam.release()
cv2.destroyAllWindows()
```

В этой программе выполняются следующие действия. В первых строках подключаются необходимые модули и активируется видеопоток с видеокамеры: `cam = cv2.VideoCapture(0, cv2.CAP_DS_HOV)`. Далее создается объект `face_detector`, в который загружается файл haarcascade_frontalface_default.xml. Это примитивы Хаара, обученные для распознавания на изображениях любых лиц человека. Затем в переменную `face_id` записывается имя человека, которого мы собираемся идентифицировать на изображениях. В нашем примере искомому объекту присвоено имя `Mark`.

Затем в программе организован цикл. Внутри этого цикла выполняется обработка видеокадров. Изображение каждого кадра записывается в переменную `img`, цветное изображение трансформируется в черно-белое изображение (в градациях серого, переменная `gray`). В изображении распознается и выделяется область лица, которая

сохраняется в отдельном файле в папке ./DataSet. Этот цикл повторяется до тех пор, пока не будут сформированы 30 изображений лица. Запустим программу на выполнение и получим набор данных с изображением лица (рис. 8.30) — программа сформировала 30 файлов с лицом объекта `Mark` с именами: user.Mark1.jpg, user.Mark2.jpg, user.Mark3.jpg, ..., user.Mark30.jpg.

Рис. 8.30. Набор данных с изображением лица персоны Mark (обучающая выборка данных)

Используя эту программу, можно сформировать обучающий набор данных для любого человека. Для этого достаточно, чтобы конкретный человек 2–3 секунды посидел перед камерой, либо подержать перед камерой фотографию нужного человека.

8.13.2. Пример программы обучения модели на основе обучающей выборки пользователя

Имея обучающий набор данных, можно приступить ко второму этапу — к тренировке модели. Для этого необходимо сформированный на предыдущем шаге набор данных (изображения) передать тренеру. В результате тренировки модели мы сформируем и сохраним файл с расширением yml. В этом файле будет записана бинарная гистограмма того человека, чьи фотографии были записаны в файлы обучающей выборки. В нашем примере это персона с именем `Mark`.

Для работы программы «тренер» необходим модуль `opencv-contrib-python`, который изначально был установлен в наш проект. В листинге 8.19 приведен программный код для обучения нашей модели.

Листинг 8.19

```
# Listing 8_19
import cv2
import numpy as np
import os

path = './DataSet/'  # папка с набором тренировочных фото
recognizer = cv2.face.LBPHFaceRecognizer_create()

# Функция чтения изображений из папки с тренировочными фото
def getImagesAndLabels(path):
    # Создаем список файлов в папке patch
    imagePaths = [os.path.join(path, f) for f in os.listdir(path)]
    face = []  # тут храним массив картинок
    ids = []  # храним id лица
    for imagePath in imagePaths:
        img = cv2.imread(imagePath)
        # Переводим изображение, тренер принимает изображения
        # в оттенках серого
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        face.append(img)  # записываем тренировочное фото в массив
        # Получаем id фото из его названия
        id = int(os.path.split(imagePath)[-1].split(".")[2])
        ids.append(id)  # записываем id тренировочного фото в массив
    return face, ids

# Чтение тренировочного набора фотографий из папки path
faces, ids = getImagesAndLabels(path)
# Тренируем модель распознавания
recognizer.train(faces, np.array(ids))
# Сохраняем результат тренировки
recognizer.write('./Mark_model/face_Mark.yml')
```

В этой программе в переменную `path` мы заносим путь к папке (`'./DataSet/'`), где хранятся изображения обучающей выборки данных. Затем создаем объект `recognizer` на основе класса:

```
cv2.face.LBPHFaceRecognizer_create()
```

В следующих строках реализована функция `getImagesAndLabels(path)`, которая считывает все фотографии из обучающего набора данных и формирует два массива:

- `face` — массив с изображениями человека из обучающей выборки;
- `ids` — массив идентификаторов изображений человека из обучающей выборки.

Затем командой `recognizer.train(faces, np.array(ids))` эти два массива передаются тренеру. По сути, в этой строке запускается процесс тренировки (обучения) моде-

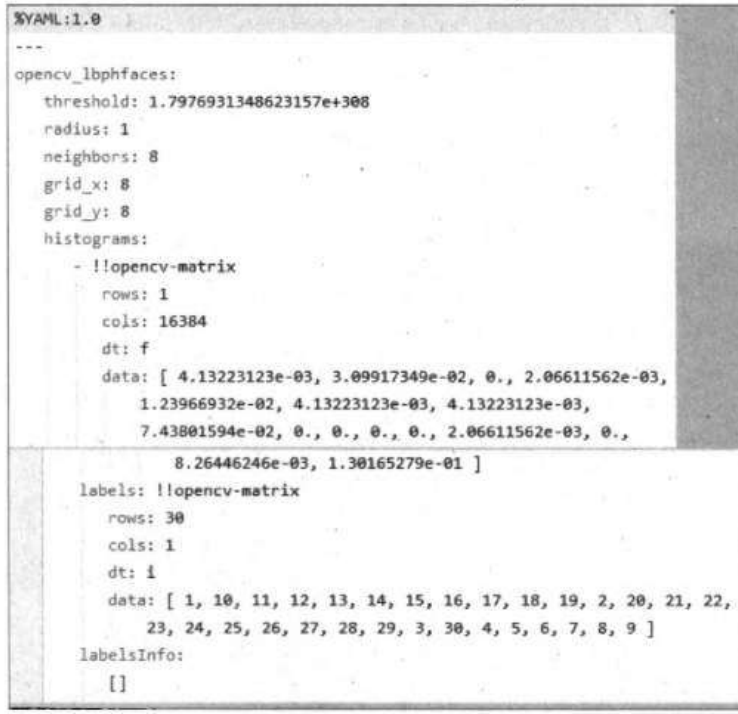


Рис. 8.31. Фрагмент содержимого файла бинарных гистограмм объекта Mark

ли. По его завершении обученная модель сохранится в файле ./Mark_model/faceMark.yml. Фрагмент содержания этого файла для объекта `Mark` приведен на рис. 8.31.

