

O'REILLY®

# Искусственный интеллект и компьютерное зрение

Реальные проекты  
на Python, Keras  
и TensorFlow



Анирад Коул  
Сиддха Ганджу, Мехер Казам

---

# Practical Deep Learning for Cloud, Mobile, and Edge

*Real-World AI and Computer-Vision Projects  
Using Python, Keras, and TensorFlow*

*Anirudh Koul, Siddha Ganju, and Meher Kasam*

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

# **Искусственный интеллект и компьютерное зрение**

Реальные проекты на Python, Keras и TensorFlow

Анирад Коул, Сиддха Ганджу, Мехер Казам



Санкт-Петербург · Москва · Минск

2023

*Анирад Коул, Сиддха Ганджу, Мехер Казам*

## **Искусственный интеллект и компьютерное зрение. Реальные проекты на Python, Keras и TensorFlow**

*Перевел с английского А. Киселев*

ББК 32.818+32.973.235

УДК 004.8+004.93

**Коул Анирад, Ганджу Сиддха, Казам Мехер**

K73 Искусственный интеллект и компьютерное зрение. Реальные проекты на Python, Keras и TensorFlow. — СПб.: Питер, 2023. — 624 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-4461-1840-3

Кем бы вы ни были — инженером-программистом, стремящимся войти в мир глубокого обучения, опытным специалистом по обработке данных или любителем, мечтающим создать «вирусное» приложение с использованием ИИ, — наверняка задавались вопросом: с чего начать? Практические примеры из этой книги научат вас создавать приложения глубокого обучения для облачных, мобильных и краевых (edge) систем. Если вы хотите создать что-то необычное, полезное, масштабируемое или просто классное — эта книга для вас.

Многолетний опыт исследований в области глубокого обучения и разработки приложений позволяют авторам научить каждого воплощать идеи в нечто невероятное и необходимое людям в реальном мире.

**16+** (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1492034865 англ.

Authorized Russian translation of the English edition of Practical Deep Learning for Cloud, Mobile and Edge ISBN 9781492034865 © 2020 Anirudh Koul, Siddha Ganju, Meher Kasam

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-5-4461-1840-3

© Перевод на русский язык ООО «Прогресс книга», 2022

© Издание на русском языке, оформление ООО «Прогресс книга», 2022

© Серия «Бестселлеры O'Reilly», 2022

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 09.2022. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 29.07.22. Формат 70×100/16. Бумага офсетная. Усл. п. л. 50,310. Тираж 700. Заказ 0000.

---

# Краткое содержание

Вступление .....	20
<b>Глава 1.</b> Обзор ландшафта искусственного интеллекта.....	32
<b>Глава 2.</b> Что на картинке: классификация изображений с помощью Keras.....	69
<b>Глава 3.</b> Кошки против собак: перенос обучения с помощью Keras в 30 строках кода .....	83
<b>Глава 4.</b> Создание механизма обратного поиска изображений. Эмбеддинги .....	112
<b>Глава 5.</b> От новичка до мастера прогнозирования: увеличение точности сверточной нейронной сети .....	151
<b>Глава 6.</b> Увеличение скорости и эффективности TensorFlow: удобный чек-лист.....	183
<b>Глава 7.</b> Практические инструменты, советы и приемы .....	225
<b>Глава 8.</b> Облачные API для компьютерного зрения: установка и запуск за 15 минут.....	237
<b>Глава 9.</b> Масштабируемый инференс в облаке с помощью TensorFlow Serving и KubeFlow .....	274
<b>Глава 10.</b> ИИ в браузере с TensorFlow.js и ml5.js.....	304
<b>Глава 11.</b> Классификация объектов в реальном времени в iOS с Core ML .....	338
<b>Глава 12.</b> Not Hotdog на iOS с Core ML и Create ML.....	379

<b>Глава 13.</b> Шазам для еды: разработка приложений для Android с помощью TensorFlow Lite и ML Kit.....	400
<b>Глава 14.</b> Создание приложения Purrfect Cat Locator с помощью TensorFlow Object Detection API.....	445
<b>Глава 15.</b> Как стать творцом: ИИ в краевых устройствах .....	487
<b>Глава 16.</b> Моделирование беспилотного автомобиля методом сквозного глубокого обучения с использованием Keras.....	519
<b>Глава 17.</b> Создание беспилотного автомобиля менее чем за час: обучение с подкреплением с помощью AWS DeepRacer .....	553
<b>Приложение.</b> Краткое введение в сверточные нейронные сети .....	596
Об авторах .....	605
Иллюстрация на обложке.....	608

---

# Оглавление

<b>Вступление.....</b>	<b>20</b>
Разработчикам.....	21
Специалистам по данным.....	22
Студентам.....	22
Преподавателям .....	23
Энтузиастам робототехники .....	23
Обзор глав .....	23
Условные обозначения .....	27
Использование исходного кода примеров.....	27
Благодарности .....	28
Коллективные благодарности.....	28
Личные благодарности .....	30
<b>Глава 1. Обзор ландшафта искусственного интеллекта.....</b>	<b>32</b>
Извинения.....	33
Настоящее вступление .....	34
Что такое ИИ? .....	34
Мотивирующие примеры .....	35
Краткая история ИИ .....	37
Захватывающее начало.....	38
Холодные и мрачные дни .....	39
Проблеск надежды .....	40
Как глубокое обучение вошло в моду.....	43
Рецепт идеального решения задачи глубокого обучения .....	46
Датасеты.....	47
Архитектура модели .....	50
Фреймворки.....	52
Ответственный ИИ.....	59
Предвзятость .....	60
Прозрачность и объяснимость .....	63
Воспроизводимость .....	64
Устойчивость .....	64
Конфиденциальность .....	65

Итоги .....	66
Часто задаваемые вопросы .....	66
<b>Глава 2. Что на картинке: классификация изображений с помощью Keras .... 69</b>	
Введение в Keras .....	70
Классификация изображений .....	71
Исследование модели.....	75
Датасет ImageNet.....	76
Зоопарк моделей.....	78
Карты активации классов .....	79
Итоги .....	82
<b>Глава 3. Кошки против собак: перенос обучения с помощью Keras в 30 строках кода .....</b>	<b>83</b>
Адаптация предварительно обученных моделей к новым задачам .....	84
Неглубокое погружение в сверточные нейронные сети .....	85
Перенос обучения.....	87
Тонкая настройка.....	88
Сколько слоев выбрать для тонкой настройки.....	89
Создание специального классификатора методом переноса обучения с использованием Keras .....	90
Организация данных .....	91
Создание пайплайна обработки данных.....	94
Количество классов.....	94
Размер пакета.....	95
Аугментация данных .....	96
Определение модели .....	99
Обучение модели .....	100
Настройка параметров обучения.....	100
Запуск обучения.....	101
Тестируем модель .....	102
Анализ результатов .....	103
Для дальнейшего чтения .....	110
Итоги .....	111
<b>Глава 4. Создание механизма обратного поиска изображений. Эмбеддинги .....</b>	<b>112</b>
Сходство изображений.....	113
Извлечение признаков .....	116
Поиск сходств.....	119

---

Визуализация кластеров изображений с помощью t-SNE.....	123
Увеличение скорости поиска сходств .....	126
Длина векторов признаков.....	126
Уменьшение длины векторов признаков с помощью метода главных компонент.....	128
Масштабирование поиска сходства с помощью метода приближительных ближайших соседей .....	132
Бенчмарк метода приближительных ближайших соседей .....	133
Какую библиотеку выбрать?.....	134
Создание синтетического датасета .....	135
Полный перебор.....	135
Annoy.....	136
NGT .....	137
Faiss .....	137
Повышение точности с помощью тонкой настройки.....	138
Тонкая настройка без полносвязных слоев .....	141
Сиамские сети для распознавания лица однократным (one-shot) обучением.....	142
Примеры из практики .....	144
Flickr .....	144
Pinterest.....	145
Двойники знаменитостей .....	146
Spotify.....	146
Описание изображений .....	148
Итоги .....	150
<b>Глава 5. От новичка до мастера прогнозирования: увеличение точности сверточной нейронной сети .....</b>	<b>151</b>
Что понадобится для работы .....	152
TensorFlow Datasets.....	153
TensorBoard.....	154
Инструмент What-If .....	157
tf-explain.....	161
Стандартные приемы для экспериментов с машинным обучением .....	163
Проверка данных .....	164
Разбиение данных на обучающую, проверочную и контрольную выборки .....	164
Ранняя остановка .....	165
Воспроизводимость экспериментов.....	166

Пример сквозного пайплайна глубокого обучения .....	166
Простой пайплайн переноса обучения .....	166
Простой пайплайн создания сети.....	169
Влияние гиперпараметров на точность .....	169
Сравнение переноса обучения и обучения с нуля.....	170
Влияние количества слоев для тонкой настройки при переносе обучения.....	171
Влияние объема данных на перенос обучения.....	172
Влияние скорости обучения .....	173
Влияние оптимизатора .....	174
Влияние размера пакета .....	175
Влияние изменения размеров .....	176
Влияние изменения соотношения сторон на перенос обучения.....	177
Инструменты автоматизации настройки моделей для достижения максимальной точности.....	178
Keras Tuner.....	178
AutoAugment .....	180
AutoKeras .....	181
Итоги .....	182
<b>Глава 6. Увеличение скорости и эффективности TensorFlow: удобный чек-лист .....</b>	<b>183</b>
Голодание GPU.....	183
nvidia-smi .....	184
Профиляровщик TensorFlow + TensorBoard .....	186
Как использовать этот чек-лист.....	187
Чек-лист настроек производительности.....	188
Подготовка данных .....	188
Чтение данных .....	188
Аугментация данных .....	188
Обучение .....	189
Инференс.....	189
Подготовка данных.....	190
Сохраните данные в формате TFRecord .....	190
Уменьшите размеры исходных данных.....	191
Используйте TensorFlow Datasets .....	191
Чтение данных .....	192
Используйте tf.data .....	192
Организуйте предварительное извлечение данных .....	193

Организуйте параллельную обработку на CPU.....	194
Организуйте параллельный ввод/вывод и обработку.....	194
Разрешите недетерминированный порядок следования данных .....	194
Кэшируйте данные.....	195
Включите экспериментальные оптимизации.....	196
Автоматическая настройка значений параметров.....	197
Аугментация данных.....	198
Используйте GPU для аугментации.....	198
Обучение.....	200
Используйте автоматическую смешанную точность .....	200
Используйте пакеты большого размера.....	201
Используйте значения, кратные восьми .....	203
Определите оптимальную скорость обучения.....	203
Используйте <code>tf.function</code> .....	205
Переобучите и научите обобщать .....	207
Установите оптимизированный программный стек для поддержки оборудования .....	209
Оптимизируйте количество потоков, выполняющихся на CPU параллельно.....	210
Используйте более производительное оборудование .....	212
Используйте распределенное обучение.....	212
Изучите отраслевые бенчмарки.....	214
Инференс .....	216
Используйте эффективную модель .....	216
Используйте квантование модели .....	219
Прореживайте модели .....	222
Используйте совмещенные операции.....	223
Включите сохранение состояния GPU .....	223
Итоги .....	224
<b>Глава 7. Практические инструменты, советы и приемы .....</b>	<b>225</b>
Установка.....	225
Обучение .....	227
Модель .....	229
Данные .....	230
Защищенность .....	233
Обучение и исследования.....	234
Последний вопрос .....	236

<b>Глава 8. Облачные API для компьютерного зрения: установка и запуск за 15 минут.....</b>	<b>237</b>
Ландшафт API распознавания образов .....	239
Clarifai .....	239
Microsoft Cognitive Services .....	240
Google Cloud Vision .....	240
Amazon Rekognition.....	242
IBM Watson Visual Recognition .....	242
Algorithmia .....	244
Сравнение API для распознавания образов .....	245
Предлагаемые услуги .....	245
Стоимость.....	246
Точность .....	247
Предвзятость .....	249
Подготовка и использование облачных API .....	252
Обучение собственного классификатора.....	255
Основные причины плохой работы классификатора .....	260
Сравнение качества работы собственных классификаторов в разных API ...	261
Настройка производительности облачных API.....	264
Влияние изменения разрешения на API разметки изображений.....	265
Влияние сжатия на API разметки изображений.....	265
Влияние сжатия на API оптического распознавания символов .....	266
Влияние изменения разрешения на API оптического распознавания символов .....	266
Примеры.....	267
New York Times .....	268
Uber .....	268
Giphy .....	268
OmniEarth .....	271
Photobucket.....	271
Staples .....	271
InDro Robotics .....	272
Итоги .....	273
<b>Глава 9. Масштабируемый инференс в облаке с помощью TensorFlow Serving и KubeFlow .....</b>	<b>274</b>
Ландшафт услуг прогнозирования с помощью ИИ .....	275
Flask: создание собственного сервера .....	277
Создание REST API с помощью Flask.....	277

Развертывание модели Keras в Flask.....	278
Плюсы использования Flask .....	279
Минусы использования Flask.....	280
Желаемые качества системы производственного уровня .....	280
Высокая доступность.....	280
Масштабируемость.....	281
Низкая задержка.....	281
Географическая доступность.....	282
Обработка сбоев.....	283
Мониторинг.....	283
Управление версиями модели .....	283
A/B-тестирование .....	284
Поддержка нескольких библиотек машинного обучения .....	284
Google Cloud ML Engine: управляемый стек облачных услуг ИИ .....	284
Плюсы использования Cloud ML Engine .....	285
Минусы использования Cloud ML Engine.....	285
Создание API классификации .....	285
TensorFlow Serving .....	292
Установка.....	293
KubeFlow.....	294
Пайплайны.....	297
Инструменты управления.....	297
Установка.....	298
Соотношение цены и производительности.....	300
Анализ затрат на управляемый стек Inference-as-a-Service .....	300
Анализ затрат на создание собственного стека.....	302
Итоги .....	303
<b>Глава 10. ИИ в браузере с TensorFlow.js и ml5.js.....</b>	<b>304</b>
Библиотеки машинного обучения на JavaScript: краткая история.....	305
ConvNetJS .....	306
Keras.js .....	307
ONNX.js .....	308
TensorFlow.js .....	308
Архитектура TensorFlow.js .....	310
Использование предварительно обученных моделей с TensorFlow.js.....	312
Преобразование модели для использования в браузере .....	314
Обучение в браузере .....	314
Извлечение признаков .....	315

Сбор данных.....	317
Обучение .....	317
Нагрузка на GPU.....	319
ml5.js.....	320
PoseNet.....	322
pix2pix.....	325
Сравнительный анализ и практические рекомендации.....	330
Размер модели .....	331
Время инференса .....	331
Примеры.....	333
Дирижер .....	333
TensorSpace.....	334
Metacar.....	335
Классификация фотографий в Airbnb .....	336
GAN Lab .....	336
Итоги .....	337
<b>Глава 11. Классификация объектов в реальном времени в iOS с Core ML.....</b>	<b>338</b>
Жизненный цикл разработки искусственного интеллекта для мобильных устройств.....	340
Краткая история Core ML .....	342
Альтернативы фреймворку Core ML.....	343
TensorFlow Lite.....	344
ML Kit .....	344
Fritz .....	345
Архитектура машинного обучения Apple .....	345
Предметно-ориентированные фреймворки .....	346
ML Framework .....	346
Оптимизированные примитивы ML .....	346
Приложение для распознавания объектов в реальном времени.....	347
Конвертация моделей в формат Core ML.....	354
Конвертация из формата Keras.....	354
Конвертация из формата TensorFlow.....	354
Развертывание динамической модели .....	356
Обучение на устройстве .....	357
Федеративное обучение .....	358
Анализ качества моделей.....	358
Бенчмарк моделей на iPhone.....	359

Оценка энергопотребления .....	363
Оценка нагрузки .....	365
Уменьшение размера приложения.....	368
Не внедряйте модели в приложение .....	368
Используйте квантование .....	369
Используйте Create ML.....	370
Примеры приложений .....	371
Magic Sudoku .....	371
Seeing AI .....	372
HomeCourt .....	373
InstaSaber + YoPuppet .....	374
Итоги .....	377
<b>Глава 12. Not Hotdog на iOS с Core ML и Create ML.....</b>	<b>379</b>
Сбор данных .....	381
Подход 1: поиск готового или создание своего датасета .....	381
Подход 2: загрузка изображений с помощью расширения Fatkun для браузера Chrome.....	382
Подход 3: загрузка с помощью Bing Image Search API.....	385
Обучение модели .....	386
Подход 1: с помощью инструментов с веб-интерфейсом.....	386
Подход 2: с помощью Create ML.....	390
Подход 3: тонкая настройка с использованием Keras .....	396
Конвертация модели с использованием Core ML Tools .....	397
Создание приложения для iOS.....	397
Что можно сделать дальше .....	398
Итоги .....	399
<b>Глава 13. Шазам для еды: разработка приложений для Android с помощью TensorFlow Lite и ML Kit .....</b>	<b>400</b>
Цикл разработки приложения для классификации блюд.....	401
Обзор TensorFlow Lite .....	403
Архитектура TensorFlow Lite .....	406
Конвертация модели в формат TensorFlow Lite .....	407
Создание приложения для распознавания объектов.....	408
ML Kit + Firebase .....	415
Классификация объектов в ML Kit .....	417
Использование своих моделей в ML Kit.....	417
Модели, размещенные в облаке .....	419

А/В-тестирование моделей, размещенных в облаке .....	423
Использование эксперимента в коде .....	426
TensorFlow Lite в iOS .....	427
Оптимизация производительности.....	428
Квантование с помощью TensorFlow Lite Converter .....	428
Набор инструментов TensorFlow для оптимизации моделей .....	428
Fritz .....	429
Целостный взгляд на цикл разработки мобильных приложений ИИ.....	432
Как собирать данные? .....	433
Как размечать данные? .....	434
Как обучить модель? .....	434
Как конвертировать модель в формат для мобильных устройств?.....	434
Как оптимизировать производительность модели? .....	435
Как повысить привлекательность для пользователей? .....	435
Как развернуть модель? .....	436
Как оценить успешность модели? .....	436
Как совершенствовать модель?.....	436
Как обновить модель на телефонах пользователей? .....	437
Самосовершенствующаяся модель .....	437
Примеры из практики .....	439
Lose It!.....	439
Режим портретной съемки на телефонах Pixel 3 .....	441
Распознавание голоса от Alibaba .....	442
Определение контуров лица с помощью ML Kit .....	442
Сегментация видео в реальном времени в YouTube Stories .....	443
Итоги .....	444
<b>Глава 14. Создание приложения Purrfect Cat Locator с помощью TensorFlow Object Detection API.....</b>	<b>445</b>
Виды задач компьютерного зрения.....	446
Классификация .....	447
Локализация .....	447
Обнаружение.....	447
Сегментация.....	448
Способы обнаружения объектов.....	450
Использование готовых облачных API обнаружения объектов.....	451
Использование предварительно обученных моделей.....	453
Получение модели .....	453

Тест-драйв модели .....	454
Развертывание на устройстве .....	455
Создание своей модели обнаружения объектов без программирования ....	456
Развитие технологии обнаружения объектов .....	460
Вопросы производительности.....	462
Ключевые термины в обнаружении объектов .....	463
Intersection over Union.....	464
Mean Average Precision.....	465
Non-Maximum Suppression.....	465
Создание своих моделей с помощью TensorFlow Object Detection API .....	466
Сбор данных.....	467
Разметка данных .....	469
Предварительная обработка данных.....	473
Исследование модели.....	474
Обучение .....	476
Конвертация модели.....	478
Сегментация изображений .....	479
Примеры из практики .....	481
Умный холодильник .....	481
Подсчет толпы.....	482
Распознавание лиц в приложении Seeing AI .....	483
Беспилотные автомобили.....	484
Итоги .....	486
<b>Глава 15. Как стать творцом: ИИ в краевых устройствах.....</b>	<b>487</b>
Обзор краевых устройств ИИ .....	488
Raspberry Pi.....	489
Intel Movidius Neural Compute Stick.....	491
Google Coral USB Accelerator.....	492
NVIDIA Jetson Nano.....	494
FPGA + PYNQ .....	496
Arduino.....	499
Сравнение краевых устройств для ИИ.....	501
Raspberry Pi.....	504
Ускорение с помощью Google Coral USB Accelerator.....	506
NVIDIA Jetson Nano .....	508
Сравнение производительности краевых устройств .....	511
Примеры из практики .....	512

JetBot .....	512
Билеты на проезд в метро за приседания .....	514
Сортировщик огурцов .....	515
Что изучать дальше .....	517
Итоги .....	517
<b>Глава 16. Моделирование беспилотного автомобиля методом сквозного глубокого обучения с использованием Keras.....</b>	<b>519</b>
Краткая история автоматизации вождения.....	521
Глубокое обучение, автономное вождение и проблема данных.....	522
«Hello, World!» в автономном вождении: управление в моделируемой среде .....	525
Инструменты и требования.....	525
Исследование и подготовка данных .....	528
Определение области интереса .....	530
Аугментация данных .....	532
Дисбаланс датасета и стратегии вождения .....	533
Обучение модели автопилота .....	538
Генератор данных.....	539
Определение модели .....	542
Развертывание модели автопилота .....	547
Что изучать дальше .....	550
Расширение датасета.....	551
Обучение на последовательных данных.....	551
Обучение с подкреплением .....	552
Итоги .....	552
<b>Глава 17. Создание беспилотного автомобиля менее чем за час: обучение с подкреплением с помощью AWS DeepRacer .....</b>	<b>553</b>
Краткое введение в обучение с подкреплением .....	554
Почему для изучения обучения с подкреплением выбран беспилотный автомобиль?.....	555
Практика глубокого обучения с подкреплением с DeepRacer.....	557
Создание первой модели обучения с подкреплением .....	560
Шаг 1: создание модели.....	561
Шаг 2: настройка процесса обучения .....	562
Шаг 3: обучение модели .....	570
Шаг 4: оценка качества модели.....	571

Обучение с подкреплением на практике.....	572
Как происходит обучение с подкреплением?.....	572
Теория обучения с подкреплением .....	577
Алгоритм обучения с подкреплением в AWS DeepRacer .....	579
Кратко о порядке глубокого обучения с подкреплением на примере DeepRacer .....	581
Шаг 5: улучшение модели обучения с подкреплением .....	582
Гонки на автомобиле AWS DeepRacer .....	587
Создание трека.....	587
Шаблон трека для AWS DeepRacer с одним поворотом .....	588
Запуск модели на автомобиле AWS DeepRacer.....	589
Автономное вождение AWS DeepRacer.....	589
Что изучать дальше .....	592
Лига DeepRacer .....	592
AWS DeepRacer с расширенными возможностями.....	592
Олимпиада автопилотов с искусственным интеллектом.....	592
DIY Robocars .....	593
Roborace.....	594
Итоги .....	595
<b>Приложение. Краткое введение в сверточные нейронные сети .....</b>	<b>596</b>
Машинное обучение.....	596
Персепtron .....	596
Функции активации.....	597
Нейронные сети.....	598
Обратное распространение ошибки .....	600
Недостатки нейронных сетей .....	600
Желаемые свойства классификатора изображений.....	600
Свертка.....	600
Объединение.....	602
Структура сверточной сети .....	602
Что изучать дальше .....	604
<b>Об авторах .....</b>	<b>604</b>
Основные авторы .....	604
Приглашенные авторы.....	606
<b>Иллюстрация на обложке.....</b>	<b>608</b>

---

# Вступление

Мы переживаем возрождение интереса к искусственному интеллекту (ИИ). Вероятно, поэтому вы листаете эту книгу. Есть множество книг о глубоком обучении, и возникает резонный вопрос: с какой целью эта книга вообще появилась? Ответим на него чуть ниже.

Занимаясь глубоким обучением с 2013 года (разрабатывая продукты в Microsoft, NVIDIA, Amazon и Square), мы стали свидетелями значительных изменений в этой сфере. Непрекращающиеся исследования были данностью, а отсутствие зрелого инструментария — правдой жизни.

Продолжая расти и перенимая опыт сообщества, мы обратили внимание, что нет каких-либо четких указаний на то, как превратить исследования в конечный продукт для обычных пользователей, которые взаимодействуют с веб-браузером, смартфоном или краевым (edge) устройством. В поисках тайных знаний мы бесконечно экспериментировали, читали блоги, обсуждали проблемы на GitHub и Stack Overflow, а также переписывались с авторами пакетов и ловили случайные инсайты. Многие книги по большей части фокусировались на теории или на особенностях использования конкретных инструментов. Лучшее, на что мы могли рассчитывать с такими книгами, — это построить игрушечную модель.

Чтобы заполнить пробел между теорией и практикой, мы с самого начала стали говорить о возможности перехода от исследований ИИ к его практическому применению. Мы старались дать мотивирующие примеры, а также показать различные уровни сложности в зависимости от уровня навыков (от любителя до инженера масштаба Google) и усилий, затраченных на внедрение глубокого обучения в продакшен. Все это было ценно и для новичков, и для экспертов.

К счастью, со временем ландшафт стал доступным для новичков, и появилось больше инструментов. Отличные онлайн-площадки, например Fast.ai и DeepLearning.ai, упростили подход к приемам обучения моделей ИИ. На рынке появились книги по основам глубокого обучения с использованием TensorFlow и PyTorch. Но даже несмотря на все это, между теорией и практикой оставалась пропасть. Мы решили восполнить этот пробел. Так и появилась эта книга.

Простым и доступным языком, на примере реальных интересных проектов в сфере компьютерного зрения книга описывает простые классификаторы, не предполагая наличия у читателя знаний о машинном обучении и ИИ. Затем сложность проектов постепенно увеличивается, повышается их точность

и скорость, открывается возможность масштабировать их до обслуживания миллионов пользователей на широком спектре оборудования и программного обеспечения. И наконец, описывается пример использования обучения с подкреплением для создания миниатюрного беспилотного автомобиля.

Почти каждая глава начинается с мотивирующего примера и вопросов, которые могут возникнуть в процессе поиска решения. Потом описываются различные подходы к решению, с разным уровнем сложности и требуемых усилий. Если вы ищете быстрое решение, то просто прочтите несколько первых страниц каждой главы — и все. Тем же, кто хочет получить более глубокое понимание предмета, следует прочитать всю главу и изучить примеры. Во-первых, они довольно интересные. Во-вторых, они показывают, как специалисты в сфере ИИ применяют идеи, описываемые в главе, для создания реальных продуктов.

Мы также обсудим проблемы, с которыми сталкиваются практики и профессионалы глубокого обучения при создании реальных приложений с использованием облачных систем, браузеров, мобильных и краевых устройств. В этой книге мы собрали ряд практических советов и приемов, а также уроков из реальной жизни, чтобы побудить читателей создавать приложения, которые смогут сделать чай-то день немного лучше.

## Разработчикам

Скорее всего, вы опытный программист. Даже если вы не знакомы с Python, мы надеемся, что с легкостью освоите его и сможете приступить к работе в кратчайшие сроки. Но мы не предполагаем у вас наличие какого-либо опыта в области машинного обучения и искусственного интеллекта. Мы поможем обрести вам эти знания и уверены, что вы извлечете пользу, узнав из этой книги:

- как создавать продукты с ИИ, ориентированные на пользователя;
- как быстро обучать модели;
- как минимизировать усилия и объем кода, нужные для создания прототипа;
- как сделать модели более производительными и энергоэффективными;
- как внедрять и масштабировать модели глубокого обучения, а также оценивать связанные с этим затраты;
- как применять ИИ на практике на примере более 40 реальных проектов;
- в каких направлениях развиваются знания о глубоком обучении;
- об обобщенных приемах, которые предлагают новые фреймворки (например, PyTorch), предметных областях (например, здравоохранение, робототехника), видах входной информации (например, видео, аудио, текст) и задачах (например, сегментация изображений, обучение с одного раза).

## Специалистам по данным

Возможно, вы уже хорошо разбираетесь в машинном обучении и знаете, как работать с моделями глубокого обучения. Тогда у нас хорошие новости. Вы сможете еще больше обогатить набор своих навыков и углубить знания в этой области, которые позволят создавать настоящие продукты. Эта книга расскажет вам, как эффективнее решать повседневные задачи, а также:

- как ускорить обучение, в том числе в кластерах с множеством нод;
- как развить интуицию, нужную для разработки и отладки моделей, включая настройку гиперпараметров для значительного повышения их точности;
- как работает модель, как выявить предвзятость в данных, а также как автоматически подобрать лучшие гиперпараметры и архитектуру модели с помощью AutoML;
- даст советы и приемы от других практиков, в том числе касающиеся быстрого сбора данных, организации экспериментов, обмена моделями с другими специалистами во всем мире и получения самой свежей информации о лучших доступных моделях для вашей задачи;
- как использовать инструменты для развертывания и масштабирования своей лучшей модели для реальных пользователей, и даже автоматически (без участия DevOps-команды).

## Студентам

Сейчас самое время подумать о будущей карьере в области ИИ — это следующая революционная технология после появления интернета и смартфонов. Здесь уже достигнуты большие успехи, но многое еще предстоит открыть. Надеемся, что эта книга станет для вас первым шагом к карьере в области ИИ и, что еще лучше, к развитию более глубоких теоретических знаний. Самое замечательное, что не нужно тратить много денег на покупку дорогостоящего оборудования — можно тренироваться на мощном оборудовании совершенно бесплатно, пользуясь лишь браузером (спасибо, Google Colab!). Надеемся, что с помощью этой книги вы:

- начнете карьеру в области ИИ, познакомившись с пакетом интересных проектов;
- обретете знания и навыки, которые помогут подготовиться к стажировке и устройству на работу;
- дадите волю своему творчеству, создавая забавные приложения, например автопилот для автомобиля;

- станете чемпионом состязания «AI for Good» (ИИ во имя добра) и используете свой творческий потенциал для решения насущных проблем, с которыми сталкивается человечество.

## Преподавателям

Надеемся, что эта книга поможет разнообразить ваш курс забавными проектами из реального мира. Мы подробно рассматриваем каждый этап пайплайна глубокого обучения, а также эффективные и действенные методы выполнения каждого этапа. Все проекты, представленные в книге, могут пригодиться для совместной или индивидуальной работы в течение семестра. А еще мы подготовили презентации на <https://github.com/PracticalDL/Practical-Deep-Learning-Book>, которые можно брать для занятий.

## Энтузиастам робототехники

Робототехника — это увлекательно. Если вы энтузиаст робототехники, то не нужно убеждать вас в том, что наделение роботов интеллектом — неминуемый шаг. Все функциональные аппаратные платформы: Raspberry Pi, NVIDIA Jetson Nano, Google Coral, Intel Movidius, PYNQ-Z2 и другие — помогают внедрению инноваций в области робототехники. По мере развития промышленной революции 4.0 некоторые из этих платформ будут становиться все более актуальными и повсеместными. Из этой книги вы узнаете:

- как создать и обучить ИИ, а затем довести его до совершенства;
- как тестировать и сравнивать краевые устройства по производительности, размеру, мощности, энергоэффективности и стоимости;
- как выбрать оптимальный алгоритм ИИ и устройство для конкретной задачи;
- как другие производители создают творческих роботов и машины;
- как добиться дальнейшего прогресса в этой области и показать достижения.

## Обзор глав

### *Глава 1. Обзор ландшафта искусственного интеллекта*

Посмотрим, как менялся ландшафт ИИ с 1950-х годов до наших дней. Поговорим о компонентах идеального рецепта глубокого обучения. Познакомимся с общепринятой терминологией и известными датасетами, а также заглянем в мир ответственного ИИ.

*Глава 2. Что на картинке: классификация изображений с помощью Keras*

Погрузимся в мир классификации изображений, написав лишь пять строк кода, использующего Keras. Затем с помощью тепловых карт узнаем, на что обращают внимание нейронные сети, делая свои прогнозы. А на десерт ознакомимся с любопытной историей Франсуа Шолле, создателя Keras. Она покажет, какое влияние может оказаться всего один человек.

*Глава 3. Кошки против собак: перенос обучения с помощью Keras в 30 строках кода*

В этой главе рассмотрим прием переноса обучения и задействуем ранее обученную сеть в новой задаче классификации, чтобы за считанные минуты получить высочайшую точность. Затем изучим результаты, чтобы оценить качество классификации. Попутно создадим пайплайн машинного обучения, который с небольшими изменениями будет использоваться во всей книге. В качестве бонуса узнаем от Джереми Ховарда, соучредителя fast.ai, о том, как сотни тысяч студентов используют перенос обучения, чтобы начать свой путь в ИИ.

*Глава 4. Создание механизма обратного поиска изображений. Эмбеддинги*

Вдохновившись примером Google Reverse Image Search, изучим эмбеддинги, представления признаков изображений, и покажем, как реализовать обратный поиск похожих изображений в десяти строках кода. А затем начнется самое интересное: изучив различные стратегии и алгоритмы, мы ускорим работу этой системы, чтобы она анализировала миллионы изображений и отыскивала похожие за микросекунды.

*Глава 5. От новичка до мастера прогнозирования: увеличение точности сверточной нейронной сети*

В этой главе изучим стратегии увеличения точности классификатора с помощью TensorBoard, What-If Tool, tfexplain, TensorFlow Datasets, AutoKeras и AutoAugment. И немного поэкспериментируем, чтобы выработать интуитивное представление о том, какие параметры могут влиять на качество работы вашей задачи ИИ.

*Глава 6. Увеличение скорости и эффективности TensorFlow: удобный чек-лист*

В этой главе увеличим скорость обучения и прогнозирования, пройдясь по чек-листву из 30 трюков, которые помогут устраниТЬ неэффективные действия и максимально использовать возможности текущего оборудования.

*Глава 7. Практические инструменты, советы и приемы*

Здесь сгруппируем практические знания и навыки, начиная от установки, сбора данных, управления экспериментами, визуализации и слежения за современными исследованиями до изучения дальнейших возможностей развития теоретических основ глубокого обучения.

*Глава 8. Облачные API для компьютерного зрения: установка и запуск за 15 минут*

Работайте не 12 часов, а головой. В этой главе мы покажем, как за 15 минут приступить к использованию возможностей облачных платформ ИИ, предлагаемых Google, Microsoft, Amazon, IBM и Clarifai. Для задач, неразрешимых для существующих API, используем специальные сервисы классификации. С их помощью, не прибегая к программированию, обучим облачные классификаторы, а затем сравним их друг с другом. Увидев победителя, вы, скорее всего, немало удивитесь.

*Глава 9. Масштабируемый инференс в облаке с помощью TensorFlow Serving и KubeFlow*

В этой главе мы перенесем нашу обученную модель из облака на уровень клиента для масштабирования обслуживания запросов от нескольких сотен до миллионов. Здесь мы познакомимся с Flask, Google Cloud ML Engine, TensorFlow Serving и KubeFlow и проанализируем усилия, сценарии и затраты.

*Глава 10. ИИ в браузере с TensorFlow.js и ml5.js*

Любой человек, использующий компьютер или смартфон, всегда имеет доступ к браузеру. Охватите всех этих пользователей с помощью библиотек глубокого обучения, работающих в браузере, таких как TensorFlow.js и ml5.js. Приглашенный автор Зайд Аляфеи (Zaid Alyafeai) познакомит нас с оценкой позы тела, применением генеративно-состязательных сетей, преобразованием изображений с помощью Pix2Pix и многими другими задачами, выполняемыми не на сервере, а в самом браузере. В качестве бонуса вы узнаете от ключевых участников о том, как развивались проекты TensorFlow.js и ml5.js.

*Глава 11. Классификация объектов в реальном времени в iOS с Core ML*

Изучим ландшафт глубокого обучения на мобильных устройствах, уделив особое внимание экосистеме Apple с Core ML. Проведем сравнительный анализ моделей на разных моделях iPhone, рассмотрим стратегии уменьшения размеров приложений и их энергопотребления, а также изучим динамическое развертывание моделей, обучение на устройстве и приемы создания профессиональных приложений.

*Глава 12. Not Hotdog на iOS с Core ML и Create ML*

Приложение Not Hotdog из сериала HBO «Кремниевая долина» — это аналог «Hello World» в сфере мобильного ИИ. Воздадим ему должное и сделаем свою версию, и не одним или двумя, а тремя разными способами.

*Глава 13. Шазам для еды: разработка приложений для Android с помощью TensorFlow Lite и ML Kit*

В этой главе перенесем ИИ на Android, используя TensorFlow Lite. Затем рассмотрим кроссплатформенную разработку с помощью ML Kit

(основанного на TensorFlow Lite) и Fritz и изучим полный жизненный цикл разработки на примере самосовершенствующегося приложения с ИИ. Попутно рассмотрим управление версиями моделей, А/В-тестирование, оценку успешности, динамическое обновление, оптимизацию модели и другие темы. А в качестве бонуса Пит Уорден (Pete Warden), техлид проекта Mobile and Embedded TensorFlow, расскажет о своем богатом опыте по внедрению ИИ в краевые устройства.

### *Глава 14. Создание приложения Purrfect Cat Locator с помощью TensorFlow Object Detection API*

Здесь мы изучим четыре метода обнаружения объектов на изображениях, познакомимся с эволюцией приемов обнаружения объектов, проанализируем компромиссы между скоростью и точностью и заложим основы для исследований в подсчете численности толпы, распознавании лиц и беспилотных автомобилей.

### *Глава 15. Как стать творцом: ИИ в краевых устройствах*

Приглашенный автор Сэм Стеркваль (Sam Sterckval) занимается реализацией моделей глубокого обучения для устройства с низким энергопотреблением. Он покажет несколько встраиваемых устройств с поддержкой ИИ, имеющих разную вычислительную мощность и стоимость, включая Raspberry Pi, NVIDIA Jetson Nano, Google Coral, Intel Movidius и PYNQ-Z2 FPGA, и приоткроет двери для проектов в сфере робототехники и умных устройств. Бонус: узнайте от команды NVIDIA Jetson Nano о том, как можно самостоятельно собрать робота!

### *Глава 16. Моделирование беспилотного автомобиля методом сквозного глубокого обучения с использованием Keras*

Приглашенные авторы Адитья Шарма (Aditya Sharma) и Митчелл Сприн (Mitchell Spryn) расскажут об обучении виртуального автомобиля в фотографически реалистичной среде моделирования Microsoft AirSim, управляя им вручную и обучая модель ИИ воспроизводить его поведение. Попутно рассмотрим ряд идей, используемых в индустрии беспилотных автомобилей.

### *Глава 17. Создание беспилотного автомобиля менее чем за час: обучение с подкреплением с помощью AWS DeepRacer*

Перейдем от виртуального мира к физическому — приглашенный автор Сунил Маллья (Sunil Mallya) покажет, как всего за час можно собрать и обучить миниатюрный автомобиль AWS DeepRacer. С помощью обучения с подкреплением автомобиль научится самостоятельно водить, штрафуя себя за ошибки и добиваясь максимального успеха. Узнаем, как применить эти знания в гонках от Olympics of AI Driving до RoboRace (с использованием полноразмерных беспилотных автомобилей). А в финале Анима Анандку-

мар (Anima Anandkumar) из NVIDIA и Крис Андерсон (Chris Anderson), основатель DIY Robocars, расскажут, куда движется индустрия беспилотных автомобилей.

## Условные обозначения

В этой книге приняты следующие условные обозначения:

### *Курсив*

Используется для обозначения новых терминов, адресов URL и электронной почты, имен файлов и расширений имен файлов.

### Моноширинный

Применяется для оформления листингов программ и программных элементов внутри обычного текста, таких как имена переменных и функций, баз данных, типов данных, переменных окружения, инструкций и ключевых слов.



Так выделяются советы и предложения.



Так обозначаются советы, предложения и примечания общего характера.



Так обозначаются предупреждения и предостережения.

## Использование исходного кода примеров

Вспомогательные материалы (примеры кода, упражнения и т. д.) доступны для загрузки по адресу <https://github.com/PracticalDL/Practical-Deep-Learning-Book>. Если у вас возникнут вопросы технического характера по использованию примеров кода, направляйте их по электронной почте на адрес [PracticalDLBook@gmail.com](mailto:PracticalDLBook@gmail.com).

В общем случае все примеры кода из книги вы можете использовать в своих программах и в документации. Вам не нужно обращаться в издательство за

разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Если вы разрабатываете программу и используете в ней несколько фрагментов кода из книги, вам не нужно обращаться за разрешением. Но для продажи или распространения примеров из книги вам потребуется разрешение от издательства O'Reilly. Вы можете отвечать на вопросы, цитируя данную книгу или примеры из нее, но для включения существенных объемов программного кода из книги в документацию вашего продукта потребуется разрешение.

Мы рекомендуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN.

За получением разрешения на использование значительных объемов программного кода из книги обращайтесь по адресу [permissions@oreilly.com](mailto:permissions@oreilly.com).

## Благодарности

### Коллективные благодарности

Мы хотели бы поблагодарить следующих людей за неоценимую помощь в работе над этой книгой. Без них она была бы невозможна.

Книга появилась на свет благодаря усилиям нашего редактора Николь Таше (Nicole Taché). Она болела за нас и на каждом этапе давала ценные рекомендации. Николь помогла правильно расставить приоритеты, отобрать материал (хотите верьте, хотите нет, но изначально книга была намного больше) и не сбиться с пути. Она была первым читателем, и поэтому нам было важно, чтобы Николь было понятно содержание несмотря на то, что она новичок в ИИ. Мы безмерно благодарны ей за поддержку.

Хотим поблагодарить и остальную часть команды O'Reilly, в том числе выпускающего редактора Кристофера Фоше (Christopher Faucher), который без устали трудился, чтобы эта книга вовремя попала в типографию. Мы также благодарны редактору Бобу Расселу (Bob Russell), который поразил нас своей молниеносной скоростью правки и вниманием к деталям. Он заставил нас осознать, насколько важно уделять внимание грамматике английского языка в школьном курсе (хотя он и опоздал на несколько лет). Хотим также поблагодарить Рэйчел Румелиотис (Rachel Roumeliotis), вице-президента по контент-стратегии, и Оливию Макдональд (Olivia MacDonald), главного редактора по развитию, за веру в проект и постоянную поддержку.

Мы безмерно признательны нашим приглашенным авторам, которые согласились поделиться опытом. Адитья Шарма (Aditya Sharma) и Митчелл Сприн (Mitchell Spryn) из Microsoft показали, что нашу любовь к гоночным видео-

играм можно использовать для обучения беспилотных автомобилей, управляя ими в среде моделирования AirSim. Сунил Малля (Sunil Mallya) из Amazon показал, как перенести эти знания в физический мир, показав на примере, что достаточно одного часа, чтобы собрать миниатюрный беспилотный автомобиль (AWS DeepRacer) и научить его ездить по треку с помощью обучения с подкреплением. Сэм Стеркваль (Sam Sterckval) из Edgise подвел итог огромному разнообразию встраиваемого оборудования для реализации ИИ, доступного на рынке, так что теперь мы можем принять участие в разработке проекта по робототехнике. И наконец, Зайд Аляфеи (Zaid Alyafeai) из Университета имени короля Фахда показал, что браузеры вполне способны поддерживать серьезные интерактивные модели ИИ (с помощью TensorFlow.js и ml5js).

Свой нынешний вид книга приобрела благодаря оперативной обратной связи наших замечательных научных редакторов, которые кропотливо изучали проекты, указывали на технические неточности и давали рекомендации по улучшению изложения идей. Благодаря их отзывам (и постоянно меняющемуся TensorFlow API) мы заново переписали почти всю книгу после выхода черновой версии. Благодарим Margaret Maynard-Reid (Margaret Maynard-Reid), эксперта по машинному обучению из Google. Возможно, вы знакомы с ней заочно, если читали документацию для TensorFlow. Благодарим Pako Nathan (Paco Nathan), более 35 лет работающего в индустрии в Derwin Inc. и познакомившего Анирудха с миром публичных выступлений, Энди Петрелла (Andy Petrella), CEO и основателя Kensu и создателя SparkNotebook, чьи технические идеи соответствуют его репутации, и Никхиту Коула (Nikhita Koul), старшего научного сотрудника в Adobe, который предлагал новые улучшения после каждой итерации чтения черновика. Фактически он прочитал несколько тысяч страниц, и благодаря ему книга стала намного доступнее. Нам очень помогли рецензенты, обладающие богатым опытом в конкретных областях, будь то ИИ в браузере, разработка мобильных приложений или создание беспилотных автомобилей. Вот поглавный список рецензентов:

- глава 1: Дхарини Чандрасекаран (Dharini Chandrasekaran), Шерин Томас (Sherin Thomas);
- глава 2: Анудж Шарма (Anuj Sharma), Чарльз Козиерок (Charles Kozierok), Манодж Парихар (Manoj Parihar), Панкеш Бамотра (Pankesh Bamotra), Пранав Кант (Pranav Kant);
- глава 3: Анудж Шарма, Чарльз Козиерок, Манодж Парихар, Панкеш Бамотра, Пранав Кант;
- глава 4: Анудж Шарма, Манодж Парихар, Панкеш Бамотра, Пранав Кант;
- глава 6: Габриэль Ибагон (Gabriel Ibagon), Иржи Симса (Jiri Simsa), Макс Кац (Max Katz), Панкеш Бамотра;
- глава 7: Панкеш Бамотра;

- глава 8: Дипеш Аггарвал (Deepesh Aggarwal);
- глава 9: Панкеш Бамотра;
- глава 10: Бретт Берли (Brett Burley), Лоран Денуэ (Laurent Denoue), Манрадж Сингх (Manraj Singh);
- глава 11: Дэвид Апгар (David Apgar), Джеймс Уэбб (James Webb);
- глава 12: Дэвид Апгар;
- глава 13: Джесси Уилсон (Jesse Wilson), Салман Гадит (Salman Gadit);
- глава 14: Акшит Аора (Akshit Arora), Пранав Кант, Рохит Танеджа (Rohit Taneja), Ронай Ак (Ronay Ak);
- глава 15: Гертруи Ван Ломмель (Geertrui Van Lommel), Джок Декуббер (Joke Decubber), Жульен Де Кок (Jolien De Cock), Марианна Ван Ломмель (Marianne Van Lommel), Сэм Хендрикс (Sam Hendrickx);
- глава 16: Дарио Салищик (Dario Salischiker), Курт Нибур (Kurt Niebuhr), Мэттью Чан (Matthew Chan), Правин Паланисами (Praveen Palanisamy);
- глава 17: Киртеш Гарг (Kirtesh Garg), Ларри Пизетт (Larry Pizette), Пьер Дюма (Pierre Dumas), Рикардо Суэйрас (Ricardo Sueiras), Сеголен Десертин-Панхард (Segolene Dessertine-panhard), Шри Элапролу (Sri Elaprolu), Тацуя Араи (Tatsuya Arai).

Во многих главах есть врезки «От создателя», в которых известные специалисты из мира ИИ приоткрывают дверь в свой мир и рассказывают, как и почему они создавали проекты, принесшие им известность. Мы благодарны Франсуа Шолле (François Chollet), Джереми Ховарду (Jeremy Howard), Питу Уордену (Pete Warden), Аниме Анандкумар (Anima Anandkumar), Крису Андерсону (Chris Anderson), Шанкину Каю (Shanqing Cai), Дэниелу Смилкову (Daniel Smilkov), Кристобалю Валенсуэле (Cristobal Valenzuela), Дэниелу Шиффману (Daniel Shiffman), Харту Вулери (Hart Woolery), Дэну Абдинуру (Dan Abdinoor), Читоку Ято (Chitoku Yato), Джону Уэлшу (John Welsh) и Дэнни Атсмо (Danny Atsmo).

