ГЛАВА 3

Кошки против собак: перенос обучения с помощью Keras в 30 строках кода

Представьте, что вы решили научиться играть на мелодике — ручном духовом инструменте с клавиатурой, напоминающей фортепианную. Без музыкального образования и без владения другими инструментами потребуется несколько месяцев обучения. Но если у вас есть навыки игры на другом инструменте, например на фортепиано, то хватит и нескольких дней. В реальной жизни мы часто используем опыт, полученный при решении одной задачи, для решения другой. Чем более схожи две задачи, тем легче адаптировать имеющийся опыт.

Справедливо это и для глубокого обучения. Проект глубокого обучения можно относительно быстро реализовать, если взять предварительно обученную модель, которая использует знания, полученные во время обучения, и адаптирует их к поставленной задаче. Это называется *переносом обучения*.

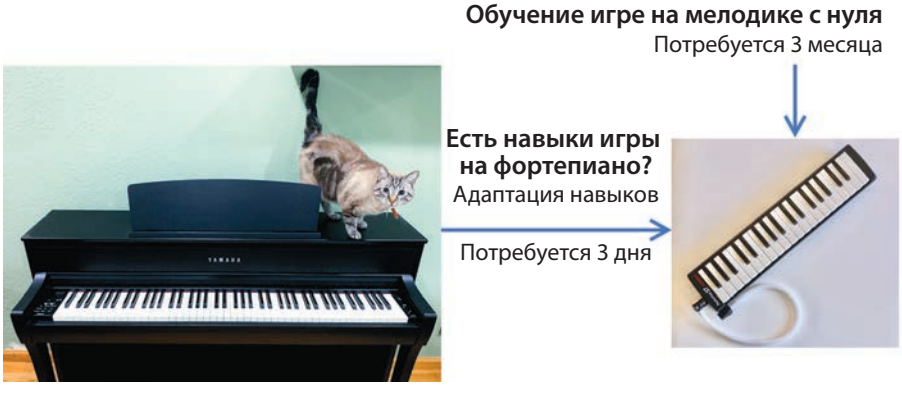

Рис. 3.1. Перенос обучения в реальной жизни

В этой главе мы используем прием переноса обучения для адаптации имеющихся моделей и за считанные минуты обучим свой классификатор с помощью Keras. К концу этой главы в нашем арсенале будет несколько инструментов, позволяющих создавать высокоточные классификаторы изображений для решения любых задач.

Адаптация предварительно обученных моделей к новым задачам

Но прежде сделаем шаг назад и рассмотрим основные причины, вызвавшие бум глубокого обучения:

- появление более крупных и качественных датасетов, таких как ImageNet;
- появление более мощной вычислительной техники, то есть более быстрых и дешевых графических процессоров;
- появление более удачных алгоритмов (архитектур моделей, оптимизаторов и процедур обучения);
- появление предварительно обученных моделей, на обучение которых ушли месяцы, но которые можно быстро использовать повторно.

От создателя

Сотни тысяч студентов изучали глубокое обучение с помощью fast.ai. Наша цель — как можно быстрее подготовить их к решению практических задач. Чему мы учим их в первую очередь? Переносу обучения.

Тысячи студентов делятся своими историями успеха на нашем форуме (*forum.fast.ai*), рассказывая, как, имея всего 30 изображений, они создали абсолютно точные классификаторы. Есть примеры студентов, побивших академические рекорды во многих областях и создавших коммерческие модели, используя этот простой метод.

Пять лет назад я создал Enlitic — первую компанию, специализирующуюся на глубоком обучении в медицине. В качестве первоначального доказательства идеи я решил разработать классификатор опухолей легких на основе снимков компьютерной томографии. Наверное, вы догадались, какой прием мы использовали. Перенос обучения. Наша библиотека с открытым исходным кодом fastai упрощает перенос обучения, и нужно написать всего три строки кода, за которыми скрыты самые передовые достижения.

Джереми Ховард (Jeremy Howard), сооснователь fast.ai,
в прошлом главный научный сотрудник в Kaggle

Последний пункт, вероятно, представляет одну из самых главных причин широкого распространения глубокого обучения. Если бы всякий раз обучение моделей занимало месяцы, то в этой области осталась бы лишь горстка исследователей с глубокими карманами. Благодаря недооцененному герою — переносу обучения, всего за несколько минут можно изменить существующую модель и приспособить ее для решения своей задачи.

В главе 2, например, мы видели, что готовая модель ResNet-50, обученная на ImageNet, определяет породы кошек и собак, а также распознает изображения из тысячи других категорий. Чтобы создать простой классификатор, распознающий обобщенные категории «кошка» и «собака» (а не конкретные породы), мы можем взять модель ResNet-50 и быстро переобучить ее для этой задачи. Для этого достаточно ввести в модель обучающий датасет с этими двумя категориями и обучить ее, что должно занять от нескольких минут до часов. Для сравнения: чтобы обучить с нуля модель, распознающую кошек и собак, может потребоваться от нескольких часов до дней.

Неглубокое погружение в сверточные нейронные сети

Мы использовали термин «модель» для обозначения той части ИИ, которая делает прогнозы. В глубоком обучении для задач компьютерного зрения роль модели обычно играет нейронная сеть особого типа, которая называется сверточной нейронной сетью (Convolutional Neural Network, CNN). Мы подробно изучим устройство CNN позже, а пока кратко опишем их с точки зрения переноса обучения.

В машинном обучении мы должны преобразовать данные в набор признаков, а затем добавить алгоритм для их оценки и классификации. Ту же задачу решают сети CNN. Они состоят из двух частей: сверточных и полносвязных слоев. Задача сверточных слоев в том, чтобы преобразовать большое количество пикселей изображения в гораздо более компактное представление, то есть в признаки. Полносвязные слои преобразуют эти признаки в вероятности. Полносвязный слой на самом деле является нейронной сетью со скрытыми слоями, как было показано в главе 1. То есть сверточные слои действуют как экстракторы признаков, а полносвязные — как классификатор. На рис. 3.2 показана общая структура сверточной сети.