В дальнейшем файл с обученной моделью faceMark.yml можно использовать для поиска этого человека на любых изображениях.

8.13.3. Программа распознавания лиц людей на основе обучающей выборки пользователя

Перейдем к последнему этапу нашей работы — напишем программу для поиска и распознавания объекта `Mark` на фотографиях и в видеоизображениях с видеокамеры. В листинге 8.20 приведен соответствующий программный код.

Листинг 8.20

```
# Listing 8_20
import cv2

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('./Mark_model/face_Mark.yml')

faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

```
# Тип шрифта
font = cv2.FONT_HERSHEY_SIMPLEX

# Список имен для id
names = ['None', 'Mark']

cam = cv2.VideoCapture(0, cv2.CAP_DSHOW)
cam.set(3, 640)  # размер видеокадра - ширина
cam.set(4, 480)  # размер видеокадра - высота

while True:
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray, scaleFactor=1.2,
        minNeighbors=5, minSize=(10, 10),)

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        id, confidence = recognizer.predict(gray[y:y + h, x:x + w])
        # print(id)

        # Проверяем, что лицо распознано
        if (confidence < 100):
            id_obj = names[1]
            confidence = "  {0}%".format(round(100 - confidence))
        else:
            id_obj = names[0]
            confidence = "  {0}%".format(round(100 - confidence))

        cv2.putText(img, str(id_obj), (x + 5, y - 5),
                    font, 1, (255, 255, 255), 2)
        cv2.putText(img, str(confidence), (x + 5, y + h - 5),
                    font, 1, (255, 255, 0), 1)

    cv2.imshow('camera', img)

    k = cv2.waitKey(10) & 0xff  # 'ESC' для Выхода
    if k == 27:
        break
cam.release()
cv2.destroyAllWindows()
```

В этой программе после импорта необходимых модулей создается объект `recognizer`, **и в него загружается наша модель, обученная распознавать объект** `Mark`:

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('./Mark_mosel/face_Mark.yml')
```

Далее мы создаем объект для расписывания лиц на основе примитивов Хаара:

```
faceCascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
```

Затем в программе задаются тип шрифта, которым будет печататься текст на распознанных изображениях, и массив с именами `names = ['None', 'Mark']`. В этот массив записано всего два значения. Первое значение: `'None'` — будет выведено на тех изображениях, где окажется неизвестная личность. Второе значение: `'Mark'` — на тех изображениях, где окажется наш объект `Mark`.

Итак, у нас все готово. Запускаем нашу программу и усаживаем перед камерой того человека, на фотографиях которого мы обучали нашу модель, а именно — объект с именем `Mark`. Результат работы программы представлен на рис. 8.32.

Рис. 8.32. Личность Mark, опознанная обученной моделью face_Mark.yml

Рис. 8.33. Личность человека, не опознанная обученной моделью face_Mark.yml как Mark

Как можно видеть, обученная модель уверенно сказала, что перед нами именно Mark.

Теперь посадим перед камерой другого человека (рис. 8.33). В этом случае обученная модель ответила, что на показанном ей изображении не Mark.

Подадим на вход обученной модели изображение с несколькими людьми. Программа выдаст следующий ответ (рис. 8.34) — она уверенно сказала, что искомая личность отсутствует на этом изображении.

И в заключение подадим на вход нашей программы изображение с двумя персонами, одна из которых является личностью с именем `Mark`, а вторая — нет. Программа выдаст следующий ответ (рис. 8.35).

Как можно видеть, на этом изображении программа корректно нашла личность с именем `Mark`. И это несмотря на то, что в обучающей выборке данных `Mark` был в очках, а на этом кадре с видеокамеры он без очков.

Итак, мы рассмотрели все три этапа создания приложений для распознавания конкретных людей: формирование обучающей выборки, обучение модели, поиск на изображениях конкретного человека.

Рис. 8.34. Личность Mark не опознана на изображении с двумя лицами

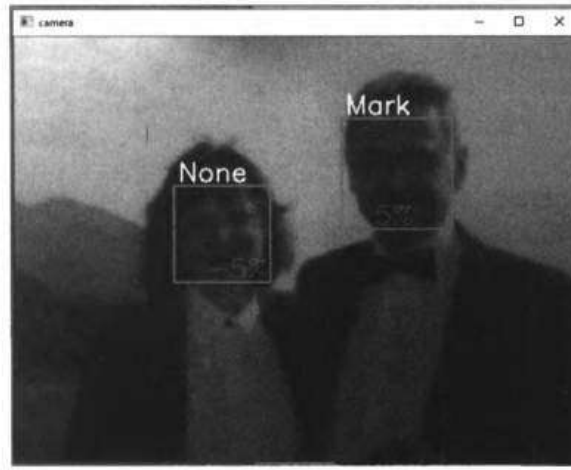


Рис. 8.35. Личность Mark, опознанная на изображении с двумя лицами

8.14. Краткие итоги главы

В этой главе представлены возможности популярной библиотеки OpenCV. Мы познакомились с алгоритмом распознавания объектов на изображениях на основе примитивов Хаара. Рассмотрели практические примеры использования возможностей библиотеки OpenCV для распознавания различных объектов на изображениях и в видеокадрах. Научились обучать модели нейронных сетей распознавать и искать на фотографиях и в видеопотоках конкретные персоны. Показали, что при использовании этой библиотеки можно значительно сократить затраты труда на создание программных модулей.