## Личные благодарности

Хочу поблагодарить свою семью — Арбинда, Сароджу и Никхиту, которые дали мне все, чтобы я мог заниматься моими увлечениями. Спасибо всем специалистам и исследователям из Microsoft, Aira и Yahoo, которые поддерживали меня и помогали воплощать идеи в прототипы и продукты. Не успехи, а трудности научили нас многому на этом пути. И эти трудности стали основной великолепного материала для этой книги в таком объеме, что можно было бы написать еще 250 страниц! Спасибо академическим сообществам университетов Карнеги — Меллона, Далхаузи и Тапара: вы научили меня многому (и не только тому, от чего зависел мой средний балл). Спасибо

сообществу слепых и слабовидящих — вы ежедневно вдохновляли меня на работу в области ИИ, показывая, что возможности людей, вооруженных правильными инструментами, действительно безграничны.

*Anirad*

Мой дедушка, сам автор, однажды сказал мне: «Писать книги труднее, чем можно подумать, но это приносит больше удовольствия, чем можно представить». Бесконечно благодарен своим бабушке и дедушке, семье, маме, папе и Шрие за то, что подталкивали меня на поиск знаний и помогли стать тем, кто я есть. Спасибо моим замечательным коллегам и наставникам из университета Карнеги — Меллона, CERN, NASA FDL, Deep Vision, NITH и NVIDIA, которые были со мной все это время. Я в большом долгу перед ними за знания и развитие научного темперамента. Хотел бы выразить благодарность своим друзьям, которые, надеюсь, еще не забыли меня, поскольку в последнее время я был очень занят. Огромное спасибо вам за невероятное терпение. Надеюсь, снова увижу вас всех. Огромное спасибо моим товарищам, которые самоотверженно рецензировали главы книги и давали обратную связь — без вас книга не сложилась бы.

*Ciddha*

Я в неоплатном долгу перед своими родителями Раджагопалом и Лакшми за их бесконечную любовь и поддержку, а также за стремление обеспечить мне хорошую жизнь и дать хорошее образование. Я благодарен своим профессорам из университета Флориды и национального технологического института Вис-весварая, которые учили меня и способствовали моей специализации в области информатики. Я благодарен своей девушке Джулии Таннер, которой почти два года приходилось терпеть бесконечные звонки по Skype по ночам и выходным, а также мои ужасные шутки (часть из которых, к сожалению, попала в эту книгу). Хотел бы также отметить своего замечательного руководителя Джоэла Кустку, поддерживавшего меня, пока я работал над этой книгой. Большой привет всем моим друзьям, которые проявили невероятное понимание, когда я не мог уделять им времени столько, сколько им хотелось бы.

*Mexep*

И последнее, но не менее важное: спасибо создателям Grammarly! Благодаря вам люди с неважными оценками по английскому могут стать публикуемыми авторами.

## ГЛАВА 1

---

# Обзор ландшафта искусственного интеллекта

Ниже приводится фрагмент из фундаментальной статьи доктора Мэй Карсон (May Carson; рис. 1.1) об изменении роли искусственного интеллекта (ИИ) в жизни человека в XXI веке:

*Искусственный интеллект часто называют электричеством XXI века. Со временем программы искусственного интеллекта смогут управлять всеми видами промышленной деятельности (включая здравоохранение), разрабатывать медицинские устройства и создавать новые типы продуктов и услуг, включая роботов и автомобили. Многие организации уже работают над такими программами искусственного интеллекта, способными выполнять свою работу и, что особенно важно, не допускать ошибок или опасных происшествий. Организациям нужен ИИ, но они также понимают, что ИИ подходит не для всех сфер человеческой деятельности.*

*Мы провели обширные исследования, чтобы выяснить, что нужно для работы искусственного интеллекта с использованием этих методов и политик, и пришли к важному выводу: количество денег, затрачиваемых на программы ИИ на одного человека в год, сравнимо с суммой, затрачиваемой на их исследование, создание и производство. Это кажется очевидным, но в действительности не совсем так.*

*Прежде всего системы искусственного интеллекта нуждаются в поддержке и обслуживании. А чтобы быть по-настоящему надежными, им нужны люди, обладающие знаниями и навыками управления такими системами и способные помогать им решать некоторые задачи. Для организаций очень важно иметь в своем штате таких сотрудников, помогающих профессиональному интеллекту в его работе. Также важно понимать людей, которые делают эту работу, особенно если ИИ сложнее людей. Например, люди особенно часто работают на должностях, где важно обладать передовыми знаниями, но не требуется владеть навыками работы с системами, которые необходимо создавать и поддерживать.*



**Рис. 1.1.** Доктор Мэй Карсон

## Извинения

А теперь признаемся: все, что мы написали выше в этой главе, — фейк. Буквально все! Весь текст (кроме первого предложения, которое мы написали в качестве затравки) был сгенерирован с помощью модели GPT-2, созданной Адамом Кингом (Adam King), на сайте *TalkToTransformer.com*. Имя автора цитаты мы сгенерировали с помощью генератора имен «Nado Name Generator» на сайте *Onitools.moe*. Вы думаете, что уж фотография автора точно должна быть настоящей, да? А вот и нет! Изображение взято с сайта *ThisPersonDoesNotExist.com*, где при каждом его посещении генерируются фотографии несуществующих людей с помощью генеративно-состязательных сетей (Generative Adversarial Networks, GAN).

Мы сомневались, правильно ли начинать книгу с мистификации, но при этом хотели показать современное состояние ИИ, когда вы, наш читатель, меньше всего этого ожидали. Вообще, способности ИИ удивляют, ошеломляют, а иногда даже пугают. Тот факт, что он в состоянии с нуля создать более осмысленный и красноречивый текст, чем некоторые мировые лидеры, говорит сам за себя.

Однако есть кое-что, чего ИИ не сможет унаследовать от нас, — нашу способность веселиться. Надеемся, что эти первые три фейковых абзаца будут самыми скучными во всей книге. В конце концов, мы не хотим, чтобы про нас сказали, что авторы скучнее машины.

## Настоящее вступление

Наверняка каждый может вспомнить какое-либо поразительное выступление иллюзиониста, вызвавшее вопрос: «Как, черт возьми, он это сделал?!» А возникли ли у вас такой вопрос, когда в новостях говорили о применении ИИ? В этой книге мы хотим вооружить вас знаниями и инструментами, которые позволят не только понять устройство ИИ, но и создать что-нибудь свое.

Шаг за шагом мы разберем устройство реальных приложений, использующих технологии ИИ, и покажем, как их создавать на самых разных платформах — от облака до браузера, от смартфонов до встраиваемых систем. Закончим же высшим достижением ИИ на сегодняшний день: беспилотными автомобилями.

Большинство глав начинается с описания мотивирующей задачи, за которым следует пошаговое создание комплексного решения. С первых глав мы будем развивать навыки, нужные для создания мозга ИИ. Но это только полдела. Истинная ценность ИИ — создание полезных приложений. И речь не об игрушечных прототипах. Мы хотим, чтобы вы создали софт, который люди смогут применять с пользой для себя. Отсюда и слова «реальные проекты» в названии книги. Поэтому мы обсудим различные доступные варианты и выберем наиболее подходящие, исходя из таких параметров, как производительность, энергопотребление, масштабируемость, надежность и конфиденциальность.

В первой главе мы отступим на шаг назад, чтобы оценить этот момент в истории ИИ. Мы узнаем, что такое искусственный интеллект, особенно в контексте глубокого обучения, и рассмотрим последовательность событий, благодаря которым глубокое обучение стало одной из самых революционных областей технического прогресса в начале XXI века. Мы также изучим основные компоненты, лежащие в основе законченного решения глубокого обучения, а в следующих главах займемся практикой.

Начнем же наше путешествие с фундаментального вопроса.

## Что такое ИИ?

В этой книге мы часто используем термины «искусственный интеллект», «машинное обучение» и «глубокое обучение», иногда как синонимы. Но, строго говоря, они имеют разные значения, и вот краткое описание каждого из них (также рис. 1.2):

### *Искусственный интеллект*

Позволяет машинам имитировать поведение человека. Известный пример ИИ — IBM Deep Blue.

### *Машинное обучение*

Раздел ИИ, в котором машины используют статистические методы для самообучения на основе имеющейся информации. Цель МО — научить машину действовать, опираясь на данные, полученные в прошлом. Если вы смотрели телешоу *Jeopardy!*, в котором IBM Watson побеждает Кена Дженнингса (Ken Jennings) и Бреда Раттера (Brad Rutter), то видели машинное обучение в действии. Более наглядный пример: когда в следующий раз в ваш почтовый ящик не попадет спам, поблагодарите за это машинное обучение.

### *Глубокое обучение*

Это подраздел МО, в котором глубокие многослойные нейронные сети используются для прогнозирования — например, в области компьютерного зрения, распознавания речи, анализа естественного языка и т. д.



**Рис. 1.2.** Связь между ИИ, машинным обучением и глубоким обучением

В этой книге основное внимание уделяется глубокому обучению.

## **Мотивирующие примеры**

А теперь к делу. Что побудило нас написать эту книгу? А зачем вы потратили свои кровно заработанные деньги<sup>1</sup> на ее покупку? Наша мотивация проста и по-

<sup>1</sup> Если вы скачали эту книгу с пиратского сайта, то разочаровали нас.

нятна: вовлечь как можно больше людей в мир ИИ. А тот факт, что вы читаете эту книгу, означает, что половина работы сделана.

Чтобы по-настоящему заинтересовать вас, представим несколько ярких примеров уже достигнутых возможностей ИИ:

- «DeepMind's AI agents conquer human pros at StarCraft II» («Агенты ИИ от компании DeepMind побеждают опытных игроков в StarCraft II»): *The Verge*, 2019.
- «AI-Generated Art Sells for Nearly Half a Million Dollars at Christie's» («Картина, созданная искусственным интеллектом, продана на аукционе Christie's почти за полмиллиона долларов»): *AdWeek*, 2018.
- «AI Beats Radiologists in Detecting Lung Cancer» («ИИ превзошел радиологов в обнаружении рака легких»): *American Journal of Managed Care*, 2019.
- «Boston Dynamics Atlas Robot Can Do Parkour» («Робот Atlas, созданный компанией Boston Dynamics, может заниматься паркуром»): *ExtremeTech*, 2018.
- «Facebook, Carnegie Mellon build first AI that beats pros in 6-player poker» («Facebook и Университет Карнеги — Меллона создали первый ИИ, превзошедший профессиональных игроков в покер на шесть игроков»): *Facebook*, 2019.
- «Blind users can now explore photos by touch with Microsoft's Seeing AI» («С помощью Microsoft Seeing AI слепые пользователи теперь смогут просматривать фотографии на ощупь»): *Tech-Crunch*, 2019.
- «IBM's Watson supercomputer defeats humans in final Jeopardy match» («Суперкомпьютер IBM Watson победил человека в финальном матче Jeopardy»): *Venture Beat*, 2011.
- «Google's ML-Jam challenges musicians to improvise and collaborate with AI» («Система машинного обучения ML-Jam, созданная в Google, предлагает музыкантам импровизировать и играть с ИИ»): *VentureBeat*, 2019.
- «Mastering the Game of Go without Human Knowledge» («Нейросеть AlphaGo Zero, созданная в DeepMind, научилась играть в го за 3 дня»): *Nature*, 2017.
- «Chinese AI Beats Doctors in Diagnosing Brain Tumors» («Китайский ИИ превзошел врачей в диагностике опухолей головного мозга»): *Popular Mechanics*, 2018.
- «Two new planets discovered using artificial intelligence» («С помощью ИИ обнаружены две новые планеты»): *Phys.org*, 2019.
- «Nvidia's latest AI software turns rough doodles into realistic landscapes» («Новейшее программное обеспечение ИИ, созданное в Nvidia, превращает наброски в реалистичные пейзажи»): *The Verge*, 2019.

Эти примеры реализации ИИ послужат нам Полярной звездой. Уровень перечисленных достижений сопоставим с победами на Олимпийских играх. Но кроме них в мире есть множество приложений, решающих практические задачи, которые можно сравнить с гонкой на 5 км. Разработка этих приложений не требует многолетнего обучения, но доставляет разработчикам огромное удовлетворение, когда они пересекают финишную черту. Мы хотим научить вас проходить эту дистанцию в 5 км.

Мы намеренно уделяем внимание широте обсуждаемых вопросов. Область ИИ меняется очень быстро, поэтому мы хотим вооружить вас правильным мышлением и набором инструментов. Помимо решения отдельных задач, посмотрим, как разные, казалось бы, никак не связанные между собой задачи имеют фундаментальные совпадения, которые можно использовать в своих интересах. Например, для распознавания речи используются сверточные нейронные сети (Convolutional Neural Networks, CNN), которые одновременно являются основой современного компьютерного зрения. Мы затронем практические аспекты нескольких областей, чтобы вы могли быстро пройти 80 % пути в решении реальных задач. Если мы заинтересовали вас, чтобы вы решили пройти еще 15 % пути, то будем считать нашу цель достигнутой. Как часто говорят, мы хотим демократизировать ИИ.

Важно отметить, что значительный прогресс в области ИИ, который трудно переоценить, произошел только в последние несколько лет. Чтобы показать, как далеко мы продвинулись, вот вам пример: пять лет назад требовалось иметь степень Ph.D., только чтобы войти в эту индустрию. Пять лет спустя не понадобится Ph.D., даже чтобы написать целую книгу по этой теме. (Серьезно, проверьте наши профили!) Современные приложения глубокого обучения кажутся удивительными, но появились они не на пустом месте. Они стоят на плечах многих гигантов отрасли, которые десятилетиями раздвигали границы. И чтобы по достоинству оценить значение этого времени, нужно заглянуть в прошлое.

## Краткая история ИИ

Итак, наша Вселенная существовала в виде горячей и бесконечно плотной точки. Потом почти 14 миллиардов лет назад началось расширение и... Так, стоп! Не будем углубляться в прошлое так далеко. На самом деле первые семена ИИ были посажены всего 70 лет назад. В 1950 году Аллан Тьюринг (Alan Turing) в своей работе «Вычислительные машины и разум» впервые задал вопрос: могут ли машины мыслить? Это положило начало более широкой философской дискуссии о разумности и о том, что значит быть человеком. Означает ли это способность сочинять музыку под воздействием собственных мыслей и эмоций? Тьюринг счел такой подход слишком ограниченным и предложил тест: если человек не может отличить машину от другого человека, то имеет ли это значение? ИИ, способный имитировать человека, по сути, является человеком.

## Захватывающее начало

Термин «искусственный интеллект» был придуман Джоном Маккарти (John McCarthy) в 1956 году в рамках летнего исследовательского проекта Дартмута (Dartmouth Summer Research Project). В то время настоящих компьютеров еще не было, поэтому особенно примечательно, что участники проекта обсуждали такие футуристические темы, как языковое моделирование, самосовершенствующиеся машины, абстракции на основе сенсорных данных и многое другое. Конечно, обсуждения носили сугубо теоретический характер. Но это был первый случай, когда ИИ стал областью исследований, а не отдельным проектом.

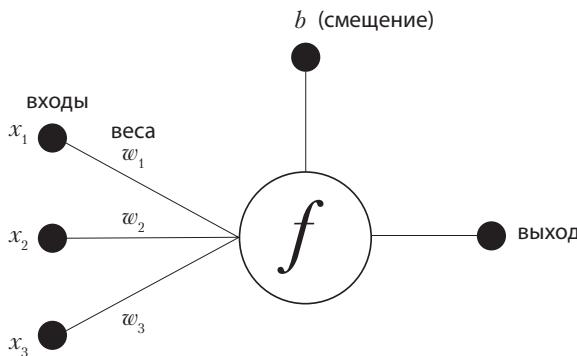
Статья Фрэнка Розенблатта (Frank Rosenblatt) «Perceptron: A Perceiving and Recognizing Automaton», опубликованная в 1957 году, заложила основы глубоких нейронных сетей. Он предположил возможность построить электронную или электромеханическую систему, способную научиться распознавать сходство между образцами оптической, электрической или тональной информации и которая будет функционировать подобно человеческому мозгу. Вместо модели на основе правил (стандартной для алгоритмов того времени) он предложил использовать статистические модели прогнозирования.



В этой книге мы снова и снова будем повторять термин *«нейронная сеть»*. Что это такое? Это упрощенная модель человеческого мозга. Как и в мозге, в ней есть *нейроны*, которые активируются, встретив что-то знакомое. Различные нейроны соединены друг с другом связями (напоминающими синапсы в нашем мозгу), которые помогают информации перетекать от одного нейрона к другому.

На рис. 1.3 пример простейшей нейронной сети: перцептрон. Математически перцептрон можно выразить так:

$$\text{выход} = f(x_1, x_2, x_3) = x_1 w_1 + x_2 w_2 + x_3 w_3 + b.$$



**Рис. 1.3.** Пример перцептрана

В 1965 году Ивахненко (Ivakhnenko) и Лапа (Lapa) представили первую действующую нейронную сеть в своей статье «Group Method of Data Handling – A Rival Method of Stochastic Approximation», ссылаясь на которую многие, несмотря на некоторые разногласия, считают Ивахненко отцом глубокого обучения.

Примерно в это же время были сделаны смелые прогнозы относительно способностей машин. Перевод с одного языка на другой, распознавание речи и многое другое будут выполняться машинами лучше, чем людьми. Эти прогнозы взволновали правительства по всему миру, и они начали финансировать подобные проекты. Золотая лихорадка началась в конце 1950-х годов и продолжалась до середины 1970-х.

## Холодные и мрачные дни

За эти годы в исследования были инвестированы миллионы долларов, и появились первые действующие системы. Но многие из первоначальных пророчеств оказались нереалистичными. Машины распознавали речь только при определенном произношении и только для ограниченного набора слов. Машинный перевод с одного языка на другой содержал множество ошибок и обходился намного дороже, чем перевод, выполненный человеком. Перцептроны (по сути, однослойные нейронные сети) очень скоро уперлись в предел надежности прогнозирования. Это ограничивало их применение для решения большинства практических задач. Проблема в том, что перцептроны — линейные функции, тогда как в реальных задачах для точного прогнозирования часто нужен нелинейный классификатор. Представьте, что вы пытаетесь аппроксимировать кривую линию прямой!

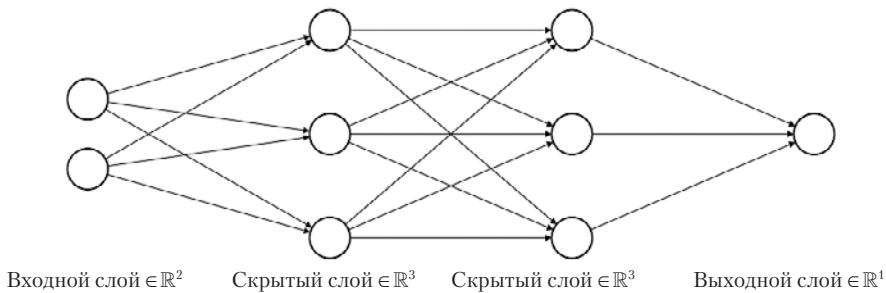
А что происходит, когда вы обещаете слишком много, но не выполняете обещаний? Правильно: теряете финансирование. Управление перспективных исследовательских проектов Министерства обороны США (Defense Advanced Research Project Agency), широко известное как DARPA (да-да, это те самые люди, которые создали сеть ARPANET, ставшую потом интернетом), профинансировало множество оригинальных проектов в США. Но отсутствие результатов за почти два десятилетия вызвало охлаждение интереса. Было легче высадить человека на Луну, чем получить пригодный к употреблению распознаватель речи!

Точно так же в 1974 году, по другую сторону Атлантики, был опубликован отчет Лайтхилла (Lighthill Report), в котором говорилось: «Универсальный робот — это мираж». Представьте, что вы в 1974 году, живете в Великобритании и наблюдаете, как авторитеты информатики обсуждают в телепередачах BBC оправданность затрат на исследования ИИ. В итоге исследования были остановлены сначала в Соединенном Королевстве, а затем и во всем мире, из-за чего

рухнули карьеры многих перспективных ученых. Эта фаза утраты веры в ИИ длилась около двух десятилетий и стала известна как «зима ИИ». Эх, если бы Нед Старк был в то время поблизости и смог бы предупредить их!

## Проблеск надежды

Но даже в те ненастные дни в области ИИ продолжалась работа. Конечно, перцептроны, будучи линейными функциями, имели ограниченные возможности. А можно ли исправить этот недостаток? Например, связать перцептроны в сеть так, чтобы выходы одних соединялись с входами других, то есть построить многослойную нейронную сеть, как показано на рис. 1.4. Чем больше слоев, тем глубже будет изучаться нелинейность, что приведет к созданию более точных прогнозов. Остается только решить: как обучать такую сеть? И тут на сцену выходит Джонни Хинтон (Geoffrey Hinton) с коллегами. В 1986 году они опубликовали статью «Learning representations by back-propagating errors» с описанием метода под названием *обратное распространение ошибки*. Как он работает? Сеть делает прогноз, смотрит, насколько он далек от реальности, и распространяет величину ошибки по сети в обратном направлении, чтобы учиться ее исправлять. Этот процесс повторяется, пока ошибка не станет несущественной. Простая, но мощная идея. Термин «обратное распространение ошибки» будет часто использоваться в этой книге.



**Рис. 1.4.** Пример многослойной нейронной сети (изображение заимствовано из статьи)

В 1989 году Джордж Кибенко (George Cybenko) представил первое доказательство *универсальной аппроксимационной теоремы* (Universal Approximation Theorem), утверждающей, что нейронная сеть с единственным скрытым слоем теоретически способна смоделировать любую задачу. Это означает способность нейронных сетей превзойти (по крайней мере, теоретически) любой другой подход к машинному обучению. И, черт возьми, они могут даже имитировать человеческий мозг! Но все это только на бумаге. Размеры таких сетей ограничены возможностями реального мира. Частично эти ограничения можно преодолеть,

создав несколько скрытых слоев и обучив такую сеть... Стоп! А как же обратное распространение ошибки?

Что касается практического воплощения, то в 1986 году команда из университета Карнеги – Меллона построила первое в истории беспилотное транспортное средство NavLab 1 (рис. 1.5). Первоначально для управления рулем в нем использовалась однослойная нейронная сеть. В итоге в рамках этого проекта в 1995 году был создан автомобиль NavLab 5. Во время демонстрации он самостоятельно проехал почти 50 миль из 2850-мильного пробега из Питтсбурга в Сан-Диего. NavLab получил водительские права еще до того, как родились многие инженеры Tesla!

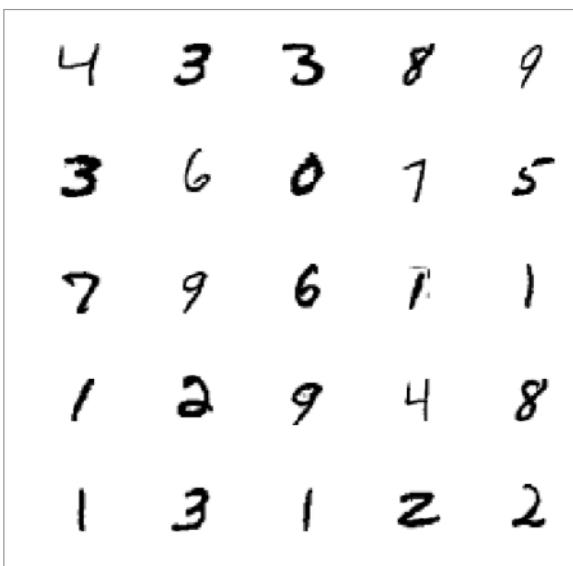


**Рис. 1.5.** Беспилотный автомобиль NavLab 1 1986 года во всей своей красе  
(изображение взято по адресу [http://www.cs.cmu.edu/Groups/ahs/navlab\\_list.html](http://www.cs.cmu.edu/Groups/ahs/navlab_list.html))

Еще один примечательный пример из 1980-х: почтовой службе США (United States Postal Service, USPS) потребовалось наладить автоматическую сортировку почтовых отправлений по индексам получателей. Поскольку большая часть писем всегда писалась от руки, оптическое распознавание символов оказалось бессильно. Чтобы решить эту задачу, Ян ЛеКун (Yann LeCun) с коллегами использовал набор рукописных образцов из Национального института стандартов и технологий (National Institute of Standards and Technology, NIST), чтобы показать способность нейронных сетей распознавать рукописные цифры. Они опубликовали результаты в своей статье «Backpropagation Applied to Handwritten Zip Code Recognition». Созданная ими сеть LeNet несколько десятков лет использовалась почтовой службой США для автоматического сканирования и сортировки почты. Пример особенно примечателен, потому что это была первая сверточная сеть, нашедшая практическое применение на производстве. Позднее, в 1990-х годах, банки начали использовать усовершенствованную версию этой сети под названием LeNet-5 для чтения рукописных цифр на чеках. Так была заложена основа современного компьютерного зрения.



Те из вас, кто читал о датасете MNIST, возможно, заметили связь с только что упомянутым институтом NIST. Дело в том, что датасет MNIST фактически включает подмножество изображений из исходного датасета NIST, к которым были применены некоторые модификации (modifications — «М» в MNIST), чтобы упростить процесс обучения и тестирования нейронной сети. Модификации, часть из которых видны на рис. 1.6, заключались в изменении размеров изображений до  $28 \times 28$  пикселей, центрировании цифр в этой области, стгаживании и т. д.



**Рис. 1.6.** Выборка рукописных цифр из датасета MNIST

Другие исследователи тоже продолжили свои работы, в том числе Юрген Шмидхубер (Jürgen Schmidhuber), предложивший сети с долгой краткосрочной памятью (Long Short-Term Memory, LSTM) с многообещающими перспективами для распознавания рукописного текста и речи.

Несмотря на существенное развитие теории, результаты тогда нельзя было показать на практике, главным образом из-за дороговизны вычислительного оборудования и сложности его масштабирования для решения более крупных задач. Даже если каким-то чудом оборудование оказывалось доступным, собрать данные, позволяющие полностью реализовать его потенциал, было очень нелегко. В конце концов, развитие интернета все еще находилось на этапе коммутируемого доступа. Метод опорных векторов (Support Vector Machine, SVM) — метод машинного обучения, разработанный для задач классификации в 1995 году — был быстрее и давал достаточно хорошие результаты на меньших объемах данных, а потому выглядел предпочтительнее.

В результате ИИ и глубокое обучение заработали плохую репутацию. Аспирантов предостерегали от проведения исследований в области глубокого обучения, потому что именно в этой области «многие умные ученые закончили свою карьеру». Люди и компании, работающие в этой области, стали использовать альтернативные обозначения и термины, такие как информатика, когнитивные системы, интеллектуальные агенты, машинное обучение и другие, чтобы отмежеваться от слов «искусственный интеллект». Похоже на то, как Министерство войны США (Department of War) было переименовано в Министерство обороны (Department of Defense), чтобы название было менее пугающим.

## Как глубокое обучение вошло в моду

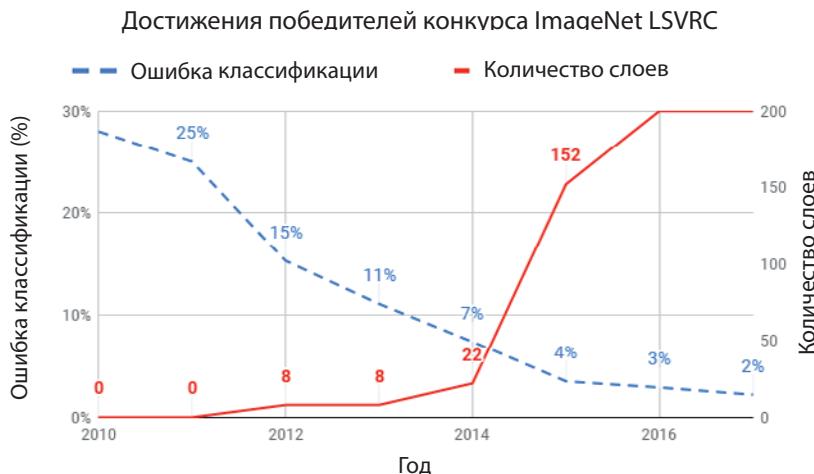
К счастью для нас, 2000-е принесли высокоскоростной интернет, смартфоны с камерами, видеоигры, Flickr и Creative Commons (что дало возможность легально использовать фотографии, созданные другими людьми). Огромное количество людей получили возможность быстро делать фотографии с помощью карманного устройства, а затем мгновенно выгружать их. Океан данных наполнялся, и постепенно стали появляться возможности искушаться в нем. В результате такого счастливого стечения обстоятельств и упорного труда Фей-Фей Ли (Fei-Fei Li, в ту пору трудившейся в Принстонском университете) и ее коллег появился датасет ImageNet с 14 миллионами изображений.

Тогда же серьезного уровня достигли компьютерные и консольные игры. Геймерам нужна была все более качественная графика. Это, в свою очередь, подтолкнуло производителей графических процессоров (Graphics Processing Unit, GPU), например NVIDIA, совершенствовать свое оборудование. При этом важно помнить, что GPU чертовски хороши в матричных операциях. Почему? Потому что этого требует математика. В компьютерной графике распространены такие задачи, как перемещение объектов, вращение объектов, изменение их формы, регулировка их освещения и т. д., и во всех этих задачах используются матричные операции. И GPU стали специализироваться на них. А знаете, где еще требуется масса матричных вычислений? В нейронных сетях. Это одно большое счастливое совпадение.

После создания ImageNet в 2010 году был организован ежегодный конкурс ImageNet Large Scale Visual Recognition Challenge (ILSVRC), побуждавший исследователей разрабатывать более совершенные методы классификации данных из этого набора. Исследователям было предложено подмножество, разделенное на 1000 категорий и включавшее примерно 1,2 миллиона изображений. Первые пять мест заняли современные на тот момент методы компьютерного зрения, такие как масштабно-инвариантное преобразование признаков (Scale-Invariant Feature Transform, SIFT) и метод опорных векторов (SVM). Они дали 28 % (в 2010-м) и 25 % (в 2011-м) ошибок (то есть

если метод ошибался в одном случае из пяти, он считался точным). А затем наступил 2012 год, и в таблице лидеров появилась запись с уровнем ошибок почти в два раза ниже — до 16 %. Эту заявку подали Алекс Крижевский (Alex Krizhevsky), Илья Суткевич (Ilya Sutskever, который позднее основал OpenAI) и Джейфри Хинтон (Geoffrey Hinton) из университета Торонто. Это была сверточная нейронная сеть (Convolutional Neural Network, CNN) под названием AlexNet, построенная по образу и подобию LeNet-5. Даже всего с восемью слоями AlexNet имела 60 миллионов параметров и 650 000 нейронов, что дало модель с размером 240 Мбайт. Ее обучали неделю с помощью двух GPU NVIDIA. Это событие прозвучало как гром среди ясного неба, доказав потенциал CNN, после чего дело сдвинулось с мертвой точки и наступила современная эра глубокого обучения.

На рис. 1.7 показан количественный прогресс, достигнутый сверточными сетями за последнее десятилетие. С момента появления глубокого обучения в 2012 году количество ошибок классификации среди победителей конкурса ImageNet LSVRC каждый год снижалось на 40 %. По мере того как сверточные сети становились все глубже, ошибка уменьшалась.



**Рис. 1.7.** Прогресс победителей конкурса ImageNet LSVRC

Имейте в виду, что история ИИ здесь значительно сокращена, а некоторые детали опущены. Но, по сути, именно это сочетание — появление больших объемов доступных данных, развитие GPU и создание более совершенных методов — привело нас к эре глубокого обучения. Прогресс распространяется и на новые сферы. Как показано в табл. 1.1, то, что раньше считалось научной фантастикой, теперь стало реальностью.

**Таблица 1.1.** Наиболее значимые этапы эпохи глубокого обучения

2012	Нейронная сеть, созданная командой Google Brain, начинает распознавать кошек в видео-роликах на YouTube
2013	<ul style="list-style-type: none"> <li>Исследователи начинают применять глубокое обучение для решения множества задач.</li> <li>word2vec привносит контекст в слова и фразы, делая еще один шаг в сторону понимания их смысла.</li> <li>Уровень ошибок при распознавании речи снизился на 25 %</li> </ul>
2014	<ul style="list-style-type: none"> <li>Изобретены генеративно-состязательные сети (Generative Adversarial Network, GAN).</li> <li>Skype переводит речь в режиме реального времени.</li> <li>Чат-бот Юджина Густмана (Eugene Goostman) преодолевает тест Тьюринга.</li> <li>Изобретена модель «последовательность в последовательность» нейронных сетей.</li> <li>Преобразование изображений в предложения с их кратким описанием</li> </ul>
2015	<ul style="list-style-type: none"> <li>Обученная 1000-слойная сеть Microsoft ResNet превзошла человека в распознавании изображений.</li> <li>Deep Speech 2, созданная в Baidu, продемонстрировала возможность сквозного (end-to-end) распознавания речи.</li> <li>Gmail запустила услугу интеллектуального ответа Smart Reply.</li> <li>YOLO (You Only Look Once) обнаруживает объекты в масштабе реального времени.</li> <li>Visual Question Answering позволяет задавать вопросы на основе изображений</li> </ul>
2016	<ul style="list-style-type: none"> <li>AlphaGo побеждает профессиональных игроков в го.</li> <li>Google WaveNets помогает создавать реалистичные звуки.</li> <li>Microsoft достигает человеческого уровня в распознавании разговорной речи</li> </ul>
2017	<ul style="list-style-type: none"> <li>AlphaGo Zero самостоятельно обучается игре в го за 3 дня.</li> <li>Capsule Nets исправляет недостатки в сверточной сети.</li> <li>Представлен тензорный процессор (Tensor Processing Unit, TPU).</li> <li>Калифорния разрешает продажу беспилотных автомобилей.</li> <li>Pix2Pix обретает возможность генерировать изображения из набросков</li> </ul>
2018	<ul style="list-style-type: none"> <li>ИИ превосходит людей в проектировании ИИ, благодаря методам поиска нейронной архитектуры Neural Architecture Search.</li> <li>Демонстрационная версия Google Duplex может бронировать столики в ресторане от нашего имени.</li> <li>Deep Fakes заменяет одно лицо другим в видео.</li> <li>Нейронная сеть BERT компании Google помогает людям переводить с одного языка на другой.</li> <li>Созданы комплекты тестов DawnBench и MLPerf для проверки качества обучения искусственного интеллекта</li> </ul>

**Таблица 1.1** (окончание)

2019	<ul style="list-style-type: none"> <li>• OpenAI Five сокрушает чемпионов мира в игре Dota2.</li> <li>• StyleGAN создает фотorealisticные изображения.</li> <li>• OpenAI GPT-2 генерирует реалистичные отрывки текста.</li> <li>• Fujitsu обучает ImageNet за 75 секунд.</li> <li>• Microsoft инвестирует миллиард долларов в OpenAI</li> </ul>
------	--

Получив представление об истории развития ИИ и глубокого обучения, вы должны понимать, почему этот момент времени важен. Важно осознавать, насколько быстро движется прогресс в этой области. Но, как мы уже видели, так было не всегда.

По словам двух пионеров этой области, еще в 1960-х годах они прогнозировали срок появления настоящего компьютерного зрения как «этим летом». Они ошиблись всего на полвека! Да, быть футуристом непросто. Исследование Александра Висснера-Гросса (Alexander Wissner-Gross) показало, что между появлением алгоритма и прорывом, к которому ведет этот алгоритм, в среднем проходило 18 лет. С другой стороны, аналогичный разрыв между появлением датасета и прорывом, которого он помог достичь, составлял в среднем всего три года! Возьмите любое открытие, сделанное в последние десять лет, и вы увидите, что датасет, позволивший сделать это открытие, скорее всего, стал доступен всего за несколько лет до этого.

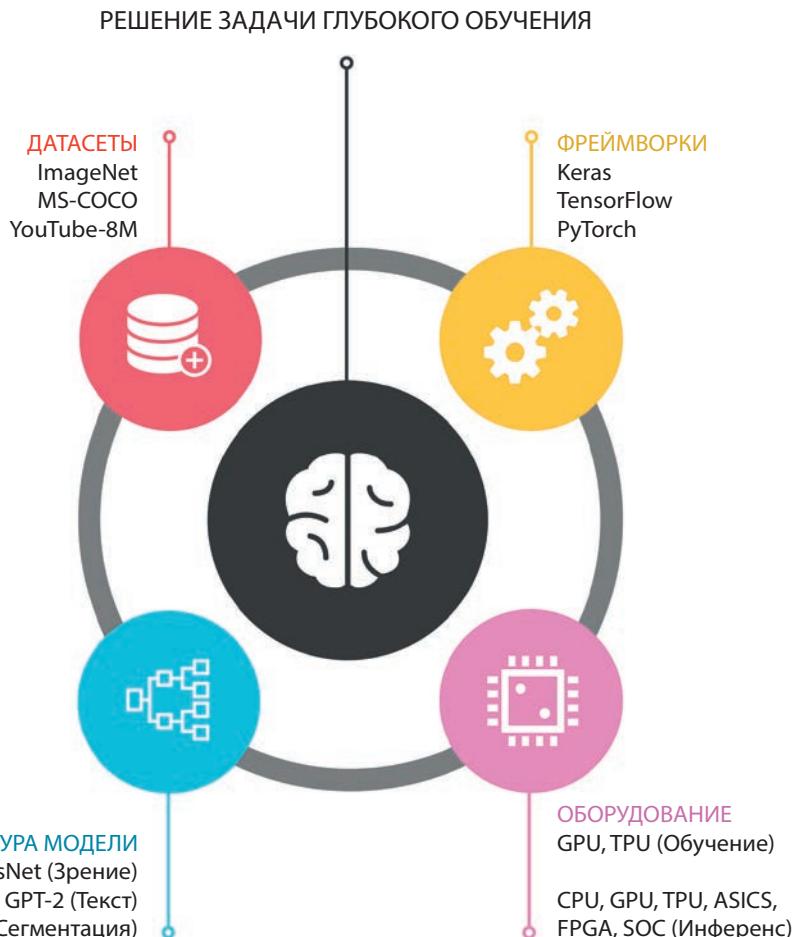
Очевидно, что нехватка данных была ограничивающим фактором. Это показывает, насколько велика роль хорошего датасета в глубоком обучении. Однако данные — не единственный фактор. Рассмотрим другие составляющие, образующие основу идеального решения глубокого обучения.

## Рецепт идеального решения задачи глубокого обучения

Прежде чем приступить к приготовлению блюд, Гордон Рамзи<sup>1</sup> проверяет наличие всех ингредиентов. То же относится к решению задач с помощью глубокого обучения (рис. 1.8). И вот ваш набор ингредиентов для глубокого обучения:

$$\begin{aligned} \text{Датасет} + \text{Модель} + \text{Фреймворк} + \text{Оборудование} = \\ = \text{Решение задачи глубокого обучения} \end{aligned}$$

<sup>1</sup> Гордон Джеймс Рамзи (Gordon James Ramsay; родился 8 ноября 1966 года) — британский шеф-повар. Его рестораны удостоены 16 звезд Мишлен. Его фирменный ресторан Restaurant Gordon Ramsay в Лондоне имеет три звезды Мишлен. Популярный ведущий британских телешоу. — Примеч. пер.



**Рис. 1.8.** Ингредиенты для идеального решения задачи глубокого обучения

Рассмотрим каждый из них поближе.

## Датасеты

Как моряк Попай страстно жаждет шпинат, так глубокое обучение страстно жаждет данных — много данных. Для выявления значимых закономерностей, которые помогают делать надежные прогнозы, нужны огромные объемы данных. В 1980-х и 1990-х годах нормой считалось традиционное машинное обучение, использовавшее сотни или тысячи образцов. Глубокие нейронные сети, построенные с нуля, напротив, требуют намного больше данных даже для типовых задач прогнозирования. Но в этом есть свой плюс — прогнозы получаются гораздо точнее.

Сейчас мы наблюдаем взрывной рост накопления данных: каждый день создают-ся квинтиллионы байтов данных — изображений, текста, видео, информации от датчиков и многое другого. Но чтобы использовать эти данные, нужны метки. Чтобы создать классификатор настроений, отделяющий положительные отзывы на Amazon от отрицательных, нужны тысячи таких отзывов с присвоенными им метками настроения. Чтобы обучить систему сегментации лица для Snapchat, нужны тысячи изображений, на которых точно отмечено расположение глаз, губ, носа и т. д. Чтобы обучить беспилотный автомобиль, нужны фрагменты видео, на которых показаны реакции человека-водителя на дорожную обстановку и его воздействие на органы управления автомобилем — тормоза, акселератор, рулевое колесо и т. д. Эти метки выступают в роли учителя нашего ИИ, и размеченные данные гораздо ценнее неразмеченных.

Создание меток может обходиться очень дорого. Неудивительно, что есть целая индустрия, которая занимается поиском исполнителей для решения задач разметки. Каждая метка стоит от нескольких центов до нескольких долларов, в зависимости от времени, потраченного исполнителем. Например, во время разработки датасета Microsoft COCO (Common Objects in Context — обычные объекты в контексте) требовалось примерно три секунды, чтобы присвоить метку с названием каждому объекту на изображении; примерно 30 секунд, чтобы определить ограничивающую рамку вокруг каждого объекта, и 79 секунд, чтобы обрисовать контур каждого объекта. Теперь представьте, что эти операции нужно произвести с сотнями тысяч изображений, и вы поймете, какие средства были вложены в создание некоторых из наиболее крупных датасетов. Отдельные компании, занимающиеся разметкой, — Appen и Scale.AI, уже оцениваются более чем в миллиард долларов каждая.

На нашем счете может не быть миллиона долларов. Но, к счастью для нас, в этой революции глубокого обучения произошло два хороших события.

- Крупные компании и университеты щедро поделились с нами, обнародовав гигантские наборы размеченных данных.
- Был разработан метод под названием «перенос обучения» (*transfer learning*), который позволяет настраивать модели с использованием небольших датасетов, насчитывающих всего несколько сотен образцов, при условии, что эти модели предварительно были обучены на более крупных наборах, аналогичных текущему. Мы неоднократно использовали этот метод в книге, в том числе в главе 5, где экспериментально доказали, что этот прием способен обеспечить достойное качество даже при наличии всего нескольких десятков обучающих образцов. Возможность переноса обучения развенчивает миф о том, что для обучения хорошей модели необходимы большие данные. Добро пожаловать в мир *крошечных данных!*

В табл. 1.2 перечислены некоторые из популярных датасетов для различных задач глубокого обучения.

**Таблица 1.2.** Общедоступные датасеты

Тип данных	Название	Описание
Изображения	Open Images V4 (Google)	<ul style="list-style-type: none"> <li>Девять миллионов изображений в 19 700 категориях.</li> <li>1,74 миллиона изображений в 600 категориях (ограничивающие рамки, bounding boxes)</li> </ul>
	Microsoft COCO	<ul style="list-style-type: none"> <li>330 000 изображений объектов в 80 категориях.</li> <li>Ограничивающие рамки, сегментация и по пять подписей к каждому изображению</li> </ul>
Видео	YouTube-8M	<ul style="list-style-type: none"> <li>6,1 миллиона видео, 3862 категории, 2,6 миллиарда аудиовизуальных признаков.</li> <li>По 3 метки на видео.</li> <li>1,53 ТБ случайно выбранных видео</li> </ul>
Видео, изображения	BDD100K (Калифорнийский университет в Беркли)	<ul style="list-style-type: none"> <li>100 000 видеороликов о рождении с суммарной продолжительностью более 1100 часов.</li> <li>100 000 изображений с ограничивающими рамками в 10 категориях.</li> <li>100 000 изображений с разметкой полос движения.</li> <li>100 000 изображений с сегментацией дорожных областей.</li> <li>10 000 изображений с пиксельной сегментацией экземпляров</li> </ul>
	Waymo Open Dataset	3000 сцен рождения с суммарной продолжительностью 16,7 часа, 600 000 кадров, примерно 25 миллионов ограничивающих 3-мерных рамок и 22 миллиона 2-мерных рамок
Текст	SQuAD	150 000 материалов из Википедии со смысловыми метками
	Yelp Reviews	Пять миллионов отзывов на Yelp
Спутниковые данные	Landsat Data	Несколько миллионов спутниковых изображений поверхности Земли (с шириной и высотой в 100 морских миль) в восьми спектральных диапазонах (с пространственным разрешением от 15 до 60 метров)
Аудио	Google AudioSet	2 084 320 10-секундных аудиозаписей с YouTube в 632 категориях
	LibriSpeech	1000 часов английской речи

## Архитектура модели

Модель — это просто функция. Она принимает один или несколько входных параметров и возвращает результат. Входными данными могут быть текст, изображения, аудио, видео или что-то еще, а результатом является прогноз. Хорошой считается модель, прогнозы которой достаточно точно соответствуют ожидаемой реальности. Точность модели, обученной на некотором датасете, является основным определяющим фактором ее пригодности для реального использования. Для большинства это все, что нужно знать о моделях глубокого обучения. Но если заглянуть внутрь модели, взору откроется множество интересных деталей (рис. 1.9).



**Рис. 1.9.** Черный ящик модели глубокого обучения

Внутри модели находится граф, состоящий из вершин и ребер. Вершины — это математические операции, а ребра представляют потоки данных между узлами. Другими словами, если выход одной вершины связать со входом одной или нескольких других вершин, то эти связи между вершинами будут представлены ребрами. Структура графа определяет потенциальную точность, скорость, количество потребляемых ресурсов (память, продолжительность вычислений и электроэнергия), а также тип входных данных, которые она может обрабатывать.

Расположение вершин и ребер определяется *архитектурой* модели. По сути, это план здания. Но кроме плана нужно еще само здание. Обучение — это строительство здания с помощью плана. Обучение модели происходит многократным выполнением нескольких этапов: 1) на вход подаются исходные данные; 2) извлекаются выходные данные; 3) определяется, насколько прогноз далек от ожидаемой реальности (то есть от меток, связанных с данными); затем 4) величина ошибки распространяется через модель в обратном направлении, чтобы она могла исправить себя. Этот процесс обучения повторяется снова и снова, пока не будет достигнут удовлетворительный уровень прогнозирования.

Результатом обучения является набор чисел (также известных как веса), которые присваиваются каждой из вершин. Эти веса являются необходимыми параметрами вершин в графе, обеспечивающими правильную обработку входных данных. Перед началом обучения весам обычно присваиваются

случайные значения. Целью процесса обучения фактически является постепенная настройка значений этих весов, пока они не обеспечат получение удовлетворительных прогнозов.

Чтобы лучше понять, как работают веса, рассмотрим следующий датасет с двумя входами и одним выходом:

**Таблица 1.3.** Пример датасета

вход <sub>1</sub>	вход <sub>2</sub>	выход
1	6	20
2	5	19
3	4	18
4	3	17
5	2	16
6	1	15

Применив методы линейной алгебры (или посчитав в уме), мы можем прийти к выводу, что этому датасету соответствует следующее уравнение:

$$\text{выход} = f(\text{вход}_1, \text{вход}_2) = 2 \times \text{вход}_1 + 3 \times \text{вход}_2.$$

В данном случае веса равны 2 и 3. Глубокая нейронная сеть имеет миллионы таких весовых параметров.

Разные типы используемых вершин определяют разные виды архитектур моделей, каждая из которых лучше подходит для входных данных определенного типа. Например, для анализа изображения и звука обычно используются сверточные нейронные сети (CNN), а для обработки текста — рекуррентные нейронные сети (RNN) и сети с долгой краткосрочной памятью (LSTM).

В общем случае обучение одной из таких моделей с нуля может потребовать довольно много времени, возможно, недели. К счастью, многие исследователи уже проделали всю тяжелую работу по обучению своих моделей на универсальных датасетах (например, ImageNet) и сделали их доступными для всех. Благодаря этому можно взять эти модели и настроить их для своего конкретного датасета. Этот процесс называется переносом обучения (transfer learning) и подходит для большинства практических потребностей.