Представьте, что нам нужно обнаружить человеческое лицо. Для классификации изображений и обнаружения лиц мы могли бы использовать сверточную сеть. Такая сеть будет состоять из нескольких слоев, связанных друг с другом. Эти слои представляют математические операции. Выход одного слоя является входом для следующего. Первый (или самый нижний) слой — это входной слой, в который вводится изображение. Последний (или самый верхний) слой — это выходной слой, который дает прогнозы.

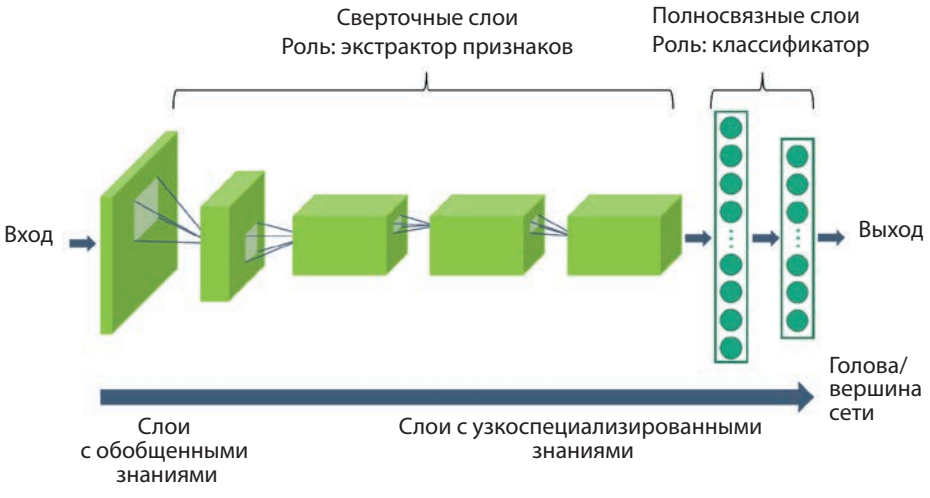

Рис. 3.2. Общая структура сверточной нейронной сети

Сеть работает примерно так: изображение вводится в сверточную сеть и проходит через последовательность слоев, каждый из которых выполняет математическую операцию и передает результат следующему слою. На выходе получается список классов изображенных объектов с их вероятностями. Например: мяч = 65 %, трава = 20 % и т. д. Если в выходных данных присутствует класс «лицо» с вероятностью 70 %, то мы можем заключить, что с вероятностью 70 % изображение содержит человеческое лицо.

Сверточную сеть можно также представить как последовательность фильтров (хотя это и слишком упрощенное представление). Как подразумевает слово «фильтр», каждый слой действует подобно ситу для информации, позволяя чему-то «проходить» сквозь него, только если это что-то распознается им. (Если вы слышали о фильтрах высоких и низких частот в электронике, то эта аналогия будет знакомой.) Мы говорим, что слой активируется при распознавании этой информации. Каждый слой активируется при встрече визуальных фрагментов, характерных для изображений кошек, собак, автомобилей и т. д. Если слой не распознает информацию (потому что не встречал ее во время обучения), то он выводит результат, близкий к нулю. Сверточные сети — это «вышибалы» в мире глубокого обучения!

Рассмотрим пример распознавания лиц. Слои нижнего уровня (рис. 3.3, а; находятся ближе к входному изображению) активируются наиболее простыми формами, например краями и кривыми. Поскольку эти слои активируются только простыми формами, их можно повторно использовать для других целей, отличных от распознавания лиц, например для распознавания автомобилей

(в конце концов, каждое изображение состоит из краев и кривых). Слои среднего уровня (рис. 3.3, b) активируются более сложными формами, такими как глаза, нос и губы. Эти слои более специализированные, чем слои нижнего уровня. Они могут быть не так полезны для распознавания автомобилей, зато могут пригодиться для распознавания животных. А слои на самом высоком уровне (рис. 3.3, c) активируются еще более сложными формами — например, большими фрагментами человеческого лица. Эти слои часто узко специализированы для конкретных задач и, следовательно, наименее пригодны для повторного использования при решении других задач классификации изображений.

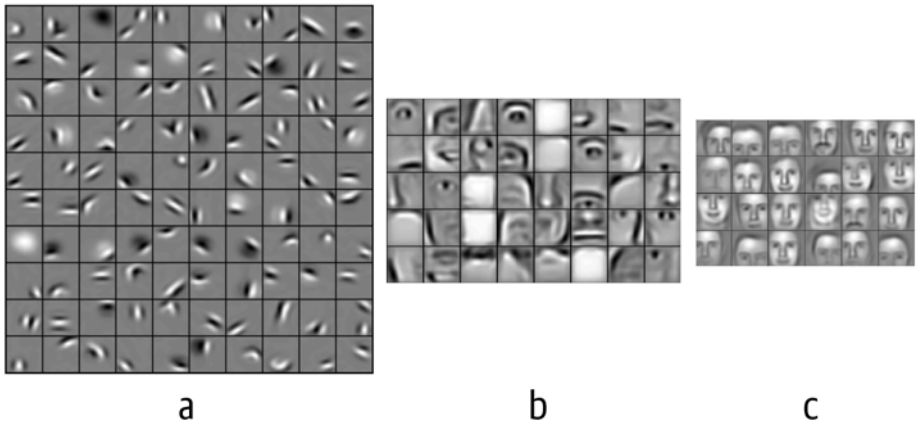

Рис. 3.3. (a) Активации низкого уровня, за ними следуют (b) активации среднего уровня и (c) активации высшего уровня (изображение взято из статьи «Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations», Lee (Ли) и др., ICML 2009)

По мере приближения к вершине сети сложность фрагментов, распознаваемых слоем, возрастает, а возможность его повторного использования уменьшается. Вы убедитесь в этом очень скоро, когда увидите, что изучают эти слои.