ПРИЛОЖЕНИЕ

Описание электронного архива

Электронный архив с материалами к этой книге можно скачать с сервера издательства «БХВ» по ссылке: **https://zip.bhv.ru/9785977518185.zip**, а также со страницы книги на сайте **https://bhv.ru/** В архиве (табл. П1) содержатся все программные модули, которые приведены в книге. Кроме них в архиве содержатся файлы с рисунками.

Таблица П1. *Содержание электронного архива, сопровождающего книгу*

Папка или файл	Описание	Глава
Listings\Glava2	**Листинги программ к главе 2**	
Listing 2_1.py	Использование ui-файла внутри программы на Python	2
Listing 2_2.py	Выполнение простейших действий на Python: сложение, вычитание, умножение, деление, возведение в степень	2
Listing 2_3.py	Пример программного кода описания и вызова функции	2
Listing 2_4.py	Программный код, который обеспечивает вывод данных	2
Listing 2_5.py	Выполнение простейших действий на Python с выводом результатов на печать	2
Listing 2_6.py	Пример описания функции	2
Listing 2_7.py	Пример обращения к функции	2
Listing 2_7_1.py	Объединенный код листингов 2_6, 2_7	2
Listing 2_8.py	Пример описания функции (параметры)	2
Listing 2_9.py	Пример обращения к функции (аргументы)	2
Listing 2_9_1.py	Объединенный код листингов 2_8, 2_9	2
Listing 2_10.py	Пример использования разветвления `if`	2
Listing 2_11.py	Пример использования разветвления `if`	2
Listing 2_12.py	Вывод таблицы квадратов первых 10 натуральных чисел	2
Listing 2_13.py	Пример программирования циклов `while`	2
Listing 2_14.py	Пример программирования циклов `for`	2
Listing 2_15.py	Пример перебора массива с помощью цикла	2

Таблица П1 (продолжение)

Папка или файл	Описание	Глава
Listing 2_16-2_20_2.py	Пример создания класса `Cat` (кошки)	2
Listing 2_21-2_23_1.py	Пример создания класса `Car` (автомобиль)	2
Listings\Glava3	**Листинги программ к главе 3**	
Listing 3_1.py	Программная реализация сумматора класса нейрона	3
Listing 3_2.py	Изменение параметров сумматора класса нейрона	3
Listing 3_2_1.py	Объединенный рабочий код листингов 3_1, 3_2	3
Listing 3_3.py	Пример программы пороговой функции единичного скачка	3
Listing 3_4.py	Пример программы пороговой сигмоидальной функции	3
Listings\Glava4	**Листинги программ к главе 4**	
Listing 4_1.py	Программная реализация искусственного нейрона (персептрон)	4
Listing 4_2.py	Пример обращения к искусственному нейрону	4
Listing 4_3.py	Полный текст программы испытания персептрона	4
Listing 4_4.py	Пример формирования обучающей выборки с идеальным изображением цифр от 0 до 9	4
Listing 4_5.py	Пример реализации простейшей функции с именем `perceptron`	4
Listing 4_6.py	Пример функции уменьшения значений весов	4
Listing 4_7.py	Пример функции увеличения значения весов	4
Listing 4_8.py	Пример модуля тренировки нейронной сети	4
Listing 4_9.py	Полный текст программы обучения сети распознаванию цифры	4
Listing 4_10.py	Проверка работы программы распознавания цифр на обучающей выборке	4
Listing 4_11.py	Пример тестовой выборки (различные варианты изображения цифры 5)	4
Listing 4_12.py	Проверка работы программы распознавания цифр на тестовой выборке	4
Listing 4_13.py	Пример тренировки сети (линейная аппроксимация)	4
Listing 4_14.py	Полный текст программы (линейная аппроксимация)	4
Listing 4_15.py	Программный код класса персептрон	4
Listing 4_16.py	Программный код проверки правильности загрузки параметров цветков ириса	4
Listing 4_17.py	Программный код формирования обучающей выборки на примере цветков ириса	4
Listing 4_18.py	Программный код визуализации обучающей выборки	4

***Таблица П1** (продолжение)*

Папка или файл	Описание	Глава
Listing 4_18_1.py	Объединенный рабочий код листингов 4_15–4_18	4
Listing 4_19.py	Программный код обучения персептрона	4
Listing 4_19_1.py	Объединенный рабочий код листингов 4_18_1, 4_19	4
Listing 4_20.py	Программный код проверки обучения персептрона	4
Listing 4_20_1.py	Объединенный рабочий код листингов 4_19_1, 4_20	4
Listing 4_21.py	Программный код вывода результатов работы персептрона	4
Listing 4_21_1.py	Объединенный рабочий код листингов 4_20_1, 4_21	4
Listing 4_22.py	Программный код визуализации разделительной границы при классификации объектов	4
Listing 4_22_1.py	Объединенный рабочий код листингов 4_21_1, 4_22	4
Listing 4_23.py	Программный код реализации адаптивного персептрона	4
Listing 4_24.py	Программный код проверки эффективности обучения адаптивного персептрона	4
Listing 4_24_1.py	Объединенный рабочий код листингов 4_23, 4_24	4
Listing 4_25.py	Программный код выполнения стандартизации обучающей выборки	4
Listing 4_25_1.py	Объединенный рабочий код листингов 4_24, 1-4_25	4
Listing 4_26.py	Программный код проверки успешности обучения адаптивного персептрона	4
Listing 4_27.py	Полный текст программ из *разд. 4.10, 4.11*	4
Listings\Glava5	**Листинги программ к главе 5**	
Listing 5_1.py	Программный код для исследования работы простейшего нейрона (влияние веса связи на значение функции активации)	5
Listing 5_2.py	Программный код для исследования работы простейшего нейрона (влияние коэффициента смещения *b* на значение функции активации)	5
Listing 5_3.py	Программный код простейшего нейрона	5
Listing 5_4.py	Программный код однослойной нейронной сети прямого распространения	5
Listing 5_5.py	Программный код расчета среднеквадратической ошибки	5
Listing 5_6.py	Программный код распознавания людей по их росту и весу	5
Listing 5_7.py	Программный код обращения к сети для распознавания пола людей по их росту и весу	5
Listing 5_7_1.py	Объединенный код листингов 5_6, 5_7	5
Listing 5_7_2.py	Объединенный рабочий код листингов 5_7, 5_7_1. Полный листинг программы распознавания пола людей по их росту и весу	5

Таблица П1 (продолжение)

Папка или файл	Описание	Глава
Listings\Glava6\	**Листинги программ к главе 6**	
Listing 6_1.py	Полный текст программы, которая создает нейронную сеть	6
Listing 6_2.py	Текст программы просмотра содержимого набора данных	6
Listing 6_3.py	Код программы формирования обучающего набора данных	6