По сравнению с обучением с нуля перенос обучения дает двойное преимущество: значительно сокращается время обучения (от нескольких минут до часов вместо недель), и он дает хорошие результаты даже при использовании относительно

небольшого датасета (от сотен до тысяч образцов вместо миллионов). В табл. 1.4 перечислены некоторые примеры известных архитектур моделей.

**Таблица 1.4.** Примеры известных архитектур моделей

Задача	Примеры архитектур моделей
Классификация изображений	ResNet-152 (2015), MobileNet (2017)
Классификация текста	BERT (2018), XLNet (2019)
Сегментация изображений	U-Net (2015), DeepLabv3 (2018)
Преобразование изображений	Pix2Pix (2017)
Распознавание объектов	YOLO9000 (2016), Mask R-CNN (2017)
Генерация речи	WaveNet (2016)

Для всех моделей, перечисленных в табл. 1.4, опубликованы показатели точности на эталонных датасетах (например, для моделей классификации — на наборе ImageNet, для моделей распознавания объектов — на наборе MS COCO). Кроме того, разные архитектуры имеют свои характерные требования к ресурсам (размер модели в мегабайтах или требование к производительности в виде количества операций с плавающей запятой в секунду — floating-point operations in second, FLOPS).

Подробнее о переносе обучения поговорим в следующих главах, а сейчас познакомимся с доступными фреймворками и сервисами глубокого обучения.



Когда в 2015 году Кайминг Хе (Kaiming He) с коллегами придумал 152-слойную архитектуру ResNet — это был настоящий подвиг для того времени, учитывая, что предыдущая самая большая модель GoogLeNet состояла из 22 слоев — у всех возник только один вопрос: «Почему не 153 слоя?» Как оказывается, Каймингу просто не хватило памяти, доступной графическому процессору.

## Фреймворки

Есть несколько библиотек глубокого обучения. Кроме того, есть фреймворки, специализирующиеся на использовании обученных моделей для прогнозирования (или инференса) и оптимизации их использования в приложениях.

Как это часто случается с софтом в целом, многие библиотеки появлялись и исчезали: Torch (2002), Theano (2007), Caffe (2013), Microsoft Cognitive Toolkit (2015), Caffe2 (2017) — и ландшафт постоянно менялся. Но уроки не проходили даром, и последующие библиотеки получались все более простыми

в использовании, более мощными, более производительными и вызывающими интерес как у новичков, так и у экспертов. В табл. 1.5 перечислены некоторые из популярных библиотек.

**Таблица 1.5.** Популярные фреймворки глубокого обучения

Фреймворк	Основное назначение	Типичная целевая платформа
TensorFlow (включая Keras)	Обучение	Настольные компьютеры, серверы
PyTorch	Обучение	Настольные компьютеры, серверы
MXNet	Обучение	Настольные компьютеры, серверы
TensorFlow Serving	Прогнозирование	Серверы
TensorFlow Lite	Прогнозирование	Мобильные и встраиваемые устройства
TensorFlow.js	Прогнозирование	Браузеры
ml5.js	Прогнозирование	Браузеры
Core ML	Прогнозирование	Устройства компании Apple
Xnor AI2GO	Прогнозирование	Встраиваемые устройства

## TensorFlow

В 2011 году в Google Brain была разработана библиотека глубокого обучения DistBelief для внутренних исследований и разработок. Она обучала сеть Inception (победительницу конкурса ImageNet Large Scale Visual Recognition Challenge 2014 года), а также улучшала качество распознавания речи в продуктах Google. Эта библиотека была тесно связана с инфраструктурой Google, имела запутанные настройки, и ею было очень сложно поделиться вне Google. Осознав ограничения, сотрудники Google приступили к созданию распределенного фреймворка МО второго поколения, который должен был получиться универсальным, масштабируемым, высокопроизводительным и переносимым на многие аппаратные платформы. И, что самое приятное, он имел открытый исходный код. В Google этот фреймворк назвали TensorFlow и объявили о его релизе в ноябре 2015 года.

В TensorFlow были реализованы многие из вышеупомянутых обещаний и сквозной процесс от разработки до развертывания. Фреймворк стал очень популярным. Сейчас у него 164 тысячи звезд на GitHub, и это не предел. Но по мере распространения пользователи справедливо критиковали TensorFlow за недостаточную простоту в использовании. Даже родилась шутка: TensorFlow была создана инженерами Google для инженеров Google, и если вы умеете использовать TensorFlow, то достаточно умны, чтобы работать в Google.

Но Google — не единственная компания, игравшая на этом поле. По правде говоря, даже в 2015 году считалось, что работа с библиотеками глубокого обучения — это не самое приятное занятие. Более того, даже простая установка некоторых из этих фреймворков заставляла людей рвать на себе волосы. (Пользователи Caffe тут? Согласны? Узнали?)

## Keras

В ответ на трудности, с которыми приходилось сталкиваться практикам глубокого обучения, в марте 2015 года Франсуа Шолле (François Chollet) выпустил библиотеку с открытым исходным кодом Keras, изменившую мир. Это решение внезапно сделало глубокое обучение доступным для новичков. Keras имела простой и понятный интерфейс и стала использоваться в других библиотеках глубокого обучения в роли внутренней вычислительной инфраструктуры. Первоначально основанная на Theano, Keras способствовала быстрому созданию прототипов и сокращала количество строк кода. Позднее ее абстракции были распространены на другие фреймворки — Cognitive Toolkit, MXNet, PlaidML и, да, TensorFlow.

## PyTorch

Примерно в то же время, в начале 2016 года, в Facebook был запущен проект PyTorch, инженеры которого столкнулись с ограничениями TensorFlow. Библиотека PyTorch изначально поддерживала конструкции Python и инструменты отладки Python, что обеспечило ей большую гибкость и простоту в использовании и быстро сделало ее фаворитом среди исследователей ИИ. Это вторая по величине сквозная (end-to-end) система глубокого обучения. Также в Facebook была создана библиотека Caffe2, позволяющая использовать модели PyTorch и развертывать их в производственной среде для обслуживания более миллиарда пользователей. В отличие от PyTorch, которая в основном предназначалась для исследований, Caffe2 применялась в основном в продакшене. В 2018 году Caffe2 вошла в состав PyTorch и был создан полноценный фреймворк.

## Постоянно развивающийся ландшафт

Если бы эта история закончилась с появлением простых и удобных библиотек Keras и PyTorch, в подзаголовке этой книги не было бы слова «TensorFlow». В команде TensorFlow осознали, что если они действительно хотят расширить возможности инструмента и демократизировать ИИ, то нужно упростить инструмент. Поэтому объявление об официальном включении Keras в состав TensorFlow стало долгожданной новостью. Это объединение позволило разработчикам использовать Keras для определения моделей и их обучения, а ядро TensorFlow — для организации высокопроизводительного пайплайна обработки данных, включая распределенное обучение и экосистему развертывания. Это

был брак, заключенный на небесах. А еще версия TensorFlow 2.0 (выщенная в 2019 году) включала поддержку собственных конструкций Python и режима «немедленного выполнения» (eager execution), хорошо знакомых по PyTorch.

При наличии нескольких конкурирующих фреймворков неизбежно встает вопрос о переносимости. Представьте, что появилась исследовательская статья, сопровождающаяся примером обученной модели PyTorch. Если вы не используете PyTorch, то для проверки этой модели придется заново реализовать и обучить ее. Разработчикам нравится возможность свободно делиться моделями и не ограничиваться какой-то конкретной экосистемой.

Первое время многие разработчики писали библиотеки для преобразования моделей из одного формата в другой. Это было простое решение, но оно привело к росту количества инструментов преобразования в геометрической прогрессии и нехватке официальной поддержки из-за огромного их числа. Чтобы решить эту проблему, Microsoft, Facebook и другие основные игроки в этой отрасли поддержали создание проекта Open Neural Network Exchange (ONNX), в рамках которого была разработана спецификация общего формата моделей, который получил официальную поддержку в ряде популярных библиотек. Также были созданы инструменты преобразования для библиотек, которые изначально не поддерживали этот формат. В результате разработчики получили возможность обучать модели в одной среде и использовать их для прогнозирования в другой.

Кроме этих фреймворков есть также несколько систем с графическим пользовательским интерфейсом, позволяющих обучать модели, не написав ни строчки кода. Используя прием переноса обучения, они создают обученные модели в нескольких форматах, которые можно использовать для прогнозирования. Используя интерфейсы «наведи и щелкни», даже ваша бабушка сможет обучать нейронные сети.

**Таблица 1.6.** Популярные инструменты с графическим интерфейсом для обучения моделей

Служба	Платформа
Microsoft CustomVision.AI	На веб-основе
Google AutoML	На веб-основе
Clarifai	На веб-основе
IBM Visual Recognition	На веб-основе
Apple Create ML	macOS
NVIDIA DIGITS	Настольные компьютеры
Runway ML	Настольные компьютеры

Так почему же для этой книги в качестве основных фреймворков мы выбрали TensorFlow и Keras? Учитывая доступность огромного объема информации, включая документацию, ответы на вопросы на Stack Overflow, онлайн-курсы, обширное сообщество участников, поддержку платформ и устройств, распространение в отрасли и, да, количество доступных вакансий (в США на 2019 год число вакансий, связанных с TensorFlow, примерно в три раза больше числа вакансий, связанных с PyTorch), можно утверждать, что сейчас TensorFlow и Keras заняли доминирующие позиции. И определенно имело смысл выбрать эту комбинацию. Тем не менее методы, обсуждаемые в книге, можно распространить и на другие библиотеки. Вам не придется тратить много времени на освоение новых фреймворков, если потребуется. Если вы действительно решите уйти в компанию, где используется только PyTorch, то не сомневайтесь и действуйте.

## Оборудование

В 1848 году, когда Джеймс Маршалл обнаружил золото в Калифорнии, эта новость распространилась по США как лесной пожар. Сотни тысяч людей отправились туда на добычу богатств. Это явление стало известно как *Калифорнийская золотая лихорадка*. Первопроходцы смогли отхватить приличный кусок, но опоздавшим повезло меньше. Лихорадка продолжалась много лет. Сможете ли вы угадать, кто за все это время заработал больше всего денег? Производители лопат!

Компании, предлагающие услуги облачных вычислений, и производители оборудования — это производители лопат XXI века. Не верите? Посмотрите, как выросли в цене акции Microsoft и NVIDIA за последнее десятилетие. Единственное отличие настоящего времени от 1848 года — ошеломляющее количество доступных лопат.

При нынешнем разнообразии доступного оборудования важно сделать правильный выбор с учетом ограничений ресурсов, задержек, бюджета, конфиденциальности и юридических требований приложения.

Обычно на этапе прогнозирования пользователь ожидает получить ответ. Это накладывает ограничения на тип оборудования, которое можно использовать, а также на его расположение. Например, Snapchat не может работать в облаке из-за задержек в сети. Кроме того, приложение должно работать практически в масштабе реального времени, чтобы соответствовать ожиданиям пользователя, и определяет минимальные требования к количеству кадров, обрабатываемых в секунду (обычно  $> 15$ ). При этом фотография, выгруженная в Google Photos, не требует немедленной классификации. Задержка в несколько секунд или даже минут вполне допустима.

С другой стороны, обучение занимает намного больше времени, от нескольких минут до часов или даже дней. В зависимости от сценария использования ис-

тинная ценность высокопроизводительного оборудования заключается в возможности ускорить эксперименты и выполнить больше итераций. Для чего-то более серьезного, чем простые нейросети, производительность оборудования может иметь огромное значение. Как правило, графические процессоры ускоряют работу в 10–15 раз по сравнению с обычными процессорами, имеют гораздо более высокую производительность на ватт и сокращают время ожидания завершения эксперимента с нескольких дней до нескольких часов. Эту разницу легко представить, сравнив продолжительность просмотра документального фильма о Гранд-Каньоне (два часа) с продолжительностью поездки в Гранд-Каньон (четыре дня).

Ниже описывается несколько основных категорий оборудования на выбор и то, как они обычно характеризуются (см. также рис. 1.10):

#### *Центральный процессор (Central Processing Unit, CPU)*

Дешевизна, универсальность, низкое быстродействие. Например, Intel Core i9-9900K.

#### *Графический процессор (GPU)*

Высокая пропускная способность, пригодность для пакетной обработки с высокой степенью распараллеливания задач, дороговизна. Например, NVIDIA GeForce RTX 2080 Ti.

#### *Программируемая логическая интегральная схема (Field-Programmable Gate Array, FPGA)*

Высокое быстродействие, низкое энергопотребление, возможность перепрограммирования для нестандартных решений, дороговизна. В числе известных производителей FPGA можно назвать: Xilinx, Lattice Semiconductor, Altera (Intel). Благодаря возможности настройки для обучения любой модели ИИ, большая часть ИИ в Microsoft Bing обучается на интегральных схемах FPGA.

#### *Специализированная интегральная схема (Application-Specific Integrated Circuit, ASIC)*

Специализированная микросхема. Чрезвычайно дорогостоящая в проектировании, но дешевая при массовом производстве. Так же как в фармацевтической промышленности, первое изделие стоит дорого из-за затрат на научно-исследовательские и конструкторские работы. Массовое производство обходится относительно недорого. Вот некоторые конкретные примеры:

#### *TPU*

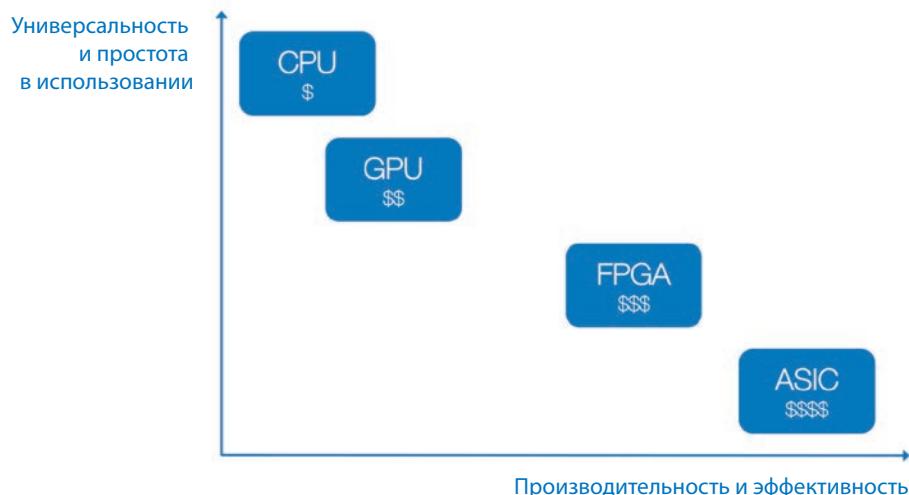
Специализированные интегральные схемы, предназначенные для работы с нейронными сетями, доступны только в Google Cloud.

### Краевой (edge) TPU

Крошечная микросхема, меньше центовой монеты, но дает значительный прирост в производительности инференса.

### Нейронный процессор (Neural Processing Unit, NPU)

Часто используется производителями смартфонов в виде отдельной микросхемы для ускорения прогнозирования с использованием нейронной сети.



**Рис. 1.10.** Сравнение разных типов оборудования по универсальности, производительности и стоимости

Перечислим типичные сценарии использования оборудования каждого типа:

- Начало обучения → CPU.
- Обучение больших сетей → GPU и TPU.
- Прогнозирование на смартфонах → мобильный CPU, GPU, процессор для цифровой обработки сигналов (Digital Signal Processor, DSP), NPU.
- Носимые устройства (например, умные очки, умные часы) → краевые TPU, NPU.
- Встраиваемые реализации ИИ (например, беспилотные летательные аппараты, автономные инвалидные коляски) → ускорители, такие как Google Coral, Intel Movidius с Raspberry Pi, или графические процессоры, такие как NVIDIA Jetson Nano, вплоть до микроконтроллеров (MCU) за 15 долларов для обнаружения активирующих слов в интеллектуальных колонках.

Многие из этих сценариев мы рассмотрим позже.

## Ответственный ИИ

После знакомства с потенциальными возможностями ИИ можно питать большие надежды на расширение наших способностей, повышение продуктивности и получение сверхспособностей.

Но с широкими возможностями приходит большая ответственность.

Насколько ИИ может помочь человечеству, настолько же он может навредить ему, если разрабатывать его без осознания важности и должного внимания (намеренно или нет). Здесь нет вины ИИ – вся ответственность лежит на разработчиках.

Рассмотрим несколько инцидентов последних лет.

«\_\_\_\_\_ якобы может определить, являетесь ли вы террористом, просто проанализировав ваше лицо» (рис. 1.11): *Computer World*, 2016;

«ИИ отправляет людей в тюрьму и... ошибается»: *MIT Tech Review*, 2019;

«\_\_\_\_\_ суперкомпьютер рекомендовал “небезопасные и ошибочные” методы лечения рака, свидетельствуют внутренние документы»: *STAT News*, 2018;

«\_\_\_\_\_ создал инструмент искусственного интеллекта для найма людей, но ему пришлось отказаться от него из-за дискриминации женщин»: *Business Insider*, 2018;

«\_\_\_\_\_ исследование искусственного интеллекта: ведущие системы распознавания объектов отдают предпочтение людям с большими деньгами»: 2019;

«\_\_\_\_\_ изображения чернокожих людей подписывал как “горилла”» *USA Today*, 2015. «Два года спустя \_\_\_\_\_ решили проблему “расистского алгоритма”, удалив подпись “горилла” из классификатора изображений»: *Boing Boing*, 2018;

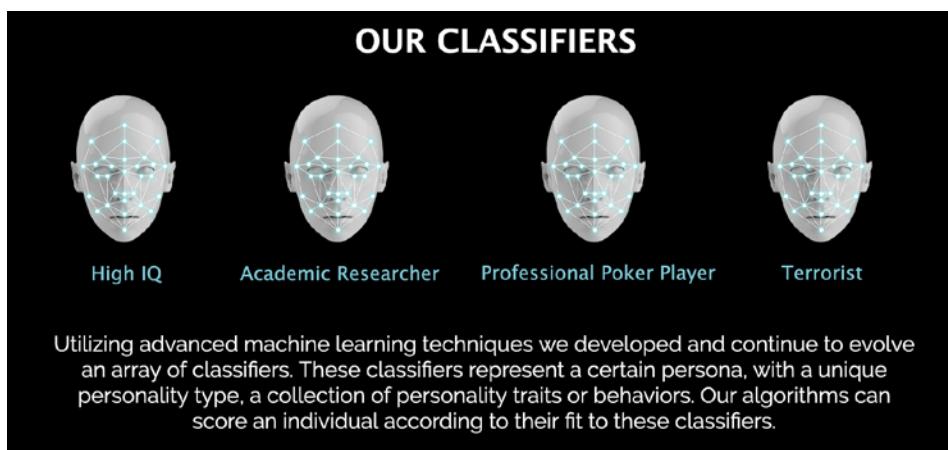
«\_\_\_\_\_ отключили своего нового бота ИИ после того, как пользователи Twitter обучили его расизму»: *TechCrunch*, 2016

«ИИ по ошибке оштрафовал известную предпринимательницу за переход на красный свет, приняв портрет на проезжавшем автобусе за реального человека.»: *Caixin Global*, 2018;

«\_\_\_\_\_ отказался от контракта с Пентагоном в области ИИ после возражений сотрудников делать “бизнес на войне”»: *Washington Post*, 2018;

«Беспилотный автомобиль-убийца Uber “заметил пешехода за шесть секунд до того, как наехал на него и убил”»: *The Sun*, 2018.

Возможно, вы сможете заполнить пробелы в этом списке. Вот несколько вариантов: Amazon, Microsoft, Google, IBM и Uber. Добавьте их, а мы подождем.



**Рис. 1.11.** Стартап, утверждающий, что способен классифицировать людей по строению лица (High IQ — высокий IQ; Academic Researcher — ученый-исследователь; Professional Poker Player — профессиональный игрок в покер; Terrorist — террорист)

Мы опустили названия компаний по одной простой причине: проблема не в конкретном человеке или компании. Эта проблема касается каждого. И хотя ошибки прошлого могут не отражать современного состояния дел, мы должны усвоить урок и постараться не повторять тех же ошибок. И самое приятное, что многие уроки действительно были усвоены.

Мы как разработчики, дизайнеры и архитекторы ИИ обязаны мыслить не только техническими категориями. Ниже приводятся несколько примеров проблем, которые имеют отношение к любой решаемой нами задаче (связанной или не связанной с ИИ). Они не должны отходить на задний план.

## Предвзятость

Часто, осознанно или нет, мы привносим в работу собственную предвзятость — результат влияния множества факторов, включая окружающую среду, воспитание, культурные нормы и даже нашу внутреннюю природу. В конце концов, ИИ и датасеты, на которых он основан, создавались не в вакууме, а людьми со своими предубеждениями. Компьютеры не имеют собственных предубеждений, они лишь отражают и усиливают существующие.

Приведем пример из первых дней существования приложения для YouTube, когда разработчики заметили, что примерно 10 % выгруженных видео перевернуты вверх ногами. Возможно, если бы это число было ниже, скажем, 1 %, то проблему можно было бы списать на ошибки пользователей. Но 10 % — это слишком большое число, чтобы его игнорировать. А вы знаете, кто составляет

10 % населения? Левши! Они держат телефоны во время съемки перевернутыми по сравнению с правшами. Но инженеры YouTube не учли этот аспект во время разработки и тестирования своего мобильного приложения, поэтому все видео выгружались на сервер YouTube в одинаковой ориентации, как для левшей, так и для правшей.

Эту проблему можно было обнаружить гораздо раньше, если бы в команде разработчиков был хоть один левша. Этот простой пример демонстрирует важность многообразия. Праворукость или леворукость — это всего лишь одна маленькая особенность человека. Есть множество других факторов, определяющих индивидуальность. Такие факторы, как пол, цвет кожи, экономическое положение, ограничения по здоровью, страна происхождения, особенности речи или даже что-то столь тривиальное, как длина волос, могут определять результаты, меняющие жизнь человека, в том числе и обработку их алгоритмами.

В гlosсарии Google по машинному обучению (<https://developers.google.com/machine-learning/glossary>) перечислено несколько форм предвзятости, которые могут повлиять на пайплайн машинного обучения. Вот лишь некоторые из них:

### *Предвзятость выборки*

Датасет не отражает реальное распределение и смещен в сторону подмножества категорий. Например, во многих виртуальных помощниках и домашних интеллектуальных колонках некоторые акценты языка представлены чрезмерно, в то время как другие вообще не включены в обучающий датасет, из-за чего большие группы населения мира остаются неохваченными.

Предвзятость выборки также может произойти из-за схожих понятий. Например, Google Translate при переводе таких предложений, как «She is a doctor. He is a nurse» (Она доктор. Он медбрать), на гендерно-нейтральный язык, скажем, турецкий, а затем обратно, меняет пол, как показано на рис. 1.12. Вероятно, это связано с тем, что датасет содержит большую выборку образцов словосочетаний, включающих мужские местоимения со словом «доктор» и женские со словом «медсестра»<sup>1</sup>.

### *Неявная предвзятость*

Этот вид предвзятости обусловлен неявными предположениями, которые мы делаем, когда видим что-то. Обратите внимание на выделенный фрагмент на рис. 1.13. Любой, увидев его, мог бы с большой долей уверенности предположить, что здесь зебра. Фактически, учитывая, насколько сети, обученные на наборе ImageNet, склонны придавать значение текстурам, большинство из

<sup>1</sup> В английском и турецком языках нет деления «медсестра/медбрать», есть одно слово «nurse» (hemşire в турецком), обозначающее сотрудника из числа младшего медицинского персонала. — Примеч. пер.

них классифицируют даже полное изображение как зебру. Но мы-то знаем, что это изображение дивана, обитого тканью с зебровидным рисунком.

The image shows two side-by-side Google Translate interface screenshots. The left screenshot shows the English input "She is a doctor" and "He is a nurse" being translated to Turkish as "O bir doktor" and "O bir hemşire". The right screenshot shows the Turkish input "O bir doktor" and "O bir hemşire" being translated back to English as "He is a doctor" and "She is a nurse". Both screenshots include language selection dropdowns at the top, and audio and copy/paste icons at the bottom.

**Рис. 1.12.** Google Translate отражает внутреннюю смешенность в данных  
(по состоянию на сентябрь 2019)



**Рис. 1.13.** Диван, обитый тканью с зебровидным рисунком; фотография Глена Эдельсона  
(Glen Edelson; <https://www.flickr.com/photos/glenirah/2781264490>)

### *Предвзятость в освещении информации*

Иногда самые громкие голоса в комнате становятся самыми резкими и доминируют в разговоре. При беглом взгляде на Twitter может показаться, что наступил конец света, хотя на самом деле большинство людей не сидят в Twitter и заняты обычными делами. К сожалению, обыденное не продается.

### *Предвзятость внутри/вне группы*

Комментатор из Восточной Азии может взглянуть на изображение статуи Свободы и присвоить ей метки, такие как «Америка» или «США», тогда как комментатор из США может сделать более подробные метки — «Нью-Йорк» или «Остров Свободы». Такова природа человека — видеть свои нюансы, в то время как другие видят что-то более обобщенное. Все это тоже отражается в наших датасетах.

## **Прозрачность и объяснимость**

Представьте, что в конце 1800-х годов Карл Бенц сказал вам, что изобрел четырехколесное устройство, на котором можно перемещаться быстрее, чем на чем-либо еще. Только он понятия не имеет, как это устройство работает. Он только знает, что оно заправляется легковоспламеняющейся жидкостью, которая взрывается внутри и толкает устройство вперед. Что заставляет устройство двигаться? Что заставляет его останавливаться? Почему человек, сидящий внутри, не сгорает? У него нет ответов на эти вопросы. И наверное, вы тогда не захотели бы садиться в эту штуковину.

Именно так можно описать положение дел в сфере ИИ. Раньше в традиционном машинном обучении специалистам по данным приходилось вручную отбирать признаки (переменные для прогнозирования), на основе которых затем обучалась модель. Этот процесс ручного отбора, хотя и обременительный, давал больше контроля и понимания, как выполняется прогноз. В глубоком обучении, напротив, признаки отбираются автоматически. Специалисты по данным создают модели, вводят в них большие объемы данных, и эти модели каким-то образом дают надежные прогнозы, по крайней мере в большинстве случаев. Но человек не знает точно, как работает модель, какие признаки она приняла во внимание, при каких обстоятельствах модель работает эффективно и, что особенно важно, при каких обстоятельствах она не работает. Такой подход, вероятно, приемлем, когда Netflix выдает рекомендации, основываясь на том, что мы просматривали раньше (хотя мы более чем уверены, что где-то в их коде есть строчка `recommendations.append("Stranger Things")`). Но сейчас ИИ используется гораздо шире, не только для подбора рекомендуемых фильмов. Полиция и судебная система начинают полагаться на алгоритмы, чтобы решить, представляет ли кто-то из подозреваемых риск для общества и следует ли задерживать его до суда. На карту поставлены жизни и свобода многих людей. Мы просто не должны отдавать важные решения на откуп бездушному черному ящику. К счастью, эту ситуацию можно изменить, вкладывая средства в «объяснимый ИИ», когда модель не только дает прогнозы, но и показывает, какие факторы заставили ее принять то или иное решение, чтобы выявлять имеющиеся ограничения.

Кроме того, городские власти (например, Нью-Йорка) начинают раскрывать свои алгоритмы перед общественностью, признавая право жителей знать, какие

алгоритмы используются для принятия жизненно важных решений и как они работают. Это позволяет экспертам проводить исследования и аудит, государственным учреждениям повышать уровень компетенции и точнее оценивать каждую добавляемую ими систему и открывает перед гражданами возможность оспорить решения, принятые алгоритмом.

## Воспроизводимость

Научные исследования получают широкое признание сообщества, только когда воспроизводимы. Любой человек, изучающий такую научную работу, может воспроизвести условия теста и получить те же результаты. Не имея возможности воспроизвести результаты, полученные моделью в прошлом, мы не сможем привлечь ее к принятию ответственных решений в будущем. Если нет воспроизводимости, то исследования уязвимы для *подделки результатов (p-hacking)*, когда параметры эксперимента подгоняются под желаемые результаты. Для исследователей чрезвычайно важно тщательно задокументировать условия эксперимента, включая датасеты, контрольные тесты и алгоритмы, и до проведения эксперимента объявить, какую гипотезу они будут проверять. Доверие к институтам и так не особенно высоко, и исследования, оторванные от реальности, но поднявшие шум в СМИ, могут еще больше подорвать это доверие. Традиционно воспроизведение исследовательских экспериментов считалось черной магией из-за отсутствия описания многих деталей реализации. Но ситуация начинает постепенно исправляться благодаря тому, что исследователи переходят на использование общедоступных эталонных тестов (вместо своих, специально подготовленных датасетов) и открывают исходный код, который использовался в исследованиях. Сообщество может использовать этот код, доказать, что он работает, усовершенствовать его и тем самым способствовать новым инновациям.

## Устойчивость

Есть целая область исследований однопиксельных (one-pixel) атак на сверточные нейронные сети. Суть в том, чтобы найти и изменить единственный пиксель в изображении и тем самым заставить сверточную сеть классифицировать измененное изображение как нечто совершенно иное. Например, изменение одного пикселя на изображении яблока может привести к тому, что сеть классифицирует его как собаку. На прогнозы может влиять множество других факторов: шум, условия освещения, ракурс камеры и многое другое, которые не влияют на способность человека правильно воспринимать изображения. Это особенно актуально для беспилотных автомобилей, потому что злоумышленник может добавить на улице какие-то объекты, которые видит машина, и заставить ее принимать неправильные решения. Так, например, исследователи из лаборатории Keen Security Lab в компании Tencent вос-

пользовались уязвимостью в Tesla AutoPilot, разместив на дороге небольшие наклейки и заставив автомобиль выехать на встречную полосу. Чтобы мы могли доверять ИИ, он должен быть устойчивым, способным противостоять постороннему шуму, незначительным отклонениям и преднамеренным манипуляциям.

## Конфиденциальность

В стремлении создать еще более совершенный ИИ компании собирают большие объемы данных. К сожалению, иногда они выходят за рамки дозволенного и собирают информации больше, чем надо для решения поставленной задачи. Компания может полагать, что использует собираемые данные только во благо. Но что, если ее приобретет другая компания, у которой нет этических границ в отношении использования данных? Информация о потребителе может быть использована в целях, выходящих за рамки первоначально намеченных. Кроме того, данные, собранные в одном месте, выглядят привлекательной целью для хакеров, которые крадут персональные данные и продают их на черном рынке преступным организациям. Более того, многие правительства уже зашли слишком далеко, пытаясь отследить каждого человека.

Все это противоречит общепризнанному праву человека на неприкосновенность частной жизни. Потребители хотят знать, какие данные о них собираются, кто имеет к ним доступ, как эти данные используются, какие механизмы для отказа от сбора данных существуют, а также как удалить данные о них, которые уже были собраны.

Как разработчики мы должны со всей ответственностью подходить к сбору данных и спрашивать себя, все ли собираемые данные действительно нужны. Чтобы свести сбор данных к необходимому минимуму, мы могли бы использовать методы машинного обучения с поддержкой конфиденциальности, например федеративное обучение (используется в Google Keyboard), которые позволяют обучать сети на устройствах пользователей без того, чтобы отправлять какие-либо персональные данные на сервер.

Оказывается, что во многих инцидентах из начала этого раздела именно негативный общественный резонанс привел к массовому осознанию этих проблем, вызвал глобальный сдвиг в мышлении и стал причиной стремления поставить ИИ под контроль, чтобы предотвратить повторение чего-то подобного в будущем. Мы должны и впредь требовать от самих себя, ученых, ведущих производителей и политиков отчета за каждую ошибку и быстро их исправлять. Каждое решение и действие могут создать прецедент на десятилетия вперед. Сфера применения ИИ постоянно расширяется, поэтому мы должны объединиться, чтобы поставить сложные вопросы и найти на них ответы, если хотим снизить потенциальный вред и получить максимальную пользу.

## Итоги

В этой главе мы изучили ландшафт удивительного мира ИИ и глубокого обучения. Познакомились с историей развития ИИ от скромных начинаний, через многообещающие периоды и мрачные зимы ИИ и до возрождения в наши дни. Попутно ответили на вопрос, почему на этот раз все складывается иначе. Затем посмотрели, какие компоненты нужны для создания решения глубокого обучения, включая датасеты, архитектуры моделей, фреймворки и оборудование. Теперь мы готовы приступить к исследованиям, которые ждут нас в следующих главах. Надеемся, вам понравится остальная часть книги.

## Часто задаваемые вопросы

1. Я только начинаю изучать эту тему. Можно ли обойтись без больших затрат на покупку мощного оборудования?

К счастью для вас, вы можете обойтись даже простым браузером. Сотрудники Google Colab (рис. 1.14) любезно согласились предоставить вычислительное время мощных графических процессоров бесплатно (до 12 часов за раз). Этого вполне достаточно, чтобы опробовать сценарии из нашего репозитория в GitHub. По мере совершенствования, когда начнете проводить больше экспериментов (особенно если будете заниматься этим профессионально или использовать большие датасеты), вы сможете получить доступ к графическому процессору, арендовав его в облаке (Microsoft Azure, Amazon Web Services, Google Cloud Platform и др.) или купив соответствующее оборудование. Но берегитесь счетов за электричество!

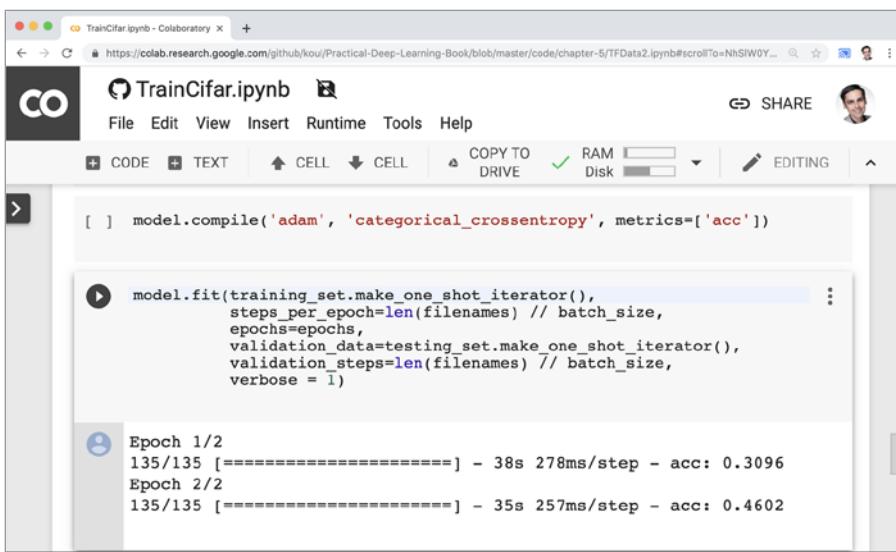
2. Colab – это хорошо, но у меня уже есть мощный компьютер, который я купил, чтобы играть в <вставьте название видеоигры>. Как мне настроить свое локальное окружение?

В идеале желательно использовать Linux, но Windows и macOS тоже подойдут. Для примеров в большинстве случаев вам понадобятся:

- Python 3 и доступ к PIP;
- пакет Tensorflow или Tensorflow-gpu из PIP (версии 2 или выше);
- библиотека Pillow.

Нам нравятся чистота и автономность, поэтому советуем использовать поддержку виртуальных окружений в Python. Устанавливайте пакеты и запускайте сценарии или блокноты (Jupyter Notebook) в виртуальном окружении. Ознакомьтесь с инструкциями по установке.

Если у вас нет графического процессора, то на этом подготовку своего окружения можно считать законченной.



**Рис. 1.14.** Скриншот с блокнотом из GitHub, открытым в Colab (в браузере Chrome)

Если у вас есть графический процессор NVIDIA, то установите соответствующие драйверы, затем библиотеку CUDA, затем библиотеку cuDNN, затем пакет Tensorflow-gpu. Для пользователей Ubuntu есть более простое решение, чем установка всех этих пакетов и библиотек вручную с риском ошибиться, от чего не застрахованы даже самые лучшие из нас: просто установите все нужное одной командой с помощью Lambda Stack (<https://lambdalabs.com/lambda-stack-deep-learning-software>).

Также нужные пакеты можно установить с помощью Anaconda Distribution. Это решение одинаково хорошо подходит для Windows, Mac и Linux.

3. Где найти код примеров из книги?

По адресу <https://github.com/PracticalDL/Practical-Deep-Learning-Book>.

4. Каковы минимальные требования для чтения этой книги?

Степень Ph.D. в математическом анализе, статистическом анализе, вариационных автокодировщиках, исследовании операций и т. д. совершенно *не* требуется для чтения этой книги (признайтесь, что уже напряглись). Для усвоения материала важны базовые навыки программирования, знакомство с Python, здоровое любопытство и чувство юмора. Некоторое понимание особенностей разработки мобильных приложений (на Swift или Kotlin) не будет лишним, но мы постарались сделать наши примеры самодостаточными и довольно простыми для развертывания, чтобы с этим могли справиться даже те, кто раньше никогда не писал мобильных приложений.

5. Какие фреймворки будут использоваться?

Keras + TensorFlow для обучения. Постепенно, глава за главой, мы рассмотрим различные фреймворки для инференса.

6. Стану ли я экспертом, когда закончу читать эту книгу?

Следуя инструкциям, вы овладеете широким кругом знаний и умений — от обучения до инференса и приемов увеличения производительности. Несмотря на то что эта книга в первую очередь посвящена компьютерному зрению, полученные здесь знания можно применить в других областях, например для анализа текста, речи и т. д.

7. Что за кот упоминался выше в этой главе?

Это кот Мехера по кличке Вейдер. В этой книге он будет сниматься в нескольких эпизодах. Не переживайте: он уже подписал согласие на использование его изображений.

8. Как с вами связаться?

Пишите нам по адресу *practicaldlbook@gmail.com*, присылайте свои вопросы, замечания и т. д. Или пишите в Твиттер: @practicaldlbook.

## ГЛАВА 2

---

# Что на картинке: классификация изображений с помощью Keras

Наверняка в литературе по глубокому обучению вы наталкивались на академические объяснения со страшными математическими формулами. Не переживайте! В этой книге мы упростили вхождение в практику глубокого обучения и взяли за основу пример классификации изображений, где всего несколько строк кода.

В этой главе познакомимся с фреймворком Keras, обсудим его место в мире глубокого обучения, а затем используем для классификации нескольких изображений современные классификаторы. Изучим работу этих классификаторов с помощью *тепловых карт* и реализуем забавный проект, где будем классифицировать объекты в видео.

Вспомните раздел «Рецепт идеального решения задачи глубокого обучения» в главе 1, где говорилось, что нужны четыре ингредиента: оборудование, датасет, фреймворк и модель. Давайте посмотрим, какую роль играет каждый из них в этой главе.

- Начнем с простого: *оборудование*. Для запуска примеров из этой главы достаточно даже недорогого ноутбука. Кроме того, код из этой главы можно запустить, открыв блокнот из GitHub в Colab. Почти в два клика.
- На первом этапе мы не будем обучать нейронную сеть, поэтому *датасет* не понадобится (кроме нескольких образцов фотографий для тестирования).
- Следующий компонент — *фреймворк*. В названии этой главы есть слово Keras — так что используем этот фреймворк. Обучать модели в Keras мы будем на протяжении большей части книги.
- Один из подходов к решению задач глубокого обучения — получить датасет, написать модель и обучающий ее код, потратить много времени и энергии (как человеческой, так и электрической) на обучение модели, а затем использовать ее для прогнозирования. Но как мы сказали, мы собираемся облегчить вхождение в глубокое обучение, и поэтому будем использовать *предварительно обученную модель*. В конце концов, исследователи пролили

много крови, пота и слез на обучение и публикацию многих стандартных моделей, которые теперь доступны всем желающим. Используем одну из наиболее известных моделей — ResNet-50, младшую сестру модели ResNet-152, победившей в конкурсе Imagenet Large Scale Visual Recognition Challenge (ILSVRC) в 2015 году.

В этой главе будет некоторый код. Как известно, лучший способ обучения — это обучение на практике. Но, может быть, вам будет интересно познакомиться с теоретическими основами. Они будут рассматриваться в последующих главах — там мы углубимся в изучение сверточных нейронных сетей, взяв за основу эту главу.

### От создателя

Изначально я создавал Keras для собственного использования. На тот момент, в конце 2014 — начале 2015 года, не было надежных и удобных фреймворков глубокого обучения с мощной поддержкой рекуррентных и сверточных сетей. В то время у меня не было мысли демократизировать глубокое обучение, я просто создавал удобный для себя инструмент. Но со временем увидел, что многие осваивают глубокое обучение с помощью Keras и используют его для решения самых разных задач, о существовании которых я даже не подозревал. Я был удивлен и восхищен, и это заставило меня осознать, что глубокое обучение можно применить в гораздо большем количестве областей, чем известно исследователям машинного обучения. Людей, которым эти технологии могли бы быть полезны, оказалось так много, что я решил позаботиться о максимальной доступности глубокого обучения. Единственный способ раскрыть возможности ИИ в полной мере — сделать его широкодоступным. Сейчас интерфейс Keras в TensorFlow 2.0 объединяет возможности глубокого обучения с целым спектром действительно эффективных и удобных рабочих процессов, подходящих для воплощения разных проектов, от исследовательских до прикладных, включая развертывание. С нетерпением жду возможности увидеть, что вы построите с его помощью!

Франсуа Шолле — создатель Keras, специалист по искусственному интеллекту, автор книги «Deep Learning with Python»<sup>1</sup>

## Введение в Keras

Фреймворк Keras появился в 2015 году как более простой в использовании по сравнению с другими библиотеками слой абстракции, открывший новые возможности для быстрого прототипирования. Его появление сделало кривую обучения новичков в глубоком обучении намного менее крутой. В то же время этот фрейм-

---

<sup>1</sup> Шолле Франсуа. Глубокое обучение на Python. СПб.: Питер. — Примеч. пер.

ворк помог экспертам в глубоком обучении повысить продуктивность и позволил быстро повторять эксперименты. Фактически большинство команд-победителей на *Kaggle.com* (где проводятся соревнования по обработке данных) использовали Keras. Наконец, в 2017 году полная реализация Keras была включена непосредственно в TensorFlow, что позволило объединить высокую масштабируемость, производительность и обширную экосистему TensorFlow с простотой Keras. В интернете можно встретить версию Keras для TensorFlow с названием `tf.keras`.

Весь код, который мы напишем в главах 2 и 3, будет использовать Keras, включая стандартные функции для чтения файлов, манипулирования изображениями и т. д. Мы сделаем это в первую очередь для простоты обучения. В главе 5 мы постепенно начнем напрямую использовать другие встроенные и высокопроизводительные функции TensorFlow для большей гибкости и контроля.

## Классификация изображений

С точки зрения непрофессионала, задача классификации изображений отвечает на вопрос: что изображено на этой картинке или фотографии? Например, «здесь изображен объект  $X$  с такой-то вероятностью», где  $X$  — одна из предопределенных категорий объектов. Если вероятность выше минимального порога, то, скорее всего, изображение содержит один или несколько экземпляров  $X$ .

Простой пайплайн классификации изображений состоит из следующих этапов:

1. Загрузка изображения.
2. Масштабирование до предопределенных размеров, например  $224 \times 224$  пикселя.
3. Нормализация значений пикселей в диапазон  $[0,1]$  или  $[-1,1]$ .
4. Выбор предварительно обученной модели.
5. Ввод подготовленного изображения в модель и получение списка категорий с их вероятностями.
6. Вывод нескольких наиболее вероятных категорий.



В репозитории GitHub (<https://github.com/PracticalDL/Practical-Deep-Learning-Book/>) перейдите в папку `code/chapter-2`. Все дальнейшие этапы подробно описаны и реализованы в блокноте Jupyter `1-predict-class.ipynb`.

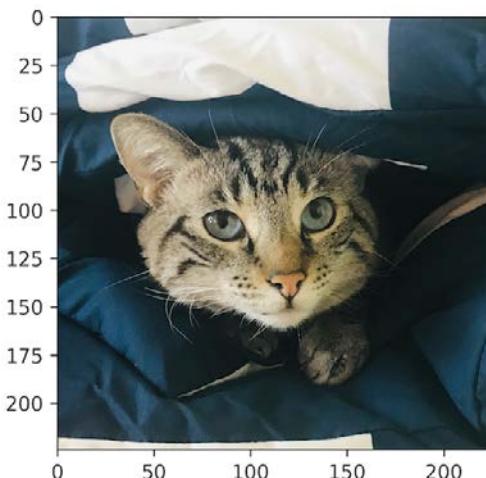
Сначала импортируем нужные модули из пакетов Keras и Python:

```
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
```

```
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
```

Затем загрузим и отобразим изображение, которое будем классифицировать (рис. 2.1):

```
img_path = "../../sample-images/cat.jpg"
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)
plt.show()
```



**Рис. 2.1.** Содержимое входного файла

Да, это кот (хотя имя файла как бы намекает на это). И в идеале наша модель должна распознать кота на этом изображении.

### Краткая справка

Прежде чем углубиться в процесс обработки изображений, было бы неплохо посмотреть, как изображения хранят информацию. В общем случае изображение — это множество пикселей, расположенных в прямоугольной сетке. В зависимости от типа изображения каждый пиксель может содержать от 1 до 4 частей (также известных как *компоненты*, или *каналы*). Эти компоненты определяют интенсивности красного, зеленого и синего цветов (Red Green Blue, RGB). Обычно каждый компонент представлен 8-битным числом, поэтому их значения находятся в диапазоне от 0 до 255 (то есть  $2^8 - 1$ ).

Прежде чем ввести изображение в Keras, нужно преобразовать его в стандартный формат. Это связано с тем, что предварительно обученные модели ожидают получить на входе данные определенного размера. В нашем случае мы должны привести изображение к стандартному размеру  $224 \times 224$  пикселя.

Большинство моделей глубокого обучения предполагают ввод целого пакета изображений. Но как быть, если есть только одно изображение? Создать пакет с одним изображением! Для этого нужно создать массив с единственным элементом. Это можно также представить как увеличение количества измерений — с трех (три канала) до четырех (дополнительное измерение для длины самого массива).

Если что-то непонятно, представьте такую ситуацию: объект, представляющий пакет из 64 изображений с размерами  $224 \times 224$  пикселя, каждый из которых содержит три канала (RGB), будет иметь форму  $64 \times 224 \times 224 \times 3$ . В следующем коде мы используем единственное изображение  $224 \times 224 \times 3$  и создаем пакет, включающий только это изображение, увеличивая количество измерений с трех до четырех. Этот пакет будет иметь форму  $1 \times 224 \times 224 \times 3$ :

```
img_array = image.img_to_array(img)
img_batch = np.expand_dims(img_array, axis=0) # Увеличить количество измерений
```

Модели МО работают эффективнее, когда на вход подаются данные в соответствующем диапазоне. Обычно используются диапазоны  $[0,1]$  и  $[-1,1]$ . Учитывая, что значения пикселей в изображении находятся в диапазоне от 0 до 255, вызов функции `preprocess_input` из Keras для входных изображений нормализует каждый пиксель до стандартного диапазона. *Нормализация*, или *масштабирование признаков*, — это один из основных этапов предварительной обработки изображений, цель которого сделать изображения пригодными для глубокого обучения.

Теперь перейдем к модели. Мы будем использовать *сверточную нейронную сеть* (*Convolutional Neural Network, CNN*) под названием ResNet-50. Самый первый вопрос, на который нужно ответить: где взять эту модель? Можно поискать в интернете и найти что-нибудь совместимое с нашим фреймворком глубокого обучения (Keras). Но у нас нет на это времени. К счастью, Keras настолько упрощает работу, что позволяет получить эту модель одним вызовом функции. При первом вызове эта функция загрузит модель с удаленного сервера и сохранит ее в локальном кэше:

```
model = tf.keras.applications.resnet50.ResNet50()
```

Когда эта модель используется в режиме прогнозирования, ее результаты включают оценки вероятности принадлежности к каждому классу. В Keras также имеется функция `decode_predictions`, которая сообщает вероятность принадлежности объектов в изображении к каждой категории.