Перенос обучения

Чтобы перенести знания из одной модели в другую, нужно повторно использовать больше универсальных слоев (расположенных ближе к входу) и меньше специализированных для конкретных задач (расположенных ближе к выходу). Иначе говоря, нужно удалить несколько последних слоев (обычно удаляются все полносвязные слои), оставить только наиболее универсальные и добавить слои, предназначенные для нашей конкретной задачи классификации. После начала обучения универсальные слои (составляющие большую часть новой модели) останутся замороженными (то есть их нельзя будет изменить), а вновь добавленные слои для конкретных задач будут доступны для изменения. Благо-

даря такому подходу прием переноса обучения помогает быстро обучать новые модели. На рис. 3.4 показан такой процесс создания модели для решения задачи Y на основе модели, предварительно обученной для задачи X.

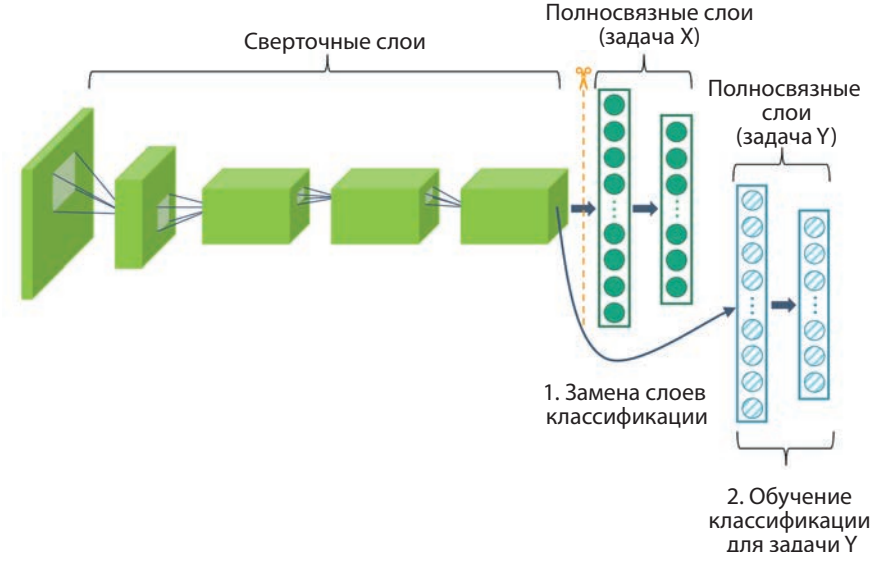

Рис. 3.4. Общая схема переноса обучения

Тонкая настройка

Даже простой перенос обучения открывает перед нами широкие возможности. Обычно, чтобы создать новую модель классификатора, достаточно добавить два-три полносвязных слоя после универсальных слоев. Но если потребуется более высокая точность, мы должны вовлечь в обучение больше слоев. Это означает разморозку некоторых слоев, которые при простом переносе обучения оставались бы в неприкосновенности. Этот прием называется *тонкой настройкой*. На рис. 3.5 показан пример, где некоторые сверточные, расположенные ближе к голове/вершине сети размораживаются и обучаются для выполнения поставленной задачи.

Очевидно, что по сравнению с простым переносом обучения прием тонкой настройки вовлекает в обучение с нашим набором данных больше слоев. Поскольку для решения задачи адаптируется большее количество слоев, чем при переносе обучения, можно добиться большей точности. Решение о том, сколько слоев подвергнуть тонкой настройке, зависит от количества имеющихся данных, а также от совместимости целевой задачи с исходным набором данных, на котором была обучена исходная модель.

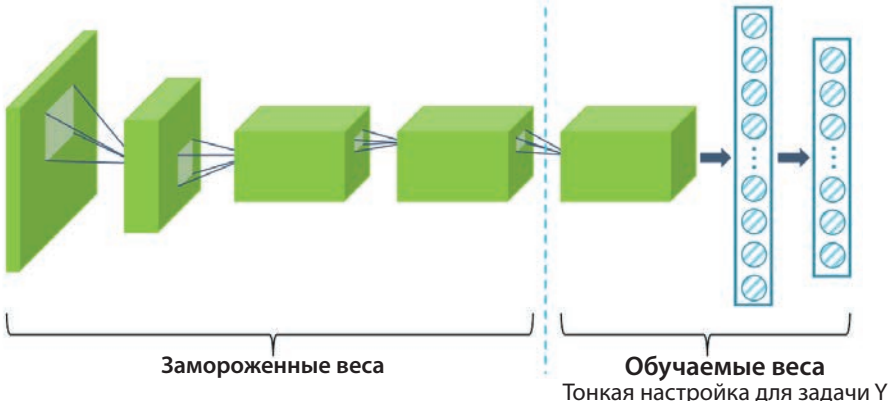

Рис. 3.5. Тонкая настройка сверточной нейронной сети

Мы часто слышим, как специалисты по обработке данных говорят: «Я провел тонкую настройку модели». Это означает, что они взяли предварительно обученную модель, удалили слои, предназначенные для конкретных задач, и добавили новые, заморозили нижние слои и обучили верхнюю часть сети на новом датасете.

В повседневной практике термины «перенос обучения» и «тонкая настройка» используются как взаимозаменяемые. При разговоре слова «перенос обучения» используются больше для описания общей идеи, тогда как «тонкая настройка» — для описания способа ее реализации.

Сколько слоев выбрать для тонкой настройки

Сколько слоев сверточной сети следует вовлекать в тонкую настройку? Ответ на этот вопрос определяется двумя факторами:

Объем имеющихся данных

При наличии всего пары сотен изображений с метками сложно обучить и протестировать модель с нуля (то есть начав с модели со случайными начальными весами), потому что для этого требуется намного больше данных. Опасность обучения на таком небольшом объеме данных состоит в том, что мощные сети могут просто запомнить их, что приведет к нежелательному эффекту переобучения (мы рассмотрим его далее в этой главе). Вместо этого лучше взять предварительно обученную сеть и настроить несколько последних ее слоев. Но если бы у нас был миллион изображений с метками, то можно было бы вовлечь в тонкую настройку все уровни сети и при не-

обходимости обучить ее с нуля. Итак, объем данных для конкретной задачи определяет, сможем ли мы выполнить тонкую настройку и в какой степени.