Listing 6_3_1.py	Полный рабочий код программы для запуска процесса обучения	6
Listing 6_4.py	Полный код данной программы формирования обучающего набора данных и обучения сети	6
Listing 6_5.py	Код программы вывода результатов обучения сети	6
Listing 6_5_1.py	Объединенный рабочий код листингов 6_4, 6_5	6
Listing 6_5_2.py	Дополненный рабочий код листинга 6_5_1	6
Listing 6_6.py	Пример программного кода сохранения сети net в файл MyNet.txt	6
Listing 6_7.py	Пример проверки корректности работы сети, загруженной из файла	6
Listing 6_8.py	Пример сохранения обученной сети в XML-файле	6
Listing 6_9.py	Полный текст программного кода создания, сохранения и использования сети	6
Listing 6_10.py	Пример реализации нейронной сети, предсказывающей успешность рыбной ловли	6
Listing 6_11.py	Пример использования нейронной сети, предсказывающей успешность рыбной ловли	6
Listing 6_12.py	Программный код для формирования различных типов матриц (массивов)	6
Listing 6_13.py	Пример построения и вывода графика зависимости двух величин	6
Listing 6_14.py	Получение наименования ключей и других сведений о структуре набора данных iris_dataset	6
Listing 6_15.py	Построение матрицы диаграмм рассеяния с использованием библиотеки seaborn	6
Listing 6_16.py	Полный код программы подготовки и анализа набора данных цветков ириса	6
Listing 6_17.py	Полный текст программы для тренировки сети и ее использования для предсказания сорта цветка ирис	6
Listing 6_18.py	Пример использования библиотеки scikit-learn для создания и обучения интеллектуального классификатора	6
Listing 6_18_1.py	Дополненный программный код листинга 6_18	6
Listing 6_19.py	Формирование обучающей выборки данных на примере цветков ириса	6

Таблица П1 (продолжение)

Папка или файл	Описание	Глава
Listing 6_20.py	Пример программы стандартизации обучающей выборки на примере цветков ириса	6
Listing 6_21.py	Пример запуска тренировки модели нейронной сети	6
Listing 6_22.py	Пример визуализации разделения классов цветков ириса Полный рабочий код листингов 6_19–6_21	6
Listing 6_22_1.py	Дополнения к листингу 6_22	6
Listing 6_23.py	Полный текст программы сети для классификации цветков ириса	6
Listing 6_24.py	Пример тренировки сети на основе логистической регрессии	6
Listing 6_24_1.py	Полный код листинга 6_24 с дополнениями	6
Listing 6_25.py	Пример предсказания вероятности принадлежности к тому или иному виду для трех образцов цветков ириса	6
Listing 6_26.py	Полный текст программы нейронной сети для классификации объектов на примере цветков ириса	6
Listing 6_27.py	Создание тренировочного и тестового набора данных MNIST для сверточной нейронной сети	6
Listing 6_28.py	Пример просмотра диапазона значения пикселов в изображении тренировочного набора данных	6
Listing 6_29.py	Пример вывода на экран части изображений цифр из тестового набора данных	6
Listing 6_30.py	Пример распечатки структуры массива меток и метки первых трех цифр (5, 0 и 4)	6
Listing 6_31.py	Пример использования функции `add()` для добавления слоев в модель сети	6
Listing 6_32.py	Пример визуализации результатов обучения нейронной сети	6
Listing 6_33.py	Добавление слоя MaxPooling2D в модель нейронной сети	6
Listing 6_34.py	Пример запуска процедуры обучения сети для пяти эпох и отображения графика точности и потерь	6
Listing 6_35.py	Пример сохранения конфигурации и параметров обученной сети	6
Listing 6_36.py	Пример использования метода `predict_classes()` для возврата меток классов	6
Listing 6_37.py	Полный текст программы формирования и использования модели нейронной сети, построенной на основе библиотеки Keras	6
Listing 6_38.py	Подключение необходимых библиотек для построения моделей сетей на основе tensorflow	6
Listing 6_39.py	Пример формирования модели сети на основе tensorflow	6
Listing 6_40.py	Программа формирования модели нейронной сети (класс `Sequential`)	6

Таблица П1 (продолжение)

Папка или файл	Описание	Глава
Listing 6_41.py	Проверка корректности выдаваемых сетью результатов	6
Listing 6_42.py	Вывод значений параметров сети для каждого уровня (веса и смещения)	6
Listing 6_43.py	Полный текст программы формирования и обучения нейронной сети на основе tensorflow	6
Listing 6_44.py	Загрузка набора данных Fashion MNIST из библиотеки TensorFlow	6
Listing 6_45.py	Пример просмотра структуры массивов, содержащихся в обучающем и тренировочном наборе данных Fashion MNIST	6
Listing 6_46.py	Пример просмотра изображений, содержащихся в тренировочном наборе данных	6
Listing 6_47.py	Пример просмотра значений пикселов в изображениях, содержащихся в тренировочном наборе данных	6
Listing 6_48.py	Вывод на экран первых 25 изображений из тестового набора данных	6
Listing 6_49.py	Создание в модели нейронной сети трех слоев	6
Listing 6_50.py	Пример предсказания класса одежды	6
Listing 6_50_1.py	Объединенный рабочий код листингов 6_49, 6_50	6
Listing 6_51.py	Пример функций, которые визуально отображают результаты предсказания	6
Listing 6_52.py	Пример обращения к функциям, которые визуально отображают результаты предсказания	6
Listing 6_53.py	Визуализация предсказания для 12-го изображения в массиве входных данных (ошибочный прогноз)	6
Listing 6_53_1.py	Объединенный рабочий код листингов 6_49–6_53	6
Listing 6_54.py	Вывод информации о вероятности правильного предсказания для первых 15 изображений из обучающего набора данных	6
Listing 6_54_1.py	Объединенный рабочий код листингов 6_53_1, 6_54	6
Listing 6_55.py	Предсказание класса на одном изображении из тестового набора данных	6
Listing 6_55_1.py	Объединенный рабочий код листингов 6_54_1, 6_55	6
Listing 6_56.py	Полный текст программы для предсказания класса одежды	6
Listings\Glava7	**Листинги программ к главе 7**	
Listing 7_1.py	Программный код с использованием класса `ImagePrediction` (модель resnet50_weights)	7
Listing 7_2.py	Программный код с использованием класса `ImagePrediction` (модель inception_v3)	7

Таблица П1 (продолжение)

Папка или файл	Описание	Глава