Посмотрим, как реализован процесс классификации:

```
def classify(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    model = tf.keras.applications.resnet50.ResNet50()
    img_array = image.img_to_array(img)
    img_batch = np.expand_dims(img_array, axis=0)
    img_preprocessed = preprocess_input(img_batch)
    prediction = model.predict(img_preprocessed)
    print(decode_predictions(prediction, top=3)[0])

classify("../sample-images/cat.jpg")
[('n02123045', 'tabby', 0.50009364),
 ('n02124075', 'Egyptian_cat', 0.21690978),
 ('n02123159', 'tiger_cat', 0.2061722)]
```

В прогнозах для этого изображения перечислены различные породы кошек. Почему модель не ответила просто «кошка»? Дело в том, что модель ResNet-50 была обучена на детализированном датасете со множеством категорий, среди которых нет обобщающей категории «кошка». Мы еще вернемся к этому датасету, но перед этим загрузим еще один образец (рис. 2.2):

```
img_path = '../sample-images/dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)
plt.show()
```



**Рис. 2.2.** Содержимое файла dog.jpg

И снова вызовем нашу удобную функцию классификации:

```
classify("../sample-images/dog.jpg")
[(u'n02113186', u'Cardigan', 0.809839),
```

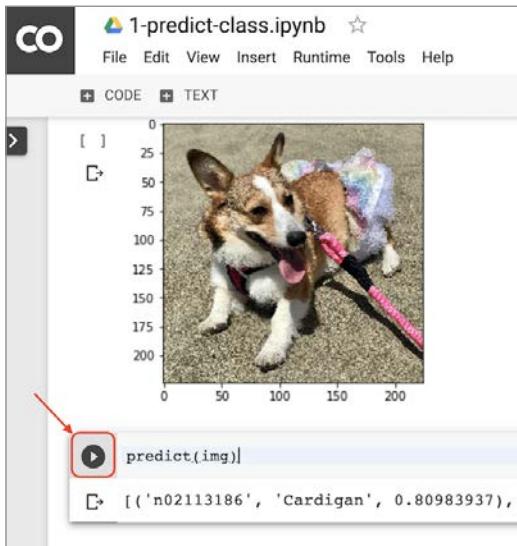
```
(u'n02113023', u'Pembroke', 0.17665945),  
(u'n02110806', u'basenji', 0.0042166105)]
```

Как и ожидалось, мы получили список разных пород собак (а не одну общую категорию «собака»). Для тех, кто не знаком с породой корги (Corgi), заметим, что слово «corgi» на валлийском языке означает буквально «карликовая собака». Кардиган (Cardigan) и пемброк (Pembroke) – это субпороды семейства корги, очень похожие друг на друга. Неудивительно, что наша модель тоже так думает.

Обратите внимание на прогнозируемую вероятность каждой категории. Обычно ответом считается прогноз с наибольшей вероятностью. Иногда ответом могут считаться все значения выше заранее установленного порога. Если в примере с собакой установить порог 0,5, то ответом будет кардиган (Cardigan).



Запустить код из этой главы можно в интерактивном режиме в браузере с помощью Google Colab. Просто отыщите ссылку «Run on Colab» (запустить в Colab) в начале любого блокнота, доступного на GitHub, с которым хотите поэкспериментировать, затем щелкните на кнопке «Run Cell» (выполнить код в ячейке), чтобы запустить код в этой ячейке (рис. 2.3).



**Рис. 2.3.** Запуск блокнота в Google Colab с использованием браузера

## Исследование модели

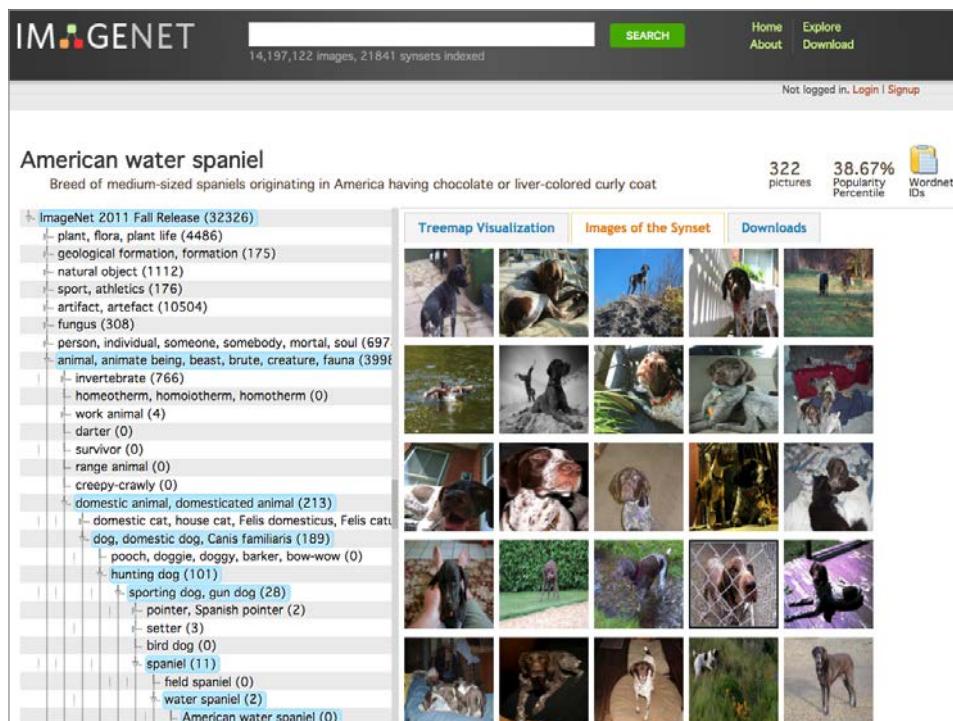
Модель вернула нам прогноз, отлично! Но как она получила этот прогноз? Чтобы понять это, нужно ответить на несколько вопросов:

- На каком датасете обучалась модель?
- Можно ли использовать другие модели? Насколько они эффективны? Где их взять?
- Почему модель вернула именно такой прогноз?

Далее мы подробно ответим на каждый из этих вопросов.

## Датасет ImageNet

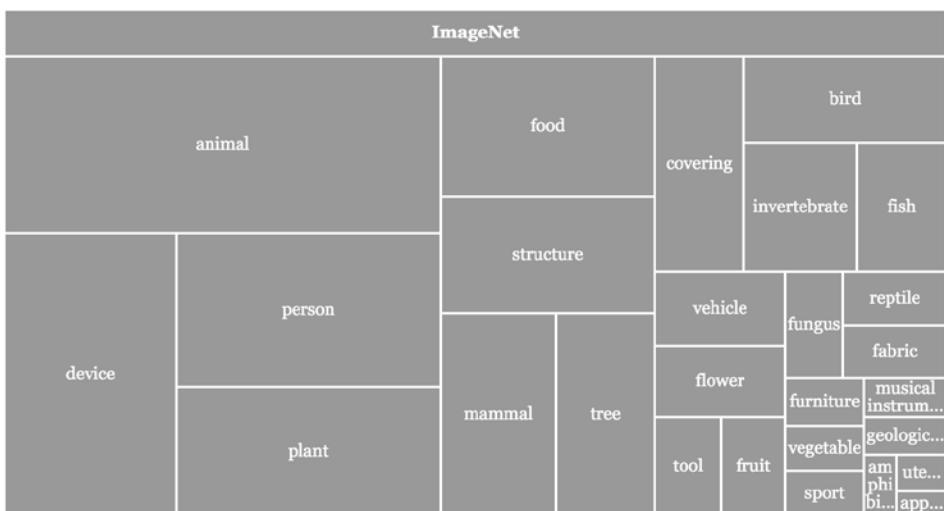
Рассмотрим датасет ImageNet, на котором была обучена модель ResNet-50. Как следует из названия, ImageNet (<http://www.image-net.org/>) — это совокупность изображений, то есть датасет с изображениями, как показано на рис. 2.4. Он имеет иерархическую организацию (по аналогии с WordNet), когда родительская категория включает все изображения из всех подкатегорий, относящихся к этой родительской категории. Например, в категории «animal» (животное) есть изображения рыб, птиц, млекопитающих, беспозвоночных и т. д. Каждая категория имеет несколько дочерних подкатегорий, те в свою очередь имеют свои



**Рис. 2.4.** Категории и подкатегории в датасете ImageNet

подкатегории и т. д. Например, категория «American water spaniel» (американский водяной спаниель) находится на восьмом уровне от корневой категории. Категория «dog» (собака) содержит 189 подкатегорий на пяти иерархических уровнях.

Чтобы визуально представить разнообразие высокоуровневых сущностей в наборе ImageNet, мы разработали древовидную диаграмму (рис. 2.5). На ней также показаны относительные доли различных категорий, составляющих датасет ImageNet.



**Рис. 2.5.** Карта датасета ImageNet и составляющие его категории

Датасет ImageNet послужил основой для знаменитого проекта ILSVRC, начатого в 2010 году для оценки прогресса в области компьютерного зрения и стимулирования исследователей к инновациям в решении задач, включая классификацию объектов. Как отмечалось в главе 1, создание набора ImageNet привело к тому, что точность классификации с каждым годом все больше и больше увеличивалась. В самом начале доля ошибок составляла почти 30 %. А сейчас всего 2,2 %, то есть теперь ИИ справляется с задачей классификации изображений лучше, чем средний человек. Этот датасет и задача классификации считаются главными причинами последних достижений в области компьютерного зрения.

Подождите, а как так получилось, что ИИ показывает более высокую точность, чем человек? Если датасет создавался людьми, то люди должны распознавать изображения со 100%-ной точностью, разве не так? Дело в том, что датасет создавался экспертами, и каждое изображение проверялось несколькими людьми. После создания набора исследователь из Стэнфорда (а теперь из Tesla) Андрей

Карпатый (Andrej Karpathy) попытался выяснить, насколько хорошо справится обычный человек с классификацией изображений из набора ImageNet-1000. В этом эксперименте точность достигла уровня 94,9 %, что намного меньше ожидаемых 100 %. Андрей целую неделю внимательно просматривал 1500 изображений, тратя примерно одну минуту на классификацию каждого из них. Но как он допустил ошибки в 5,1 % случаев? Тому есть несколько причин.

### *Детальность классификации*

Мало кто сможет отличить сибирского хаски от аляскинского маламута. Тот, кто действительно хорошо разбирается в породах собак, сможет отличить их друг от друга, потому что видит более тонкие детали различия. Как оказалось, нейронные сети способны фиксировать такие тонкости намного точнее, чем люди.

### *Малая известность категорий*

Не все знают, что есть 120 пород собак, которые к тому же делятся на 1000 разновидностей. Но ИИ знает. Ведь его специально этому обучили.



Похожие эксперименты проводились с наборами речевых данных, такими как Switchboard, в ходе которых выяснилось, что доля ошибок транскрипции речи составили 5,1 % (по совпадению то же самое число, что и в ImageNet). Понятно, что человеческие возможности имеют предел, и ИИ постепенно начинает превосходить нас.

Еще одна причина такого быстрого совершенствования в том, что исследователи могли свободно делиться моделями, обученными на таких датасетах, как ImageNet. В следующем разделе поближе познакомимся с этой возможностью.

## **Зоопарк моделей**

Зоопарк моделей — это место, куда организации или частные лица могут выгружать созданные ими модели для повторного использования другими. Эти модели могут быть обучены с помощью любых фреймворков (например, Keras, TensorFlow, MXNet) для любых задач (классификация, обнаружение и т. д.) или на любом датасете (например, ImageNet, Street View House Numbers (SVHN)).

Традиция создания зоопарков моделей началась с Caffe — одного из первых фреймворков глубокого обучения, разработанного в Калифорнийском университете в Беркли. Обучение модели с нуля на датасете с миллионами изображений требует много времени (до нескольких недель) и больших вычислительных мощностей, в том числе с привлечением GPU, что делает эту задачу весьма сложной. Сообщество признало это узким местом, и организации, участвовавшие в конкурсе ImageNet, разместили свои обученные модели на сайте Caffe. Вскоре этому примеру последовали и другие фреймворки.

Приступая к новому проекту, советуем сначала выяснить, есть ли модель, которая решает аналогичную задачу и была обучена на похожем датасете.

Зоопарк моделей в Keras (<https://keras.io/applications/>) включает обширный набор моделей с разными архитектурами, обученных с Keras на датасете ImageNet. Мы перечислили их в табл. 2.1.

**Таблица 2.1.** Архитектурные детали некоторых моделей, предварительно обученных на ImageNet

Модель	Размер	Топ-1 точности	Топ-5 точности	Параметров	Глубина
VGG16	528 Мбайт	0,713	0,901	138 357 544	23
VGG19	549 Мбайт	0,713	0,9	143 667 240	26
ResNet-50	98 Мбайт	0,749	0,921	25 636 712	50
ResNet-101	171 Мбайт	0,764	0,928	44 707 176	101
ResNet-152	232 Мбайт	0,766	0,931	60 419 944	152
Inceptionv3	92 Мбайт	0,779	0,937	23 851 784	159
InceptionResNetv2	215 Мбайт	0,803	0,953	55 873 736	572
NASNetMobile	23 Мбайт	0,744	0,919	5 326 716	–
NASNetLarge	343 Мбайт	0,825	0,96	88 949 818	–
MobileNet	16 Мбайт	0,704	0,895	4 253 864	88
MobileNetV2	14 Мбайт	0,713	0,901	3 538 984	88

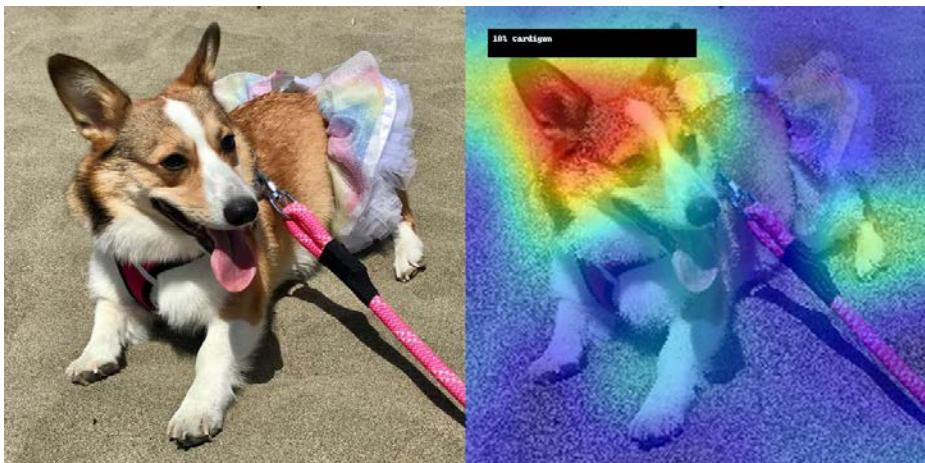
Столбец «Топ-1 точности» показывает, сколько раз ответ с наибольшей оценкой был правильным, а столбец «Топ-5 точности» — сколько раз хотя бы один из пяти ответов с наибольшими оценками был правильным. «Глубина» сети — это количество слоев. В столбце «Параметров» указывается количество параметров, то есть количество отдельных весов в модели: чем больше параметров, тем «тяжелее» модель и тем медленнее она вычисляет прогноз. В этой книге часто используется модель ResNet-50 (наиболее распространенная архитектура, упоминаемая в исследовательских статьях как одна из моделей с высокой точностью), а также семейство моделей MobileNet (дающее хороший баланс между скоростью, размером и точностью).

## Карты активации классов

Термин «значимые детали изображения», хорошо известный в исследованиях пользовательского восприятия, обозначает вопрос: на какую часть изображения пользователи обращают внимание? Для ответа на него исследователи фиксируют движение глаз пользователя и выводят результат в виде тепловых

карт. Например, крупный жирный шрифт или лица людей обычно привлекают больше внимания, чем фон. Нетрудно догадаться, насколько полезны такие тепловые карты для дизайнеров и маркетологов, которые могут адаптировать контент и привлечь внимание пользователя. Но разве не интересно узнать, на какие части изображения обращает внимание нейронная сеть? Именно этим и займемся.

В нашем эксперименте мы будем накладывать *карту активации классов* (или, в просторечии, *тепловую карту*) на видео, чтобы узнать, на какие детали сеть обращает внимание. Тепловая карта сообщит примерно следующее: «на этом изображении вот эти пиксели ответственны за предсказание класса “собака”», где «собака» — это категория, получившая наибольшую прогнозную оценку. «Горячие» пиксели на карте будут представлены более теплыми цветами, такими как красный, оранжевый, желтый, а «холодные» пиксели — оттенками синего. Чем «горячее» пиксель, тем мощнее сигнал, который он дает для предсказания. Изображение на рис. 2.6 дает более ясную картину.



**Рис. 2.6.** Исходное изображение собаки и то же изображение с наложенной тепловой картой

В репозитории с примерами для книги перейдите в папку `code/chapter-2`. Там будет блокнот `2-class-activation-map-on-video.ipynb`, который описывает следующие шаги.

Прежде всего, нужно установить `keras-vis` с помощью `pip` (в дополнение к библиотекам, упоминавшимся выше):

```
$ pip install keras-vis --user
```

Затем запустить сценарий визуализации с единственным изображением, чтобы сгенерировать тепловую карту:

```
$ python visualization.py --process image --path ../sample-images/dog.jpg
```

Он создаст файл *dog-output.jpg*, на котором бок о бок размещены исходное изображение и его тепловая карта. Как видно на рис. 2.6, правая половина изображения показывает «горячие области» вместе с правильным предсказанием «Cardigan» (кардиган, то есть вельш-корги).

Следующий шаг — наложение тепловой карты на кадры в видео. Для этого понадобится FFmpeg, мультимедийный фреймворк с открытым исходным кодом. Бинарный файл с инструкциями по установке для вашей ОС можно найти на сайте [www.ffmpeg.org](http://www.ffmpeg.org).

Используем ffmpeg, чтобы разбить видео на отдельные кадры (следующие с частотой 25 кадров в секунду), а затем обработаем каждый кадр с помощью скрипта визуализации. Создадим каталог для хранения кадров и передадим его имя команде ffmpeg:

```
$ mkdir kitchen  
$ ffmpeg -i video/kitchen-input.mov -vf fps=25 kitchen/thumb%04d.jpg  
        -hide_banner
```

После этого запустим скрипт визуализации, передав ему путь к каталогу с кадрами, полученными на предыдущем шаге:

```
$ python visualization.py --process video --path kitchen/
```

В результате появится новый каталог *kitchen-output*, содержащий тепловые карты для всех кадров в исходном каталоге *kitchen*.

В заключение соберем видео из полученных изображений с помощью ffmpeg:

```
$ ffmpeg -framerate 25 -i kitchen-output/result-%04d.jpg kitchen-output.mp4
```

Отлично! Теперь исходное видео расположено рядом с копией, на которую наложена тепловая карта. Этот полезный прием можно использовать, например, чтобы выяснить, насколько качественно обучилась модель и не обнаружила ли она случайные артефакты во время обучения.

Представьте себе создание тепловых карт для анализа сильных и слабых сторон модели или сторонней предварительно обученной модели.

Проведите этот эксперимент самостоятельно, сняв видео на камеру смартфона и использовав вышеупомянутые сценарии. Не забудьте разместить видео в Twitter с тегом @PracticalDLBook.



Тепловые карты — отличный способ визуально выявить предвзятость данных. Качество прогнозов модели во многом зависит от данных, на которых она обучена. Если в данных есть предвзятость, это отразится на прогнозах. Прекрасным примером (хотя, скорее всего, это лишь городская легенда) может служить попытка армии США использовать нейронные сети для обнаружения вражеских танков, замаскированных среди деревьев<sup>1</sup>. Исследователи, создававшие модель, сделали фотографии, на половине которых были замаскированные танки, а на другой половине — только деревья. Обученная модель показала 100%-ную точность. Повод для праздника? К сожалению, во время полевых испытаний в армии США модель работала очень плохо — не лучше чем наугад. Расследование показало, что фотографии с танками были сделаны в пасмурные дни, а фотографии без танков — в ясные и солнечные. В результате нейросеть стала искать небо вместо танков. Если бы исследователи визуализировали внимание модели с помощью тепловых карт, то обнаружили бы эту проблему практически сразу.

Собирая данные, следует проявлять особую осторожность, чтобы избежать потенциальной предвзятости, которая может отрицательно сказаться на обучении модели. Например, собирая изображения для будущего классификатора продуктов питания, нужно убедиться, что другие артефакты, такие как тарелки и прочая посуда, не будут восприниматься моделью как значимые детали. Иначе какое-нибудь блюдо может быть классифицировано как чау-мейн (рагу по-китайски) всего лишь из-за наличия палочек. Еще один термин, определяющий подобные ситуации, — «совместная встречааемость» (*cooccurrence*). На изображениях с едой очень часто бывают столовые приборы. Поэтому следите, чтобы такие артефакты не просачивались в обучающие наборы классификаторов.

## Итоги

В этой главе мы получили представление о вселенной глубокого обучения с Keras. Это простой в использовании и в то же время очень мощный фреймворк, который мы продолжим использовать в следующих нескольких главах. Теперь вы знаете, что зачастую нет нужды собирать миллионы изображений и использовать мощные графические процессоры для обучения модели, потому что можно взять предварительно обученную. Познакомившись чуть ближе с наборами данных, вы узнали, какие категории могут предсказывать модели, обученные на этих наборах. Вы также узнали, как искать готовые модели в зоопарках моделей, имеющихся для большинства фреймворков.

В главе 3 мы покажем, как настроить предварительно обученную модель для прогнозирования классов, которых нет в первоначальном обучающем наборе. И мы снова обойдемся без миллиона изображений и большого количества аппаратных ресурсов для обучения классификатора.

---

<sup>1</sup> Eliezer Yudkowsky, «Artificial Intelligence as a Positive and Negative Factor in Global Risk».

## 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.

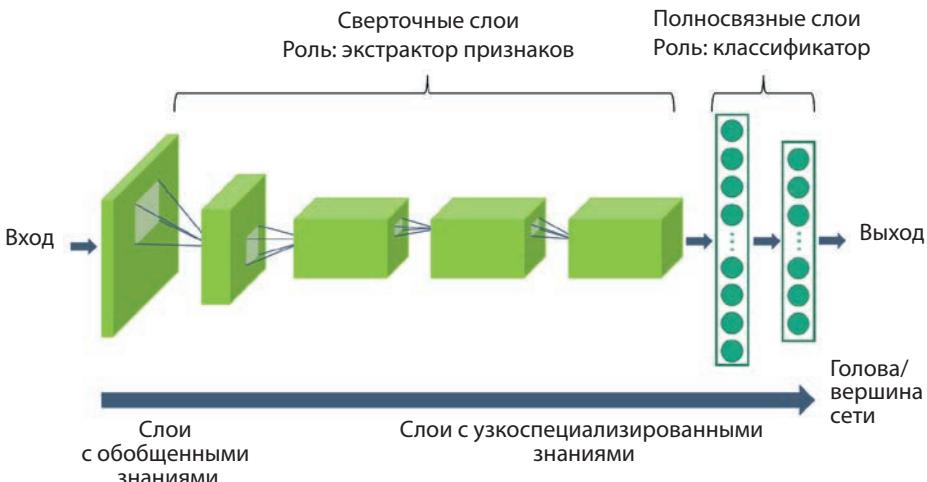
В главе 2, например, мы видели, что готовая модель ResNet-50, обученная на ImageNet, определяет породы кошек и собак, а также распознает изображения из тысячи других категорий. Чтобы создать простой классификатор, распознавающий обобщенные категории «кошка» и «собака» (а не конкретные породы), мы можем 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

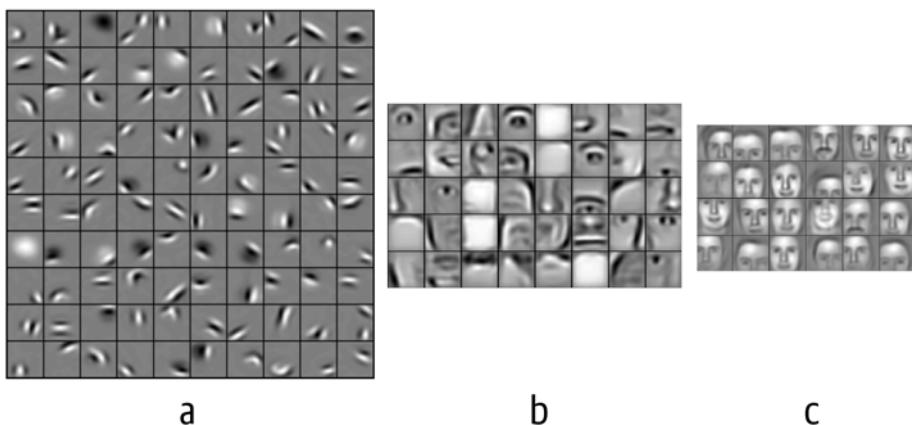
Сеть работает примерно так: изображение вводится в сверточную сеть и проходит через последовательность слоев, каждый из которых выполняет математическую операцию и передает результат следующему слою. На выходе получается список классов изображенных объектов с их вероятностями. Например: мяч = 65 %, трава = 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.



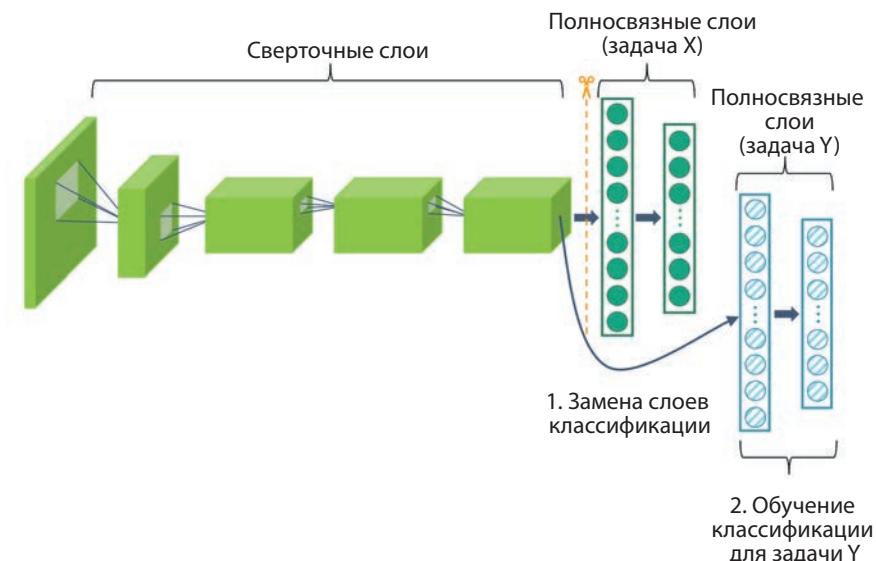
**Рис. 3.3.** (a) Активации низкого уровня, за ними следуют (b) активации среднего уровня и (c) активации высшего уровня (изображение взято из статьи «Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations», Lee (Ли) и др., ICML 2009)

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.

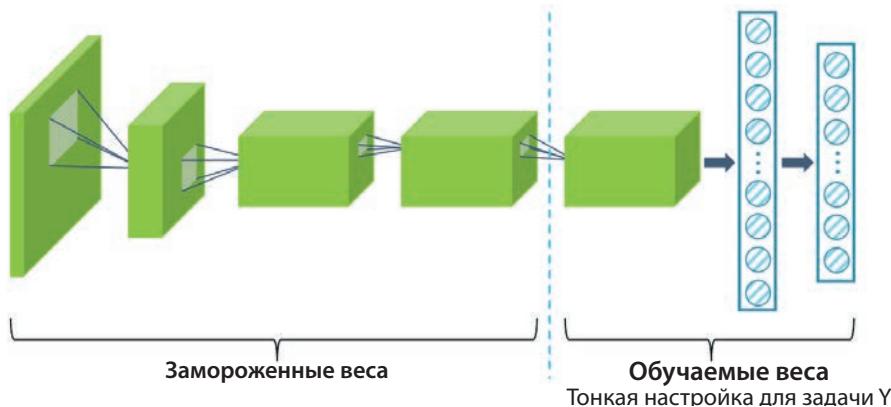


**Рис. 3.4.** Общая схема переноса обучения

## 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 размогжаются 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 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

	<b>Большое сходство датасетов</b>	<b>Малое сходство датасетов</b>
<b>Большой объем обучающих данных</b>	Тонкая настройка всех слоев	Обучение с нуля или тонкая настройка всех слоев
<b>Малый объем обучающих данных</b>	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.

Смысль 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.)<sup>1</sup>. 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

<sup>1</sup> ЕГЭ в России. — Примеч. пер.

задания, оценивать уровень своих знаний и корректировать свой учебный план. Наконец, он приходит на финальный стандартизованный 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.

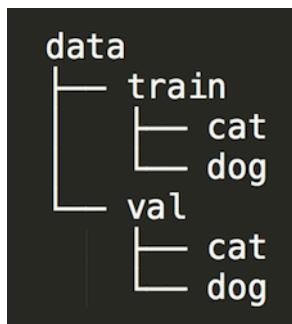
Модель учится на обучающих данных и использует проверочный набор для оценки качества своих прогнозов. Практики МО воспринимают этот процесс как систему с обратной связью, способствующую постоянному улучшению их моделей, подобно тому как студенты улучшают свою подготовку с помощью контрольных и самостоятельных 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 префиксами «`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 tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense, Dropout,
GlobalAveragePooling2D
from tensorflow.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  $1 - P(\text{кошка})$ . 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 «недостатка памяти». Обычно размеры пакетов выбираются как степени двойки. В большинстве задач лучше начать с 64, а затем можем поиграть с ним, увеличивая или уменьшая.

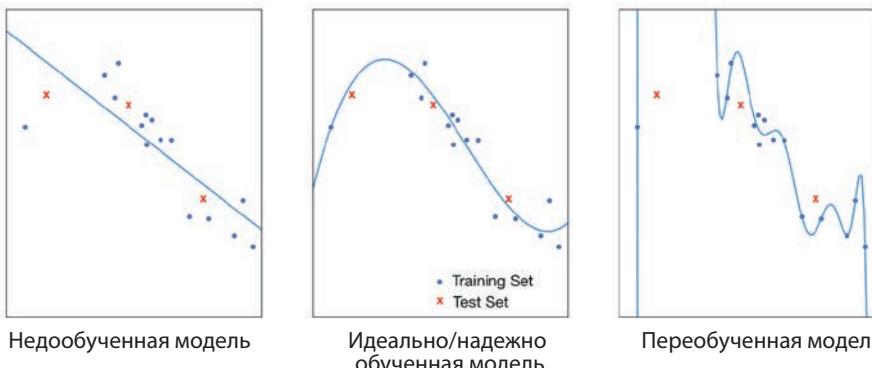
## Аугментация данных

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 высохшее 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*

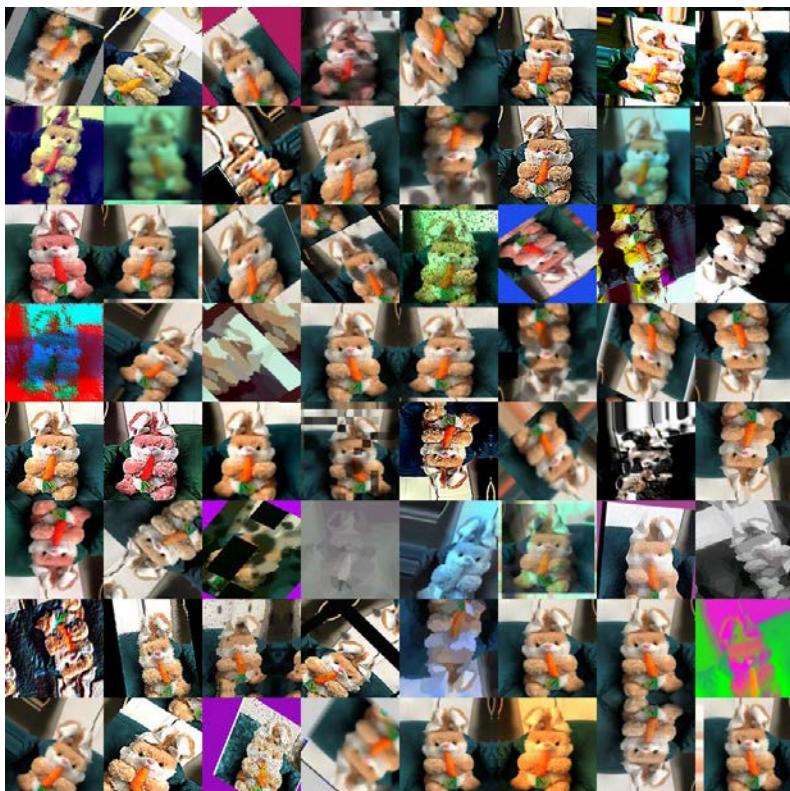
Изображения можно немного сместить влево или вправо.

### *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 ( <code>class_mode</code> )	Функция потерь (loss)	Функция активации в последнем слое (activation)
1 или 2 класс	<code>binary</code>	<code>binary_crossentropy</code>	<code>sigmoid</code>
Многоклассовая, единственная метка	<code>categorical</code>	<code>categorical_crossentropy</code>	<code>softmax</code>
Многоклассовая, несколько меток	<code>categorical</code>	<code>binary_crossentropy</code>	<code>sigmoid</code>

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?

Прежде чем перейти к ответам на эти вопросы, сделаем прогнозы для всех изображений в проверочном датасете. Настроим пайплайн:

```
# Переменные
IMG_WIDTH, IMG_HEIGHT = 224, 224
VALIDATION_DATA_DIR = 'data/val_data/'
VALIDATION_BATCH_SIZE = 64

# Генераторы данных
validation_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)
validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DATA_DIR,
    target_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=VALIDATION_BATCH_SIZE,
    shuffle=False,
    class_mode='categorical')
ground_truth = validation_generator.classes
```

Затем получим прогнозы:

```
predictions = model.predict_generator(validation_generator)
```

Чтобы упростить анализ, создадим словарь, в котором для каждого изображения сохраним индекс и его истинный класс (ожидаемый прогноз):

```
# prediction_table - это словарь (dict) с индексом, прогнозом и истинным
# классом (ground truth)
prediction_table = {}
for index, val in enumerate(predictions):
    # получить индекс argmax
    index_of_highest_probability = np.argmax(val)
    value_of_highest_probability = val[index_of_highest_probability]
    prediction_table[index] = [value_of_highest_probability,
                               index_of_highest_probability,
                               ground_truth[index]]
assert len(predictions) == len(ground_truth) == len(prediction_table)
```

Далее приводится типовой код, который мы будем использовать в этой книге.

Вот скрипт вспомогательной функции для поиска изображений с наибольшим/наименьшим значением вероятности для данной категории (функция определена в репозитории GitHub (<https://github.com/PracticalDL/Practical-Deep-Learning-Book/>) с примерами для книги в папке code/chapter-3). Используем еще одну вспомогательную функцию — `display()` — для вывода изображений в виде сетки на экране:

```
def display(sorted_indices, message):
    similar_image_paths = []
    distances = []
    for name, value in sorted_indices:
        [probability, predicted_index, gt] = value
```

```

similar_image_paths.append(VALIDATION_DATA_DIR + fnames[name])
distances.append(probability)
plot_images(similar_image_paths, distances, message)

```

А теперь самое интересное!

Какие изображения уверенно классифицируются как собаки? Найдем изображения с высокими значениями вероятности (то есть близкими к 1,0; см. рис. 3.9) и с предсказанным классом «собака» (то есть 1):

```

# Прогнозы относительно класса 'dog' с наибольшими значениями вероятности
indices = get_images_with_sorted_probabilities(prediction_table,
get_highest_probability=True, label=1, number_of_items=10,
only_false_predictions=False)
message = 'Images with the highest probability of containing dogs'
display(indices[:10], message)

```



**Рис. 3.9.** Изображения, которые модель уверенно распознала как изображения собак

На этих изображениях действительно собаки. Одна из причин столь высокой прогнозируемой вероятности, возможно, заключается в наличии нескольких собак на изображении, а также в четкости и однозначности изображений. А теперь найдем изображения, опознанные моделью как собаки, но с малой долей уверенности (рис. 3.10):

```

# Прогнозы относительно класса 'dog' с невысокими значениями вероятности
indices = get_images_with_sorted_probabilities(prediction_table,
get_highest_probability=False, label=1, number_of_items=10,
only_false_predictions=False)
message = 'Images with the lowest probability of containing dogs'
display(indices[:10], message)

```



**Рис. 3.10.** Изображения, которые модель неуверенно распознала как изображения собак

Отметим, что это изображения, которые классификатор с неуверенностью определил как изображения собак. Большинство этих прогнозов находятся в переходной точке (то есть с вероятностью 0,5 или чуть выше), что и говорит о неуверенности. Имейте в виду: чтобы отнести изображение к классу «кошка», вероятность должна быть чуть ниже, около 0,49. По сравнению с предыдущими результатами, животные на этих изображениях часто меньше и менее четкие. И такие изображения часто приводят к неверным прогнозам — только 2 из 10 изображений в этом примере были опознаны правильно. Один из возможных способов добиться большего — провести обучение с большим набором изображений.

Если вы переживаете за эти ошибочные попытки классификации, то не стоит. Мы легко можем повысить точность классификации, установив более высокий порог принятия решения классификатором, например 0,75. Если классификатор не уверен в категории изображения, его результат будет отброшен. Как подобрать оптимальный порог, поговорим в главе 5.

Ошибочные предсказания появляются, когда изображение распознается классификатором с низкой долей уверенности (то есть когда вероятность близка к 0,5 в задаче с двумя классами). Но хотелось бы также исключить ошибки, когда классификатор действительно уверен в своем выборе. Проверим, какие изображения кошек были уверенно опознаны классификатором как собаки (рис. 3.11):

```
# Ошибочные прогнозы относительно класса 'dog'
indices = get_images_with_sorted_probabilities(prediction_table,
get_highest_probability=True, label=1, number_of_items=10,
only_false_predictions=True)
message = 'Images of cats with the highest probability of containing dogs'
display(indices[:10], message)
```



**Рис. 3.11.** Изображения кошек, которые уверенно были распознаны как собаки

Хм... Оказывается, на половине этих изображений и кошки, и собаки, и наш классификатор правильно отнес их к категории «собака», потому что собаки на этих изображениях крупнее кошек. Так что здесь ошибка не в классификаторе, а в данных. Это обычное дело для больших датасетов. Другая половина неправильно классифицированных изображений содержит нечеткие и относительно мелкие объекты (хотя в идеале хотелось бы иметь простое снижение оценки достоверности прогноза для этих трудно идентифицируемых изображений).

Зададим те же вопросы в отношении классификации кошек. Какие изображения уверенно классифицируются как кошки (рис. 3.12)?

```
# Прогнозы относительно класса 'cat' с наибольшими значениями вероятности
indices = get_images_with_sorted_probabilities(prediction_table,
get_highest_probability=True, label=0, number_of_items=10,
only_false_predictions=False)
message = 'Images with the highest probability of containing cats'
display(indices[:10], message)
```

Интересно отметить, что на многих из них есть несколько кошек. Это подтверждает предыдущую гипотезу о том, что наличие на изображении нескольких кошек, а также четкость и однозначность изображений могут обеспечить высокую вероятность. А теперь найдем изображения, опознанные моделью как кошки, но с небольшой долей уверенности (рис. 3.13).

```
# Прогнозы относительно класса 'cat' с невысокими значениями вероятности
indices = get_images_with_sorted_probabilities(prediction_table,
get_highest_probability=False, label=0, number_of_items=10,
only_false_predictions=False)
message = 'Images with the lowest probability of containing cats'
display(indices[:10], message)
```



**Рис. 3.12.** Изображения, которые модель уверенно распознала как изображения кошек



**Рис. 3.13.** Изображения, которые модель неуверенно распознала как изображения кошек

Как было замечено выше, ключевые объекты на этих изображениях мелкие, некоторые изображения довольно нечеткие, а некоторые слишком контрастные или объект слишком яркий, что не соответствует большинству обучающих изображений. Например, распознавание восьмой (dog.6680) и десятой (dog.1625) фотографий на рис. 3.13 затруднило использование вспышки. На шестой фото-

графии собака находится перед диваном того же цвета. На двух изображениях — клетки.

И наконец, какие изображения собак были уверенно опознаны классификатором как кошки (рис. 3.14)?

```
# Ошибочные прогнозы относительно класса 'cat'
indices = get_images_with_sorted_probabilities(prediction_table,
get_highest_probability=True, label=0, number_of_items=10,
only_false_predictions=True)
message = 'Images of dogs with the highest probability of containing cats'
display(indices[:10], message)
```



**Рис. 3.14.** Изображения собак, которые уверенно были распознаны как кошки

Количество таких ошибочных прогнозов хотелось бы уменьшить. В некоторых случаях ошибка очевидна, но есть и такие, которые сбивают с толку по понятным причинам. Шестое изображение (dog.4334) на рис. 3.14, по всей видимости, было ошибочно отнесено к категории собак. На седьмом и десятом изображении ключевой объект сливаются с фоном. На первом и десятом изображениях нет деталей, что способствовало ошибочной их классификации с большой долей уверенности. А на некоторых изображениях, как на втором и четвертом, собаки слишком мелкие.

Проанализировав полученные результаты, заключаем, что неправильные прогнозы обычно вызываются низкой освещенностью, нечетким и трудно различимым фоном, отсутствием деталей и малой занимаемой площадью на изображении.

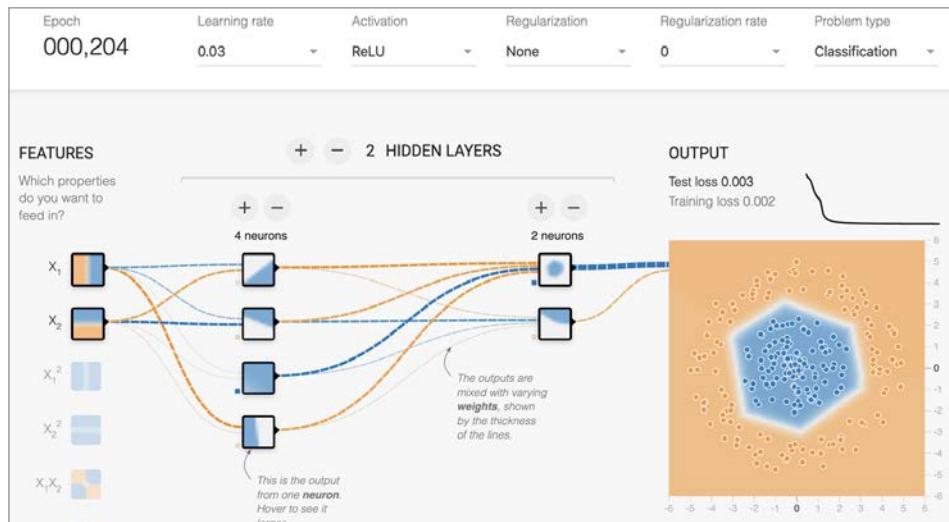
Анализ прогнозов помогает понять, чему научилась модель, что у нее плохо получается, и определить дальнейшие шаги для повышения ее прогнозирующей силы. Увеличение размеров обучающих образцов и обогащение набора поможет улучшить классификацию. Важно отметить, что обучение модели на реальных изображениях (похожих на те, которые будут использоваться приложением) значительно повышает ее точность. В главе 5 сделаем наш классификатор более надежным.

## Для дальнейшего чтения

На нашем сайте есть гайд, который поможет погрузиться в тему нейронных сетей и CNN. Там вы найдете рекомендуемые ресурсы: видеолекции, блоги и, что более интересно, интерактивные визуальные инструменты, позволяющие поиграть с различными сценариями в браузере без необходимости установки каких-либо пакетов.

Если вы только начинаете изучать глубокое обучение, рекомендуем изучить этот гайд, чтобы укрепить базовые знания. Он охватывает теорию, знание которой потребуется, чтобы развить интуицию для решения будущих задач. Для реализации нейронных сетей мы используем TensorFlow Playground (рис. 3.15), а для реализации сверточных сетей – ConvNetJS (рис. 3.16).

В приложении вы найдете краткое руководство, которое обобщает сведения о сверточных сетях.