Сходство данных

Если данные для конкретной задачи подобны данным, использовавшимся при создании предварительно обученной сети, то в тонкую настройку можно вовлечь лишь несколько последних слоев. Но если перед нами стоит задача идентификации различных костей на рентгеновских снимках и мы решили взять за основу сеть, обученную на наборе ImageNet, то большое различие между обычными изображениями в ImageNet и рентгеновскими снимками потребует обучения почти всех слоев.

В табл. 3.1 приводится простая шпаргалка, которая поможет ответить на поставленный вопрос.

Таблица 3.1. Когда и в каком объеме выполнять тонкую настройку

	Большое сходство датасетов	Малое сходство датасетов
Большой объем обучающих данных	Тонкая настройка всех слоев	Обучение с нуля или тонкая настройка всех слоев
Малый объем обучающих данных	Тонкая настройка нескольких последних слоев	Не везет так не везет! Попробуйте обучить небольшую сеть на обогащенном датасете или постарайтесь собрать больше данных

Но хватит теории, перейдем теперь к практике.

Создание специального классификатора методом переноса обучения с использованием Keras

Как и обещали, приступаем к созданию нашего высокоточного классификатора в 30 строках кода или даже меньше. Для этого сделаем вот что:

1. Организуем данные. Загрузим изображения кошек и собак с метками, а затем разделим изображения на обучающую и проверочную выборки.
2. Построим пайплайн обработки данных. Определим пайплайн для чтения данных, включая предварительную обработку изображений (например, изменение размеров) и группировку изображений в пакеты.
3. Проведем аугментацию данных . При недостатке обучающих изображений можно попробовать обогатить набор, копируя изображения и внося в ко-

пии небольшие изменения (дополнения), такие как поворот, масштабирование и т. д., чтобы увеличить вариативность обучающих данных.

4. Определим модель. Возьмем предварительно обученную модель, удалим последние несколько слоев, связанных с конкретной задачей, и добавим новый слой классификатора. Заморозим веса исходных слоев (то есть сделаем их неизменяемыми). Выберем алгоритм оптимизатора и метрику для оценки (например, точность).
5. Обучим и проверим. Выполним несколько итераций, пока точность на этапе проверки не станет достаточно высокой. Сохраним модель, чтобы потом иметь возможность загрузить ее в любое приложение для получения прогнозов.

Смысл этих шагов очень скоро станет понятен. А пока подробно рассмотрим этот процесс.

Решение насущной проблемы компьютерного зрения

В начале 2014 года в Microsoft Research выяснили, как решить самую насущную на тот момент проблему: различение кошек и собак. (Откуда бы мы еще взяли идею для этой главы?) Имейте в виду, что в ту пору проблема компьютерного зрения выглядела намного сложнее. Для этого в Microsoft Research создали датасет Asirra (Animal Species Image Recognition for Restricting Access — распознавание изображений видов животных для ограничения доступа). Мотивом для создания Asirra послужила разработка достаточно сложной системы CAPTCHA. *Petfinder.com* предоставил Microsoft Research более трех миллионов изображений с метками, созданных приютами для животных со всей территории США. Максимальная точность первого решения этой задачи составляла около 80 %. При использовании глубокого обучения всего за несколько недель точность выросла до 98 %! Эта задача (теперь относительно простая) демонстрирует силу глубокого обучения.

Организация данных

Важно понимать разницу между обучающими, проверочными и контрольными данными. Давайте рассмотрим аналогию с учеником, готовящимся к стандартизованным экзаменам (например, SAT в США, Gaokao в Китае, JEE в Индии, CSAT в Корее и т. д.)[1]. Работа в классе и выполнение домашних заданий подобна процессу обучения. Решение контрольных и проверочных заданий в школе эквивалентно процессу проверки: ученик может часто решать такие

[1] ЕГЭ в России. — *Примеч. пер.*

задания, оценивать уровень своих знаний и корректировать свой учебный план. Наконец, он приходит на финальный стандартизированный экзамен, где у него есть только один шанс. Финальный экзамен эквивалентен процессу контроля — учащийся не имеет возможности улучшить свои знания (если не учитывать возможность пересдачи экзамена). Это его единственный шанс показать свой уровень знаний.

Мы преследуем ту же цель — генерировать лучшие прогнозы в реальном мире. Для этого мы разделим наши данные на три части: обучающую, проверочную и контрольную. Обычно в обучающую выборку включается 80 % данных, а в проверочную и контрольную — по 10 %. Обратите внимание: чтобы гарантировать наименьшее количество систематических ошибок, которые могут возникнуть непредумышленно, мы случайным образом разделим наши данные на эти три части. Окончательная точность модели будет определяться точностью прогнозирования на контрольной выборке, так же как оценка учащегося определяется только на основе результатов сдачи стандартизированного экзамена.

Модель учится на обучающих данных и использует проверочный набор для оценки качества своих прогнозов. Практики МО воспринимают этот процесс как систему с обратной связью, способствующую постоянному улучшению их моделей, подобно тому как студенты улучшают свою подготовку с помощью контрольных и самостоятельных работ. В нашем распоряжении есть несколько регуляторов для настройки качества прогнозов, например количество слоев в сети.

Во многих конкурсах МО (включая *Kaggle.com*) участники отдельно получают контрольный набор, не связанный с данными, которые они могут использовать для обучения модели. Это гарантирует равные условия среди конкурентов в оценке точности их моделей. Участники должны разделить доступные данные на обучающий и проверочный наборы. Точно так же в ходе наших экспериментов в этой книге мы будем делить данные на эти два набора, не забывая о важности контрольного набора для оценки реального положения дел.

Зачем вообще нужен контрольный набор? Нередко сбор исходных данных сопряжен с большими сложностями, так почему бы не использовать все доступные данные для обучения, а затем сообщить о точности, полученной на них? Конечно, в процессе обучения модель постепенно будет увеличивать точность прогнозов на обучающем датасете (так называемая точность на этапе обучения). Но глубокие нейронные сети могут быть чрезвычайно мощными и потенциально способны запомнить обучающие данные, что иногда приводит даже к 100 %-ной точности на обучающих данных. Но реальное качество прогнозирования будет оставаться довольно низким. Это как если бы студент знал, какие вопросы будут задаваться на экзамене еще до его начала. Вот почему проверочный набор, не используемый для обучения модели, дает реалистичную оценку качества ее работы. Даже при том что мы можем выделить лишь 10–15 % данных для

проверочного набора, он будет иметь большое значение для определения того, насколько хорошо обучена наша модель.