Listing 7_3.py	Пример модели нейронной сети для обработки нескольких изображений	7
Listing 7_4.py	Пример модели нейронной сети YOLOv3 для обработки изображений	7
Listing 7_5.py	Пример модели нейронной сети YOLOv3 для обработки изображений, расположенных в различных папках компьютера	7
Listing 7_6.py	Пример модели нейронной сети YOLOv3 для обработки ограниченного числа объектов на изображении	7
Listing 7_7.py	Пример модели нейронной сети YOLOv3 для поиска множества объектов на изображении (оживленная улица)	7
Listing 7_8.py	Пример программного кода для обработки видеофайла (поиск автомобилей в потоке)	7
Listing 7_9.py	Пример программного кода для обработки видеофайла (файлы расположены в различных папках компьютера)	7
Listing 7_10.py	Пример программного кода поиска объектов в потоковых видео с видеокамер	7
Listing 7_11.py	Пример использования пользовательских функций при обработке видеофайлов	7
Listing 7_12.py	Пример пользовательской функции, которая обрабатывает видеофайл покадрово, находит в нем различные объекты и в режиме реального времени визуализирует результаты этой обработки	7
Listing 7_13.py	Пример пользовательской функции, которая обрабатывает видеофайл по секундам, находит в нем различные объекты и в режиме реального времени визуализирует результаты этой обработки	7
Listing 7_14.py	Пример программного модуля посекундной обработки видеофайла	7
Listing 7_15.py	Пример пользовательской функции, которая обрабатывает видеофайл по минутам, находит в нем различные объекты и в режиме реального времени визуализирует результаты этой обработки	7
Listing 7_16.py	Пример программного кода использования параметра, который позволяет указать продолжительность (количество секунд) обработки изображения с видеокамеры	7
Listing 7_17.py	Пример программы формирования структуры нейронной сети и ее обучения для работы с пользовательским набором данных (дорожные знаки)	7
Listing 7_18.py	Пример программного кода использования класса `imageai.Prediction.Custom` (распознавание дорожных знаков)	7

Таблица П1 (продолжение)

Папка или файл	Описание	Глава
Listing 7_19.py	Программный код для обучения пользовательских моделей нейронных сетей для обнаружения объектов в изображениях на наборе данных, созданных пользователем (модель YOLOv3)	7
Listing 7_20.py	Программный код для оценки в баллах точности сохраненных моделей обученных нейронных сетей	7
Listing 7_21.py	Программный код для испытания работы обученной модели нейронной сети (на примере обнаружения гарнитуры виртуальной реальности)	7
Listing 7_22.py	Программный код обнаружения объектов на видео и получения аналитических данных из видео с использованием собственной пользовательской модели YOLOv3	7
Listing 7_23.py	Пример пользовательской функции (вывод информации о каждом обработанном кадре видеоизображения)	7
Listing 7_24.py	Пример применения пользовательской функции (вывод информации о каждой обработанной секунде в видеоизображении)	7
Listing 7_25.py	Пример применения пользовательской функции (вывод информации о каждой обработанной минуте в видеоизображении)	7
Listing 7_26.py	Пример применения пользовательской функции (вывод информации о найденных объектах во всем видеофайле)	7
Listing 7_27.py	Программный код для формирования и обучения пользовательской нейронной сети (модель YOLOv3)	7
Listing 7_28.py	Программный код проверки работы обученной нейронной сети, которую натренировали на распознавание дорожных знаков «стоп» и «пешеходный переход»	7
Listings\Glava8	**Листинги программ к главе 8**	
Listing 8_1.py	Программа распознавания лиц на фото	8
Listing 8_2.py	Программа поиска лиц в видеопотоке с видеокамер	8
Listing 8_3.py	Программа распознавания глаз на фото	8
Listing 8_4.py	Программа распознавания улыбки на лице (с видеокамеры)	8
Listing 8_5.py	Программа распознавания российских автомобильных номеров на изображениях	8
Listing 8_6.py	Программа для выделения на видеокадрах обнаруженных автомобильных номеров	8
Listing 8_7.py	Программа формирования отдельного изображения с автомобильным номером на видеокадрах	8
Listing 8_8.py	Программа обнаружения автомобилей в потоке транспортных средств с видеокамеры	8
Listing 8_9.py	Программа распознавания различных объектов на изображениях из одного программного кода	8

Таблица П1 *(окончание)*

Папка или файл	Описание	Глава
Listing 8_10.py	Программа для обнаружения пешеходов в изображении	8
Listing 8_11.py	Программа-детектор пешеходов для изображений с видеокамеры	8
Listing 8_12.py	Программа формирования локального бинарного шаблона для поиска конкретных людей на изображениях	8
Listing 8_13.py	Пример функции, которая находит по определенному пути во всех фотографиях лица людей и сохраняет их	8
Listing 8_14.py	Программный код, который научит компьютер распознавать лица конкретных людей	8
Listing 8_15.py	Полный текст программы обучения нейронной сети распознавать лица конкретных людей	8
Listing 8_16.py	Программный код для проверки качества обучения нейронной сети распознаванию лиц конкретных людей (на примере тестовых фотографий)	8
Listing 8_17.py	Полный текст программы распознавания на фотографиях лиц конкретных людей	8
Listing 8_18.py	Программный код для формирования пользовательской обучающей выборки (для поиска конкретного человека на изображениях)	8
Listing 8_19.py	Пример программы обучения модели поиску конкретного человека (на основе обучающей выборки пользователя)	8
Listing 8_20.py	Программа поиска и распознавания конкретного человека на фотографиях и видеоизображениях с видеокамеры	8
Pictures\ Ris_Glava1	Рисунки к главе 1	1
Pictures\ Ris_Glava2	Рисунки к главе 2	2
Pictures\ Ris_Glava3	Рисунки к главе 3	3
Pictures\ Ris_Glava4	Рисунки к главе 4	4
Pictures\ Ris_Glava5	Рисунки к главе 5	5
Pictures\ Ris_Glava6	Рисунки к главе 6	6
Pictures\ Ris_Glava7	Рисунки к главе 7	7
Pictures\ Ris_Glava8	Рисунки к главе 8	8

Список литературы

Книги

1. Джоши П. Искусственный интеллект с примерами на Python: пер. с англ. — М.: Диалектика, 2019. — 448 с.
2. Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения: пер с англ. — СПб.: Питер, 2017. — 496 с.
3. Мюллер А. П., Гвидо С. Введение в машинное обучение с помощью Python: пер. с англ. — М.: Вильямс, 2017. — 480 с.
4. Мюллер Дж. П., Массарон Л. Искусственный интеллект для чайников: пер. с англ. — М.: Диалектика, 2019. — 384 с.
5. Прохоренок Н. А., Дронов В. А. Python 3 и PyQt 5. Разработка приложений. — СПб.: БХВ-Петербург, 2016. — 832 с.
6. Рашка С. Python и машинное обучение: пер. с англ. — М.: ДМК Пресс, 2017. – 418 с.
7. Реза Босаг З. Р., Бхарат Р. TensorFlow для глубокого обучения: пер. с англ. — СПб.: БХВ-Петербург, 2020. — 256 с.
8. Шолле Ф. Глубокое обучение на Python: пер. с англ. — СПб.: Питер, 2018. — 400 с.

Электронные ресурсы

1. Автоматическое отслеживание объекта на Python [Электронный ресурс] // Все о робототехнике : [сайт]. URL: https://robotos.in/uroki/avtomaticheskoe-otslezhivanie-ob-ekta-na-python (дата обращения: 11.12.2020).
2. Библиотеки Python для нейронных сетей [Электронный ресурс]. URL: https://otus.ru/nest/post/738/ (дата обращения: 11.12.2020).
3. Детектирование пешеходов беспилотным автомобилем. URL: https://www.youtube.com/watch?v=HWH3sz0lx9Y (дата обращения: 11.12.2020).

4. Как извлечь и распознать номер автомобиля с помощью Python? [Электронный ресурс]. URL: https://fooobar.com/questions/16977622/how-to-extract-and-recognize-the-vehicle-plate-number-with-python (дата обращения: 11.12.2020).
5. Как программисту стать специалистом по искусственному интеллекту [Электронный ресурс]. URL: https://tproger.ru/blogs/coder-to-artificial-intelligence/ (дата обращения: 11.12.2020).
6. Как начать работу с Keras, Deep Learning и Python [Электронный ресурс]. URL: https://www.reg.ru/blog/keras/ (дата обращения: 11.12.2020).
7. Лучшие дата-сеты для машинного обучения и анализа данных [Электронный ресурс]. URL: https://tproger.ru/translations/the-best-datasets-for-machine-learning-and-data-science/ (дата обращения: 11.12.2020).
8. Машинное обучение для начинающих: создание нейронных сетей [Электронный ресурс] // Изучаем Python 3 на примерах : [сайт]. URL: https://python-scripts.com/intro-to-neural-networks (дата обращения: 11.12.2020).
9. Научись работать с компьютерным зрением и программировать беспилотный автомобиль [Электронный ресурс] // Академия высоких технологий : [сайт]. URL: http://newgen.education/rosdc (дата обращения: 11.12.2020).
10. Обнаружение и распознавание лица на Python [Электронный ресурс] // Все о робототехнике : [сайт]. URL: https://robotos.in/uroki/obnaruzhenie-i-raspoznavanie-litsa-na-python (дата обращения: 11.12.2020).
11. Обнаружение лица и выделение характерных точек (Face Detection in Python) [Электронный ресурс] // Моделирование и распознавание 2D/3D-образов (Modeling and recognition of 2D/3D images) : [сайт]. URL: https://api-2d3d-cad.com/python-face-detection/ (дата обращения: 11.12.2020).
12. Обучи свою первую нейросеть: простая классификация [Электронный ресурс] // TensorFlow : [сайт]. URL: https://www.tensorflow.org/tutorials/keras/classification?hl=ru (дата обращения: 11.12.2020).
13. Пишем свою нейросеть: пошаговое руководство [Электронный ресурс] // proglib : [сайт]. URL: https://proglib.io/p/neural-nets-guide (дата обращения: 11.12.2020).
14. Программа OpenCV Python для обнаружения транспортных средств в видеокадре [Электронный ресурс] // Портал информатики для гиков : [сайт]. URL: http://espressocode.top/opencv-python-program-vehicle-detection-video-frame/ (дата обращения: 11.12.2020).