**Рис. 3.15.** Создание нейронной сети в TensorFlow Playground

**Instantiate a Network and Trainer**

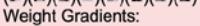
```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'conv', sx:5, filters:8, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:3, stride:3});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```

**change network**

**Network Visualization**

<b>input (24x24x1)</b> max activation: 1, min: 0 max gradient: 0.00537, min: -0.00722	<b>Activations:</b>  <b>Activation Gradients:</b> 
<b>conv (24x24x8)</b> filter size 5x5x1, stride 1 max activation: 3.04579, min: -3.36012 max gradient: 0.00328, min: -0.00289 parameters: 8x5x5x1+8 = 208	<b>Activations:</b>  <b>Activation Gradients:</b>  <b>Weights:</b>  <b>Weight Gradients:</b> 
<b>relu (24x24x8)</b> max activation: 3.04579, min: 0 max gradient: 0.00328, min: -0.00289	<b>Activations:</b> 

**Рис. 3.16.** Определение сверточной сети и визуализация вывода каждого слоя во время обучения в ConvNetJS

## Итоги

В этой главе мы познакомились с идеей переноса обучения. Мы использовали предварительно обученную модель, чтобы создать классификатор кошек и собак, написали менее 30 строк кода и использовали для обучения всего 500 изображений. Это позволило достигнуть высочайшей точности за несколько минут. Написав этот код, мы также развенчали миф о том, что для обучения хорошего классификатора нужны миллионы изображений и мощные GPU (хотя это не было бы лишним). Теперь с этими навыками вы наконец сможете ответить на извечный вопрос: что здесь изображено?

В последующих главах эти знания помогут глубже понять сверточные сети и поднять точность модели на новый уровень.

## ГЛАВА 4

---

# Создание механизма обратного поиска изображений. Эмбеддинги

Боб только что купил новый дом и хочет обставить его современной мебелью. Он постоянно листает каталоги и ходит по мебельным салонам, но пока не нашел то, что ему понравилось бы. Но вот однажды он видит диван мечты — необычный современный белый диван Г-образной формы в приемной офиса. Теперь он знает, чего хочет, но вот беда — он не знает, где его купить. На диване нет этикетки с названием и артикулом. Не помогло и обращение к менеджеру. Поэтому Боб сделал несколько снимков с разных ракурсов, чтобы показать их продавцам в местных мебельных магазинах, но ему не повезло: никому этот диван не был знаком. Поиск в интернете по таким ключевым словам, как «белый Г-образный», «современный диван», дал ему тысячи результатов, но очень далекие от искомого.

Алиса, прознав про неудачу Боба, спросила его: «Почему бы тебе не найти его через поиск по изображению?» Боб загрузил фотографии в Google и Bing и быстро нашел похожий диван в интернет-магазине. Взяв более четкое изображение с сайта, он продолжил поиски и нашел другие магазины, предлагающие такой же диван, но по более низкой цене. Через несколько минут Боб заказал диван своей мечты!

*Обратный поиск по изображению* (или, как его называют на техническом языке, *поиск по экземплярам, instance retrieval*) позволяет разработчикам и исследователям реализовать сценарии, выходящие за рамки простого поиска по ключевым словам. Во всех случаях — от обнаружения визуально похожих объектов на Pinterest до рекомендаций на Spotify и поиска продуктов на Amazon с помощью камеры — используется аналогичный класс технологий. Такие сайты, как TinEye, предупреждают фотографов о нарушении авторских прав, когда их фотографии без согласия публикуются в интернете. Даже функция распознавания лиц в некоторых системах безопасности использует аналогичную идею для установления личности человека.

Самое замечательное, что, обладая нужными знаниями, вы за несколько часов сможете самостоятельно создать действующие аналоги многих из этих продуктов. Давайте займемся этим прямо сейчас!

Вот что мы сделаем в этой главе:

1. Реализуем извлечение признаков и поиск сходств в датасетах Caltech101 и Caltech256.
2. Узнаем, как масштабировать свои разработки до уровня больших датасетов (с миллиардами изображений).
3. Повысим точность системы и оптимизируем ее.
4. Проанализируем конкретные примеры и посмотрим, как описанные здесь идеи используются в популярных продуктах.

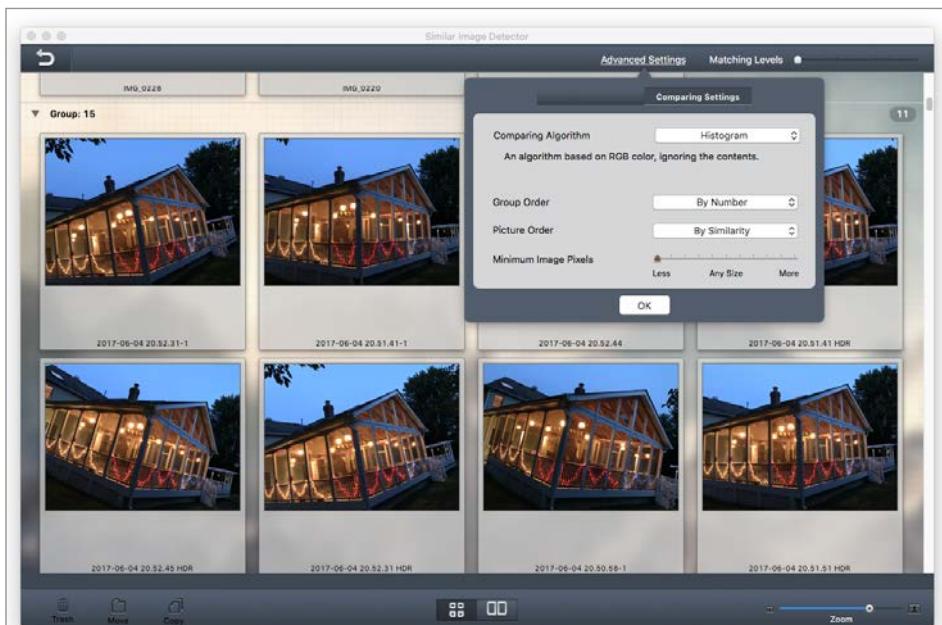
## Сходство изображений

Первый и, пожалуй, самый главный вопрос: как определить сходство двух изображений?

Есть несколько решений. Первое — сравнить отдельные участки этих двух изображений. Оно находит одинаковые или почти одинаковые изображения (которые могли быть просто по-разному обрезаны), но даже небольшой поворот одного из них приведет к несходству. Сохраняя хеши участков, можно искать дубликаты изображения. Один из вариантов использования этого решения — выявление плагиата фотографий.

Другой подход — построить RGB-гистограммы и сравнить их. Это позволит найти почти похожие фотографии, снятые в одинаковых условиях, не имеющие существенных отличий в содержимом. Например, как показано на рис. 4.1, этот метод используется в ПО, предназначенном для поиска серий похожих фотографий на жестком диске. Пользуясь таким ПО, можно выбрать лучший снимок и удалить остальные. Конечно, с ростом датасета вероятность ложных срабатываний возрастает. Еще один недостаток этого решения — небольшие изменения цвета, оттенка или баланса белого затрудняют распознавание.

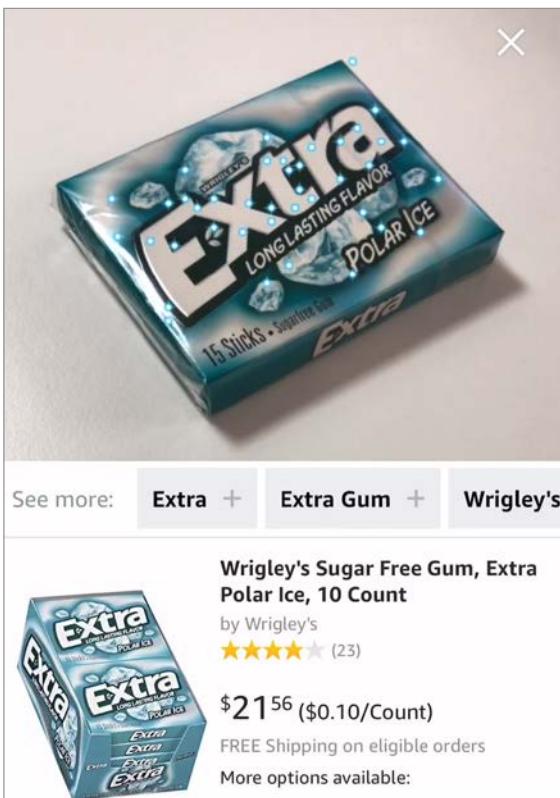
Более надежное традиционное решение, основанное на компьютерном зрении, состоит в том, чтобы выявить характерные визуальные признаки на границе объектов с помощью алгоритмов Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF) и Oriented FAST and Rotated BRIEF (ORB), а затем сравнить количество одинаковых признаков, общих для двух фотографий. Это поможет перейти с обобщенного уровня анализа всего изображения на более надежный уровень анализа объектов. Такое решение отлично подходит для изображений объектов с неизменными формами, имеющих меньше вариаций, как, например, коробки с хлопьями, которые почти всегда выглядят одинаково,



**Рис. 4.1.** Поиск похожих изображений по RGB-гистограммам в программе «Similar Image Detector»

но оно почти непригодно для сравнения изображений изменчивых объектов, таких как люди и животные, которые могут принимать разные позы. Для примера посмотрите, какие признаки используются приложением Amazon при поиске товаров с помощью камеры. Они отмечаются синими точками (рис. 4.2). Обнаружив достаточное количество признаков, приложение отправляет их на серверы Amazon для поиска информации о продукте.

Еще одно решение — определить категорию изображения (например, «диван») с помощью глубокого обучения, а затем отыскать другие изображения в той же категории. Это эквивалентно извлечению метаданных из изображения для последующего индексирования и использования в типичном поиске по текстовым запросам. Такое решение легко масштабировать, сохраняя метаданные в поисковых системах с открытым исходным кодом вроде Elasticsearch. Многие сайты электронной коммерции показывают рекомендации, опираясь на теги, извлеченные из изображения, когда выполняют внутренний поиск по запросу. Однако, извлекая теги, мы теряем значительный объем информации, такой как цвет, поза, положение объектов относительно друг друга и т. д. Кроме того, основным недостатком этого подхода является необходимость иметь огромный объем размеченных данных для обучения классификатора, который потом будет классифицировать новые изображения. И каждый раз, когда понадобится добавить новую категорию, модель нужно будет обучить повторно.



**Рис. 4.2.** Сканер товаров в приложении Amazon с отмеченными визуальными признаками

Поскольку наша цель — поиск среди миллионов изображений, нужен некоторый способ обобщения информации, содержащейся в миллионах пикселей изображения, в более компактное представление (скажем, имеющее несколько тысяч признаков), чтобы при сравнении обобщенные представления похожих изображений оказывались близко друг к другу, а разных — далеко.

В этом помогут глубокие нейронные сети. Как мы видели в главах 2 и 3, сверточная нейронная сеть принимает входное изображение и преобразует его в тысячечемерный вектор признаков, которые затем передаются на вход классификатора для определения наиболее вероятных категорий, которым может принадлежать изображение (например, «собака» или «кошка»). *Векторы признаков* (их называют также *эмбеддингами*, или *ключевыми признаками*) — это, по сути, набор из нескольких тысяч числовых значений с плавающей запятой. Прохождение информации через слои свертки и объединения в сверточной сети — это фактически процесс редукции, или упрощения, осуществляющий фильтрацию информации в изображении и оставляющий наиболее важные

и заметные компоненты, которые, в свою очередь, образуют эмбеддинги. В процессе обучения сверточной сети эти значения формируются таким образом, что элементы, принадлежащие к одному классу, оказываются на небольшом евклидовом расстоянии друг от друга (евклидово расстояние — это просто квадратный корень из суммы квадратов разностей соответствующих значений), а элементы из разных классов разделяются большими расстояниями. Это важное свойство помогает решить множество задач, в которых нельзя использовать классификатор, особенно в задачах обучения без учителя, в которых отсутствуют размеченные данные.



Иdealный способ реализовать поиск похожих изображений — использовать *перенос обучения* (*transfer learning*). Например, можно пропустить сравниваемые изображения через предварительно обученную сверточную нейронную сеть, такую как ResNet-50, извлечь признаки, а затем использовать некоторую метрику, такую как евклидово расстояние, для вычисления величины ошибки.

Но хватит разговоров, давайте писать код!

## Извлечение признаков

Изображение стоит тысячи слов признаков.

В этом разделе мы поэкспериментируем с идеей извлечения признаков, использовав сначала датасет Caltech 101 (131 Мбайт, примерно 9000 изображений), а затем — Caltech 256 (1,2 Гбайт, примерно 30 000 изображений). Набор Caltech 101 включает примерно 9000 изображений, разбитых на 101 категорию, от 40 до 800 изображений в каждой. Важно отметить, что есть 102-я категория с названием «BACKGROUND\_Google», включающая случайные изображения, не входящие в первые 101 категорию, которую нужно удалить перед началом экспериментов. Помните, что весь код, который мы пишем, также доступен в репозитории GitHub (<https://github.com/PracticalDL/Practical-Deep-Learning-Book/>).

Загрузим датасет:

```
$ wget http://www.vision.caltech.edu/Image_Datasets/Caltech101/101_
ObjectCategories.tar.gz
$ tar -xvf 101_ObjectCategories.tar.gz
$ mv 101_ObjectCategories caltech101
$ rm -rf caltech101/BACKGROUND_Google
```

Теперь импортируем все необходимые модули:

```
import numpy as np
from numpy.linalg import norm
```

```
import pickle
from tqdm import tqdm, tqdm_notebook
import os
import time
from tf.keras.preprocessing import image
from tf.keras.applications.resnet50 import ResNet50, preprocess_input
```

Загрузим модель ResNet-50 без верхних слоев, составляющих классификатор, чтобы на выходе получить только ключевые признаки. Затем определим функцию, принимающую на вход путь к изображению, загружающую его, изменяющую размеры изображения до поддерживаемых моделью ResNet-50, извлекающую признаки и затем нормализующую их:

```
model = ResNet50(weights='imagenet', include_top=False,
                   input_shape=(224, 224, 3))
def extract_features(img_path, model):
    input_shape = (224, 224, 3)
    img = image.load_img(img_path, target_size=(
        input_shape[0], input_shape[1]))
    img_array = image.img_to_array(img)
    expanded_img_array = np.expand_dims(img_array, axis=0)
    preprocessed_img = preprocess_input(expanded_img_array)
    features = model.predict(preprocessed_img)
    flattened_features = features.flatten()
    normalized_features = flattened_features / norm(flattened_features)
    return normalized_features
```



Функция в предыдущем примере является ключевой, и мы будем использовать ее почти везде, где потребуется извлекать признаки в Keras.

Вот и все! Давайте посмотрим, какой размер имеют векторы признаков, извлекаемые моделью:

```
features = extract_features(' ../../sample_images/cat.jpg', model)
print(len(features))
> 2048
```

Из заданного изображения модель ResNet-50 извлекла 2048 признаков. Каждый признак — это число с плавающей запятой в диапазоне от 0 до 1.



Если вы решите использовать другую модель, обученную или дообученную на датасете, отличном от ImageNet, переопределите шаг `preprocess_input(img)` соответствующим образом. Средние значения, используемые в функции, относятся к датасету ImageNet. Каждая модель в Keras имеет свою собственную функцию предварительной обработки, поэтому используйте правильную.

Теперь извлечем признаки для всего датасета. Сначала получим все имена файлов с помощью следующей вспомогательной функции, которая выполняет рекурсивный поиск всех файлов изображений (по расширениям) в каталоге:

```
extensions = ['.jpg', '.JPG', '.jpeg', '.JPEG', '.png', '.PNG']
def get_file_list(root_dir):
    file_list = []
    counter = 1
    for root, directories, filenames in os.walk(root_dir):
        for filename in filenames:
            if any(ext in filename for ext in extensions):
                file_list.append(os.path.join(root, filename))
                counter += 1
    return file_list
```

Для этого определим путь к нашему датасету и вызовем функцию:

```
# путь к датасету
root_dir = '../..../datasets/caltech101'
filenames = sorted(get_file_list(root_dir))
```

Далее определим переменную для хранения всех признаков, выполним обход всех имен файлов в датасете, извлечем их признаки и добавим в эту переменную:

```
feature_list = []
for i in tqdm_notebook(range(len(filenames))):
    feature_list.append(extract_features(filenames[i], model))
```

Если использовать только CPU, эта операция займет около часа. На GPU на ее выполнение потребуется всего несколько минут.



Чтобы видеть ход операции, воспользуйтесь удобным инструментом `tqdm`, который показывает индикатор выполнения (рис. 4.3), скорость обработки, а также прошедшее и ожидаемое время завершения. Для этого заключите итерируемый объект в вызов `tqdm`; например `tqdm(range(10))`. Вариант этой функции в Jupyter Notebook называется `tqdm_notebook`.



**Рис. 4.3.** Индикатор выполнения, созданный с помощью `tqdm_notebook`

Наконец, запишем полученные признаки в файл хранилища, чтобы использовать их в будущем без необходимости извлекать вновь:

```
pickle.dump(feature_list, open('data/features-caltech101-resnet.pickle', 'wb'))
pickle.dump(filenames, open('data/filenames-caltech101.pickle', 'wb'))
```

Вот и все! Мы закончили с извлечением признаков.

## Поиск сходств

Наша цель — по имеющейся фотографии найти в нашем датасете другую, похожую, фотографию. Начнем с загрузки предварительно извлеченных признаков:

```
filenames = pickle.load(open('data/filenames-caltech101.pickle', 'rb'))
feature_list = pickle.load(open('data/features-caltech101-resnet.pickle', 'rb'))
```

Воспользуемся библиотекой машинного обучения `scikit-learn` для Python и с ее помощью отыщем *ближайших соседей* по заданному набору признаков; то есть признаков, представляющих заданное изображение. Для этого обучим модель методом ближайших соседей, используя алгоритм полного перебора для поиска ближайших пяти соседей на основе евклидова расстояния (чтобы установить библиотеку `scikit-learn`, выполните команду `pip3 install sklearn`):

```
from sklearn.neighbors import NearestNeighbors
neighbors = NearestNeighbors(n_neighbors=5, algorithm='brute',
                             metric='euclidean').fit(feature_list)
distances, indices = neighbors.kneighbors([feature_list[0]])
```

Теперь у нас есть индексы и расстояния до пяти соседей, ближайших к самому первому набору признаков (представляющему первое изображение). Обратите внимание, насколько быстро выполняется первый шаг — шаг обучения. В отличие от большинства моделей, обучение которых может занять от нескольких минут до часов на больших датасетах, создание экземпляра модели ближайшего соседа происходит мгновенно, потому что во время обучения выполняется не так много операций. Это один из примеров *ленивого*, или *отложенного, обучения*, потому что основные вычисления перекладываются на этап классификации.

Получив индексы, посмотрим, какие изображения скрываются за ними. Сначала выберем изображение с индексом 0, которое использовалось как эталонное:

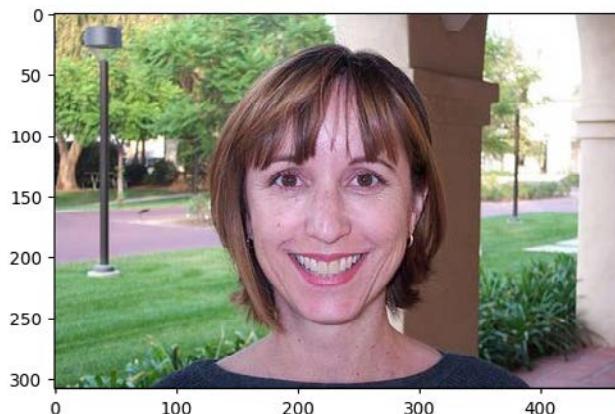
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline # Показывать графики как ячейки внутри блокнота Jupyter
Notebook
plt.imshow(mpimg.imread(filenames[0]))
```

Результат показан на рис. 4.4.

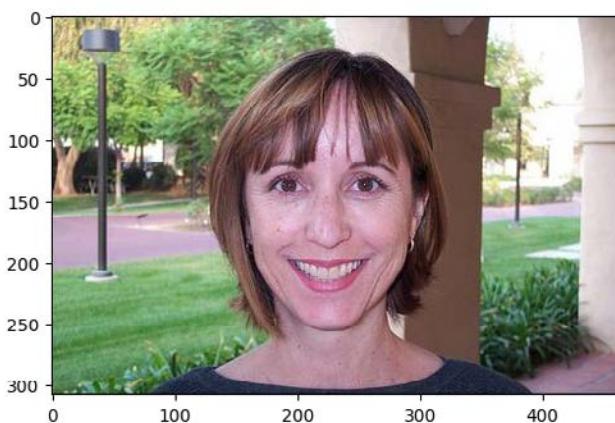
Посмотрим на первого ближайшего соседа.

```
plt.imshow(mpimg.imread(filenames[indices[0]]))
```

Результат показан на рис. 4.5.



**Рис. 4.4.** Искомое изображение из Caltech-101



**Рис. 4.5.** Первый ближайший сосед к искомому изображению

Подождите, разве это не дубликат? На самом деле первым ближайшим индексом является само искомое изображение, потому что оно есть в датасете:

```
for i in range(5):
    print(distances[0][i])
```

0.0  
0.8285478  
0.849847  
0.8529018

Наше подозрение подтверждается нулевым расстоянием до первого результата. А теперь посмотрим, как выглядит настоящий первый ближайший сосед:

```
plt.imshow(mpimg.imread(filenames[indices[1]]))
```

Взгляните на результат, показанный на рис. 4.6.



**Рис. 4.6.** Второй ближайший сосед к искомому изображению

Определенно это изображение похоже на искомое. Оно соответствует идеи сходства — относится к той же категории (лица), на нем изображен человек того же пола и присутствует похожий фон с колоннами и растительностью. На самом деле это один и тот же человек!

В будущем нам еще не раз потребуется выводить похожие изображения, поэтому мы написали вспомогательную функцию `plot_images()` для вывода искомого изображения и его ближайших соседей. Теперь вызовем эту функцию, чтобы увидеть ближайших соседей для шести случайно выбранных изображений. Обратите внимание, что при каждом запуске следующего фрагмента кода он будет выводить разные изображения (рис. 4.7), потому что каждый раз он случайно выбирает индекс искомого изображения.

```
for i in range(6):
    random_image_index = random.randint(0,num_images)
    distances, indices = neighbors.kneighbors([featureList[random_image_index]])
    # отбросить первого ближайшего соседа, так как это то же самое изображение
    similar_image_paths = [filenames[random_image_index]] +
                          [filenames[indices[0][i]] for i in range(1,4)]
    plot_images(similar_image_paths, distances[0])
```

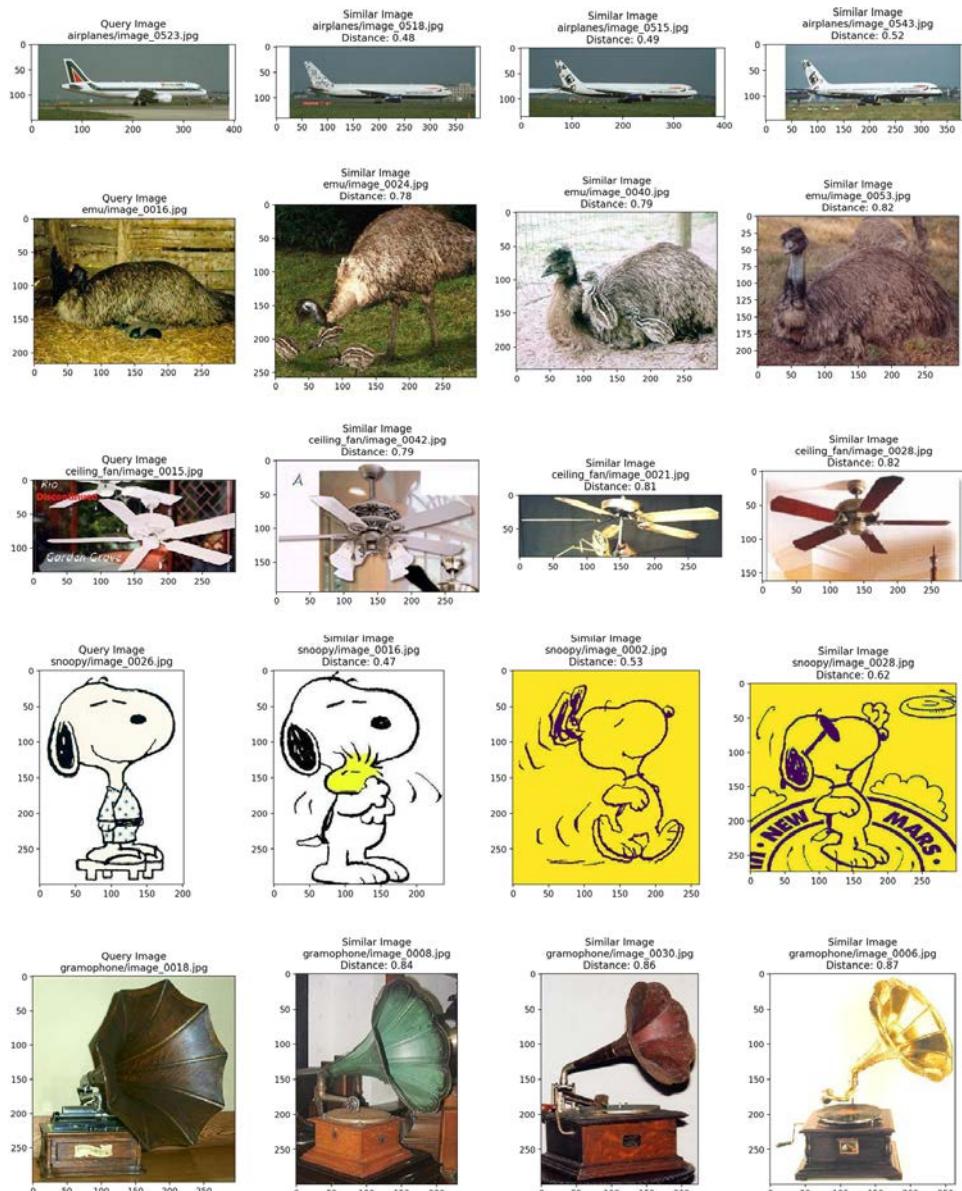


Рис. 4.7. Ближайшие соседи для нескольких случайно выбранных изображений

# Визуализация кластеров изображений с помощью t-SNE

Сделаем еще шаг и визуализируем весь датасет!

Для этого придется уменьшить размерность векторов признаков, потому что невозможно изобразить 2048-мерный вектор (количество признаков) в двумерной плоскости (на листе бумаги). Алгоритм t-распределенного стохастического эмбеддинга соседей (t-distributed Stochastic Neighbor Embedding, t-SNE) уменьшает размерность многомерного вектора признаков до двух измерений, обеспечивая возможность обзора датасета с высоты птичьего полета, что помогает увидеть кластеры и близлежащие изображения. Алгоритм t-SNE трудно масштабировать для обработки больших датасетов, поэтому рекомендуется сначала уменьшить размерность с помощью метода главных компонент (Principal Component Analysis, PCA), а затем использовать t-SNE:

```
# Применить метод главных компонент к векторам признаков
num_feature_dimensions=100 # Задать число признаков
pca = PCA(n_components = num_feature_dimensions)
pca.fit(featureList)
feature_list_compressed = pca.transform(featureList)

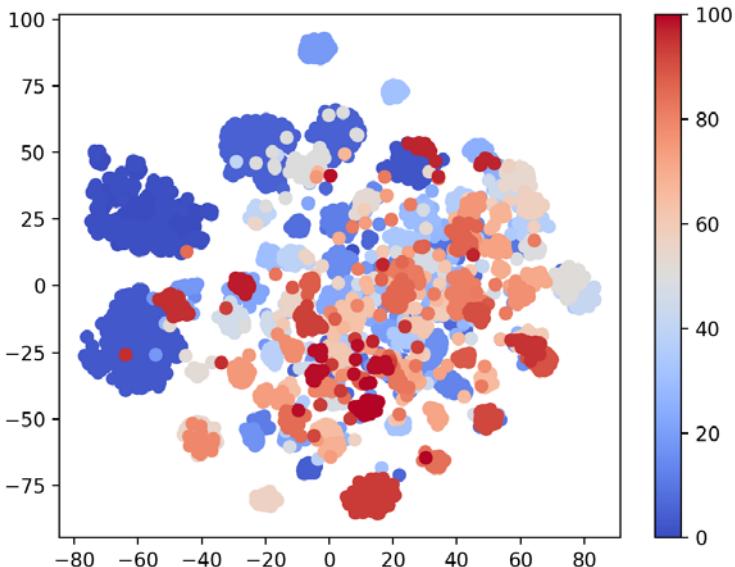
# Для скорости проанализируем только первую половину датасета.
selected_features = feature_list_compressed[:4000]
selected_class_ids = class_ids[:4000]
selected_filenames = filenames[:4000]

tsne_results =
    TSNE(n_components=2,verbose=1,metric='euclidean')
    .fit_transform(selected_features)

# Построить диаграмму рассеяния из результатов, полученных с помощью
# алгоритма t-SNE
colormap = plt.cm.get_cmap('coolwarm')
scatter_plot = plt.scatter(tsne_results[:,0],tsne_results[:,1],
                           c = selected_class_ids, cmap=colormap)
plt.colorbar(scatter_plot)
plt.show()
```

Мы еще вернемся к методу главных компонент. Для масштабирования до большего числа размерностей можно использовать алгоритм равномерного приближения и проекции многообразия (Uniform Manifold Approximation and Projection, UMAP).

На рис. 4.8 показаны кластеры похожих классов и их расположение относительно друг друга.



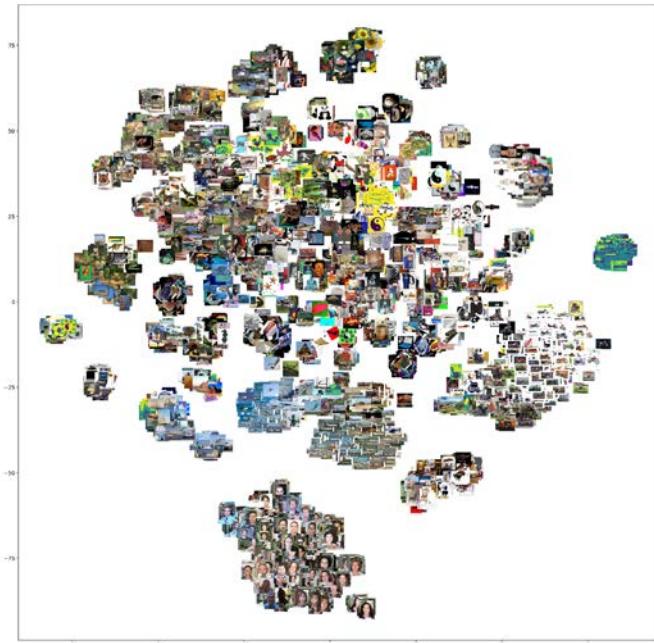
**Рис. 4.8.** Алгоритм t-SNE позволяет визуализировать кластеры признаков изображений, где каждый кластер представляет один класс объектов и отмечен одним цветом

Каждый цвет на рис. 4.8 обозначает отдельный класс. Для большей ясности можно использовать другую вспомогательную функцию, `plot_images_in_2d()`, которая добавляет изображения в эти кластеры, как показано на рис. 4.9.

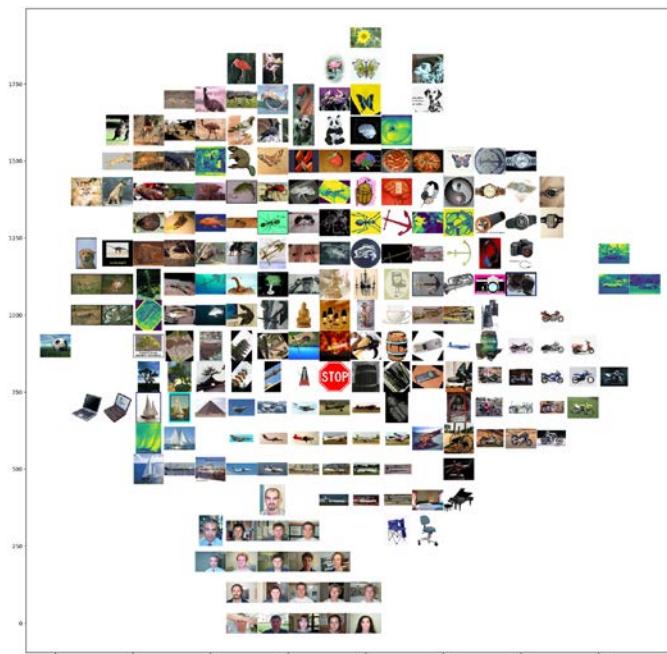
Здорово! Здесь четко видны кластеры, объединяющие изображения людей, цветов, старинных автомобилей, кораблей, мотоциклов, и еще обширный кластер земных и морских животных. График на рис. 4.9 получен наложением большого количества изображений друг на друга, что несколько затрудняет его анализ, поэтому давайте попробуем другой подход к визуализации с использованием ясно видимых плиток, использовав вспомогательную функцию `tsne_to_grid_plotter_manual()`. Получившийся результат можно увидеть на рис. 4.10.

```
tsne_to_grid_plotter_manual(tsne_results[:,0], tsne_results[:,1],
                           selected_filenames)
```

Определенно так намного понятнее. Как видите, похожие изображения размещены в кластерах, объединяющих человеческие лица, стулья, велосипеды, самолеты, корабли, ноутбуки, животных, часы, цветы, наклонные башни, старинные автомобили, якоря и фотоаппараты, и все они находятся близко к себе подобным. Изображения птиц действительно находятся в одной области!



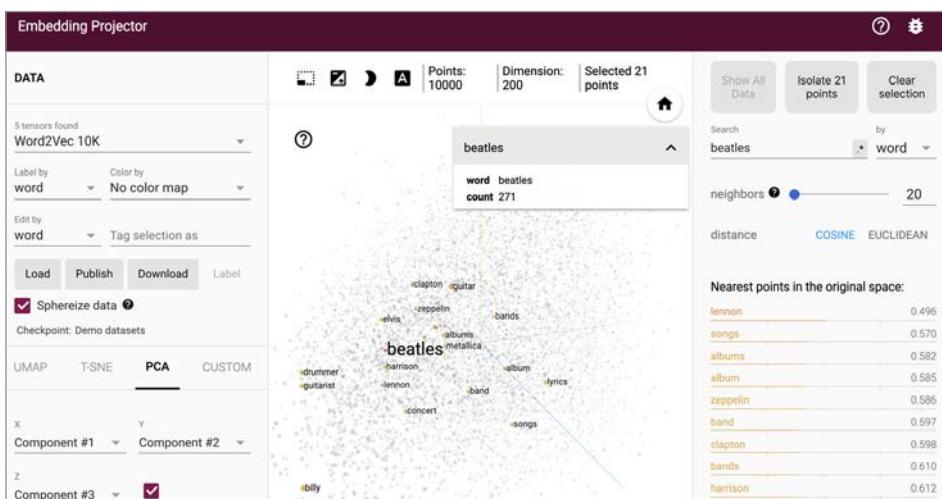
**Рис. 4.9.** Визуализация кластеров изображений, созданных с помощью алгоритма t-SNE; в каждом кластере сгруппированы похожие изображения



**Рис. 4.10.** Визуализация кластеров изображений, созданных с помощью алгоритма t-SNE, где изображения упорядочены мозаикой; похожие изображения находятся ближе друг к другу



Двумерная диаграмма с кластерами выглядят замечательно, но трехмерная диаграмма выглядела бы еще лучше, особенно если бы ее можно было вращать, приближать и отдалять с помощью мыши, и при этом не требовалось написать ни строчки кода. Приятным дополнением могла бы стать возможность выполнять поиск ближайших соседей в интерактивном режиме. Все эти и многие другие возможности предлагает проектор векторных представлений TensorFlow Embedding Projector (<https://projector.tensorflow.org/>) – инструмент с графическим интерфейсом, работающий в браузере. Предварительно загруженные эмбеддинги из датасетов изображений и текста помогают лучше понять возможности этих самых эмбеддингов. И, как показано на рис. 4.11, глубокое обучение определило, что в английском языке John Lennon (Джон Леннон), Led Zeppelin и Eric Clapton (Эрик Клэптон) близки к слову Beatles.



**Рис. 4.11.** Трехмерное представление 10 000 распространенных английских слов в интерфейсе TensorFlow Embedding Projector; здесь выделены слова, близкие к слову «Beatles»

## Увеличение скорости поиска сходств

Есть несколько способов увеличить скорость поиска сходств, и все они основаны на двух стратегиях: уменьшить длину признака или применить более эффективный алгоритм поиска. Рассмотрим эти стратегии по отдельности.

## Длина векторов признаков

В идеале, чем меньше объем данных для поиска, тем быстрее должен выполнять-ся поиск. Напомним, что модель ResNet-50 создает 2048 признаков. Поскольку каждый признак является 32-битным числом с плавающей запятой, каждое изо-

брожение представляется вектором признаков размером 8 Кбайт. Суммарный объем векторов признаков для миллиона изображений составит почти 8 Гбайт. Представьте, насколько долго будет выполняться поиск в 8-гигабайтном массиве признаков. Чтобы вы могли получить более полное представление о нашем сценарии, в табл. 4.1 перечислены различные модели и длины генерируемых ими векторных представлений.

**Таблица 4.1.** Топ-1 % (то есть наиболее вероятный ответ модели совпадает с правильным) по метрике точности и длина вектора признаков для разных моделей CNN

Модель	Длина вектора признаков	Топ-1 точности на ImageNet
VGG16	512	71,5%
VGG19	512	72.7%
MobileNet	1024	66.5%
InceptionV3	2048	78.8%
ResNet-50	2048	75.9%
Xception	2048	79.0%



Многие модели, доступные в `tf.keras.applications`, генерируют векторы с несколькими тысячами признаков. Например, InceptionV3 генерирует массивы признаков с формой  $1 \times 5 \times 5 \times 2048$ , что соответствует 2048 картам признаков со свертками  $5 \times 5$  и общим количеством признаков, равным 51 200. Очевидно, что такой большой вектор желательно уменьшить, используя слой объединения по средним или максимальным значениям в свертках. Такой слой будет преобразовывать каждую свертку (например,  $5 \times 5$ ) в одно значение. Вот как его можно определить во время создания экземпляра модели:

```
model = InceptionV3(weights='imagenet', include_top=False,
                     input_shape = (224,224,3), pooling='max')
```

Почти во всех примерах кода используется такой способ объединения при работе с моделями, генерирующими большое количество признаков. В табл. 4.2 показано влияние объединения по максимальному значению до и после него на количество признаков в различных моделях.

**Таблица 4.2.** Количество признаков до и после объединения для разных моделей

Модель	Количество признаков до объединения	Количество признаков после объединения
ResNet-50	[1,1,1,2048] = 2048	2048
InceptionV3	[1,5,5,2048] = 51200	2048
MobileNet	[1,7,7,1024] = 50176	1024

Как видите, почти все модели генерируют большое количество признаков. Представьте, насколько быстрее выполнялся бы поиск, если можно было бы сократить количество признаков до 100 (это огромное сокращение, от 10 до 20 раз!) без ущерба для качества результатов. Такое сокращение размеров векторов поможет сэкономить не только на операциях, имеющих прямое отношение к поиску, но и на сопутствующих им, потому что позволит загрузить в память сразу весь объем признаков, а не читать их с диска порциями. В этом нам поможет метод главных компонент.

## Уменьшение длины векторов признаков с помощью метода главных компонент

Метод главных компонент (Principal Component Analysis, далее PCA) — это статистическая процедура, которая предполагает, что не все признаки, представляющие данные, одинаково важны и, возможно, в наборе признаков есть избыточные, которые можно удалить и получить аналогичные результаты классификации. PCA считается одним из наиболее популярных методов уменьшения размерности. Обратите внимание, что он не удаляет избыточные признаки, а генерирует новый набор признаков, являющийся линейной комбинацией исходных признаков. Эти новые признаки ортогональны друг другу, а значит, избыточных признаков нет. Эти новые признаки называют *главными компонентами*.

Реализовать PCA довольно просто, достаточно воспользоваться библиотекой `scikit-learn`:

```
import sklearn.decomposition.PCA as PCA
num_feature_dimensions=100
pca = PCA(n_components = num_feature_dimensions)
pca.fit(feature_list)
feature_list_compressed = pca.transform(feature_list)
```

Также PCA позволяет оценить относительную важность каждого признака. Самый первый признак вносит наибольший вклад в дисперсию, и каждый последующий признак вносит все меньший вклад:

```
# Оценка важности первых 20 признаков
print(pca.explained_variance_ratio_[0:20])

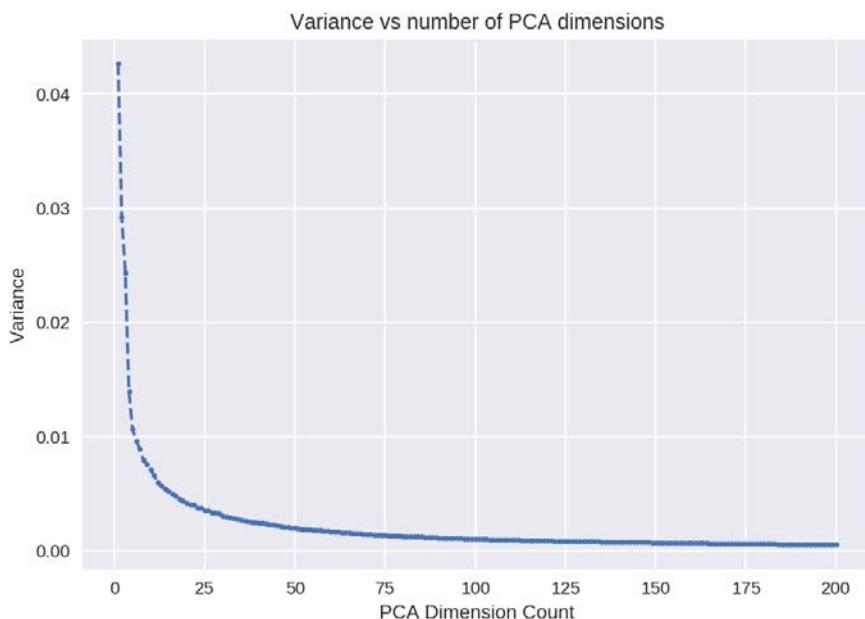
[ 0.07320023  0.05273142  0.04310822  0.03494248  0.02166119  0.0205037
 0.01974325  0.01739547  0.01611573  0.01548918  0.01450421  0.01311005
 0.01200541  0.0113084   0.01103872  0.00990405  0.00973481  0.00929487
 0.00915592  0.0089256 ]
```

Интересно, а почему мы решили сократить 2048 исходных признаков до 100? Почему не до 200? PCA преобразует исходный вектор признаков в новый вектор, но с меньшими размерами. Каждое новое измерение вносит все меньший

вклад в представление исходного вектора (то есть несет все меньше полезной информации об исходных данных) и занимает много места в памяти. Попробуем найти баланс между полнотой описания исходных данных и желаемым количеством признаков в уменьшенном наборе. Давайте определим вклад, скажем, первых 200 измерений.

```
pca = PCA(200)
pca.fit(feature_list)
matplotlib.style.use('seaborn')
plt.plot(range(1,201),pca.explained_variance_ratio_,'o--', markersize=4)
plt.title ('Variance for each PCA dimension')
plt.xlabel('PCA Dimensions')
plt.ylabel('Variance')
plt.grid(True)
plt.show()
```

Результаты показаны на рис. 4.12.

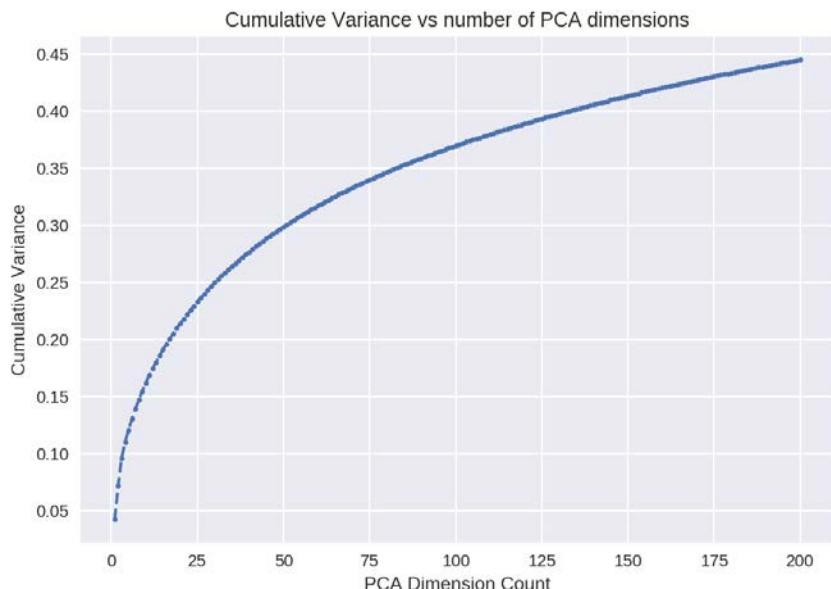


**Рис. 4.12.** Дисперсия для каждого измерения PCA

Индивидуальный вклад в общую дисперсию определяет важность новых добавляемых признаков. Например, главные компоненты, следующие за первой сотней, почти не вносят вклада в общую дисперсию (их вклад близок к 0) и ими можно пренебречь. Даже не проверяя точность, можно с уверенностью предположить, что модель со 100 главными компонентами будет достаточно надежной.

На эту проблему можно взглянуть с другой стороны: посмотреть, какая часть общей дисперсии объясняется ограниченным числом главных компонент, построив график накопленной объясняемой дисперсии (рис. 4.13).

```
plt.plot(range(1,201),pca.explained_variance_ratio_.cumsum(),'o--',
         markersize=4)
plt.title ('Cumulative Variance with each PCA dimension')
plt.xlabel('PCA Dimensions')
plt.ylabel('Variance')
plt.grid(True)
plt.show()
```



**Рис. 4.13.** Кумулятивная дисперсия с каждым измерением PCA

Как и ожидалось, дополнительные 100 измерений (от 100 до 200) объясняют всего 10 % от кумулятивной дисперсии и постепенно выходят на плато. Для справки: использование всех 2048 признаков приведет к объяснению доли общей дисперсии, равной 1.

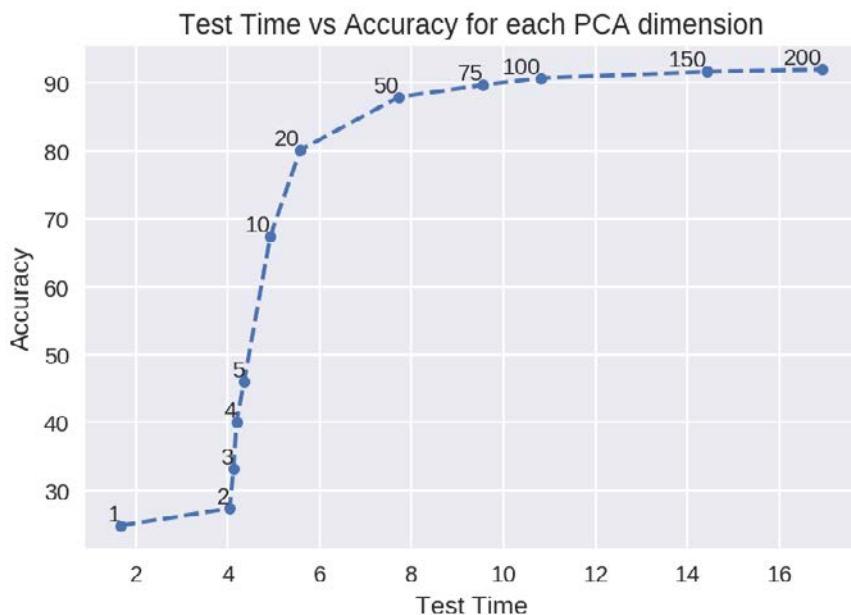
Количество измерений в PCA — важный параметр, который можно настроить для решения поставленной задачи. Один из способов выбрать хороший порог — найти приемлемый баланс между количеством признаков и их влиянием на точность и скорость:

```
pca_dimensions = [1,2,3,4,5,10,20,50,75,100,150,200]
pca_accuracy = []
pca_time = []
```

```
for dimensions in pca_dimensions:  
    # найти главные компоненты  
    pca = PCA(n_components = dimensions)  
    pca.fit(feature_list)  
    feature_list_compressed = pca.transform(feature_list[:])  
    # Вычислить точность, обеспечивающую сокращенным набором признаков  
    accuracy, time_taken = accuracy_calculator(feature_list_compressed[:])  
    pca_time.append(time_taken)  
    pca_accuracy.append(accuracy)  
    print("For PCA Dimensions = ", dimensions, ",\tAccuracy = ",accuracy,"%",  
",\tTime = ", pca_time[-1])
```

Отобразив полученные результаты на графике (рис. 4.14), можно увидеть, что после определенного количества измерений дальнейшее увеличение их числа не приводит к более высокой точности:

```
plt.plot(pca_time, pca_accuracy, 'o--', markersize=4)  
for label, x, y in zip(pca_dimensions, pca_time, pca_accuracy):  
    plt.annotate(label, xy=(x, y), ha='right', va='bottom')  
plt.title ('Test Time vs Accuracy for each PCA dimension')  
plt.xlabel('Test Time')  
plt.ylabel('Accuracy')  
plt.grid(True)  
plt.show()
```



**Рис. 4.14.** Прирост времени поиска и точности с увеличением числа главных компонент

Как видно на графике, добавление новых признаков сверх 100, выбранных первоначально, дает весьма небольшое увеличение точности. Зато, имея размер (100) почти в 20 раз меньше исходного (2048), новый вектор признаков обеспечивает значительно более высокую скорость поиска практически с любым алгоритмом и сопоставимую (а иногда и немного большую) точность. Следовательно, 100 – это, пожалуй, идеальная длина вектора признаков для этого датасета. Это также означает, что первые 100 главных компонент содержат большую часть информации о датасете.

Подобное сокращенное представление дает такие преимущества, как эффективное использование вычислительных ресурсов, удаление шума, лучшее обобщение за счет меньшего количества измерений и повышенную производительность алгоритмов машинного обучения, обучающихся на этих данных. Ограничивааясь расстояниями между наиболее важными признаками, можно даже немного повысить точность результатов. Это связано с тем, что ранее все 2048 признаков вносили равный вклад в вычисление расстояний, а теперь расстояния определяются только 100 наиболее важными признаками. Но что особенно важно, этот прием избавляет нас от *проклятия размерности*. На практике замечено, что с увеличением числа признаков отношение евклидовых расстояний между двумя ближайшими и двумя самыми дальними точками стремится к 1. В пространстве с очень большим числом измерений большинство точек из реального датасета оказываются на одинаковом расстоянии друг от друга, равном или близком к 1, и метрика евклидова расстояния не позволяет отличать схожие и несхожие элементы. РСА помогает вернуть здравомыслие.