Для качественного обучения мы должны сохранить датасет в соответствующей структуре папок. Разделим изображения на два набора: обучающий и проверочный. В случае с файлами изображений Keras автоматически будет определять имена *классов* (категорий) по именам папок, в которых эти изображения находятся. На рис. 3.6 показана идеальная структура.

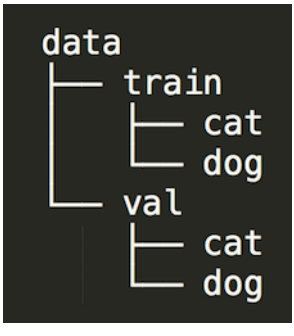

Рис. 3.6. Пример структуры папок для обучающих и проверочных данных с разными классами

Такая последовательность команд загрузит данные и разместит их согласно этой структуре папок:

```
$ wget https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/download/
train.zip
$ unzip train.zip
$ mv train data
$ cd data
$ mkdir train val
$ mkdir train/cat train/dog
$ mkdir val/cat val/dog
```

Теперь у нас в папке data есть 25 000 файлов с именами, начинающимися с префиксов «cat» (кошка) и «dog» (собака). Переместим файлы в соответствующие каталоги. Чтобы сократить время для первого эксперимента, выберем 250 случайных файлов из каждого класса и поместим их в папки для обучающего (train) и проверочного (val) датасетов. Можно увеличить или уменьшить это число в любое время, чтобы поэкспериментировать с компромиссом между точностью и скоростью:

```
$ ls | grep cat | sort -R | head -250 | xargs -I {} mv {} train/cat/
$ ls | grep dog | sort -R | head -250 | xargs -I {} mv {} train/dog/
$ ls | grep cat | sort -R | head -250 | xargs -I {} mv {} val/cat/
$ ls | grep dog | sort -R | head -250 | xargs -I {} mv {} val/dog/
```

Создание пайплайна обработки данных

Напишем программу на Python и добавим инструкции импорта нужных библиотек:

```
import tensorflow as tf
from tf.keras.preprocessing.image import ImageDataGenerator
from tf.keras.models import Model
from tf.keras.layers import Input, Flatten, Dense, Dropout,
GlobalAveragePooling2D
from tf.keras.applications.mobilenet import MobileNet, preprocess_input
import math
```

Затем добавим строки с настройками, которые можно изменить, если изменится размер датасета:

```
TRAIN_DATA_DIR = 'data/train_data/'
VALIDATION_DATA_DIR = 'data/val_data/'
TRAIN_SAMPLES = 500
VALIDATION_SAMPLES = 500
NUM_CLASSES = 2
IMG_WIDTH, IMG_HEIGHT = 224, 224
BATCH_SIZE = 64
```

Количество классов

При наличии двух классов задачу можно рассматривать как:

- задачу бинарной классификации;
- задачу многоклассовой классификации.

Бинарная классификация

Важно отметить, что в случае бинарной классификации задача «кошки против собак» фактически сводится к задаче «кошки против не кошек». Собака классифицируется как «не кошка», так же как письменный стол или мяч. Для каждого конкретного изображения модель даст единственное значение — вероятность соответствия классу «кошка», — следовательно, вероятность «не кошка» равна $1 - P(\textit{кошка})$. Если вероятность больше 0,5, мы будем считать, что изображение принадлежит к классу «кошка»; в противном случае «не кошка». Для простоты предположим, что контрольный набор гарантированно содержит только изображения кошек или собак. Поскольку «кошка против не кошек» — это задача бинарной классификации, установим количество классов равным 1, и этим единственным классом будет класс «кошка». Все, что не может быть классифицировано как «кошка», будет классифицироваться как «не кошка».

Keras обрабатывает входные данные в алфавитном порядке имен папок, а поскольку «cat» (кошка) в алфавитном порядке находится выше «dog» (собака), то нашим первым классом будет класс «кошка». В задаче мультиклассовой классификации мы можем применить ту же идею и определить индекс каждого класса на основе порядка сортировки папок. Обратите внимание, что нумерация индексов классов начинается с 0, то есть первый класс имеет индекс 0.

Мультиклассовая классификация

В гипотетическом мире, где бывают только кошки и собаки, «не кошка» всегда будет собакой. То есть метку «не кошка» можно просто заменить меткой «собака». Но в реальном мире многообразие объектов шире. Как объяснялось выше, мяч или диван в случае бинарной классификации тоже будут классифицироваться как «собака», что неверно. Поэтому для реального сценария намного полезнее рассматривать нашу задачу как задачу многоклассовой классификации. В задаче многоклассовой классификации мы прогнозируем вероятность для каждого класса, и класс с наибольшей вероятностью считается победителем. В случае задачи «кошки против собак» мы устанавливаем количество классов (переменная `NUM_CLASSES`) равным двум. Чтобы код был переиспользуемый, будем рассматривать эту задачу как задачу многоклассовой классификации.

Размер пакета

В обобщенном случае процесс обучения включает такие этапы:

1. Получить прогнозы на наборе изображений (*прямой проход*).
2. Выявить неверные прогнозы и передать в сеть разницу между прогнозом и истинным значением (*обратное распространение ошибки*).
3. Повторять раз за разом, пока прогнозы не станут достаточно точными.

Вполне вероятно, что первая итерация даст точность, близкую к 0 %. Но повторение этого процесса несколько раз может дать очень точную модель (>90 %).

Размер пакета (переменная `BATCH_SIZE`) определяет, сколько изображений будет получать модель одновременно. Важно, чтобы в каждом пакете было как можно больше разных изображений, чтобы предотвратить большие колебания в оценке точности между итерациями. Для этого пакет должен быть достаточно большого размера, но не должен быть слишком большим, иначе изображения не поместятся в памяти GPU, что приведет к сбою «нехватка памяти». Обычно размеры пакетов выбираются как степени двойки. В большинстве задач лучше начать с 64, а затем можем поиграть с ним, увеличивая или уменьшая.