15. Простая нейронная сеть в 9 строк кода на Python [Электронный ресурс] // Neurohive : [сайт]. URL: https://neurohive.io/ru/tutorial/prostaja-nejronnaja-set-python/ (дата обращения: 11.12.2020).
16. Распознавание автомобильных номеров Opencv [Электронный ресурс] // PVSM.RU : [сайт]. URL: https://www.pvsm.ru/net/274725 (дата обращения: 11.12.2020).

17. Распознавание лица в OpenCV. Python [Электронный ресурс] // LinuxBlog.РФ : [сайт]. URL: https://линуксблог.рф/raspoznavanie-lica-v-opencv-python/ (дата обращения: 11.12.2020).

18. Распознавание номеров. Практическое пособие. Часть 1 [Электронный ресурс] // Habr : [сайт]. URL: https://habr.com/ru/post/432444/ (дата обращения: 11.12.2020).

19. Распознавание объектов на Python / Глубокое машинное обучение [Электронный ресурс] // </> itProger : [сайт]. URL: https://itproger.com/news/174 (дата обращения: 11.12.2020).

20. Распознаем лица на фото с помощью Python и OpenCV [Электронный ресурс] // Habr : [сайт]. URL: https://habr.com/ru/post/301096/ (дата обращения: 11.12.2020).

21. Сверточная нейронная сеть на Python и Keras [Электронный ресурс] // LinuxBlog.РФ : [сайт]. URL: https://xn--90aeniddllys.xn--p1ai/svertochnaya-nejronnaya-set-na-python-i-keres/ (дата обращения: 11.12.2020).

22. Сиборн — краткое руководство [Электронный ресурс] // CoderLessons.com : [сайт]. URL: https://coderlessons.com/tutorials/python-technologies/izuchai-siborna/siborn-kratkoe-rukovodstvo (дата обращения: 11.12.2020).

23. Создаем нейронную сеть InceptionV3 для распознавания изображений [Электронный ресурс] // Habr : [сайт]. URL: https://habr.com/ru/post/321834/ (дата обращения: 11.12.2020).

24. Построение сверточной нейронной сети (CNN) в Keras [Электронный ресурс]. URL: https://www.machinelearningmastery.ru/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 (дата обращения: 11.12.2020).

25. Топ 8 библиотек Python для машинного обучения и искусственного интеллекта [Электронный ресурс] // pythonist.ru : [сайт]. URL: https://yandex.ru/turbo?text=https%3A%2F%2Fpythonist.ru%2Ftop-8-bibliotek-python-dlya-mashinnogo-obucheniya-i-iskusstvennogo-intellekta%2F (дата обращения: 11.12.2020).

26. Топ-10 фреймворков для искусственного интеллекта. Часть 1 [Электронный ресурс] // vc.ru : [сайт]. URL: https://vc.ru/ml/80391-top-10-freymvorkov-dlya-iskusstvennogo-intellekta-chast-pervaya (дата обращения: 11.12.2020).

27. Учебник по нейронным сетям [Электронный ресурс]. URL: https://neuralnet.info/book/ (дата обращения: 11.12.2020).

28. AI Новости: Беспилотные автомобили, грузовые машины, автобусы 2018 [Электронный ресурс]. URL: https://ai-news.ru/bespilotnyj_avtomobili.html (дата обращения: 11.12.2020).

29. Face Recognition using Python and OpenCV [Electronical resource] // hanzaTech : [site]. URL: http://hanzratech.in/2015/02/03/face-recognition-using-opencv.html (date of access: 11.12.2020).

30. Face Recognition with OpenCV and Python [Electronical resource] // GitHub : [site]. URL: https://github.com/informramiz/opencv-face-recognition-python (date of access: 11.12.2020).
31. Haar Cascade Object Detection Face & Eye OpenCV Python Tutorial [Electronical resource] // PythonProgramming.net : [site]. URL: https://pythonprogramming.net/haar-cascade-face-eye-detection-python-opencv-tutorial/ (date of access: 11.12.2020).
32. ImageAI. State-of-the-art Recognition and Detection AI with few lines of code [Electronical resource]. URL: http://imageai.org/#about (date of access: 11.12.2020).
33. ImageAI: Custom Prediction Model Training [Electronical resource] // GitHub : [site]. URL: https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai/Prediction/CUSTOMTRAINING.md (date of access: 11.12.2020).
34. Keras Tutorial: Руководство для начинающих по глубокому обучению на Python [Электронный ресурс]. URL: https://python.ivan-shamaev.ru/keras-tutorial-beginner-guide-to-deep-learning-in-python/ (дата обращения: 11.12.2020).
35. LabelImg [Electronical resource] // GitHub : [site]. URL: https://github.com/tzutalin/labelImg (date of access: 11.12.2020).
36. Official English Documentation for ImageAI! [Electronical resource]. URL: https://imageai.readthedocs.io/en/latest/ (date of access: 11.12.2020).
37. OpenCV Tutorials, Resources, and Guides [Electronical resource] // PyImageSearch : [site]. URL: https://www.pyimagesearch.com/opencv-tutorials-resources-guides/ (date of access: 11.12.2020).
38. OpenPilot [Electronical resource] // GitHub : [site]. URL: https://github.com/commaai/openpilot (date of access: 11.12.2020).
39. Pedestrian Detection using OpenCV-Python [Electronical resource] // GeeksforGeeks : [site]. URL: https://www.geeksforgeeks.org/pedestrian-detection-using-opencv-python/?ref=rp (date of access: 11.12.2020).
40. perceprton-python [Electronical resource] // GitHub : [site]. URL: https://github.com/FyzHsn/perceptron-python/blob/master/perceptron_script.py (date of access: 11.12.2020).
41. Python. Урок 1. Установка [Электронный ресурс] // Devpractice : [сайт]. URL: https://devpractice.ru/python-lesson-1-install/ (дата обращения: 11.12.2020).