Поэкспериментируйте с другими метриками расстояний: расстоянием Минковского, манхэттенским расстоянием, расстоянием Жаккара и взвешенным евклидовым расстоянием (где веса признаков равны их вкладам и сохраняются в `pca.explained_variance_ratio_`).

Теперь используем этот сокращенный набор признаков, чтобы ускорить поиск.

## Масштабирование поиска сходства с помощью метода приблизительных ближайших соседей

Что мы хотим? Найти ближайших соседей. С чем будем сравнивать? С поиском методом полного перебора. Метод перебора легко реализовать — достаточно написать всего две строки кода. Но он изучает каждый элемент в датасете, поэтому затраты времени на его выполнение линейно растут с увеличением размеров датасета (количества элементов и количества измерений). Если методом главных компонент мы уменьшим размерность вектора признаков с 2048 до 100, то получим не только 20-кратное уменьшение объема анализируемых данных,

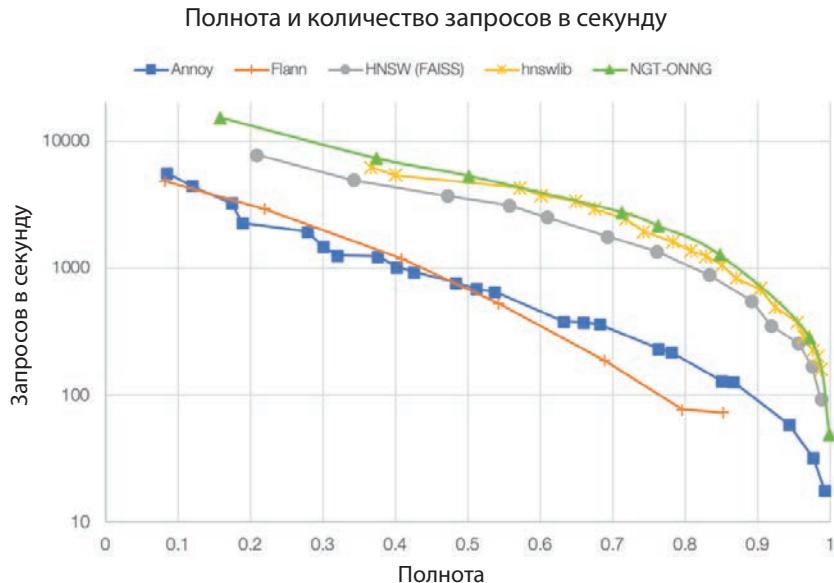
но также 20-кратное увеличение скорости при использовании метода полного перебора. Затраты на внедрение РСА определенно окупаются!

Предположим, что поиск сходств в небольшой коллекции из 10 000 изображений, представленных теперь 100-мерными векторами, занимает примерно 1 мс. На первый взгляд кажется, что поиск среди 10 000 элементов производится очень быстро, но в реальной системе, хранящей намного больше данных, например 10 миллионов элементов, поиск займет больше секунды. Такая система не сможет обработать больше одного запроса в секунду на ядро процессора. Если от пользователей поступает по 100 запросов в секунду, то даже если они будут обрабатываться машинами с процессорами, имеющими несколько ядер (и загружают поисковый индекс для каждого потока), потребуется несколько машин, чтобы справиться с обслуживанием трафика. Неэффективный алгоритм приведет к большим затратам на оборудование.

Метод полного перебора — это наша точка отсчета, на которую мы будем ориентироваться. Как и в большинстве случаев, этот метод оказывается самым медленным. Теперь, когда у нас есть точка отсчета (базовое решение), изучим алгоритмы приблизительных ближайших соседей. Эти алгоритмы не гарантируют абсолютной точности, как метод полного перебора, но *обычно* дают правильный результат, потому что... являются алгоритмами аппроксимации (приблизительной оценки). Большинство алгоритмов предлагают некоторые возможности настройки баланса между правильностью и скоростью. Качество результатов можно оценить, сравнив их с результатами алгоритма перебора.

## Бенчмарк метода приблизительных ближайших соседей

Есть несколько библиотек, реализующих алгоритмы приблизительных ближайших соседей (Approximate Nearest-Neighbor, ANN), в том числе такие известные, как Spotify Annoy, FLANN, Facebook Faiss, Yahoo NGT и NMSLIB. Тестирование каждой из них превратилось бы в утомительную задачу (даже если вы уже установили некоторые из них). К счастью, бравые парни из *ann-benchmarks.com* (Мартин Аумюллер (Martin Aumüller), Эрик Бернхардссон (Erik Bernhardsson) и Алек Фейтфулл (Alec Faitfull)) проделали за нас всю работу, протестирував 19 библиотек на больших общедоступных датасетах. Для сравнения мы решили взять набор эмбеддингов слов (вместо изображений) под названием GloVe. Этот датасет имеет объем 350 Мбайт и состоит из 400 000 векторов признаков, представляющих слова в 100-мерном пространстве. На рис. 4.15 показано, как меняется производительность библиотек в зависимости от качества модели (по метрике полноты). Производительность измеряется способностью библиотеки отвечать на запросы каждую секунду. Напомним, что полнота — это доля топ-*n* самых близких элементов в полученном результате, которые действительно являются топ-*n* самыми близкими. Фундаментальная истина определяется методом полного перебора.



**Рис. 4.15.** Сравнение библиотек поиска методом приблизительных ближайших соседей (данные взяты с сайта [ann-benchmarks.com](http://ann-benchmarks.com))

Самые быстрые библиотеки из числа испытуемых способны обрабатывать до нескольких тысяч запросов в секунду при приемлемом значении полноты 0,8. Для сравнения: поиск методом полного перебора обрабатывает менее 1 запроса в секунду. Самые быстрые из этих библиотек (например, NGT) могут обрабатывать до 15 000 запросов в секунду (хотя и с невысоким уровнем полноты, что делает их непрактичными для использования).

## Какую библиотеку выбрать?

Разумеется, что выбор библиотеки во многом будет зависеть от сценария использования. Каждая библиотека дает определенный компромисс между скоростью поиска, точностью, размером индекса, потреблением памяти, использованием оборудования (CPU/GPU) и простотой настройки. В табл. 4.3 представлены некоторые сценарии использования и рекомендации в выборе библиотеки, которая лучше всего подходит для каждого сценария.

Более подробные примеры использования некоторых библиотек ищите на страницах книги в GitHub (<https://github.com/PracticalDL/Practical-Deep-Learning-Book/>), а здесь мы рассмотрим только библиотеку Annoy и сравним ее с поиском методом полного перебора в синтетическом датасете. Кроме того, кратко коснемся библиотек Faiss и NGT.

**Таблица 4.3.** Рекомендации по выбору библиотек поиска методом приблизительных ближайших соседей

Сценарий	Рекомендации
Для коротких экспериментов в Python. Важны минимум настроек и высокая скорость	Annoy или NMSLIB
Требуется высокая скорость на огромном датасете (до 10 миллионов элементов или до размерности пространства в несколько тысяч измерений)	NGT
Для поиска в гигантском датасете (более 100 миллионов элементов) на кластере графических процессоров	Faiss
Хочу построить базовое решение со 100%-ной точностью, а затем использовать наиболее быструю библиотеку, чтобы впечатлить шефа ускорением на порядки и получить премию	Метод полного перебора

## Создание синтетического датасета

Для корректного сравнения библиотек создадим датасет на миллион элементов, состоящих из случайных значений с плавающей запятой со средним 0 и дисперсией 1. И генерируем случайный вектор признаков, для которого будем искать ближайших соседей:

```
num_items = 1000000
num_dimensions = 100
dataset = np.random.randn(num_items, num_dimensions)
dataset /= np.linalg.norm(dataset, axis=1).reshape(-1, 1)

random_index = random.randint(0,num_items)
query = dataset[random_index]
```

## Полный перебор

Сначала определим время поиска с помощью алгоритма полного перебора. Он последовательно просматривает все данные, вычисляя расстояние между каждым элементом в наборе и искомым вектором. Для замера времени используем команду `timeit`. Сначала создадим поисковый индекс для получения пяти ближайших соседей, а затем выполним поиск:

```
neighbors = NearestNeighbors(n_neighbors=5, algorithm='brute',
metric='euclidean').fit(dataset)
%timeit distances, indices = neighbors.kneighbors([query])

> 177 ms ± 136 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```



Команда `timeit` — удобный инструмент. Чтобы измерить время выполнения одной операции, добавьте перед ней эту команду. По сравнению с командой `time`, которая выполняет оператор один раз, `timeit` может запустить следующую за ней строку несколько раз, чтобы получить более точную агрегированную оценку вместе со стандартным отклонением. По умолчанию она отключает сборку мусора, делая результаты более сопоставимыми. Однако эти результаты могут не совпадать с производительностью реальных производственных нагрузок, когда включена сборка мусора.

## Annoy

Annoy (Approximate Nearest Neighbours Oh Yeah) — это библиотека, написанная на C++ и имеющая интерфейс для Python, предназначенная для приближенного поиска ближайших соседей. Эта библиотека, отличающаяся высокой скоростью обработки, был создана Spotify и используется для музыкальных рекомендаций. Несмотря на название<sup>1</sup>, работать с библиотекой легко и приятно.

Чтобы использовать библиотеку Annoy, установите ее через pip:

```
$ pip install annoy
```

Библиотека удивительно проста в использовании. Сначала построим поисковый индекс с двумя гиперпараметрами: количеством измерений в датасете и количеством деревьев:

```
from annoy import AnnoyIndex
annoy_index = AnnoyIndex(num_dimensions) # Длина векторов признаков элементов
for i in range(num_items):
    annoy_index.add_item(i, dataset[i])
annoy_index.build(40) # 40 деревьев
```

Теперь посмотрим, сколько времени потребуется, чтобы найти пять ближайших соседей для одного изображения:

```
%timeit indexes=t.get_nns_by_vector(query, 5, include_distances=True)
> 34.9 µs ± 165 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Очень быстро! Запросы к нашему датасету с миллионом элементов эта библиотека сможет обслуживать со скоростью почти 28 000 запросов в секунду на одном ядре процессора. Учитывая, что большинство процессоров имеют несколько ядер, одна система почти наверняка сможет обрабатывать более 100 000 в секунду. Самое приятное, что библиотека позволяет нескольким процессам совместно использовать один и тот же индекс в памяти. Таким образом, огромный индекс,

---

<sup>1</sup> Акроним «Анноу» совпадает с английским словом «annoy» — противный, раздражающий. — Примеч. пер.

занимающий почти всю оперативную память, сможет обслуживать сразу несколько запросов в одной системе.

Еще одно преимущество: библиотека генерирует индекс довольно скромного размера. Более того, она отделяет создание индекса от его загрузки, поэтому индекс можно создать на одном компьютере, передать его, а затем на другом компьютере загрузить индекс в память и обрабатывать запросы.



Хотите знать, как правильно выбрать количество деревьев? Чем больше деревьев, тем выше точность, но тем больше размер индекса. Обычно для достижения максимальной точности требуется не более 50 деревьев.

## NGT

Библиотека Neighborhood Graph and Tree (NGT), созданная в Yahoo Japan, сейчас лидирует в большинстве тестов и лучше всего подходит для больших датасетов (с миллионами элементов) с большим количеством измерений (до нескольких тысяч). Библиотека появилась в 2016 году, но тестировать ее начали только в 2018-м, после реализации алгоритма ONNG (сокращенно от «Optimization of indexing based on k-Nearest Neighbor Graph» — оптимизация индексирования на основе графа k-ближайших соседей). Учитывая, что библиотека NGT может вызываться одновременно из нескольких потоков выполнения на сервере, она позволяет разместить индекс в общей памяти с помощью файлов, отображаемых в память, что помогает уменьшить потребление памяти, но увеличивает время загрузки.

## Faiss

Faiss — это эффективная библиотека поиска сходств, созданная в Facebook. Она умеет масштабироваться до поиска среди миллиардов векторов в ОЗУ на одном сервере, сохраняя векторы признаков в компактном представлении (с квантованием) вместо исходных значений. Она лучше всего подходит для плотных векторов, что особенно заметно на машинах с графическими процессорами (GPU), потому что индекс хранится в памяти графического процессора (VRAM). Может использовать один или несколько GPU. Позволяет настраивать производительность, точность, потребление памяти и время индексации. Это одна из самых быстрых известных реализаций поиска методом приблизительных ближайших соседей на GPU. Если библиотеки достаточно для разработки в Facebook, то ее достаточно и большинству из нас (при условии, что у нас есть достаточно данных).

Не будем показывать весь процесс установки библиотеки Faiss, но отметим, что проще всего сделать это с помощью инсталлятора Anaconda или контейнеров Docker.

## Повышение точности с помощью тонкой настройки

Многие модели, предварительно обученные на датасете ImageNet, предлагают отличную отправную точку для оценки сходства в большинстве ситуаций. Но если вы адаптируете эти модели для своей конкретной задачи, они будут давать еще более точные результаты при поиске похожих изображений.

В этой части определим категории с наихудшими результатами, визуализируем их с помощью алгоритма t-SNE, проведем тонкую настройку и посмотрим, как изменится график t-SNE.

Какой показатель лучше подходит для проверки качества поиска похожих изображений?

### *Трудоемкий вариант 1*

Просмотреть весь набор изображений и вручную оценить, действительно ли найденные изображения выглядят одинаково.

### *Более простой вариант 2*

Вычислить точность. То есть определить, принадлежат ли найденные похожие изображения к той же категории, что и искомое. Мы будем использовать эту оценку точности.

Итак, какие категории в нашем наборе худшие? Почему они худшие? Чтобы ответить на этот вопрос, мы заранее определили вспомогательную функцию `worst_classes`. Для каждого изображения в датасете она отыскивает ближайших соседей, используя алгоритм полного перебора, а затем возвращает шесть классов, показавших наименьшую точность. Чтобы увидеть эффект тонкой настройки, запустим анализ на более сложном датасете: Caltech-256. Вот классы, показавшие худшую точность, которые вернула функция:

```
names_of_worst_classes_before_finetuning, accuracy_per_class_before_finetuning =  
worst_classes(feature_list[:])
```

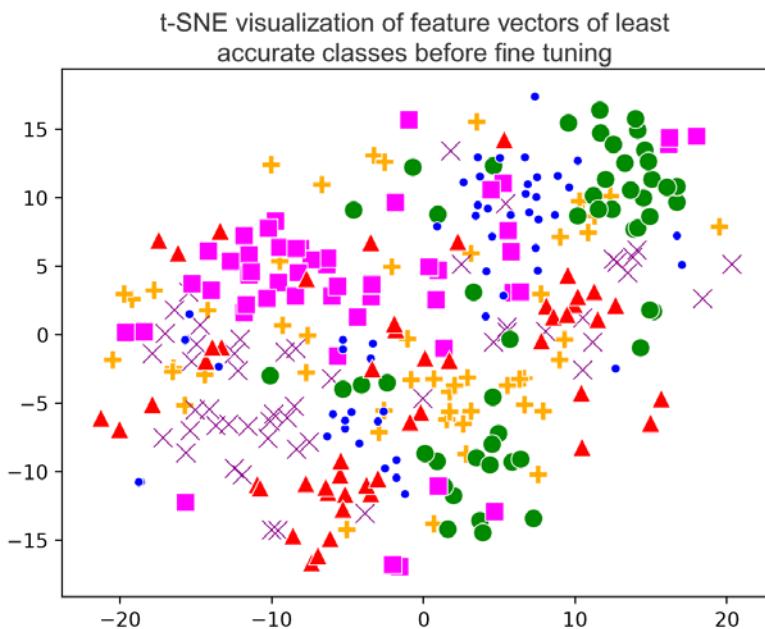
```
Accuracy is 56.54  
Top 6 incorrect classifications  
059.drinking-straw    Accuracy: 11.76%  
135.mailbox            Accuracy: 16.03%  
108.hot-dog             Accuracy: 16.72%  
163.playing-card        Accuracy: 17.29%  
195.soda-can            Accuracy: 19.68%  
125.knife               Accuracy: 20.53%
```

Понять причину низкой точности при работе с некоторыми классами поможет диаграмма t-SNE, визуализирующая эмбеддинги в двумерном пространстве (рис. 4.16). Чтобы не перегружать диаграмму мелкими деталями, мы использовали только по 50 элементов из этих 6 классов.



Чтобы упростить чтение диаграммы, можно определить разные метки и использовать разные цвета для разных классов. Matplotlib предоставляет широкий выбор меток и цветов.

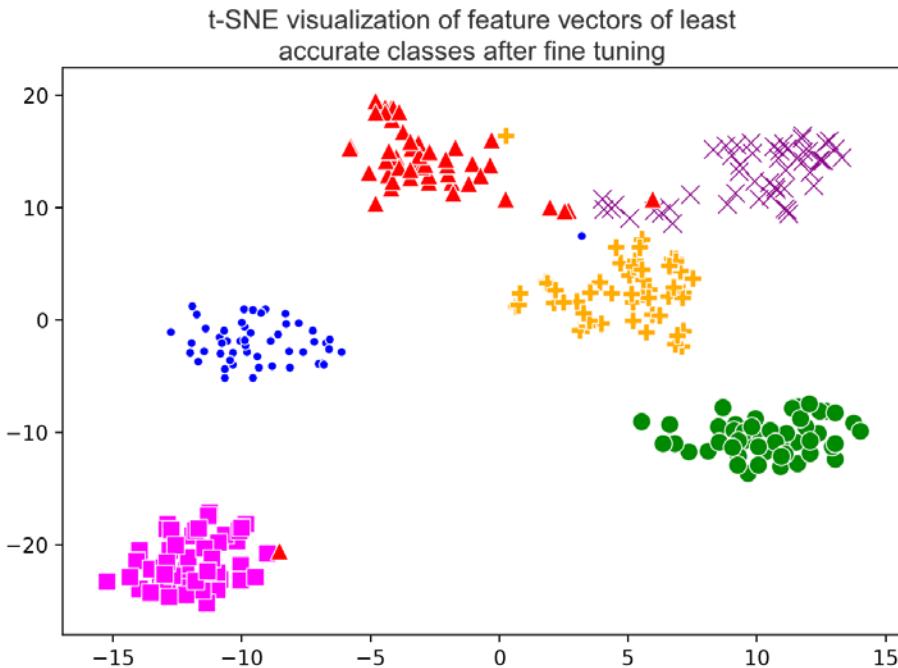
```
markers = [ '^', '.', 's', 'o', 'x', 'P' ]
colors = ['red', 'blue', 'fuchsia', 'green',
          'purple', 'orange']
```



**Рис. 4.16.** Диаграмма t-SNE векторов признаков для классов, показавших наименьшую точность до тонкой настройки

Так вот в чем дело! Векторы признаков разбросаны повсюду и накладываются друг на друга. Использование этих векторов, например, в задачах классификации может оказаться не лучшей идеей, потому что будет трудно найти четкую поверхность, разделяющую их. Неудивительно, что они показали плохие результаты в этом teste классификации ближайших соседей.

Как вы думаете, что получится, если повторить эти шаги с моделью после тонкой настройки? Считаем, что результат получится интересным; взгляните на рис. 4.17 и убедитесь сами.



**Рис. 4.17.** Диаграмма t-SNE векторов признаков для классов, показавших наименьшую точность после тонкой настройки

Картина получилась намного более четкой. После небольшой тонкой настройки, как показано в главе 3, эмбеддинги сгруппировались. Сравните зашумленные/рассеянные эмбеддинги, полученные предварительно обученными моделями, с результатами, полученными после тонкой настройки. Классификатору в модели МО теперь будет гораздо легче найти поверхность, разделяющую эти классы, что обеспечит более высокую точность классификации, а также обнаружение более похожих изображений, когда классификатор не используется. И имейте в виду, что это были классы с наибольшей ошибкой в классификации; а теперь представьте, насколько хорошо после тонкой настройки будут определяться классы с изначально более высокой точностью классификации.

Эмбеддинги, генерируемые предварительно обученной моделью, обеспечивали точность 56 %. Новые эмбеддинги, полученные после тонкой настройки, обеспечивают колоссальную точность 87 %! Небольшая толика волшебства дала удивительный результат.

Единственное ограничение для тонкой настройки — наличие размеченных данных, которые есть не всегда. Поэтому в зависимости от сценария использования может потребоваться самостоятельно создать размеченный датасет.

Но здесь есть небольшая хитрость, о которой поговорим в следующем разделе.

## Тонкая настройка без полно связанных слоев

Как мы уже знаем, нейронная сеть состоит из трех частей:

- сверточные слои, которые создают векторы признаков;
- полно связанные слои;
- последний слой классификатора.

Тонкая настройка, как следует из названия, предполагает подстройку нейронной сети для адаптации к новому датасету. Обычно для этого из сети исключаются полно связанные (верхние) слои, на их место подставляются новые, а затем производится обучение этой воссозданной нейронной сети с использованием нового датасета. Обучение подобным образом вызывает:

- значительное изменение весов во всех вновь добавленных полно связанных слоях;
- незначительное изменение весов в сверточных слоях.

Полно связанные слои выполняют основную работу для достижения максимальной точности классификации, а большая часть сети, которая генерирует векторы признаков, изменяется незначительно. То есть в ходе тонкой настройки векторы признаков изменяются незначительно.

Цель в том, чтобы для похожих на вид объектов генерировались более близкие векторы признаков, что почти невозможно при тонкой настройке, описанной ранее. Вовлекая в обучение для конкретной задачи еще и сверточные слои, можем добиться гораздо лучших результатов, как было показано. Но как это сделать? Нужно удалить все полно связанные слои и поместить слой классификатора сразу после сверточных слоев (которые генерируют векторы признаков). Так мы оптимизируем модель для поиска сходства, но не для классификации.

Чтобы сравнить процесс тонкой настройки модели для задач классификации с тонкой настройкой модели для поиска сходства, давайте вспомним, как в главе 3 мы настраивали модель для классификации:

```
from tensorflow.keras.applications.resnet50 import ResNet50
model = ResNet50(weights='imagenet', include_top=False,
input_shape = (224,224,3))
input = Input(shape=(224, 224, 3))
x = model(input)
```

```
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(NUM_CLASSES, activation='softmax')(x)
model_classification_optimized = Model(inputs=input, outputs=x)
```

А вот как настраивается модель для поиска сходства. Обратите внимание на отсутствующий скрытый полносвязный слой в середине:

```
from tensorflow.keras.applications.resnet50 import ResNet50
model = ResNet50(weights='imagenet', include_top=False,
input_shape = (224,224,3))
input = Input(shape=(224, 224, 3))
x = model(input)
x = GlobalAveragePooling2D()(x)
# Здесь нет ни полносвязанного слоя (Dense),
# ни слоя прореживания (Dropout)
x = Dense(NUM_CLASSES, activation='softmax')(x)
model_similarity_optimized = Model(inputs=input, outputs=x)
```

После тонкой настройки, чтобы использовать `model_similarity_optimized` для получения признаков вместо вероятностей классов, просто удалите (`pop`) последний слой:

```
model_similarity_optimized.layers.pop()
model = Model(model_similarity_optimized.input,
model_similarity_optimized.layers[-1].output)
```

Важно отметить, что если бы мы использовали обычный процесс тонкой настройки, то получили бы более низкую точность в определении сходства, чем дает `model_similarity_optimized`. Очевидно, что для сценариев классификации предпочтительнее использовать `model_classification_optimized`, а для извлечения признаков с целью поиска похожих изображений — `model_similarity_optimized`.

Теперь, обретя новые знания, вы сможете создать как быструю, так и точную модель поиска сходства для любого сценария. А теперь посмотрим, как гиганты индустрии искусственного интеллекта создают свои продукты.

## Сиамские сети для распознавания лица однократным (one-shot) обучением

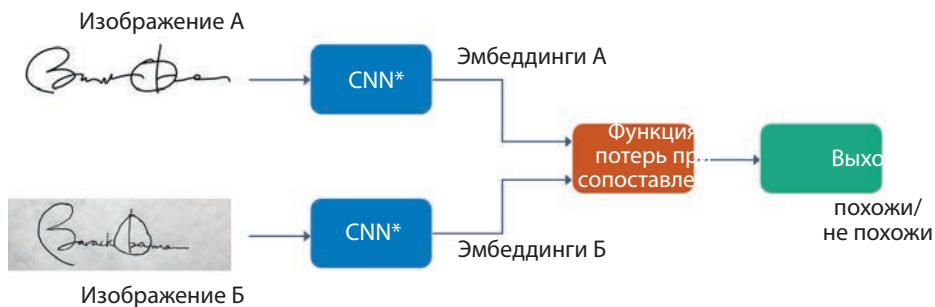
Система проверки лиц обычно пытается установить — по двум картинкам, — являются ли они изображением одного и того же человека. Это высокоточный бинарный классификатор, который должен надежно сравнивать людей, не взирая на разное освещение, одежду, прическу, фон и выражение лица. Задача может усложняться еще вот чем: в базе данных сотрудников, например, могут

храниться изображения большого количества людей, но только очень небольшое число изображений одного и того же человека. Системы идентификации подписи в банках и продуктов на Amazon сталкиваются с той же проблемой — ограниченное количество изображений каждого товара.

Как бы вы подошли к обучению такого классификатора? Выбор эмбеддингов из модели, подобной ResNet, предварительно обученной на датасете ImageNet, может не различать эти прекрасные черты лица. Одно из решений состоит в том, чтобы выделить каждого человека в отдельный класс, а затем выполнить обучение, как в случае обычной сети. Но возникают две ключевые проблемы:

- если бы у нас был миллион человек, обучение по миллиону категорий было бы просто невозможно;
- обучение с использованием небольшого количества изображений в каждом классе неизбежно приведет к переобучению.

Другое решение: вместо обучения на разных категориях можно обучить сеть сравнивать изображения и определять, похожи ли пары изображений или нет, давая ей подсказки о сходстве во время обучения. И это ключевая идея, лежащая в основе сиамских сетей: ввести в модель два изображения, извлечь два эмбеддинга и затем вычислить расстояние между ними. Если расстояние ниже порога, изображения можно считать похожими, иначе — нет. Вводя пары изображений с соответствующей меткой похожи/не похожи и обучая сеть от начала до конца, эмбеддинги начинают выявлять детали во входных данных. Этот подход прямой оптимизации метрики расстояния, показанный на рис. 4.18, называется *метрическим обучением*.



\* Одна и та же сеть CNN

**Рис. 4.18.** Сиамская сеть для проверки подписи; обратите внимание, что для обработки обоих изображений используется одна и та же сверточная сеть (CNN)

Можно развить эту идею и даже подавать на вход три изображения: одно эталонное, одно похожее (из той же категории) и одно непохожее (из другой категории) изображение.

Обучим эту сеть минимизации расстояния между похожими изображениями и максимизации расстояния между разными изображениями. Функция потерь, которая поможет в этом, называется *тройной функцией потерь (triplet loss)*. Функция потерь в предыдущем примере с парой изображений называется *контрастивной функцией потерь (contrastive loss)*. Тройная функция потерь дает лучшие результаты.

После обучения сети понадобится только одно эталонное изображение лица, чтобы на этапе проверки решить, изображен ли на них один и тот же человек. Эта методология открывает двери для обучения распознавания по одному снимку. К другим распространенным применением можно отнести распознавание подписи и логотипа. Сакет Махешвари (Saket Maheshwary) и Хемант Мисра (Hemant Misra) предлагают использовать сиамскую сеть для сопоставления резюме с соискателями путем вычисления семантического сходства между ними.

## Примеры из практики

Рассмотрим несколько интересных примеров, которые показывают, как то, что мы узнали, применяется в отрасли.

### Flickr

Flickr — один из крупнейших фотохостингов, особенно популярный среди профессиональных фотографов. Чтобы помочь фотографам найти вдохновение, а также показать снимки, которые могут быть интересны пользователям, на сайте Flickr реализована функция поиска похожих фотографий, основанная на сходстве семантического значения. Как показано на рис. 4.19, поиск фотографий из категории «пустыня» вернул несколько похожих снимков. Под капотом Flickr использует



**Рис. 4.19.** Похожие фотографии из категории «пустыня» (изображения взяты с сайта <https://code.flickr.com/>)

алгоритм ANN под названием Locally Optimized Product Quantization (LOPQ), исходный код которого был открыт в Python, а также в реализациях Spark.

## Pinterest

Pinterest — это веб-приложение, ставшее популярным благодаря возможностям визуального поиска, в частности функциям «Похожие пины» и «Связанные пины». Например, Baidu и Alibaba поддерживают аналогичные системы визуального поиска. Zappos, Google Shopping и *like.com* используют компьютерное зрение для подбора рекомендаций.

В Pinterest тема «женская мода» — одна из самых популярных, а функция «Похожие пины» (рис. 4.20) помогает людям находить похожие товары. Кроме

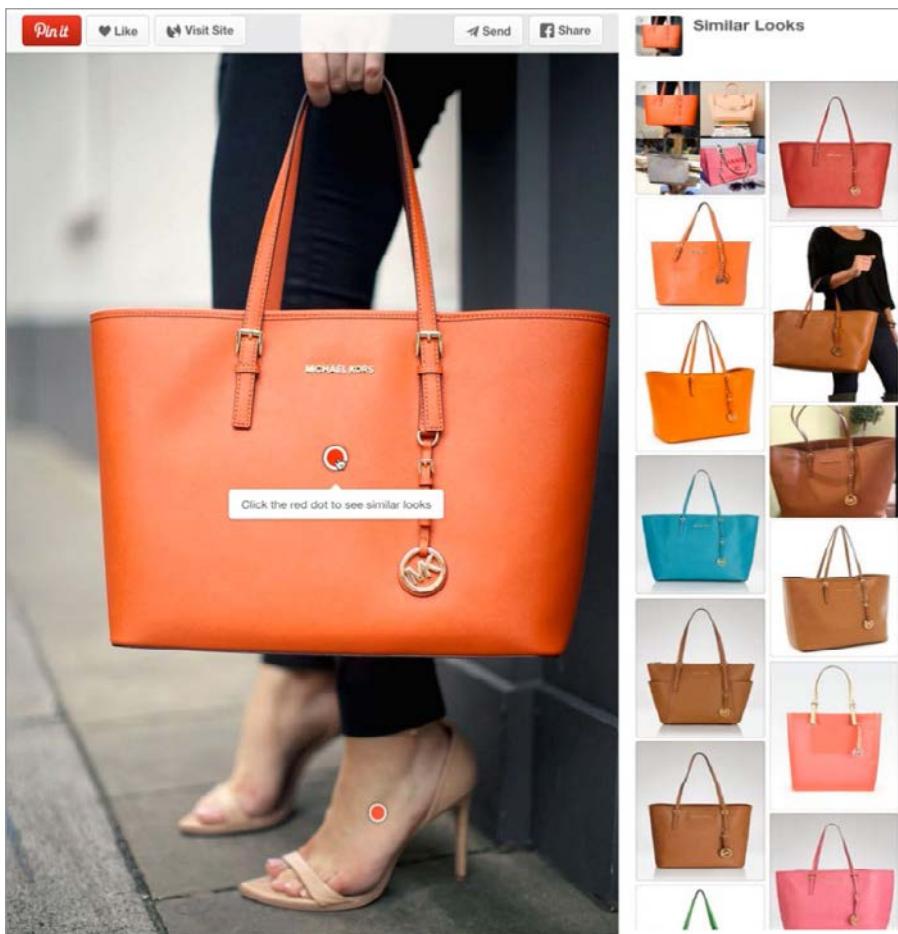


Рис. 4.20. Функция «Похожие идеи» в приложении Pinterest (изображение взято из блога Pinterest)

того, Pinterest сообщает, что его функция «Связанные пины» увеличила частоту повторов. Не каждый пин на Pinterest имеет связанные метаданные, что делает рекомендацию сложной проблемой из-за отсутствия контекста. Разработчики Pinterest решили эту проблему, используя визуальные признаки для поиска связанных пинов. Кроме того, в Pinterest реализована служба инкрементного снятия отпечатков, которая генерирует новые цифровые подписи, если загружается новое изображение или происходит эволюция признаков (из-за внесения инженерами улучшений или изменений в базовые модели).

## Двойники знаменитостей

Приложения вроде *Celebslike.me*, ставшие вирусными в 2015 году, ищут «ближайшего соседа» среди знаменитостей, как показано на рис. 4.21. Аналогичный подход был применен в приложении Google Arts & Culture в 2018 году, которое показывает реальный портрет, наиболее похожий на вас. «Twins or not» (<https://www.twinsornot.net/>) — еще одно аналогичное приложение.

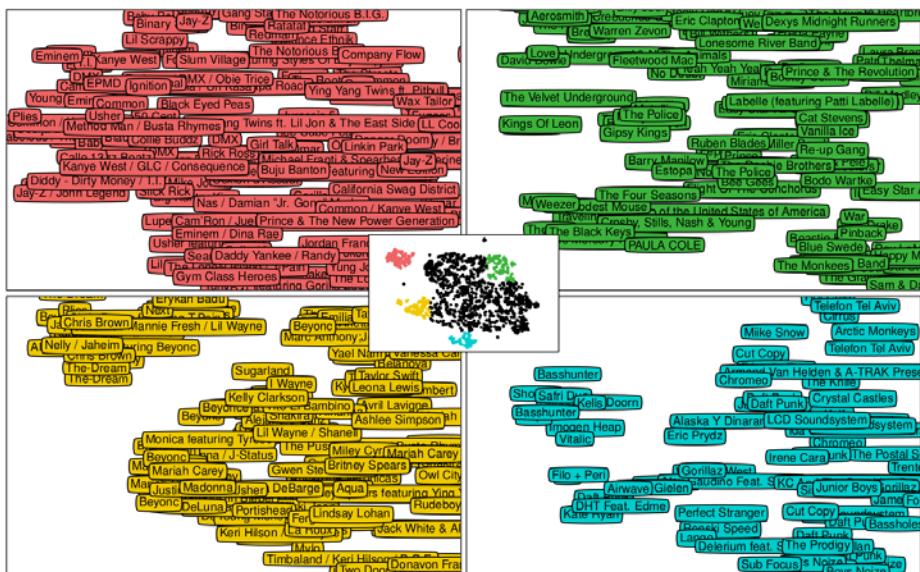


**Рис. 4.21.** Проверка фотографии нашего друга Пита Уордена (Pete Warden), технида команды разработки TensorFlow для мобильных и встраиваемых устройств в Google, на веб-сайте celebslike.me

## Spotify

Spotify использует алгоритм поиска ближайших соседей для рекомендаций в выборе музыки и автоматического создания плейлистов и радиостанций на

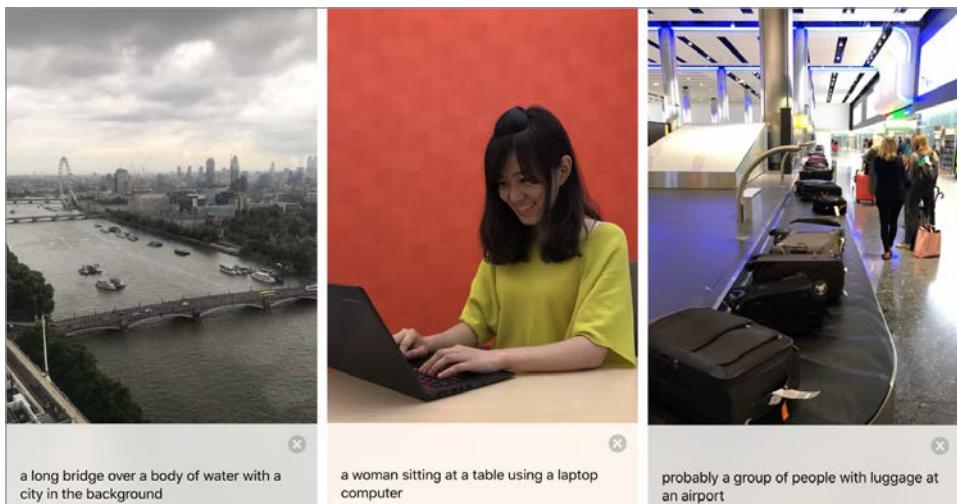
основе текущего набора прослушанных треков. Обычно методы совместной (коллаборативной) фильтрации, которые используются для создания рекомендаций, например фильмов на Netflix, не зависят от контента — то есть рекомендации основываются на том, что большие группы пользователей со схожими вкусами смотрят похожие фильмы или слушают похожую музыку. И это проблема для нового и пока не набравшего популярность контента, потому что пользователям будут рекомендовать имеющийся популярный контент. Это проблема холодного запуска. Решение — использовать скрытое представление контента. Как и в случае с изображениями, можно создавать векторы признаков для музыки, используя коэффициенты MFCC (Mel Frequency Cepstral Coefficients — коэффициенты косинусного преобразования Фурье для частот чистых тонов), генерирующие двумерную спектrogramму, которую можно рассматривать как изображение и использовать для генерации векторов признаков. Треки разбиваются на трехсекундные фрагменты, и их спектrogramмы используются для создания признаков. Затем эти признаки усредняются и вместе образуют представление всего трека. На рис. 4.22 показаны исполнители, треки которых группируются в определенных областях. Можно заметить хип-хоп (вверху слева), рок (вверху справа), поп (внизу слева) и электронную музыку (внизу справа). Как уже говорилось, под капотом Spotify использует библиотеку Annoy.



**Рис. 4.22.** Визуализация t-SNE распределения прогнозируемых шаблонов использования с помощью скрытых факторов, предсказанных по аудио (источник изображения: статья «Deep content-based music recommendation» Аарона ван ден Оорда (Aaron van den Oord), Сандера Дилемана (Sander Dieleman), Бенджамина Шраувена (Benjamin Schrauwen), NIPS 2013)

## Описание изображений

Описание изображений (image captioning) — это наука о переводе изображений в предложения (как показано на рис. 4.23). Она не умещается в рамки простой разметки объектов и требует более глубокого визуального анализа всего изображения и взаимосвязей между объектами. Для обучения моделей в этой сфере в 2014 году был выпущен открытый датасет MS COCO, который включает более 300 000 изображений вместе с категориями объектов, описывающими их предложениями, визуальными парами «вопрос-ответ» и рамками объектов. Он служит бенчмарком для ежегодного конкурса, где можно отслеживать прогресс в области добавления подписей к изображениям, обнаружения объектов и сегментации.

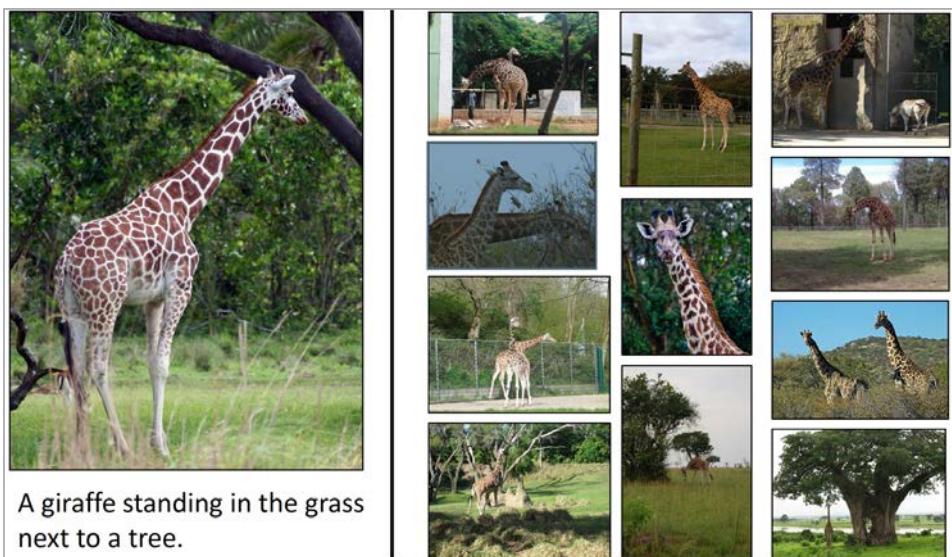


**Рис. 4.23.** Функция описания изображений в приложении для слепых Seeing AI: the Talking Camera App

Общая стратегия, применявшаяся в первом состязании (2015), заключалась в добавлении языковой модели (LSTM/RNN) к сверточной сети (CNN) так, чтобы вектор признаков на выходе CNN передавался на вход языковой модели (LSTM/RNN). Эта комбинированная модель обучалась от начала до конца и показала результаты, впечатлившие мир. Все исследовательские лаборатории вступили в гонку, пытаясь превзойти друг друга, но позже выяснилось, что простой поиск ближайшего соседа может дать результаты ничуть не хуже. Суть состояла в том, чтобы для данного изображения найти похожие изображения, опираясь на эмбеддинги. Затем выбрать общие слова в подписях к похожим изображениям

и сконструировать из них подпись для изображения. Проще говоря, ленивые вычисления оказались ничуть не хуже самых современных решений и помогли выявить критическую предвзятость в датасете.

Эта предвзятость была придумана Ларри Зитником (Larry Zitnick) для задачи «жираф и дерево». Поиските по слову «жираф» изображение, и вы увидите, что кроме жирафа почти на каждом изображении есть трава. Конечно, большинство таких изображений можно описать как «жираф, стоящий в траве». Точно так же для изображения в запросе, как на рис. 4.24 слева, где жираф стоит рядом с деревом, почти все найденные похожие изображения (справа) можно описать как «жираф, стоящий в траве рядом с деревом». Даже без глубокого анализа изображения можно получить вполне правильную подпись, выполнив простой поиск ближайшего соседа. Этот пример показывает, что для оценки реального интеллекта системы нужно больше семантически новых/оригинальных изображений в проверочном наборе.



**Рис. 4.24.** Задача «жираф и дерево» (источник изображения: статья «Measuring Machine Intelligence Through Visual Question Answering» К. Лоуренса Зитника (C. Lawrence Zitnick), Айшварии Агравала (Aishwarya Agrawal), Станислава Антоля (Stanislaw Antol), Маргарет Митчелл (Margaret Mitchell), Дхрува Батра (Dhruv Batra), Деви Париха (Devi Parikh))

Вывод прост: не стоит недооценивать простой подход на основе метода поиска ближайших соседей!

## Итоги

Подошла к концу наша экспедиция, в ходе которой мы рассмотрели задачу поиска похожих изображений с помощью эмбеддингов. Но мы не остановились на достигнутом и узнали, как масштабировать поиск от нескольких тысяч до нескольких миллиардов документов с помощью алгоритмов и библиотек приближенных ближайших соседей, включая Annoy, NGT и Faiss. Мы также узнали, что тонкая настройка модели для конкретного датасета может повысить точность и репрезентативность эмбеддингов в моделях обучения с учителем. В заключение мы познакомились с сиамскими сетями, которые используют мощь эмбеддингов для распознавания методом однократного (one-shot) обучения, например для систем проверки лиц. Наконец, мы увидели, как алгоритмы поиска ближайших соседей используются на практике. Ближайшие соседи — это простой, но мощный инструмент, который нужно иметь в своем арсенале.

## ГЛАВА 5

---

# От новичка до мастера прогнозирования: увеличение точности сверточной нейронной сети

В главе 1 мы говорили о важности ответственного подхода к развитию ИИ. Один из вопросов, который мы обсуждали, — это устойчивость (робастность) моделей. Пользователи будут доверять нашему продукту, только если будут уверены, что ИИ, с которым они ежедневно сталкиваются, работает точно и надежно. Очевидно, что большое значение имеет контекст приложения. Вполне допустимо, если классификатор еды иногда по ошибке будет классифицировать макароны как хлеб. Но для беспилотного автомобиля ошибочная классификация пешехода как полосы движения очень опасна. Мы обсудим очень важную цель — создание максимально точных моделей.

В этой главе вы разовьете интуицию для распознавания возможностей повышения точности модели, когда начнете ее обучение в следующий раз. Сначала познакомимся с инструментами, которые избавят вас от необходимости действовать наугад. После этого перейдем к экспериментам: зададим базовое решение, выделим отдельные параметры для настройки и посмотрим, как они влияют на качество модели и скорость обучения. Большая часть кода из этой главы собрана в одном блокноте Jupyter вместе со списком примеров. Он создавался с учетом возможности переиспользования на тот случай, если вы решите включить его в свой следующий учебный проект.

Рассмотрим несколько вопросов, часто возникающих во время обучения модели.

- Какой подход предпочтительнее для моего сценария: перенос обучения или создание и обучение собственной сети с нуля?

- Какой наименьший объем данных я могу ввести в пайплайн обучения, чтобы получить приемлемые результаты?
- Как добиться, чтобы модель выявляла правильные закономерности и не обнаруживала ложных корреляций?
- Как гарантировать, что я (или кто-то другой) буду получать одни и те же результаты, повторяя эксперимент снова и снова? Как обеспечить воспроизводимость экспериментов?
- Влияет ли изменение размеров входных изображений на прогнозы?
- Насколько сильно влияет уменьшение размеров входного изображения на прогнозы?
- Какой процент слоев следует настраивать при переносе обучения, чтобы достичь желаемого баланса точности модели и времени ее обучения?
- Сколько слоев должно быть в моей модели, если я решу обучить ее с нуля?
- Как выбрать оптимальное значение «скорости обучения» для модели?
- Слишком много всего надо запомнить. Как автоматизировать эту работу?

Ответим на все эти вопросы в экспериментах на нескольких датасетах. В идеале вы посмотрите на результаты, прочтете выводы и получите некоторое представление об идее, которую проверял эксперимент. А если вы любите приключения, то можете провести эксперименты с помощью блокнотов Jupyter сами.

## Что понадобится для работы

Один из приоритетов этой главы — сократить объем кода и сэкономить усилия, чтобы достичь высокой точности в экспериментах. Есть целый арсенал инструментов, помогающих сделать это путешествие более приятным:

### *TensorFlow Datasets*

Обеспечивают быстрый и легкий доступ примерно к 100 датасетам. В TensorFlow доступны все известные датасеты, начиная с самого маленького MNIST (несколько мегабайт) и заканчивая самыми большими MS COCO, ImageNet и Open Images (несколько сотен гигабайт). Также доступны наборы медицинских данных — Colorectal Histology и Diabetic Retinopathy.

### *TensorBoard*

Около 20 простых в использовании методов визуализации многих аспектов обучения, включая визуализацию графика, отслеживание экспериментов,

просмотр изображений, текста и аудиоданные, которые проходят через сеть во время обучения.

#### *Инструмент What-If*

Параллельно проводит эксперименты на отдельных моделях и выявляет различия, сравнивая их на определенных точках данных. Изменяет отдельные точки данных, чтобы показать, как это влияет на обучение модели.

#### *tf-explain*

Анализирует решения, принимаемые сетью, помогает выявить признаки предвзятости и неточности в датасете. Использует тепловые карты для визуализации частей изображений, на которых активировалась сеть.

#### *Keras Tuner*

Библиотека, созданная для `tf.keras`. Автоматически настраивает гиперпараметры в TensorFlow 2.0.

#### *AutoKeras*

Автоматизирует поиск нейронной архитектуры (Neural Architecture Search, NAS) для различных задач, таких как классификация изображений, текста и аудио, а также обнаружение объектов на изображениях.

#### *AutoAugment*

Использует технологию обучения с подкреплением для аугментации наборов обучающих данных, что способствует повышению точности.

Рассмотрим эти инструменты поближе.

## TensorFlow Datasets

TensorFlow Datasets включает почти 100 готовых к использованию датасетов. Он позволяет создавать высокоэффективные пайплайны ввода данных для обучения моделей TensorFlow. Вместо того чтобы загружать датасеты вручную, а затем выяснить, как читать их метки, TensorFlow Datasets стандартизирует данные и позволяет легко заменить один датасет другим, часто всего одной строчкой кода. Как вы увидите, выполнение таких операций, как деление датасета на обучающую, проверочную и контрольную выборки, также реализуется с помощью всего одной строки кода. В следующей главе мы вернемся к TensorFlow Datasets и изучим его с точки зрения производительности.