Аугментация данных

Говоря о глубоком обучении, часто представляют наборы данных, насчитывающие миллионы изображений. Те 500 изображений, которые отобрали мы, бывают недостаточными для реального обучения. Глубокие нейронные сети очень мощные, даже слишком мощные для небольших объемов данных. Опасность ограниченности набора обучающих изображений заключается в том, что нейронная сеть может просто запомнить все обучающие данные и показать высочайшее качество прогнозирования на обучающем наборе, но крайне низкую точность на проверочном наборе. В таких случаях говорят, что модель переобучена и не обобщает полученные знания на изображения, которые прежде не видела. А нам этого точно не нужно.

Часто при попытках обучить нейронную сеть на небольшом датасете получается модель, которая показывает хорошие результаты на самих обучающих данных, но плохо справляется с прогнозированием на данных, которые она не видела раньше. Такую модель называют *переобученной*, а саму проблему — *проблемой переобучения*.

На рис. 3.7 это явление показано на примере распределения точек, близкого к синусоидальному (с небольшим шумом). Точки представляют обучающие данные, а крестики — контрольные данные, которые сеть не видела во время обучения. С одной стороны, простая (недостаточно обученная) модель, например линейная, не в состоянии достаточно точно представить распределение, что вызывает высокий уровень ошибок как на обучающих, так и на контрольных данных. С другой стороны, мощная (переобученная) модель (например, глубокая нейронная сеть) может запомнить обучающие данные и показать действительно

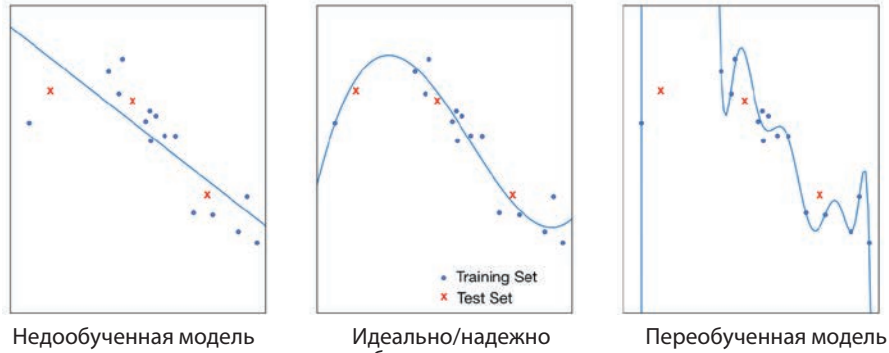

Рис. 3.7. Недообучение, переобучение и идеальное обучение на точках с распределением, близким к синусоидальному

низкий уровень ошибок на обучающих данных, но на контрольных данных уровень ошибок по-прежнему будет высоким. Нужна золотая середина, когда ошибки на этапах обучения и контроля будут умеренно низкими, что в идеале гарантирует способность модели работать в реальном мире так же хорошо, как и во время обучения.

С большой силой приходит большая ответственность. Мы несем ответственность за то, чтобы наша мощная глубокая нейронная сеть не переобучилась на данных. Переобучение — обычное дело, когда есть не очень много обучающих данных. Вероятность переобучения можно снизить:

- попробовав получить больше данных;
- проведя аугментацию существующих данных;
- уменьшив количество слоев, подвергающихся тонкой настройке.

Ситуации, когда данных оказывается недостаточно, нередки. Например, так бывает при работе над узкоспециализированной задачей, для которой сложно получить подходящие данные. Но есть несколько способов искусственно расширить датасет:

Повернуть

В этом примере мы можем повернуть 500 случайно отобранных изображений на 20 градусов в любом направлении и получить до 20 000 в чем-то уникальных изображений.

Добавить случайный сдвиг

Изображения можно немного сместить влево или вправо.

Масштабировать

Можно немного увеличить и уменьшить масштаб изображения.

Комбинируя поворот, сдвиг и масштабирование, программа может сгенерировать почти бесконечное количество уникальных изображений. Этот важный шаг называется *аугментацией (обогащением) данных*. Аугментация данных может пригодиться не только для расширения датасета, но также для обучения более надежных моделей в реальных задачах. Например, не на всех изображениях кошка находится в центре или под идеальным углом в 0 градусов. Keras предоставляет функцию `ImageDataGenerator`, которая проводит аугментацию данных в ходе их загрузки из каталога.

На рис. 3.8 показаны примеры изображений, сгенерированных библиотекой imgaug (*https://github.com/aleju/imgaug*) на основе изначального изображения. (Обратите внимание: использовать библиотеку imgaug для обучения нашей модели мы не будем.)