42. Python. Урок 16. Установка пакетов в Python [Электронный ресурс] // Devpractice : [сайт]. URL: https://devpractice.ru/python-lesson-16-install-packages/ (дата обращения: 11.12.2020).
43. Python: распознавание объектов в реальном времени [Электронный ресурс] // proglib : [сайт]. URL: https://proglib.io/p/real-time-object-detection/ (дата обращения: 11.12.2020).
44. Resources for setting up your coding environment [Electronical resource] // GitHub : [site]. URL: https://github.com/rasbt/python-machine-learning-book/tree/master/code (date of access: 11.12.2020).

Предметный указатель

О

П

Р

С

Т

У

Ф

Ц

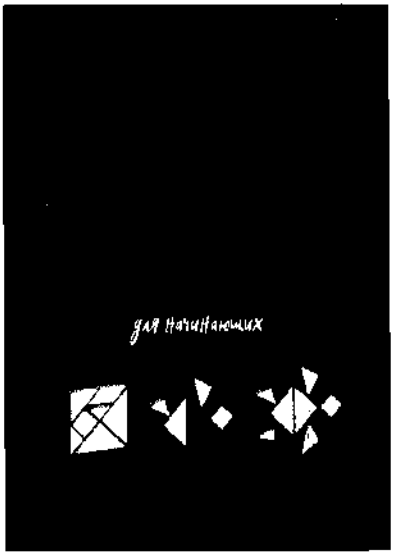
Ш

www.bhv.ru

Постолит А.

Python, Django и Bootstrap для начинающих

Отдел оптовых поставок:
e-mail: opt@bhv.ru



- Веб-технологии
- Инструментальные средства для разработки веб-приложений
- Знакомство с фреймворком Django
- Знакомство с фреймворком Bootstrap
- Интерактивная среда разработки PyCharm
- Обработка и маршрутизация запросов
- Шаблоны веб-страниц
- Формы и модели данных
- Веб-сайт и веб-интерфейс для пользователей
- Встроенная панель для администрирования сайта
- Пользовательские формы
- Публикация сайта в Интернете

Книга посвящена вопросам разработки веб-приложений с использованием языка Python, фреймворков Django, Bootstrap и интерактивной среды разработки PyCharm. Рассмотрены основные технологии и рабочие инструменты создания веб-приложений. Описаны фреймворки Django, Bootsrtap и структура создаваемых веб-приложений. На простых примерах показана обработка и маршрутизация запросов пользователей, формирование ответных веб-страниц. Рассмотрено создание шаблонов веб-страниц и форм для пользователей. Показано взаимодействие пользователей с различными типами баз данных через модели. Описана работа с базами данных через встроенные в Django классы без использования SQL-запросов. Приведен пошаговый пример создания сайта — от его проектирования до формирования программных модулей и развертывания сайта в Интернете с базами данных SQLite и MySQL. Электронный архив на сайте издательства содержит коды всех примеров.

Постолит Анатолий Владимирович, доктор технических наук, профессор, академик Российской академии транспорта, лауреат Всероссийского конкурса «Инженер года». Профессиональный программист, автор книг компьютерной тематики, в том числе «Основы искусственного интеллекта в примерах на Python», «Python, Django и PyCharm для начинающих» и более 100 научных публикаций. Преподавал в Московском государственном автомобильно-дорожном техническом университете (МАДИ). Занимался разработкой и внедрением информационных систем для транспортного комплекса Москвы и Московской области, для транспортного обслуживания зимних Олимпийских игр в г. Сочи, систем оплаты проезда и информирования пассажиров городского общественного транспорта. Специализируется на создании информационных систем на основе MS SQL Server, MS Visual Studio, Bluetooth-технологий, а также инновационных проектов с использованием Python, Django, Bootstrap, PyCharm и различных специализированных библиотек.

самоучитель

ОСНОВЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ПРИМЕРАХ НА

Python

Решение практических задач на простых и понятных примерах

Постолит Анатолий Владимирович — доктор технических наук, профессор, академик Российской академии транспорта, лауреат Всероссийского конкурса «Инженер года». Профессиональный программист, автор книг компьютерной тематики, в том числе «Python, Django и Bootstrap для начинающих» и более 120 научных публикаций. Профессор Московского государственного автомобильно-дорожного технического университета (МАДИ). Занимался разработкой и внедрением информационных систем для транспортного комплекса Москвы и Московской области, для транспортного обслуживания зимних Олимпийских игр в г. Сочи, систем оплаты проезда и информирования пассажиров городского общественного транспорта. Специализируется на создании информационных систем на основе MS SQL Server, MS Visual Studio, Bluetooth-технологий, веб-приложений, а также систем искусственного интеллекта, обработки изображений и компьютерного зрения.

Книга посвящена вопросам применения искусственного интеллекта для решения практических задач с использованием языка Python и различных дополнительных библиотек, которые в значительной степени облегчают процесс программирования. Она должна помочь как начинающим программистам, так и имеющим опыт работы с различными инструментальными средами, понять, как научить компьютеры решать нетрадиционные задачи, и с минимальными потерями времени перейти к практическому программированию в новой инструментальной среде, которая упрощает создание приложений, работающих в различных операционных системах и на разных аппаратных средствах. В книге читатель найдет все материалы, необходимые для практического программирования, пройдет все этапы проектирования нейронных сетей — от формулирования задачи до использования обученных моделей. Основное внимание уделено примерам разработки различных типов приложений с элементами искусственного интеллекта и машинного обучения. Во 2-м издании обновлены программные коды и версии библиотек, улучшены рисунки, учтены пожелания читателей и исправлены ошибки.

Примеры из книги можно скачать по ссылке https://zip.bhv.ru/9785977518185.zip, а также на странице книги на сайте bhv.ru.

ISBN 978-5-9775-1818-5

9 785977 518185

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

bhv®