Получить полный список доступных датасетов можно с помощью следующей команды (для экономии места ниже приводится короткий фрагмент вывода):

```
import tensorflow_datasets as tfds
print(tfds.list_builders())

['amazon_us_reviews', 'bair_robot_pushing_small', 'bigearthnet', 'caltech101',
'cats_vs_dogs', 'celeb_a', 'imagenet2012', ..., 'open_images_v4',
'oxford_flowers102', 'stanford_dogs', 'voc2007', 'wikipedia', 'wmt_translate',
'xnli']
```

Посмотрим, насколько просто загрузить датасет. Позже мы добавим к этому фрагменту полноценный пайплайн обработки:

```
# Импортировать необходимые пакеты
import tensorflow_datasets as tfds

# Скачать и загрузить в память датасет
dataset = tfds.load(name="cats_vs_dogs", split=tfds.Split.TRAIN)

# Создать высокопроизводительный пайплайн передачи данных
dataset = dataset.map(preprocess).cache().repeat().shuffle(1024).batch(32).
prefetch(tf.data.experimental.AUTOTUNE)

model.fit(dataset, ...)
```



`tfds` создает большое количество индикаторов выполнения, занимающих много места на экране, и использование `tfds.disable_progress_bar()` может оказаться неплохой идеей.

## TensorBoard

TensorBoard — это универсальное решение для всех ваших потребностей в визуализации. Оно предлагает около 20 инструментов для понимания, проверки и улучшения обучения модели.

Чтобы отслеживать прогресс, мы сохраняем значения потерь и точности, полученные в каждой эпохе, а затем строим график их изменения с помощью `matplotlib`. Недостаток такого подхода в том, что все это делается не в режиме реального времени. Обычно мы наблюдаем за ходом обучения, просматривая информацию в текстовом виде, а после завершения обучения должны написать дополнительный код, чтобы построить график в `matplotlib`. TensorBoard предлагает панель мониторинга, которая действует в режиме реального времени (рис. 5.1) и помогает визуализировать все фиксируемые показатели (например, точность и значение функции потерь на этапах обучения и проверки), позволяя следить за ходом обучения. Еще одно преимущество — возможность сравнивать прогресс текущего эксперимента с предыдущим, что позволяет видеть, как изменение параметров повлияло на общую точность.



**Рис. 5.1.** По умолчанию панель мониторинга TensorBoard отображает метрики обучения в реальном времени (затененные линии представляют процесс обучения, выполнявшийся перед текущим)

Чтобы TensorBoard отображал ход обучения модели, нужно зарегистрировать информацию об обучении с `summary writer`:

```
summary_writer = tf.summary.FileWriter('./logs')
```

Чтобы можно было наблюдать за ходом обучения в режиме реального времени, нужно загрузить TensorBoard до начала обучения модели с помощью команд:

```
# Подготовить TensorBoard к работе
%load_ext tensorboard

# Запустить TensorBoard
%tensorboard --logdir ./log
```

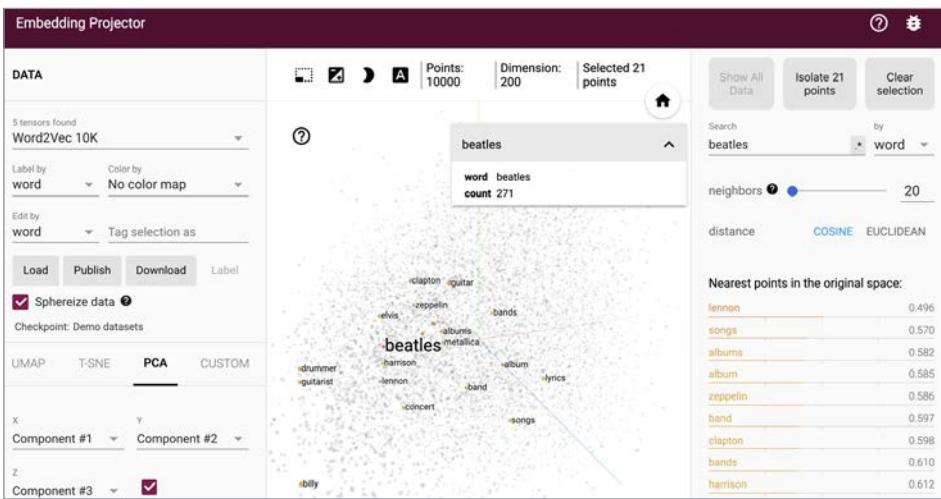
Поскольку многим компонентам TensorFlow требуется визуальный пользовательский интерфейс, они повторно используют TensorBoard, действуя подобно встраиваемым плагинам. Обратите внимание на раскрывающееся меню **Inactive** (Неактивно) в TensorBoard — там вы найдете различные профили и инструменты TensorFlow. Некоторые из них перечислены в табл. 5.1.

TensorBoard можно использовать не только с TensorFlow, но и с другими фреймворками, такими как PyTorch, scikit-learn и др., в зависимости от плагина. Для поддержки плагина нужно организовать запись метаданных, которые требуется визуализировать. Например, TensorBoard поддерживает инструмент TensorFlow

Projector для кластеризации изображений, текста или звука с помощью алгоритма t-SNE (подробно о нем в главе 4). Кроме вызова TensorBoard следует также записать метаданные, такие как эмбеддинги изображений, чтобы TensorFlow Projector мог использовать их для кластеризации, как показано на рис. 5.2.

**Таблица 5.1.** Плагины для TensorBoard

Имя плагина	Описание
Default Scalar (Стандартный скаляр)	Визуализация скалярных данных, таких как точность классификации
Custom Scalar (Настраиваемый скаляр)	Визуализация скалярных данных, определяемых пользователем. Например, разных весов для разных классов, которые могут быть недоступны среди стандартных метрик
Image (Изображение)	Щелкнув на вкладке Image (Изображение), можно просмотреть вывод каждого слоя
Audio (Аудио)	Визуализация аудиоданных
Debugging tools (Инструменты отладки)	Позволяет выполнять отладку визуально и устанавливать точки останова, срабатывающие по условию (например, «тензор содержит Nan или Infinity»)
Graphs (Графы)	Выводит архитектуру модели в графическом виде
Histograms (Гистограммы)	Показывает, как изменяется распределение весов в слоях модели по мере обучения. Это особенно полезно для проверки эффекта сжатия модели с квантованием
Projector (Проектор)	Визуализация прогнозов с помощью t-SNE, метода главных компонент и др.
Text (Текст)	Визуализация текстовых данных
PR curves (Кривые ТП)	Показывает графики кривых «точность/полнота»
Profile (Профилирование)	Оценивает скорость всех операций и слоев в модели
What-If Tool (Инструмент What-If )	Используется для исследования модели путем манипуляции данными и проверки ее качества. Особенно полезен для выявления предвзятости
HParams	Помогает выяснить, какие параметры и при каких значениях являются наиболее важными, позволяет журналировать все параметры, поддерживаемые сервером (подробно обсуждается в репозитории GitHub)
Mesh (Меш)	Визуализация трехмерных данных (включая облака точек)



**Рис. 5.2.** TensorFlow Embedding Projector отображает данные в кластерах (может использоваться как плагин TensorBoard)

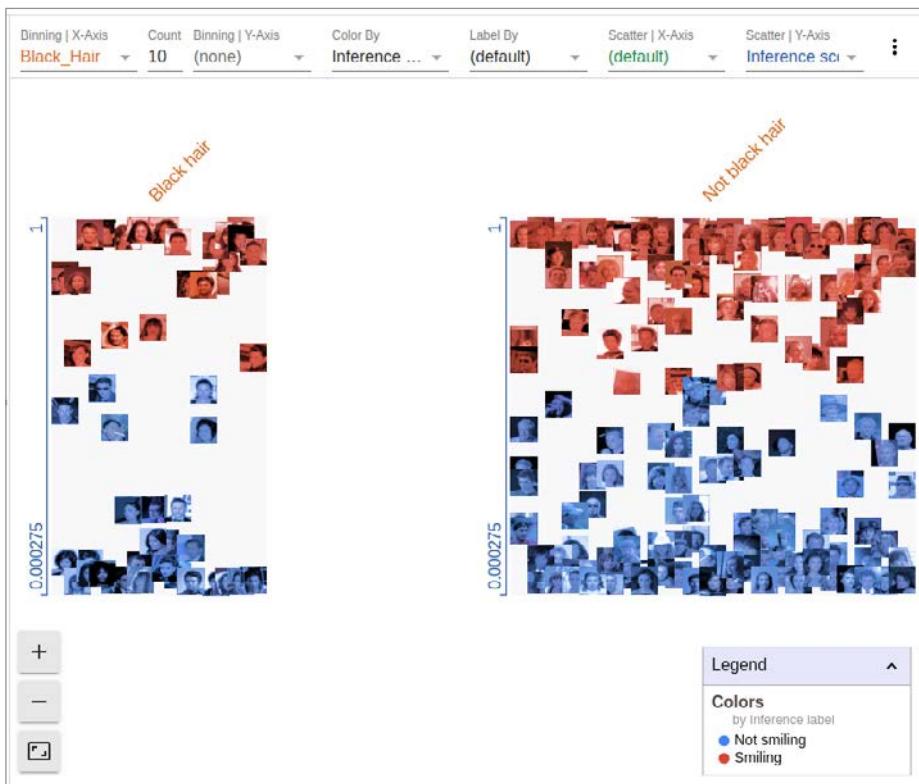
## Инструмент What-If

Представьте, что с помощью средств визуализации можно проверить прогнозы модели. Найти лучший порог для модели, максимизирующий точность и полноту. Проанализировать данные вместе с прогнозами, сделанными нашей моделью, и увидеть, в чем она хороша и какие есть возможности для улучшения. Сравнить две модели и выяснить, какая из них действует лучше. И все это и многое другое можно сделать с помощью нескольких щелчков мыши в браузере. Такая возможность выглядит весьма заманчиво! Инструмент What-If (рис. 5.3 и рис. 5.4), созданный в рамках инициативы Google People + AI Research (PAIR), помогает вскрывать черные ящики моделей ИИ и выяснить, как они работают.

Чтобы воспользоваться инструментом What-If, понадобятся датасет и модель. Как мы видели чуть выше, TensorFlow Datasets упрощает скачивание и загрузку данных (в формате `tfrrecord`). Все, что мы должны сделать, — найти файл данных. Также мы должны сохранить модель в том же каталоге:

```
# Сохранить модель для инструмента What If  
tf.saved_model.save(model, "/tmp/model/1")
```

Следующие строки кода лучше выполнять в локальной системе, а не в блокноте Colab, потому что интеграция Colab и инструмента What-If все еще далека от совершенства.



**Рис. 5.3.** Редактор точек данных инструмента What-If позволяет фильтровать и визуализировать данные в соответствии с учетом аннотаций в датасете и меток, присвоенных классификатором



**Рис. 5.4.** Кривые точность/полнота в разделе Performance and Fairness (Качество и достоверность) инструмента What-If помогают интерактивно подобрать оптимальный порог для получения максимальных оценок точности и полноты

Запустим TensorBoard:

```
$ mkdir tensorboard
$ tensorboard --logdir ./log --alsologtostderr
```

Теперь в новом окне терминала создадим каталог для экспериментов с What-If:

```
$ mkdir what-if-stuff
```

Переместим в него обученную модель и данные. Общая структура каталогов выглядит примерно так:

```
$ tree .
├── colo
│   └── model
│       └── 1
│           ├── assets
│           ├── saved_model.pb
│           └── variables
```

Мы будем испытывать модель с помощью Docker во вновь созданном каталоге:

```
$ sudo docker run -p 8500:8500 \
--mount type=bind,source=/home/{your_username}/what-if-stuff/colo/model/,\
target=/models/colo -e MODEL_NAME=colo -t tensorflow/serving
```

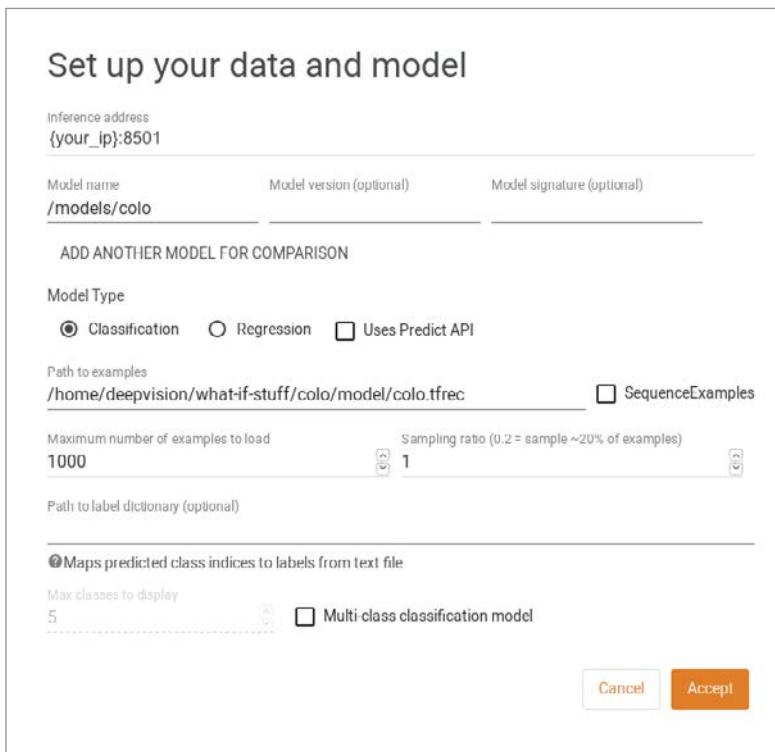
Предупреждение: порт 8500 — это обязательное условие, и все параметры должны вводиться в точности так, как показано в этом примере.

Щелкните на кнопке в правом верхнем углу (с изображением шестеренки) и добавьте значения, перечисленные в табл. 5.2.

**Таблица 5.2.** Настройки для инструмента What-If

Параметр	Значение
Inference address (Адрес инференса)	ip_addr:8500
Model type (Тип модели)	Classification (Классификация)
Path to examples (Путь к примерам)	/home/{ваше_имя_пользователя}/what_if_stuff/colo/models/colo.tfrec (Обратите внимание, что это должен быть абсолютный путь)

Теперь можно открыть инструмент What-If в браузере в TensorBoard, как показано на рис. 5.5.



**Рис. 5.5.** Диалоговое окно с настройками инструмента What-If

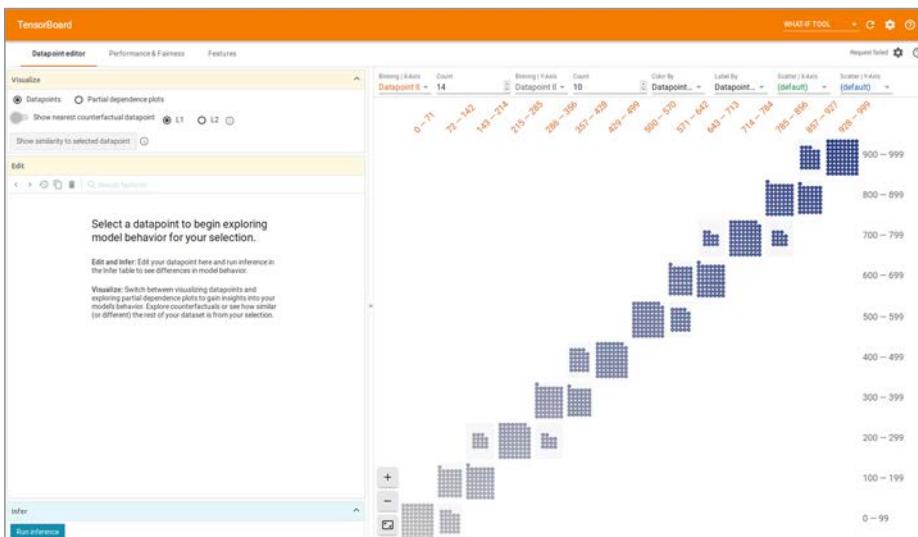
Инструмент What-If можно использовать для визуализации распределения данных в наборах по интервалам, как показано на рис. 5.6, а также для выбора наиболее эффективной модели на определенном датасете из нескольких имеющихся с помощью функции `set_compare_estimator_and_feature_spec`.

```
from witwidget.notebook.visualization import WitConfigBuilder

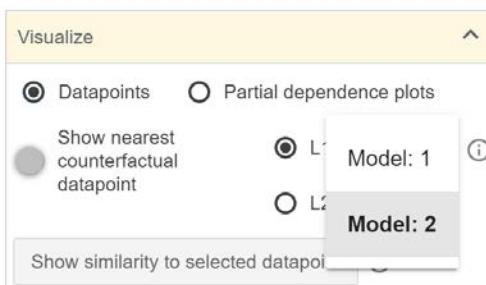
# features -- это контрольные примеры, которые мы хотим загрузить в инструмент
models = [model2, model3, model4]
config_builder =
WitConfigBuilder(test_examples).set_estimator_and_feature_spec(model1, features)

for each_model in models:
    config_builder =
    config_builder.set_compare_estimator_and_feature_spec(each_model, features)
```

Теперь можно загрузить TensorBoard и в разделе **Visualize** (Визуализация) выбрать модель для анализа, как показано на рис. 5.7. Этот инструмент предлагает массу возможностей для исследований.



**Рис. 5.6.** Инструмент What-If позволяет использовать несколько метрик, визуализировать данные и многое другое



**Рис. 5.7.** Выбор модели для анализа в инструменте What-If

## tf-explain

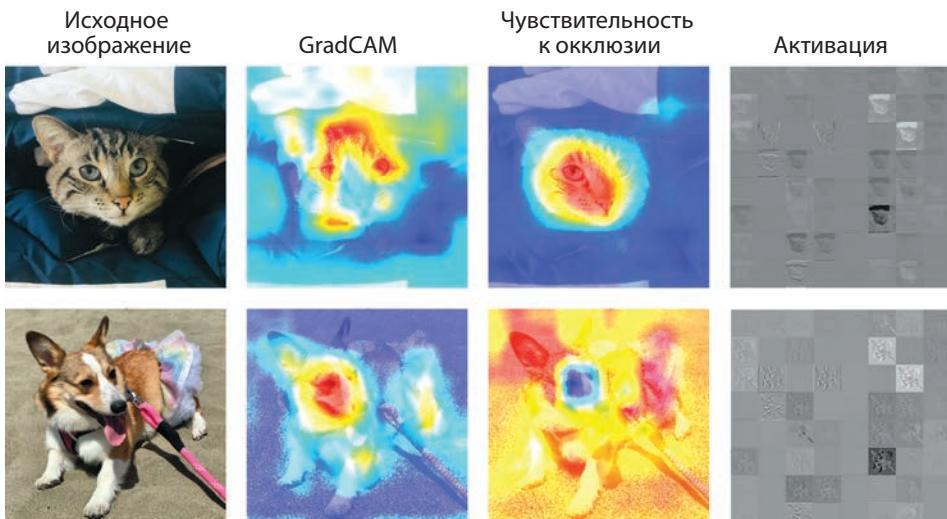
Модели глубокого обучения традиционно являются черными ящиками, и до сих пор мы оценивали качество их работы, наблюдая за вероятностями классов и точностью на этапе проверки. Чтобы сделать эти модели более интерпретируемыми, а их результаты объяснимыми, используем тепловые карты. Показывая области изображения, которые оказывают наибольшее влияние на прогноз, тепловые карты отображают, чему обучилась модель. Например, животное на снегу может распознаваться с высочайшей точностью, но если в датасете есть только снимки этого животного на снегу, модель может решить, что характерной чертой является снег, а не животное. Такой датасет страдает предвзятостью и делает

прогнозы не слишком устойчивыми (и потенциально опасными) после переноса классификатора в реальный мир. Тепловые карты могут быть особенно полезными для выявления такой предвзятости, так как в модель могут просачиваться ложные корреляции, если датасет поддерживается недостаточно тщательно.

`tf-explain` (автор Рафаэль Медек (Raphael Meudec)) помогает понять, как работает нейронная сеть и ее результаты, и выявить предвзятость в датасетах. Можно добавить несколько обратных вызовов на этапе обучения или использовать программный интерфейс этого инструмента для генерации событий TensorFlow, которые позже можно будет загрузить в TensorBoard. Для получения результатов нужно передать в `tf-explain` изображение, его идентификатор в наборе ImageNet и модель. Идентификатор необходим, чтобы `tf.explain` знал, что активируется для этого конкретного класса. В `tf.explain` поддерживается несколько подходов к визуализации:

### *Grad CAM*

Градиентно-взвешенное отображение активации классов (Gradient-weighted Class Activation Mapping, Grad CAM) визуализирует влияние частей изображения на прогноз нейронной сети, просматривая карты активации. Тепловая карта (рис. 5.8) создается на основе градиентов идентификатора из последнего сверточного слоя. Grad CAM – это генератор тепловых карт широкого спектра, устойчивый к шумам, и поэтому может использоваться с множеством моделей CNN.



**Рис. 5.8.** Визуализация результатов работы модели MobileNet с помощью `tf-explain`

### *Чувствительность к окклюзии*

Закрывает часть изображения (выбирая небольшой случайный квадратный участок), чтобы определить, насколько надежна сеть. Если прогноз остается верным, то это говорит о том, что в среднем сеть устойчива. Закрытие самой теплой области изображения (окрашенной в красный цвет на тепловой карте) оказывает наибольшее влияние на прогноз.

### *Активации*

Визуализирует активации сверточных слоев.

Как показано в следующем примере, для создания таких визуализаций требуется совсем немного кода. Сняв видео, сгенерировав отдельные кадры, обработав их с помощью tf-explain в режиме Grad CAM и объединив результаты с исходными кадрами в новый видеоролик, можно получить полное представление о том, как нейронная сеть будет реагировать, например, на изменение ракурса съемки.

```
from tf_explain.core.grad_cam import GradCAM
from tensorflow.keras.applications import MobileNet

model = MobileNet(weights='imagenet', include_top=True)

# Выбрать систему Grad CAM
explainer = GradCAM()

# Изображение для обработки
IMAGE_PATH = 'dog.jpg'
dog_index = 263
img = tf.keras.preprocessing.image.load_img(IMAGE_PATH, target_size=(224, 224))
img = tf.keras.preprocessing.image.img_to_array(img)
data = ([img], None)

# Передать изображение в Grad CAM
grid = explainer.explain(data, model, 'conv1', index)
name = IMAGE_PATH.split(".jpg")[0]
explainer.save(grid, '/tmp', name + '_grad_cam.png')
```

## **Стандартные приемы для экспериментов с машинным обучением**

В первых нескольких главах мы уже рассмотрели вопросы обучения моделей. А сейчас расскажем о некоторых важных моментах, которые следует помнить при проведении экспериментов.

## Проверка данных

Первое серьезное препятствие при проверке данных — определение структуры. В TensorFlow Datasets этот шаг относительно прост: у всех доступных датасетов одинаковый формат и структура и они могут использоваться эффективно. Все, что нужно сделать, — загрузить датасет в инструмент What-If и использовать уже поддерживаемые способы проверки данных. Например, датасет SMILE можно визуализировать в соответствии с аннотациями, такими как изображения людей в очках и без очков, как показано на рис. 5.9. Изображения людей без очков более распространены, что говорит о наличии предвзятости в данных из-за их несбалансированности. Эту проблему можно решить, изменив с помощью инструмента соответствующие веса.



**Рис. 5.9.** Анализ структуры данных на основе прогнозов и реальных категорий

## Разбиение данных на обучающую, проверочную и контрольную выборки

Разбиение датасета на обучающую, проверочную и контрольную выборки — важный шаг, потому что качество работы классификатора оценивается по результатам классификации данных, которых он до этого не видел (то есть данных из контрольной выборки). TensorFlow Datasets упрощает скачивание, загрузку

и разделение датасета на эти три части. Некоторые датасеты уже разбиты на три части по умолчанию. Но при желании данные можно разбить по процентам. Следующий код показывает использование выборок по умолчанию:

```
dataset_name = "cats_vs_dogs"
train, info_train = tfds.load(dataset_name, split=tfds.Split.TRAIN,
                             with_info=True)
```

В датасете `cats_vs_dogs` в `tfds` предопределена только обучающая выборка. Аналогично некоторые другие датасеты в TensorFlow Datasets тоже не имеют предопределенной проверочной выборки. При работе с такими датасетами можно взять небольшую часть из предопределенной обучающей выборки и использовать ее как проверочную выборку. Также разбиение датасета с помощью функции `weighted_splits` обеспечит рандомизацию и перемешивание данных перед созданием выборок:

```
# Датасет для загрузки
dataset_name = "cats_vs_dogs"

# Разбить данные на обучающую (80%), проверочную (10%) и контрольную (10%)
# выборки
split_train, split_val, split_test = tfds.Split.TRAIN.subsplit(weighted=
    [80, 10, 10])
train, info_train = tfds.load(dataset_name, split=split_train, with_info=True)
val, info_val = tfds.load(dataset_name, split=split_val, with_info=True)
test, info_test = tfds.load(dataset_name, split=split_test, with_info=True)
```

## Ранняя остановка

Ранняя остановка (early stopping) помогает избежать переобучения сети, определяя момент, когда дальнейшее обучение перестает улучшать точность. Представьте, что модель настроена на обучение в течение 1000 эпох и достигла точности 90 % на 10-й эпохе, и затем следующие 10 эпох она перестала улучшаться. Дальнейшее обучение будет пустой тратой ресурсов. Если количество эпох, в ходе которых не наблюдается значимого улучшения точности, превышает заданный порог (параметр `patience`), то обучение прекращается, даже если потрачено меньше эпох, чем задано.

Ранняя остановка определяет момент, когда обучение перестает приносить пользу, и прекращает его. Метрику, оценивающую прогресс, можно изменить с помощью параметра `monitor` и добавить раннюю остановку в список обратных вызовов модели:

```
# Определить обратный вызов для оценки необходимости ранней остановки
earliest_stop_callback = tf.keras.callbacks.EarlyStopping(monitor='val_acc',
                                                          min_delta=0.0001, patience=10)
# Добавить обратный вызов в обучаемую модель
model.fit_generator(... callbacks=[earliest_stop_callback])
```

## Воспроизводимость экспериментов

Обучите сеть один раз. Затем обучите ее снова, не меняя код или параметры. Можно заметить, что точность в этих двух экспериментах будет немного отличаться. Эта разница связана со случайными величинами. Чтобы сделать эксперименты воспроизводимыми в разных сериях, нужно контролировать эту случайность. Инициализация весов моделей, перемешивание данных и т. д. — во всех этих операциях используются алгоритмы рандомизации. Как известно, генераторы случайных чисел можно заставить воспроизводить одну и ту же последовательность снова и снова, инициализируя их одним и тем же начальным числом. Именно этим мы и займемся. В разных фреймворках предусмотрены свои способы установки начального числа. Вот некоторые из них:

```
# Установка начального числа в Tensorflow
tf.random.set_seed(1234)

# Установка начального числа в Numpy
import numpy as np
np.random.seed(1234)

# Установка начального числа в Keras
seed = 1234
fit(train_data, augment=True, seed=seed)
flow_from_dataframe(train_dataframe, shuffle=True, seed=seed)
```



Начальное число необходимо устанавливать во всех используемых фреймворках и библиотеках, потому что начальные числа не передаются между фреймворками.

## Пример сквозного пайплайна глубокого обучения

Объединим несколько инструментов и создадим основу для пайплайна, куда будем добавлять параметры, слои, функции и многое другое, чтобы понять, что происходит. Код, представленный ниже, можно найти в репозитории книги (<https://github.com/PracticalDL/Practical-Deep-Learning-Book>) и запустить в браузере с более чем 100 датасетами, благодаря поддержке Colab. Код можно изменить, приспособив для своих задач классификации.

### Простой пайpline переноса обучения

Создадим сквозной пайpline для переноса обучения.

```
# Импортировать необходимые пакеты
import tensorflow as tf
```

```
import tensorflow_datasets as tfds

# в tfds создает большое количество индикаторов выполнения, занимающих много
# места на экране, поэтому отключим их
tfds.disable_progress_bar()

tf.random.set_seed(1234)

# Переменные
BATCH_SIZE = 32
NUM_EPOCHS= 20
IMG_H = IMG_W = 224
IMG_SIZE = 224
LOG_DIR = './log'
SHUFFLE_BUFFER_SIZE = 1024
IMG_CHANNELS = 3

dataset_name = "oxford_flowers102"

def preprocess(ds):
    x = tf.image.resize_with_pad(ds['image'], IMG_SIZE, IMG_SIZE)
    x = tf.cast(x, tf.float32)
    x = (x/127.5) - 1
    return x, ds['label']

def augmentation(image,label):
    image = tf.image.random_brightness(image, .1)
    image = tf.image.random_contrast(image, lower=0.0, upper=1.0)
    image = tf.image.random_flip_left_right(image)
    return image, label

def get_dataset(dataset_name):
    split_train, split_val = tfds.TFRecordDataset('oxford_flowers102').split([9,1])
    train, info_train = tfds.load(dataset_name, split=split_train,
                                 with_info=True)
    val, info_val = tfds.load(dataset_name, split=split_val, with_info=True)
    NUM_CLASSES = info_train.features['label'].num_classes
    assert NUM_CLASSES >= info_val.features['label'].num_classes
    NUM_EXAMPLES = info_train.splits['train'].num_examples * 0.9
    IMG_H, IMG_W, IMG_CHANNELS = info_train.features['image'].shape
    train = train.map(preprocess).cache()
        .repeat().shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE)
    train = train.map(augmentation)
    train = train.prefetch(tf.data.experimental.AUTOTUNE)
    val = val.map(preprocess).cache().repeat().batch(BATCH_SIZE)
    val = val.prefetch(tf.data.experimental.AUTOTUNE)
    return train, info_train, val, info_val, IMG_H, IMG_W, IMG_CHANNELS,
           NUM_CLASSES, NUM_EXAMPLES

train, info_train, val, info_val, IMG_H, IMG_W, IMG_CHANNELS, NUM_CLASSES,
NUM_EXAMPLES = get_dataset(dataset_name)

# Включить обратные вызовы TensorBoard
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(LOG_DIR,
                                                    histogram_freq=1,
                                                    write_graph=True,
                                                    write_grads=True,
                                                    batch_size=BATCH_SIZE,
                                                    write_images=True)

def transfer_learn(train, val, unfreeze_percentage, learning_rate):
    mobile_net = tf.keras.applications.ResNet50(
        input_shape=(IMG_SIZE, IMG_SIZE, IMG_CHANNELS),
        include_top=False)
    mobile_net.trainable=False

    # Разморозить некоторые слои, в зависимости от используемого датасета
    num_layers = len(mobile_net.layers)
    for layer_index in range(int(num_layers - unfreeze_percentage*num_layers),
                             num_layers ):
        mobile_net.layers[layer_index].trainable = True
    model_with_transfer_learning = tf.keras.Sequential(
        [mobile_net,
         tf.keras.layers.GlobalAveragePooling2D(),
         tf.keras.layers.Flatten(),
         tf.keras.layers.Dense(64),
         tf.keras.layers.Dropout(0.3),
         tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')],)

    model_with_transfer_learning.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
        loss='sparse_categorical_crossentropy',
        metrics=["accuracy"])
    model_with_transfer_learning.summary()
    earlystop_callback = tf.keras.callbacks.EarlyStopping(
        monitor='val_accuracy',
        min_delta=0.0001,
        patience=5)
    model_with_transfer_learning.fit(train,
                                     epochs=NUM_EPOCHS,
                                     steps_per_epoch=int(NUM_EXAMPLES/
                                                         BATCH_SIZE),
                                     validation_data=val,
                                     validation_steps=1,
                                     validation_freq=1,
                                     callbacks=[tensorboard_callback,
                                                earlystop_callback])

    return model_with_transfer_learning

# Запустить TensorBoard
%tensorboard --logdir ./log

# Выбрать % последних слоев для дообучения в ходе переноса обучения.
# Эти слои - ближайшие к выходным слоям.
unfreeze_percentage = .33
learning_rate = 0.001

model = transfer_learn(train, val, unfreeze_percentage, learning_rate)
```

## Простой пайплайн создания сети

Помимо переноса обучения с помощью предварительно обученных моделей мы также можем экспериментировать и набираться опыта, создавая свои сети. Используем их вместо моделей, создаваемых предыдущим кодом переноса обучения:

```
def create_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
            input_shape=(IMG_SIZE, IMG_SIZE, IMG_CHANNELS)),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(rate=0.3),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(rate=0.3),
        tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
    ])
    return model

def scratch(train, val, learning_rate):
    model = create_model()
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
        learning_rate),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

    earlystop_callback = tf.keras.callbacks.EarlyStopping(
        monitor='val_accuracy',
        min_delta=0.0001,
        patience=5)

    model.fit(train,
        epochs=NUM_EPOCHS,
        steps_per_epoch=int(NUM_EXAMPLES/BATCH_SIZE),
        validation_data=val,
        validation_steps=1,
        validation_freq=1,
        callbacks=[tensorboard_callback, earlystop_callback])
    return model
```

Теперь поэкспериментируем с нашим пайплайном.

## Влияние гиперпараметров на точность

В этом разделе изменим различные параметры пайплайна глубокого обучения — от количества слоев для тонкой настройки до выбора функции активации —

и посмотрим, как они влияют на точность на этапе проверки. Также посмотрим, как некоторые параметры влияют на скорость обучения и время, нужное для достижения максимальной точности (то есть сходимости).

Так выглядит наше экспериментальное окружение:

- чтобы сократить время для экспериментов, мы использовали в этой главе самую быструю архитектуру — MobileNet;
- чтобы еще больше ускорить обучение, мы уменьшили размеры входных изображений до  $128 \times 128$  пикселей, но в продакшене советуем использовать изображения с большими размерами (не менее  $224 \times 224$ );
- используется прием ранней остановки, если точность модели не увеличивается в течение 10 эпох подряд;
- при работе с моделями переноса обучения размораживаем последние 33 % слоев;
- скорость обучения равна 0,001 и используется оптимизатор Adam;
- в большинстве экспериментов, если явно не указано иное, для контроля используется датасет Oxford Flowers 102; мы выбрали этот датасет, потому что он достаточно сложен для обучения из-за большого количества классов (102) и сходства между многими классами, что заставляет обучаемую сеть развивать точное распознавание признаков;
- для корректного сравнения результатов мы будем выбирать максимальное значение точности в конкретном эксперименте и нормализовать все другие значения точности в этом эксперименте относительно этого максимального значения.

На основе этих и других экспериментов мы составили список полезных советов, они пригодятся вам в будущих приключениях в мире обучения моделей. Ищите советы в репозитории нашей книги (<https://github.com/PracticalDL/Practical-Deep-Learning-Book>) вместе с интерактивными примерами. А если хотите поделиться своими советами, присылайте их в Твиттер @PracticalDLBook или туда же в репозиторий книги на GitHub.

## Сравнение переноса обучения и обучения с нуля

### Условия эксперимента

На одном и том же датасете обучить две модели: одну с переносом обучения, а другую с нуля.

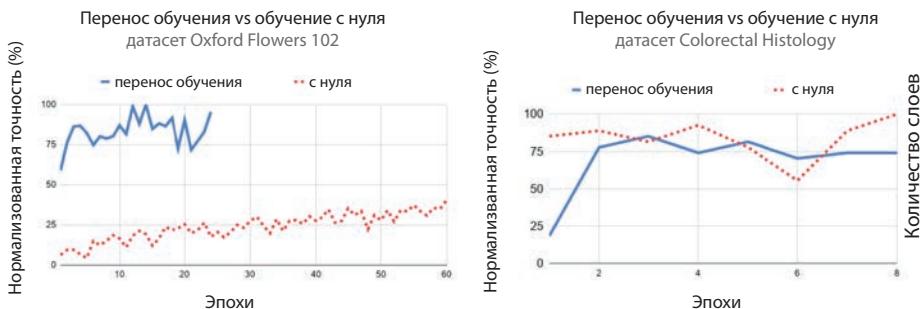
### Используемые датасеты

Oxford Flowers 102, Colorectal Histology.

### *Используемые архитектуры*

Предварительно обученная модель MobileNet, своя нестандартная модель.

Результаты эксперимента показаны на рис. 5.10.



**Рис. 5.10.** Сравнение переноса обучения и обучения с нуля на разных датасетах

Основные выводы:

- Перенос обучения позволяет быстрее достичь высокой точности за счет повторного использования ранее изученных признаков.
- Как предполагается, перенос обучения (основанный на использовании моделей, предварительно обученных на наборе ImageNet) должен давать хорошие результаты, когда целевой датасет также представляет собой коллекцию естественных изображений. Но закономерности, зафиксированные нижними слоями сети, на удивление хорошо работают с датасетами, отличными от ImageNet. Это не гарантирует получение лучших результатов, но они будут весьма близки к лучшим. Если изображения в целевом датасете близки к изображениям реального мира, на которых была обучена модель, то мы получим относительно быстрое достижение высокой точности.

## **Влияние количества слоев для тонкой настройки при переносе обучения**

### *Условия эксперимента*

Изменять процент дообучаемых слоев от 0 до 100 %.

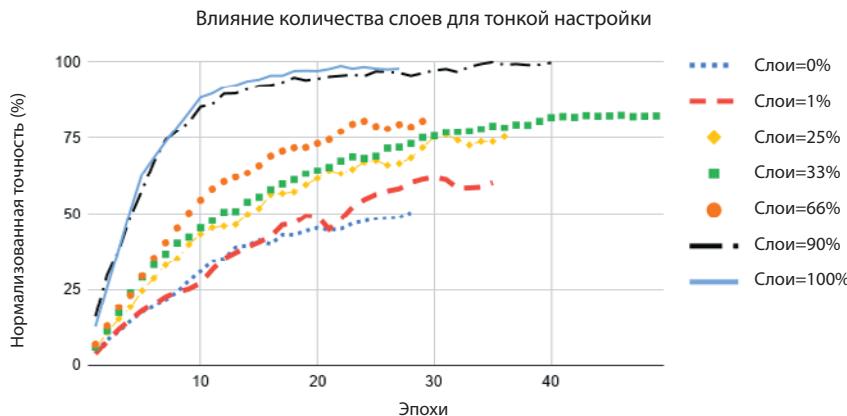
### *Используемые датасеты*

Oxford Flowers 102.

### *Используемые архитектуры*

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.11.



**Рис. 5.11.** Влияние количества слоев для тонкой настройки при переносе обучения

Основные выводы:

- Чем больше слоев подвергается тонкой настройке, тем меньше эпох требуется для достижения сходимости и тем выше точность.
- Чем больше слоев подвергается тонкой настройке, тем больше времени требуется на обучение в каждой эпохе из-за большего количества вычислений и обновлений.
- В случаях с датасетами для анализа изображений, в которых потребовалось конкретизировать задачу: чем больше слоев будет разморожено, тем точнее получится модель.

## Влияние объема данных на перенос обучения

### Условия эксперимента

Добавлять изображения по одному и анализировать качество обучения.

### Используемые датасеты

Датасет cats\_vs\_dogs.

### Используемые архитектуры

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.12.



**Рис. 5.12.** Влияние объемов данных в категориях на точность модели

Основные выводы:

- Даже с тремя изображениями в каждом классе модель смогла достичь точности, близкой к 90 %. Это показывает, насколько мощным может быть перенос обучения в плане снижения требований к данным.
- Поскольку в ImageNet есть несколько кошек и собак, то сети, предварительно обученные на наборе ImageNet, прекрасно подходят для тонкой настройки на нашем датасете. Для сетей, обученных на более сложных датасетах, например Oxford Flowers 102, требуется изучить гораздо больше изображений, чтобы достичь аналогичной точности.

## Влияние скорости обучения

*Условия эксперимента*

Провести обучение со скоростями 0,1, 0,01, 0,001 и 0,0001.

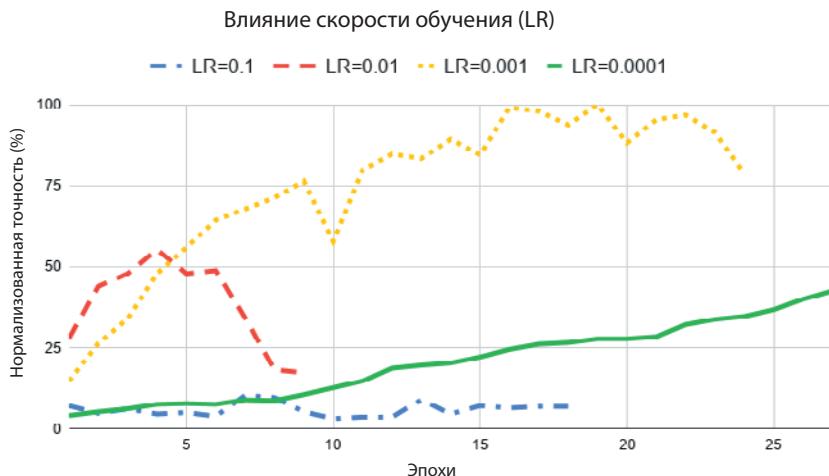
*Используемые датасеты*

Oxford Flowers 102.

*Используемые архитектуры*

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.13.



**Рис. 5.13.** Влияние скорости обучения модели на ее точность и время до достижения сходимости

Основные выводы:

- При слишком высокой скорости обучения модель может никогда не достигнуть сходимости.
- При слишком низкой скорости обучения для достижения сходимости может потребоваться много времени.

Баланс крайне важен для быстрого обучения.

## Влияние оптимизатора

### Условия эксперимента

Попробовать использовать доступные оптимизаторы, включая AdaDelta, AdaGrad, Adam, Gradient Descent, Momentum и RMSProp.

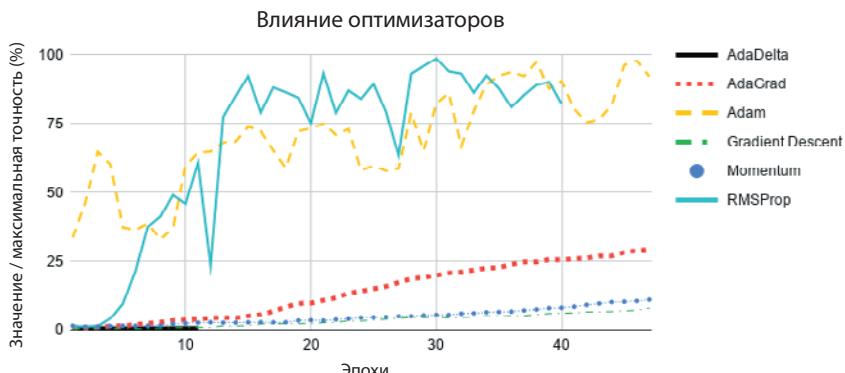
### Используемые датасеты

Oxford Flowers 102.

### Используемые архитектуры

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.14.



**Рис. 5.14.** Влияние разных оптимизаторов на скорость сходимости

Основные выводы:

- Оптимизатор Adam – отличный выбор, когда требуется высокая скорость сходимости при высокой точности.
- RMSProp лучше подходит для рекуррентных сетей.

## Влияние размера пакета

*Условия эксперимента*

Опробовать пакеты разных размеров, равных степени двойки.

*Используемые датасеты*

Oxford Flowers 102.

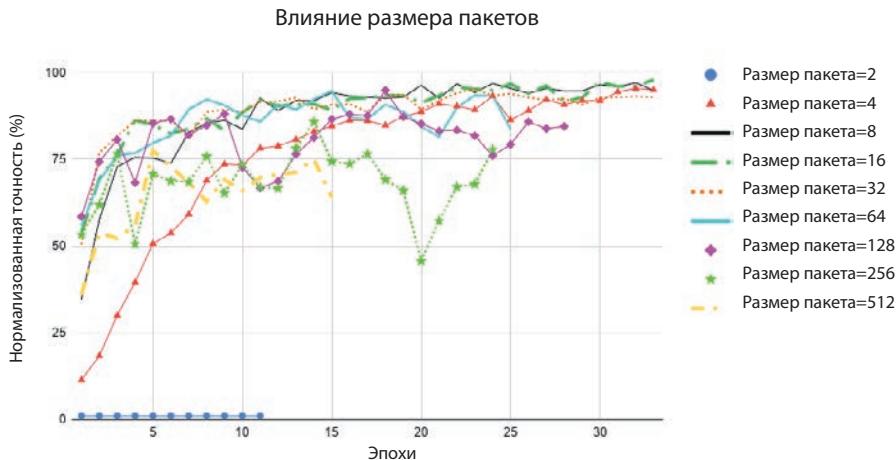
*Используемые архитектуры*

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.15.

Основные выводы:

- Чем больше размер пакета, тем хуже стабильность результатов от эпохи к эпохе – больше их амплитуда. Но при этом увеличивается точность и эффективность использования GPU, что сокращает продолжительность каждой эпохи.
- Слишком маленький размер партии замедляет рост точности.
- 16/32/64 – хорошие начальные размеры пакетов.



**Рис. 5.15.** Влияние размера пакетов на точность и скорость сходимости

## Влияние изменения размеров

Условия эксперимента

Сравнить результаты обучения на изображениях размером  $128 \times 128$  и  $224 \times 224$ .

*Используемые датасеты*

Oxford Flowers 102.

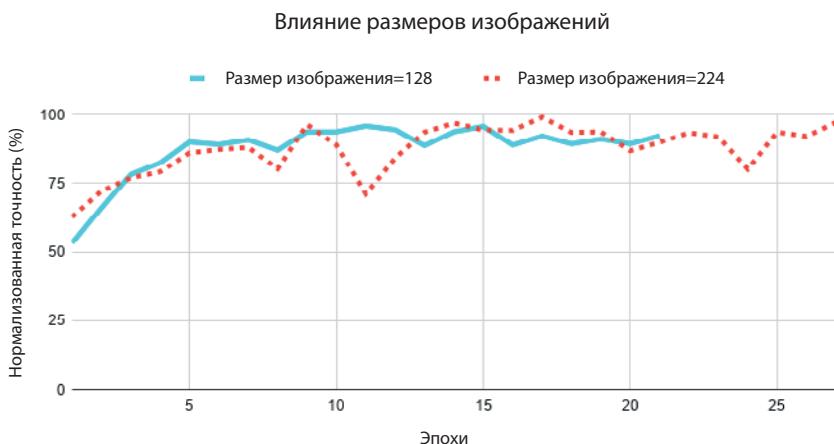
*Используемые архитектуры*

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.16.

Основной вывод:

- Даже когда осталась только треть пикселей, точность на этапе проверки упала незначительно. С одной стороны, это свидетельствует о надежности сверточных сетей. Отчасти это может быть связано с тем, что в датасете Oxford Flowers 102 цветы сняты крупным планом. Для датасетов, в которых объекты занимают гораздо меньшие части изображений, результаты могут оказаться хуже.



**Рис. 5.16.** Влияние размеров изображений на точность

## Влияние изменения соотношения сторон на перенос обучения

*Условия эксперимента*

Взять изображения с разным соотношением сторон (ширины к высоте) и преобразовать их в квадратную форму (с соотношением сторон 1:1).

*Используемые датасеты*

Cats vs. Dogs.