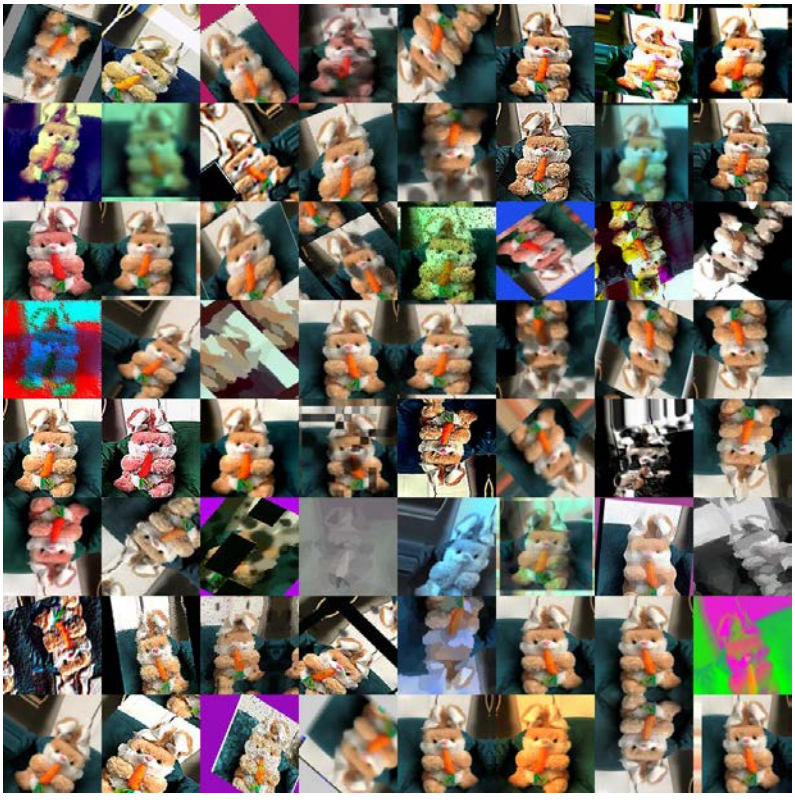

Рис. 3.8. Аугментация датасета на примере единственного изображения

Пиксели цветных изображений обычно имеют три канала: красный, зеленый и синий. Интенсивность каждого цвета определяется значением в диапазоне от 0 до 255. Для нормализации (то есть для приведения значений в диапазон от 0 до 1) мы используем функцию `preprocess_input` (которая, помимо всего прочего, делит значение каждого канала в пикселе на 255):

```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   zoom_range=0.2)
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

Иногда знание меток обучающих изображений помогает определить подходящие способы аугментации. Например, при обучении распознаванию цифр можно для обогащения применить поворот на 180 градусов к изображениям цифры «8», но такой поворот неприменим к цифрам «6» и «9».

Проверочный набор, в отличие от обучающего, мы не будем расширять с помощью аугментации. Причина в том, что при динамической аугментации проверочный набор будет меняться в каждой итерации, в результате чего оценка точности будет непоследовательной, из-за чего возникнут сложности при сравнении результатов разных итераций.

А теперь загрузим данные из каталогов. Обучение на одном изображении за раз может быть довольно неэффективным, поэтому объединим их в группы. Чтобы сделать процесс обучения более случайным, мы будем постоянно перемешивать изображения перед формированием пакетов. А чтобы обеспечить воспроизводимость результатов в разных запусках одной и той же программы, инициализируем генератор случайных чисел начальным значением (seed value):

```
train_generator = train_datagen.flow_from_directory(
                        TRAIN_DATA_DIR,
                        target_size=(IMG_WIDTH, IMG_HEIGHT),
                        batch_size=BATCH_SIZE,
                        shuffle=True,
                        seed=12345,
                        class_mode='categorical')
validation_generator = val_datagen.flow_from_directory(
                        VALIDATION_DATA_DIR,
                        target_size=(IMG_WIDTH, IMG_HEIGHT),
                        batch_size=BATCH_SIZE,
                        shuffle=False,
                        class_mode='categorical')
```

Определение модели

Теперь, когда данные обработаны, мы подошли к самому важному этапу процесса обучения: определению модели. В следующем коде мы повторно используем сверточную сеть, ранее обученную на датасете ImageNet (в данном случае MobileNet). Мы отбросим последние несколько слоев, которые называются полносвязными (это слои классификатора, характерные для ImageNet), и заменим их собственным классификатором, предназначенным для решения поставленной задачи.

Для переноса обучения «заморозим» веса исходной модели, то есть объявим эти слои неизменяемыми, чтобы в ходе обучения изменялись только новые слои добавленного нами классификатора. Здесь для ускорения решения мы используем сеть MobileNet, но вообще этот подход можно использовать для любой нейронной сети. В строках кода ниже есть термины Dense, Dropout и т. д. В этой главе мы не будем обсуждать их, а все пояснения вы найдете в приложении.

```
def model_maker():
    base_model = MobileNet(include_top=False, input_shape =
                    (IMG_WIDTH,IMG_HEIGHT,3))
```

```
for layer in base_model.layers[:]:
    layer.trainable = False # Freeze the layers
input = Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3))
custom_model = base_model(input)
custom_model = GlobalAveragePooling2D()(custom_model)
custom_model = Dense(64, activation='relu')(custom_model)
custom_model = Dropout(0.5)(custom_model)
predictions = Dense(NUM_CLASSES, activation='softmax')(custom_model)
return Model(inputs=input, outputs=predictions)
```

Обучение модели

Настройка параметров обучения

Теперь, когда данные и модель готовы, осталось только обучить модель. Процесс обучения еще называют *подгонкой модели к данным*. Для обучения модели нужно настроить несколько параметров обучения.

Функция потерь

Функция потерь определяет величину штрафа за ошибочные прогнозы в процессе обучения. Наша цель — минимизировать значение этой функции. Например, в задаче прогнозирования цен на жилье функцией потерь может быть среднеквадратичная ошибка (Root-Mean-Square Error, RMSE).

Оптимизатор

Это алгоритм, помогающий минимизировать функцию потерь. Мы используем один из самых быстрых алгоритмов: Adam.

Скорость обучения

Обучение идет постепенно. Скорость обучения сообщает оптимизатору, насколько большим должен быть следующий шаг в процессе обучения в направлении минимизации потерь. Если сделать шаг слишком большим, обучение будет производиться с большим размахом и постоянно пролетать мимо цели. Если сделать шаг слишком маленьким, может потребоваться очень много времени, прежде чем будет достигнуто целевое значение потерь. Важно выбрать оптимальную скорость обучения, чтобы гарантировать достижение цели обучения в разумные сроки. В нашем примере мы установили скорость обучения равную 0,001.

Метрика

Определяет, как будет оцениваться качество прогнозов обучаемой модели. Точность — хорошая и понятная метрика, особенно когда классы сбалансированы, то есть когда для всех классов имеются примерно равные объемы

данных. Обратите внимание, что метрика не связана с функцией потерь и используется в основном для отчетности, а не для обратной связи в модели.

В этом фрагменте кода мы создаем модель с помощью функции `model_maker`, которую написали выше, и задаем параметры, описанные здесь, чтобы настроить эту модель для нашей задачи классификации кошек и собак:

```
model = model_maker()
model.compile(loss='categorical_crossentropy',
              optimizer= keras.optimizers.Adam(lr=0.001),
              metrics=['acc'])
num_steps = math.ceil(float(TRAIN_SAMPLES)/BATCH_SIZE)
model.fit_generator(train_generator,
                    steps_per_epoch = num_steps,
                    epochs=10,
                    validation_data = validation_generator,
                    validation_steps = num_steps)
```

Возможно, вы обратили внимание на термин epoch (эпоха) в предыдущем коде. Одна эпоха представляет полный шаг обучения, в ходе которого сеть просмотрела весь датасет. Одна эпоха может состоять из нескольких мини-пакетов.

Запуск обучения

Запустите программу, и да начнется волшебство! Если у вас нет GPU, заварите себе чашечку кофе — ожидание может занять от 5 до 10 минут. Хотя зачем ждать, если есть возможность запустить блокнот для этой главы (на GitHub) в Colab и бесплатно воспользоваться GPU в облаке?

По завершении обратите внимание на четыре статистических показателя: `loss` и `acc` на обоих датасетах, обучающем и проверочном. Мы болеем за `val_acc`:

```
> Epoch 1/100 7/7 [====] - 5s - loss: 0.6888 - acc: 0.6756 - val_loss: 0.2786 -
val_acc: 0.9018
> Epoch 2/100 7/7 [====] - 5s - loss: 0.2915 - acc: 0.9019 - val_loss: 0.2022 -
val_acc: 0.9220
> Epoch 3/100 7/7 [====] - 4s - loss: 0.1851 - acc: 0.9158 - val_loss: 0.1356 -
val_acc: 0.9427
> Epoch 4/100 7/7 [====] - 4s - loss: 0.1509 - acc: 0.9341 - val_loss: 0.1451 -
val_acc: 0.9404
> Epoch 5/100 7/7 [====] - 4s - loss: 0.1455 - acc: 0.9464 - val_loss: 0.1637 -
val_acc: 0.9381
> Epoch 6/100 7/7 [====] - 4s - loss: 0.1366 - acc: 0.9431 - val_loss: 0.2319 -
val_acc: 0.9151
> Epoch 7/100 7/7 [====] - 4s - loss: 0.0983 - acc: 0.9606 - val_loss: 0.1420 -
val_acc: 0.9495
> Epoch 8/100 7/7 [====] - 4s - loss: 0.0841 - acc: 0.9731 - val_loss: 0.1423 -
val_acc: 0.9518
```

```
> Epoch 9/100 7/7 [====] - 4s - loss: 0.0714 - acc: 0.9839 - val_loss: 0.1564 -
val_acc: 0.9509
> Epoch 10/100 7/7 [====] - 5s - loss: 0.0848 - acc: 0.9677 - val_loss: 0.0882 -
val_acc: 0.9702
```

На самую первую эпоху обучения было потрачено пять секунд, точность на проверочном наборе составила 90 %. Эти результаты были достигнуты всего с 500 обучающими изображениями. Неплохо! К 10-му шагу *точность на проверочном наборе* приблизилась к 97 %. Вот она, истинная мощь переноса обучения!

Остановимся и оценим происходящее. Имея всего 500 изображений и написав немного кода, мы смогли достичь высокого уровня точности за несколько секунд. Если бы модели, предварительно обученной на наборе ImageNet, не было, то для получения точной модели потребовалось бы от пары часов до нескольких дней и огромный объем данных.

Это весь код, необходимый для обучения современного классификатора в любой задаче. Поместите данные в папки с именами классов и измените соответствующие значения в переменных конфигурации. Если задача имеет более двух классов, то в качестве функции потерь используйте `category_crossentropy` и замените функцию активации в последнем слое на `softmax`, как показано в табл. 3.2.

Таблица 3.2. Выбор функций потерь и активации в зависимости от стоящей задачи

Тип классификации	Режим классификации (`class_mode`)	Функция потерь (`loss`)	Функция активации в последнем слое (`activation`)
1 или 2 класс	`binary`	`binary_crossentropy`	`sigmoid`
Многоклассовая, единственная метка	`categorical`	`categorical_crossentropy`	`softmax`
Многоклассовая, несколько меток	`categorical`	`binary_crossentropy`	`sigmoid`

Пока не забыли, сохраним обученную модель, чтобы воспользоваться ею потом:

```
model.save('model.h5')
```

Тестируем модель

Теперь, имея обученную модель, мы сможем использовать ее в нашем приложении. Для этого достаточно загрузить эту модель и классифицировать изображение. Функция `load_model`, как можно догадаться по имени, загружает модель:

```
from keras.models import load_model
model = load_model('model.h5')
```

Теперь загрузим начальный образец изображения и посмотрим, как модель с ним справится:

```
img_path = '../../sample_images/dog.jpg'
img = image.load_img(img_path, target_size=(224,224))
img_array = image.img_to_array(img)
expanded_img_array = np.expand_dims(img_array, axis=0)
preprocessed_img = preprocess_input(expanded_img_array) # Preprocess the image
prediction = model.predict(preprocessed_img)
print(prediction)
print(validation_generator.class_indices)
[[0.9967706]]
{'dog': 1, 'cat': 0}
```

Как видите, вероятность составила 0,996. Это вероятность того, что данное изображение принадлежит к классу «1», то есть к классу «dog» (собака). Поскольку вероятность больше 0,5, изображение классифицируется как изображение собаки.

Вот, собственно, и все, что нужно для обучения собственных классификаторов. В этой книге мы будем переиспользовать этот код (с минимальными изменениями) для обучения. Вы тоже можете использовать этот код в своих проектах. Поиграйте с количеством эпох и изображений и посмотрите, как это повлияет на точность. Поэкспериментируйте с любыми другими данными, которые сможете найти в интернете. Нет ничего проще!

Анализ результатов

Проанализируем, насколько успешно обученная модель справляется с классификацией изображений в проверочном датасете. Помимо простых показателей точности, обзор ошибочно классифицированных изображений поможет получить интуитивное представление о том, действительно ли эти изображения трудно классифицировать или модель оказалась недостаточно сложной.

По каждой категории (кошка, собака) нужно ответить на три вопроса:

- Какие изображения уверенно (с большим значением вероятности) классифицируются моделью как принадлежащие к классу кошка/собака?
- Какие изображения неуверенно (с небольшим значением вероятности) классифицируются моделью как принадлежащие к классу кошка/собака?
- Какие изображения неправильно классифицированы, несмотря на высокую уверенность?