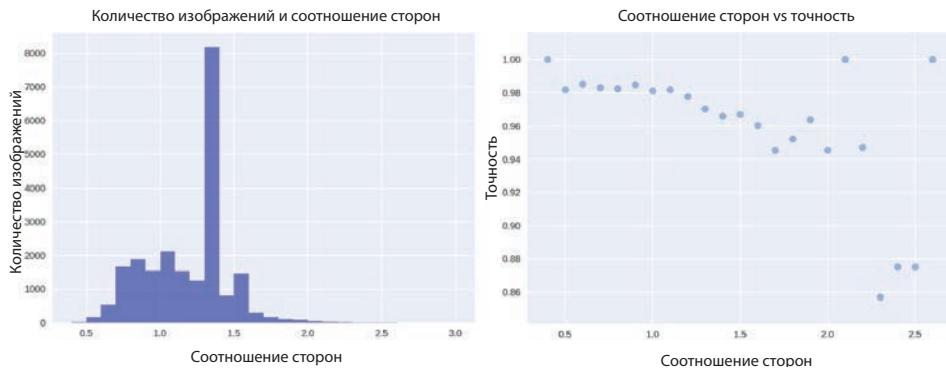
*Используемые архитектуры*

Предварительно обученная модель MobileNet.

Результаты эксперимента показаны на рис. 5.17.

Основные выводы:

- Чаще всего встречаются изображения с соотношением сторон 4:3; то есть 1,33, тогда как нейронные сети обычно обучаются на изображениях с соотношением сторон 1:1.
- Нейронные сети относительно устойчивы к незначительным изменениям соотношения сторон до квадратной формы: соотношения до 2,0 дают вполне достойные результаты.



**Рис. 5.17.** Распределение изображений с разным соотношением сторон и соответствующая точность

## Инструменты автоматизации настройки моделей для достижения максимальной точности

Начиная с XIX века автоматизация всегда приводила к увеличению производительности. В этом разделе рассмотрим инструменты, помогающие автоматизировать поиск лучшей модели.

### Keras Tuner

При таком большом количестве возможных комбинаций гиперпараметров, доступных для настройки, создание оптимальной модели может оказаться утомительным процессом. Часто два или более параметров могут совокупно влиять на общую скорость сходимости, а также на точность на этапе проверки, поэтому настройка параметров по одному может не дать желаемого эффекта. И если любопытство возьмет верх, то можно поэкспериментировать со всеми гиперпараметрами сразу.

Keras Tuner позволяет автоматизировать подбор гиперпараметров. Для этого следует определить алгоритм поиска, потенциальные значения для каждого параметра (например, дискретные значения или диапазон) и цель для максимизации (например, точность на этапе проверки), после чего можно откинуться на спинку стула и наблюдать, как протекает обучение. Keras Tuner проведет множество экспериментов, изменения и сохраняя параметры, с которыми модель показала наилучшие результаты. Пример кода ниже, адаптированный из документации Keras Tuner, демонстрирует поиск по разным архитектурам модели (различающимся количеством слоев от 2 до 10), а также значениям скорости обучения (от 0,1 до 0,001):

```
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

from kerastuner.engine.hypermodel import HyperModel
from kerastuner.engine.hyperparameters import HyperParameters

# Входные данные
(x, y), (val_x, val_y) = keras.datasets.mnist.load_data()
x = x.astype('float32') / 255.
val_x = val_x.astype('float32') / 255.

# Определение гиперпараметров
hp = HyperParameters()
hp.Choice('learning_rate', [0.1, 0.001])
hp.Int('num_layers', 2, 10)

# Определение модели с изменяемым количеством слоев
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28)))
    for _ in range(hp.get('num_layers')):
        model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(
        optimizer=keras.optimizers.Adam(hp.get('learning_rate')),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
    return model

hypermodel = RandomSearch(
    build_model,
    max_trials=20, # Количество допустимых комбинаций
    hyperparameters=hp,
    allow_new_entries=False,
    objective='val_accuracy')

hypermodel.search(x=x,
                  y=y,
                  epochs=5,
                  validation_data=(val_x, val_y))

# Показать параметры, соответствующие лучшей модели
hypermodel.results_summary()
```

Примерно таким будет вывод после каждого эксперимента:

```
> Hp values:
|-learning_rate: 0.001
|-num_layers: 6
```

Name	Best model	Current model
accuracy	0.9911	0.9911
loss	0.0292	0.0292
val_loss	0.227	0.227
val_accuracy	0.9406	0.9406

В конце серии экспериментов выводится общая сводка результатов и дополнительные метаданные:

```
Hypertuning complete - results in ./untitled_project
[Results summary]
|-Results in ./untitled_project
|-Ran 20 trials
|-Ran 20 executions (1 per trial)
|-Best val_accuracy: 0.9406
```

Еще одно большое преимущество — возможность наблюдать за экспериментами в режиме реального времени и получать информацию о ходе их выполнения. Для этого зайдите на страницу <http://keras-tuner.appspot.com>, получите ключ API (из Google App Engine) и введите следующую строку в нашу программу вместе с действительным ключом API:

```
tuner.enable_cloud(api_key=<ключ_API>)
```

Пространство всех возможных комбинаций может оказаться чересчур обширным, поэтому случайный поиск будет предпочтительнее, чем поиск по сетке, так как это более практичный способ получить хорошее решение при ограниченном времени на выполнение экспериментов. Но есть более быстрые способы, в том числе Hyperband (Lisha Li (Лиша Ли) и др.), реализация которого также доступна в Keras Tuner.

Для решения задач компьютерного зрения Keras Tuner предлагает готовые к использованию настраиваемые приложения, например HyperResNet.

## AutoAugment

Другой пример гиперпараметров — аугментация. Какие приемы аугментации датасета лучше использовать? До какой степени производить аугментацию данных? Не ухудшит ли ситуацию чрезмерная аугментация? Решение этих вопросов лучше не оставлять людям, а передать их ИИ. Модуль AutoAugment использует обучение с подкреплением и выбирает параметры аугментации (например, смещение, вращение, сдвиг), а также применяемые вероятности и величины, чтобы максимизировать точность на этапе проверки. (Этот метод был применен Экином Кубуком (Ekin Cubuk) и другими для оценки параметров аугментации

ImageNet.) Исследовав лучшую комбинацию параметров обогащения ImageNet, можно легко применить ее к нашей задаче.

Применить ранее выявленную стратегию аугментации ImageNet довольно просто:

```
from PIL import Image
from autoaugment import ImageNetPolicy

img = Image.open("cat.jpg")
policy = ImageNetPolicy()
imgs = [policy(img) for _ in range(8) ]
```

Результаты показаны на рис. 5.18.



**Рис. 5.18.** Варианты аугментации датасета ImageNet, подобранные путем обучения с подкреплением

## AutoKeras

Искусственный интеллект все шире используется для автоматизации рабочих мест, поэтому неудивительно, что его наконец научили автоматизировать и проектирование архитектур ИИ. В подходах к поиску нейронной архитектуры (Neural Architecture Search, NAS) используют обучение с подкреплением, когда архитектурные блоки объединяются, пока не максимизируют целевую функцию, то есть точность на этапе проверки. Все современные сети созданы с использованием NAS, без прямого участия человека. Исследования в этой области показали многообещающие результаты в 2017 году и начали ускоряться в 2018 году. Теперь и мы с помощью AutoKeras (Haifeng Jin (Хайфэн Цзинь) и др.) можем относительно легко использовать этот передовой прием на наших конкретных датасетах.

Для создания новых архитектур моделей с помощью AutoKeras нужно предоставить размеченные изображения, а также определить время, в течение которого создание должно быть выполнено. Для поиска оптимальной архитектуры AutoKeras использует несколько алгоритмов оптимизации, включая байесовский:

```
!pip3 install autokeras
!pip3 install graphviz
from keras.datasets import mnist
from autokeras.image.image_supervised import ImageClassifier
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(x_train.shape + (1,))
x_test = x_test.reshape(x_test.shape + (1,))

clf = ImageClassifier(path=".",
verbose=True, augment=False)
clf.fit(x_train, y_train, time_limit= 30 * 60) # 30 минут
clf.final_fit(x_train, y_train, x_test, y_test, retrain=True)
y = clf.evaluate(x_test, y_test)
print(y)

# Сохранить модель в файл
clf.export_autokeras_model("model.pkl")

visualize('.)
```

После обучения всегда интересно узнать, как выглядит архитектура новой модели. В отличие от большинства архитектур, которые мы обычно видим, автоматически созданная модель выглядит довольно запутанной. Но мы верим, что она обеспечивает высокую точность.

## Итоги

Мы рассмотрели инструменты и методы, позволяющие увеличить точность сверточной сети. Создавая основу для итеративных экспериментов, вы узнали, как настройка гиперпараметров может способствовать повышению качества прогнозов. Определив, насколько широким может быть круг гиперпараметров, мы рассмотрели автоматизированные методы их настройки с помощью AutoKeras, AutoAugment и Keras Tuner. Самое замечательное, что основной код, представленный в этой главе и объединяющий несколько инструментов в одном файле Colab, доступен в репозитории <https://github.com/PracticalDL/Practical-Deep-Learning-Book>. Его легко адаптировать для работы с более чем 100 датасетами заменой всего одной строки и запустить в браузере. Еще мы составили список полезных советов вместе с интерактивными примерами, чтобы дать вашей модели дополнительные преимущества. Надеемся, что информация из этой главы сделает ваши модели более надежными и объяснимыми, уменьшив их предвзятость и позволит внести свой вклад в развитие ответственного ИИ.

## ГЛАВА 6

---

# Увеличение скорости и эффективности TensorFlow: удобный чек-лист

В жизни важно уметь обходиться тем, что имеешь, и помогает в этом оптимизация — то есть разумное использование своих ресурсов. Может, мы на самом деле хотим купить Ferrari, но бюджет позволяет лишь Hyundai. Но знаете что? Правильно настроив работу всех узлов и агрегаторов, можно и на Hyundai принять участие в гонке NASCAR!

Что это значит с точки зрения глубокого обучения? Google с его инженерной мощью и стойками TPU, способными вскипятить океан, установил рекорд скорости, обучив свою сеть на наборе ImageNet всего за 30 минут. И все же всего несколько месяцев спустя разношерстная команда из трех исследователей (Эндрю Шоу (Andrew Shaw), Ярослав Булатов (Yaroslav Bulatov) и Джереми Ховард (Jeremy Howard)) с 40 долларами в кармане и доступом к публичному облаку смогли обучить свою сеть на наборе ImageNet всего за 18 минут!

Эти примеры показывают, что количество имеющихся ресурсов не так важно, как их использование. Все дело в том, чтобы уметь делать больше с меньшими затратами. И эта глава послужит вам удобным чек-листом возможных оптимизаций производительности на всех этапах глубокого обучения, который еще не раз пригодится на протяжении книги. В этой главе мы обсудим оптимизацию подготовки данных, чтения данных, аугментации данных, обучения и, наконец, инференса.

Эта история начинается и заканчивается двумя словами...

## Голодание GPU

Разработчики ИИ часто задают вопрос: почему обучение моей сети такое медленное? Чаще всего причина кроется в голодании (starvation) GPU.

Графические процессоры — основа глубокого обучения. Нередко они являются самым дорогим компонентом компьютерной системы. По этой причине их нужно использовать с максимальной эффективностью. Это означает, что GPU не должен ждать (голодать), пока другие компоненты передадут данные для обработки. Когда GPU будет готов к работе, подготовленные данные должны быть уже доступны и ждать своей очереди. Однако в реальности узким местом часто оказываются центральный процессор (CPU), память и устройства хранения, что влечет неоптимальное использование GPU. Иначе говоря, следует организовать работу так, чтобы узким местом был GPU, а не другие компоненты.

Покупка дорогих GPU за тысячи долларов может быть оправданна, только если GPU является узким местом. В противном случае это будет пустая траты денег.

Рассмотрим рис. 6.1. В пайплайне глубокого обучения CPU и GPU работают в паре, передавая данные друг другу. CPU читает данные, выполняет предварительную обработку, включая аугментацию, а затем передает их в GPU для обучения сети. Их сотрудничество напоминает эстафету, с той лишь разницей, что один из участников эстафеты — олимпийский чемпион, ожидающий, пока посредственный бегун передаст эстафетную палочку. Чем дольше GPU простоявает, тем больше ресурсов расходуется впустую.



**Рис. 6.1.** Голодание GPU: графический процессор простоявает, ожидая, пока CPU закончит подготовку данных

Большая часть этой главы посвящена приемам, помогающим сократить время простоя GPU и CPU.

Возникает логичный вопрос: как узнать, голодает ли GPU? В этом помогут два удобных инструмента:

### **nvidia-smi**

Эта команда показывает статистику использования GPU.

### *Профилировщик TensorFlow + TensorBoard*

Эта пара выводит временную шкалу выполнения программы в TensorBoard.

### **nvidia-smi**

Имя **nvidia-smi** происходит от NVIDIA System Management Interface (интерфейс управления системой NVIDIA). Эта команда выводит подробную

статистику о наших драгоценных графических процессорах, включая объем памяти, степень загруженности, температуру, мощность и многое другое. Мечта компьютерного гика.

Давайте попробуем воспользоваться ею:

```
$ nvidia-smi
```

Результат показан на рис. 6.2.

```
Every 0.5s: nvidia-smi                               deepvision: Tue Aug 20 04:05:01 2019

Tue Aug 20 04:05:01 2019
+-----+
| NVIDIA-SMI 410.78     Driver Version: 410.78      CUDA Version: 10.0 |
| GPU  Name        Persistence-M | Bus-Id      Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====|
|  0  GeForce GTX 108... Off  | 00000000:01:00.0 On |          N/A   |
| 31%   57C    P2    104W / 250W | 10762MiB / 11177MiB | 51%      Default |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU  PID  Type  Process name             Usage      |
|=====+=====+=====+=====
|  0    1060   G  /usr/lib/xorg/Xorg           184MiB  |
|  0    1331   G  /usr/bin/gnome-shell         211MiB  |
|  0    20342   C  /usr/bin/python3            10351MiB |
|  0    28460   G  /usr/lib/firefox/firefox       2MiB  |
+-----+
```

**Рис. 6.2.** Вывод команды `nvidia-smi` в окне терминала; рамкой выделен уровень загруженности GPU

Ключевым показателем при обучении сети для нас является степень загруженности GPU, которая определяется (согласно документации) как процент времени за последнюю секунду, в течение которого было задействовано одно или несколько ядер графического процессора. Пятьдесят один процент — это, откровенно говоря, не так много. Но эта степень загруженности соответствует моменту запуска команды `nvidia-smi`. А можно ли наблюдать за этой величиной продолжительное время? Чтобы лучше оценить загруженность GPU, можно организовать обновление метрики каждые полсекунды, воспользовавшись командой `watch` (эта команда стоит того, чтобы запомнить ее):

```
$ watch -n .5 nvidia-smi
```



Загруженность GPU является хорошим показателем для оценки эффективности пайплайна, но сама по себе эта метрика не лучший показатель того, насколько хорошо используется GPU — для фактической работы может использоваться лишь малая часть ресурсов GPU.

Следить за меняющимся числом в окне терминала — не лучшее времяпрепровождение. Намного эффективнее получать показатель загруженности GPU раз в секунду и записывать его в файл. Запустите следующую команду примерно на 30 секунд, пока в системе выполняется любой процесс, использующий GPU, и остановите ее нажатием комбинации Ctrl+C:

```
$ nvidia-smi --query-gpu=utilization.gpu --format=csv,noheader,nounits -f gpu_utilization.csv -l 1
```

Теперь найдите медиану загруженности GPU в сгенерированном файле:

```
$ sort -n gpu_utilization.csv | grep -v '^0$' | datamash median 1
```

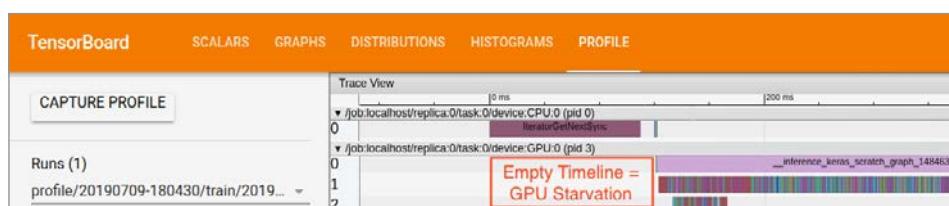


**datamash** — удобный инструмент командной строки, который выполняет базовые числовые, текстовые и статистические операции с текстовыми файлами данных. Инструкции по его установке можно найти по адресу <https://www.gnu.org/software/datamash/>.

**nvidia-smi** — это самый удобный способ оценить загруженность GPU в командной строке. А можно ли провести более глубокий анализ? Как оказывается, TensorFlow предлагает опытным пользователям мощный набор инструментов.

## Профилировщик TensorFlow + TensorBoard

В состав TensorFlow входит **tfprof** (рис. 6.3) — профилировщик TensorFlow, который помогает изучать и анализировать процесс обучения на более глубоком уровне. Этот инструмент, например, позволяет создавать подробные отчеты для анализа каждой операции в модели. Но в командной строке порой довольно сложно ориентироваться. К счастью, есть набор инструментов визуализации **TensorBoard** для использования в браузере, поддерживающий плагин для профилировщика. Плагин позволяет организовать интерактивную отладку сети всего несколькими щелчками мыши. Этот набор включает в себя **Trace Viewer** — компонент, показывающий события на временной шкале. Он помогает точно выяснить, как использовались ресурсы в тот или иной период времени, и выявить неэффективность.



**Рис. 6.3.** Временная шкала профилировщика в TensorBoard показывает простой GPU, пока CPU готовит данные, а также простой CPU, пока GPU обрабатывает очередную порцию данных



На момент написания этих строк TensorBoard полностью поддерживается только в Google Chrome; он может не отображать результаты профилирования в других браузерах, например Firefox.

По умолчанию профилировщик включен в TensorBoard. Чтобы его активировать, достаточно добавить простую функцию обратного вызова:

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="/tmp",
                                                       profile_batch=7)

model.fit(train_data,
          steps_per_epoch=10,
          epochs=2,
          callbacks=[tensorboard_callback])
```

Если при инициализации обратного вызова параметр `profile_batch` не указан явно, профилирование начинается со второго пакета. Почему со второго? Потому что первый пакет обычно обрабатывается медленнее, чем остальные, из-за накладных расходов на инициализацию.



Отметим, что профилирование с TensorBoard лучше подходит для опытных пользователей TensorFlow. Если вы только начинаете осваивать этот инструмент, то лучше использовать `nvidia-smi`. (Хотя `nvidia-smi` обладает гораздо более широкими возможностями, кроме предоставления информации об использовании GPU, что обычно интересует большинство практиков.) Пользователям, желающим иметь более полный доступ к показателям использования аппаратных ресурсов, лучше подойдет замечательный инструмент NVIDIA Nsight.

С помощью этих инструментов мы выяснили, что наша программа нуждается в доработке для увеличения эффективности. Как это сделать, подробно рассмотрим в следующих разделах.

## Как использовать этот чек-лист

Бизнесмены часто говорят: «Невозможно улучшить то, что нельзя измерить». Это в равной степени относится и к пайплайнам глубокого обучения. Настройка производительности похожа на научный эксперимент. Вы определяете точку отсчета, поворачиваете ручку, оцениваете эффект и снова поворачиваете ее, пока наблюдается улучшение. Пункты в чек-листе — это наши ручки: некоторые из них реализуются легко и просто, другие более сложны.

Чтобы с максимальной пользой использовать этот чек-лист, сделайте следующее:

1. Изолируйте часть пайплайна, которую хотите улучшить.
2. Найдите нужный пункт в чек-листе.

3. Реализуйте его, поэкспериментируйте и понаблюдайте, сокращается ли время выполнения. Если нет, то отмените внесенные изменения.
4. Повторяйте шаги с 1-го по 3-й, пока не дойдете до конца списка.

Некоторые из улучшений могут быть незначительными, другие — более радикальными. Но в совокупности все эти изменения должны обеспечить более быстрое и эффективное выполнение и, что самое главное, более полное использование оборудования. Рассмотрим каждый участок пайплайна глубокого обучения, включая подготовку данных, чтение данных, аугментацию данных, обучение и, наконец, инференс.

## Чек-лист настроек производительности

### Подготовка данных

- Сохраните данные в формате TFRecord.
- Уменьшите размеры исходных данных.
- Используйте TensorFlow Datasets.

### Чтение данных

- Используйте tf.data.
- Организуйте предварительное извлечение данных.
- Организуйте параллельную обработку на CPU.
- Организуйте параллельный ввод/вывод и обработку.
- Разрешите недетерминированный порядок следования данных.
- Кэшируйте данные.
- Включите экспериментальные оптимизации.
- Автоматическая настройка значений параметров.

### Аугментация данных

- Используйте GPU для аугментации.

## Обучение

- Используйте автоматическую смешанную точность.
- Используйте пакеты большого размера.
- Используйте значения, кратные восьми.
- Определите оптимальную скорость обучения.
- Используйте tf.function.
- Переобучите и научите обобщать:
  - постепенно увеличивайте размер выборки;
  - постепенно добавляйте аугментацию данных;
  - постепенно изменяйте разрешение данных.
- Установите оптимизированный программный стек для поддержки оборудования.
- Оптимизируйте количество потоков, выполняющихся на CPU параллельно.
- Используйте более производительное оборудование.
- Используйте распределенное обучение.
- Изучите отраслевые бенчмарки.

## Инференс

- Используйте эффективную модель.
- Используйте квантование модели.
- Прореживайте модель.
- Используйте комбинированные операции.
- Включите сохранение состояния GPU.



Скачать этот чек-лист можно в репозитории <https://github.com/PracticalDL/Practical-Deep-Learning-Book/blob/master/code/chapter-6/Performance-Checklist.pdf>. Используйте его как справочник при обучении или развертывании модели. Или поделитесь им со своими друзьями, коллегами и, что особенно важно, со своим руководителем.

## Подготовка данных

Некоторые оптимизации можно применить еще до начала обучения, и все они связаны с подготовкой данных.

### Сохраните данные в формате TFRecord

Датасеты с изображениями обычно состоят из тысяч маленьких файлов размарами в несколько килобайт. Пайплайн глубокого обучения должен прочитать каждый файл отдельно. Тысячекратное выполнение этой операции — это огромные накладные расходы и замедление обучения. Проблема усугубляется, когда в системе используются вращающиеся жесткие диски, в которых магнитная головка должна быть установлена в начало каждого файла. А если файлы хранятся в удаленном хранилище, например в облаке, то проблема становится еще более серьезной. Это первое препятствие.

Одна из идей — объединить тысячи файлов в несколько файлов большого размера, чтобы ускорить чтение. Именно это и делает TFRecord. Он хранит данные в эффективных объектах Protocol Buffer (protobuf), ускоряя их чтение. Посмотрим, как создавать файлы TFRecord:

```
# Создать файлы в формате TFRecord
import tensorflow as tf
from PIL import Image
import numpy as np
import io

cat = "cat.jpg"
img_name_to_labels = {'cat' : 0}
img_in_string = open(cat, 'rb').read()
label_for_img = img_name_to_labels['cat']

def getTFRecord(img, label):
    feature = {
        'label': _int64_feature(label),
        'image_raw': _bytes_feature(img),
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

with tf.compat.v1.python_io.TFRecordWriter('img.tfrecord') as writer:
    for filename, label in img_name_to_labels.items():
        image_string = open(filename, 'rb').read()
        tf_example = getTFRecord(image_string, label)
        writer.write(tf_example.SerializeToString())
```

Теперь посмотрим, как читать файлы TFRecord:

```
# Чтение файлов TFRecord
dataset = tf.data.TFRecordDataset('img.tfrecord')
```

```
ground_truth_info = {
    'label': tf.compat.v1.FixedLenFeature([], tf.int64),
    'image_raw': tf.compat.v1.FixedLenFeature([], tf.string),
}

def map_operation(read_data):
    return tf.compat.v1.parse_single_example(read_data, ground_truth_info)

imgs = dataset.map(map_operation)

for image_features in imgs:
    image_raw = image_features['image_raw'].numpy()
    label = image_features['label'].numpy()
    image = Image.open(io.BytesIO(image_raw))
    image.show()
    print(label)
```

Но почему бы не объединить все данные, например все изображения из ImageNet, в один файл? Дело в том, что производительности вредит не только чтение тысяч крошечных файлов из-за больших накладных расходов, но и чтение гигантских файлов. Они уменьшают возможности для параллельного чтения и параллельной отправки сетевых запросов. Наилучшие результаты дает размещение большого датасета в файлах TFRecord размером около 100 Мбайт.

## Уменьшите размеры исходных данных

Большие изображения нужно уменьшить перед передачей в GPU. Это означает:

- повторяющиеся циклы CPU на каждой итерации;
- пропускная способность повторяющихся операций ввода/вывода ограничена сильнее, чем нужно в пайплайне.

Сэкономить процессорное время можно, однократно выполнив предварительную обработку всего датасета (например, изменив размеры всех изображений) и сохранив результаты в файлы TFRecord для последующих запусков.

## Используйте TensorFlow Datasets

Загрузить часто используемые общедоступные датасеты (нередко в виде нескольких файлов архивов), такие как MNIST (11 Мбайт), CIFAR-100 (160 Мбайт), MS COCO (38 Гбайт) и Google Open Images (565 Гбайт), довольно сложно. Представьте свое разочарование, когда после длительной загрузки 95 % файла соединение разрывается. В этом нет ничего необычного, потому что эти файлы обычно размещаются на университетских серверах или загружаются из разных источников, таких как Flickr (как в случае с набором ImageNet 2012, который дает URL-адреса, откуда можно загрузить более 150 Гбайт изображений). Разрыв связи нередко означает, что придется начинать все сначала.

Если вы думаете, что загрузка — это самое сложное, то ошибаетесь. Настоящие сложности только начинаются. После успешной загрузки каждого нового датасета нужно найти документацию и определить, как организованы данные, чтобы начать их чтение и обработку. Затем нужно разделить данные на обучающую, проверочную и контрольную выборки (желательно с преобразованием в TFRecord). А когда объем данных настолько велик, что они не помещаются в память, придется использовать разные ухищрения, чтобы обеспечить эффективное чтение и передачу данных в пайплайн обучения. Мы и не говорили, что будет легко.

Но всех этих сложностей можно избежать, если использовать готовый высокопроизводительный пакет TensorFlow Datasets. Он эффективно загружает, компонует и передает известные датасеты в обучающий пайплайн, требуя от нас всего нескольких строк кода.

Посмотрим, какие датасеты доступны.

```
import tensorflow_datasets as tfds
# Показать доступные датасеты
print(tfds.list_builders())
===== Output =====
['abstract_reasoning', 'bair_robot_pushing_small', 'caltech101', 'cats_vs_dogs',
'celeb_a', 'celeb_a_hq', 'chexpert', 'cifar10', 'cifar100', 'cifar10_corrupted',
'cnn_dailymail', 'coco2014', 'colorectal_histology',
'colorectal_histology_large', 'cycle_gan' ...]
```

На момент написания этих строк в списке было более 100 датасетов, и это число постоянно увеличивается. Загрузим и извлечем обучающий датасет CIFAR-10:

```
train_dataset = tfds.load(name="cifar100", split=tfds.Split.TRAIN)
train_dataset = train_dataset.shuffle(2048).batch(64)
```

Вот и все! При первом запуске этот код загрузит датасет и поместит его в кэш на нашем компьютере. При каждом следующем запуске он будет пропускать загрузку из сети и сразу будет читать данные из кэша.

## Чтение данных

Когда с подготовкой данных закончено, поищем возможности увеличить пропускную способность пайплайна чтения данных.

### Используйте `tf.data`

Можно вручную прочитать каждый файл из датасета, использовав встроенную библиотеку ввода/вывода. Достаточно просто вызвать `open` для каждого файла, и все будет хорошо, так ведь? Увы, нет. Основной недостаток этого подхода в том,

что GPU будет простоявать во время чтения файлов — он будет вынужден ждать, пока мы прочитаем файл. Затем, пока GPU обрабатывает полученные данные, мы будем ждать завершения обработки, прежде чем начать читать следующий файл с диска. Все это нерационально.

Если бы из этой главы можно было вынести только один урок, он бы был таким: `tf.data` — это путь к созданию высокопроизводительного пайплайна обучения. В следующих разделах мы рассмотрим некоторые особенности `tf.data`, которые можно использовать для повышения скорости обучения.

Для начала настроим базовый пайплайн чтения данных:

```
files = tf.data.Dataset.list_files("./training_data/*.tfrecord")
dataset = tf.data.TFRecordDataset(files)

dataset = dataset.shuffle(2048)
    .repeat()
    .map(lambda item: tf.io.parse_single_example(item, features))
    .map(_resize_image)
    .batch(64)
```

## Организуйте предварительное извлечение данных

В примере пайплайна, представленном выше, GPU ждет, пока CPU сгенерирует данные, а затем CPU ждет, пока GPU завершит вычисления, после чего генерирует данные для следующего цикла. Эта циклическая зависимость вынуждает центральный и графический процессоры периодически простоявать, а это очень неэффективно.

Эту проблему решает функция `prefetch`, которая отделяет генерирование данных (центральным процессором) от их обработки (графическим процессором). Фоновый поток выполнения позволяет асинхронно передавать данные в промежуточный буфер, где они доступны для GPU. Теперь CPU сможет приступить к подготовке следующего пакета данных, не дожидаясь GPU. Точно так же GPU, завершив предыдущие вычисления, сможет приступить к обработке очередной порции данных, как только она появится в буфере.

От нас требуется только вызвать функцию `prefetch` для датасета в самом конце пайплайна и передать ей параметр `buffer_size`, определяющий максимальный объем данных, который можно сохранить. Обычно `buffer_size` — небольшое число; во многих случаях достаточно значения 1:

```
dataset = dataset.prefetch(buffer_size=16)
```

Скоро мы покажем, как найти оптимальное значение для этого параметра.

Таким образом, `prefetch` позволяет организовать параллельную работу CPU и GPU.

## Организуйте параллельную обработку на CPU

Было бы напрасной тратой ресурсов заставлять CPU с несколькими ядрами делать всю работу только на одном ядре. Почему бы не задействовать остальные? В этом поможет аргумент `num_parallel_calls` в функции `map`:

```
dataset = dataset.map(lambda item: tf.io.parse_single_example(item, features),  
                      num_parallel_calls=4)
```

Этот вызов функции `map()` запустит несколько параллельных потоков для обработки. Если допустить, что в системе не выполняются другие тяжелые приложения, то в параметре `num_parallel_calls` можно передать число, равное количеству ядер CPU. Если задать большее число, это ухудшит производительность из-за накладных расходов на переключение контекста.

## Организуйте параллельный ввод/вывод и обработку

Чтение файлов с диска или, что еще хуже, из Сети — самое узкое место. У нас могут быть самые мощные в мире CPU и GPU, но если не оптимизировать чтение файлов, их возможности останутся недоиспользованными. Одно из решений — распараллеливание ввода/вывода и последующей обработки (этот прием также известен как *чредование*).

```
dataset = files.interleave(map_func, num_parallel_calls=4)
```

Обратите внимание, что эта команда:

- загружает входные данные, используя несколько параллельных потоков (по умолчанию их количество равно количеству ядер CPU в системе);
- параметр `num_parallel_calls` позволяет вызывать функцию `map_func` в нескольких параллельных потоках и асинхронно читать входные данные.

Если параметр `num_parallel_calls` опустить, то даже несмотря на параллельное чтение данных `map_func` будет выполняться синхронно в одном потоке. Если `map_func` справляется с обработкой поступающих данных, то это не проблема. Но если `map_func` становится узким местом, то желательно задать более высокое значение в параметре `num_parallel_calls`.

## Разрешите недетерминированный порядок следования данных

Для многих датасетов порядок чтения неважен. Так или иначе можно прибегнуть к randomизации порядка следования данных. По умолчанию при параллельном

чтении файлов `tf.data` все еще пытается возвращать данные в *фиксированном порядке, созданном алгоритмом кругового обслуживания*. Но при таком подходе можно столкнуться с проблемой «отставших» операций (то есть операций, которые занимают намного больше времени, чем другие, и задерживают все остальные операции). Это как очередь в магазине, когда человек, стоящий перед вами, долго рассчитывается на кассе наличными и скрупулезно пересчитывает сдачу, в то время как все остальные платят банковскими картами. Чтобы не блокировать все последующие операции, готовые вернуть прочитанные данные, можно разрешить пропускать отставших, пока они не закончат свою работу. Это нарушит порядок следования данных, зато сократит потери времени на ожидание более медленных операций:

```
options = tf.data.Options()
options.experimental_deterministic = False

dataset = tf.data.Dataset.list_files("./training_data/")
dataset = dataset.with_options(options)
dataset = dataset.interleave(tf.data.TFRecordDataset, num_parallel_calls=4)
```

## Кэшируйте данные

Функция `Dataset.cache()` позволяет создавать копии данных в памяти или в файле на диске. Кэшировать данные стоит по двум причинам.

- Чтобы избежать повторного чтения с диска после первой эпохи. Очевидно, что это эффективно, только когда кэш находится в оперативной памяти и может уместиться в ней.
- Чтобы не выполнять многоократно дорогостоящие операции с данными (например, уменьшать размеры больших изображений).



Кэш лучше использовать для хранения данных, которые не меняются. Рекомендуется вызывать `cache()` перед операциями аугментации и перемешивания; в противном случае кэширование будет приводить к созданию точно таких же датасетов с тем же порядком следования при каждом запуске.

В зависимости от сценария используем одну из двух инструкций:

```
dataset = dataset.cache() # в памяти
dataset = dataset.cache(filename='tmp.cache') # на диске
```

Кэш в памяти хранится только во время выполнения и поэтому дает прирост производительности лишь со второй эпохи в каждом запуске. Кэш в файлах, напротив, ускоряет каждый запуск обучения (после самой первой эпохи самого первого запуска).



В разделе «Уменьшите размеры исходных данных» выше мы обсуждали предварительную обработку данных и сохранение ее результатов в файлах TFRecord, которые будут служить источниками данных для будущих пайплайнов. Вызов функции `cache()` сразу после этапа предварительной обработки в вашем пайплайне даст аналогичный эффект.

## Включите экспериментальные оптимизации

TensorFlow имеет множество встроенных оптимизаций, часто экспериментальных и потому отключенных по умолчанию. В зависимости от сценария использования можно включить некоторые из них, чтобы немного повысить производительность пайплайна. Многие из оптимизаций подробно описаны в документации для `tf.data.experimental.OptimizationOptions`.



Ниже приводится краткое описание операций фильтрации и отображения.

### Фильтрация

Просматривает список элементов и отбирает те из них, которые соответствуют заданному условию. Условие определяется как лямбда-выражение, возвращающее логическое значение.

### Отображение

Принимает элемент, выполняет вычисления с ним (например, изменяет размеры изображения) и возвращает результат.

Рассмотрим некоторые из доступных экспериментальных оптимизаций, включая возможность объединения двух последовательных операций в одну.

## Объединение операций фильтрации

Иногда нужно выполнить фильтрацию по нескольким атрибутам, например отобрать только изображения, на которых есть и собака, и кошка. Или отобрать из датасета с результатами переписи только семьи с доходом выше определенного порога и проживающие на определенном расстоянии от центра города. В таких случаях ускорить выполнение операций может помочь `filter_fusion`. Рассмотрим следующий пример:

```
dataset = dataset.filter(lambda x: x < 1000).filter(lambda x: x % 3 == 0)
```

Первый фильтр выполняет обход датасета и возвращает элементы, размер которых меньше 1000. Второй фильтр выполняет второй обход элементов, остав-

шихся после первого фильтра, и удаляет элементы, не кратные трем. Вместо двух обходов одних и тех же элементов мы могли бы выполнить две операции фильтрации в одном обходе, используя операцию И (AND). Именно это обеспечивает оптимизация `filter_fusion` — она объединяет несколько операций фильтрации и выполняет их в одном проходе. По умолчанию эта оптимизация выключена. Ее можно включить, как показано ниже:

```
options = tf.data.Options()  
options.experimental_optimization.filter_fusion = True  
dataset = dataset.with_options(options)
```

## Объединение операций отображения и фильтрации

Рассмотрим следующий пример:

```
dataset = dataset.map(lambda x: x * x).filter(lambda x: x % 2 == 0)
```

В этом примере функция `map` выполняет обход всего датасета, вычисляя квадраты элементов. Затем функция `filter` отбрасывает элементы с нечетными значениями. Вместо двух обходов (что особенно расточительно в данном примере) мы могли бы объединить операции отображения и фильтрации, включив параметр `map_and_filter_fusion`:

```
options.experimental_optimization.map_and_filter_fusion = True
```

## Объединение операций отображения

Как и в двух предыдущих примерах, объединение двух или более операций отображения может сократить количество обходов одних и тех же данных до одного:

```
options.experimental_optimization.map_fusion = True
```

## Автоматическая настройка значений параметров

Возможно, вы заметили, что во многих примерах этого раздела используются жестко заданные значения некоторых параметров. Настраивая эти параметры, можно обеспечить максимальную эффективность выполнения конкретных задач на конкретном оборудовании. Но как их настроить? Один из очевидных способов — вручную пробовать настраивать каждый параметр в отдельности и смотреть, как он влияет на общую производительность. Так в итоге можно получить набор оптимальных параметров. Но количество настраиваемых параметров растет очень быстро из-за комбинаторного взрыва. Кроме того, наш точно настроенный сценарий неизбежно будет столь же эффективен на другом компьютере из-за различий в оборудовании, например из-за другого количества ядер CPU, отсутствия GPU и т. д. И даже в одной и той же системе,

в зависимости от использования ресурсов другими программами, оптимальные значения параметров могут быть разными в разных прогонах.

Можно ли решить эту проблему? Да, если использовать автоматическую настройку. Применяя оптимизации алгоритмов поиска восхождения к вершине (разновидность алгоритмов эвристического поиска), можно автоматически настраивать идеальные комбинации параметров для многих функций в `tf.data`. Используйте `tf.data.experimental.AUTOTUNE` вместо конкретных числовых значений. С помощью этого единственного значения можно управлять всеми параметрами. Рассмотрим следующий пример:

```
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

Разве это не элегантное решение? Этот же прием можно использовать с другими функциями в пайплайне `tf.data`. Вот пример объединения нескольких оптимизаций из раздела «Чтение данных» для создания высокопроизводительного пайплайна:

```
options = tf.data.Options()
options.experimental_deterministic = False

dataset = tf.data.Dataset.list_files("/path/*.tfrecord")
dataset = dataset.with_options(options)
dataset = files.interleave(tf.data.TFRecordDataset,
                           num_parallel_calls=tf.data.experimental.AUTOTUNE)
dataset = dataset.map(preprocess,
                      num_parallel_calls=tf.data.experimental.AUTOTUNE)
dataset = dataset.cache()
dataset = dataset.repeat()
dataset = dataset.shuffle(2048)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

## Аугментация данных

Иногда данных бывает недостаточно для запуска пайплайна обучения. Но даже когда данных довольно много, все равно полезно провести аугментацию, чтобы повысить надежность модели. Посмотрим, можно ли ускорить этот этап.

### Используйте GPU для аугментации

Пайплайны предварительной обработки данных бывают настолько сложными, что о них можно написать целую книгу. Операции преобразования изображений — изменение размера, кадрирование, преобразование цветов, размытие и др. — обычно выполняются с данными сразу после их чтения с диска в память. Учитывая, что все эти операции матричные, с ними прекрасно справится GPU.

OpenCV, Pillow и встроенные функции аугментации в Keras — наиболее часто используемые библиотеки для обработки изображений в области компьютерного зрения. Но есть одно серьезное ограничение. Они обрабатывают изображения в основном на CPU (исключением является только библиотека OpenCV, которую можно скомпилировать с поддержкой CUDA). А это означает, что пайплайн может использовать не все возможности, которые есть у оборудования.



На момент написания этих строк (в августе 2019 года) в проекте Keras уже были начаты работы по реализации функций аугментации изображений с использованием GPU.

Есть несколько вариантов поддержки аугментации с использованием GPU.

## Инструмент аугментации в `tf.image`

`tf.image` предлагает несколько удобных функций для аугментации наборов изображений, которые легко подключить к пайплайну `tf.data`. К их числу относятся функции переворота изображения, изменения цветовой палитры (оттенка, насыщенности, яркости, контраста), масштабирования и поворота. Вот пример изменения оттенка изображения:

```
updated_image = tf.image.adjust_hue(image, delta = 0.2)
```

Недостаток `tf.image` — намного более ограниченные возможности по сравнению с OpenCV, Pillow и даже Keras. Например, функция поворота изображения в `tf.image` поддерживает поворот только на 90 градусов против часовой стрелки. Если потребуется поворачивать изображения на произвольный угол, например на 10 градусов, то вам придется самим написать такую функцию. Keras, напротив, предлагает эту возможность из коробки.

## NVIDIA DALI

Еще одна альтернатива пайплайну `tf.data` — библиотека NVIDIA Data Loading Library (DALI), предлагающая возможность быстрой загрузки данных и пайплайн ускоренной предварительной обработки данных на GPU. Как показано на рис. 6.4, DALI реализует несколько основных этапов, включая изменение размеров изображений и их аугментацию на GPU, предшествующих обучению. Библиотека DALI поддерживает несколько фреймворков глубокого обучения, включая TensorFlow, PyTorch, MXNet и др., и предлагает переносимый пайплайн предварительной обработки.

Даже декодирование формата JPEG (довольно тяжеловесная операция) может частично выполняться на GPU, что тоже дает некоторое ускорение. Эта возможность есть в библиотеке nuJPEG, поддерживающей использование

GPU для декодирования JPEG. При наличии нескольких GPU эта библиотека позволяет масштабировать задачи почти линейно с увеличением количества используемых GPU.



**Рис. 6.4.** Конвейер NVIDIA DALI

Усилия NVIDIA завершились рекордом по версии MLPerf (тестирует производительность оборудования, программного обеспечения и служб машинного обучения): им удалось обучить модель ResNet-50 за 80 секунд.

## Обучение

Для тех, кто только начинает интересоваться оптимизацией производительности, отметим, что оптимизация пайплайнов предварительной обработки данных приносит самые быстрые выгоды и относительно легко реализуется. Теперь посмотрим, как оптимизировать пайпайн обучения, получающий подготовленные данные и непосредственно обучающий модель.

### Используйте автоматическую смешанную точность

«Одна строка, способная ускорить обучение в два-три раза!»

Веса в моделях глубокого обучения обычно хранятся в виде вещественных значений одинарной точности, то есть 32-битных чисел с плавающей запятой, или, как их еще называют, FP32. Размещение этих моделей в устройствах с ограниченным объемом памяти, например в смартфонах, может оказаться сложной задачей. Простой способ уменьшить модель — преобразовать ее веса из одинарной точности (FP32) в половинную точность (FP16). Конечно, представительная мощность этих весов снижается, но, как будет показано далее в этой главе (см. раздел «Используйте квантование модели»), нейронные сети устойчивы к небольшим изменениям весов и к шуму в изображениях. Поэтому мы можем получить более эффективную модель без сильной потери точности. Как будет показано в следующих главах, представить веса можно даже в виде 8-битных целых чисел (INT8) без значительной потери точности.

Напрашивается вопрос: если можно использовать представление с пониженной точностью в инференсе, то можно ли использовать такое же представление на

этапе обучения? Переход от 32- к 16-битному представлению фактически означает удвоение пропускной способности памяти, удвоение размера модели или удвоение размера пакета. К сожалению, простое использование представления FP16 *во время обучения* приведет к значительной потере точности модели и даже не позволит получить оптимальное решение. Это обусловлено ограниченностью диапазона представления чисел FP16. Из-за недостаточной точности любые малые обновления модели во время обучения окажутся незафиксированными. Представьте, что к значению веса 1,1 добавляется 0,00006. При использовании представления FP32 вес правильно обновится до 1,10006. Но при использовании FP16 вес останется равным 1,1. И наоборот, любые активации из таких слоев, как блок линейной ректификации (Rectified Linear Unit, ReLU), могут оказаться черезчур высокими, вызвать переполнение представления FP16 и достичь бесконечности (`Nan` в Python).

Самый простой способ решить эти проблемы — обучение с автоматической смешанной точностью (automatic mixed precision). В этом методе сохраняется модель с весами в представлении FP32, которая служит главной копией, и выполняются прямые/обратные проходы обучения с использованием представления FP16. Обновления весов, полученные после каждого шага обучения, масштабируются обратно в представление FP32 и затем применяются к главной копии. Это помогает избежать ловушек для арифметики FP16, уменьшить объем используемой памяти и ускорить обучение (как показали эксперименты, скорость может увеличиться в два-три раза) при сохранении точности, сопоставимой с обучением только с представлением FP32. Примечательно, что новые архитектуры GPU, такие как NVIDIA Volta и Turing, оптимизируют операции FP16.

Чтобы включить автоматическую смешанную точность во время обучения, достаточно добавить такую строку в начало сценария:

```
os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
```

## Используйте пакеты большого размера

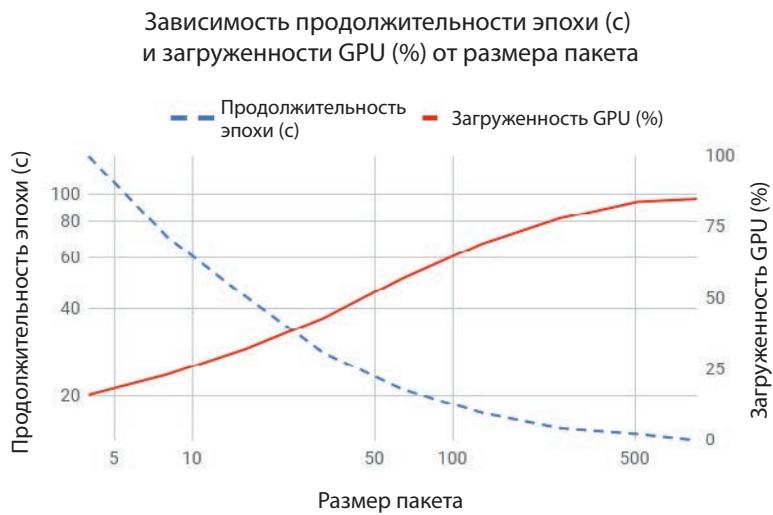
При обучении мы вводим в модель обучающие данные не все сразу, а пакетами. Это делается по двум причинам:

- весь обучающий датасет (в виде единственного пакета) может не поместиться в памяти GPU;
- той же точности обучения можно достичь, передавая данные порциями, как большими, так и маленькими.

Ввод данных небольшими пакетами может привести к недоиспользованию памяти GPU, поэтому важно поэкспериментировать с этим параметром, посмотреть, как он влияет на загруженность GPU (с помощью команды `nvidia-smi`), и выбрать размер пакета, обеспечивающий максимальную его загрузку.

Потребительские графические процессоры, например NVIDIA 2080 Ti, поставляются с 11 Гбайт памяти, чего достаточно для эффективного обучения таких моделей, как MobileNet.

Например, на оборудовании с видеокартой 2080 Ti обучение модели MobileNetV2 на изображениях размером  $224 \times 224$  и с использованием GPU можно производить пакетами, включающими до 864 изображений. На рис. 6.5 показано, как влияет изменение размера пакета от 4 до 864 на загруженность GPU (сплошная линия) и продолжительность одной эпохи (пунктирная линия): чем больше размер пакета, тем выше загруженность GPU и тем меньше продолжительность эпохи обучения.



**Рис. 6.5.** Влияние изменения размера пакета на продолжительность эпохи (в секундах) и степень загруженности GPU (по обеим осям используется логарифмический масштаб)



Выше были примеры использования пакетов размером до нескольких сотен, но в промышленных окружениях обучение моделей выполняется в распределенных системах сразу на нескольких нодах и с использованием пакетов гораздо большего размера, благодаря приему, который называется послойным адаптивным масштабированием скорости (Layer-wise Adaptive Rate Scaling, LARS). Например, Fujitsu Research обучила сеть ResNet-50 на наборе ImageNet до 75 % в топ-1 точности всего за 75 секунд. Для этого они использовали 2048 графических процессоров Tesla V100 и пакеты колоссального размера — 81 920.

Даже при максимальном размере пакета 864 (ограниченном объемом памяти) загруженность GPU не превышает 85 %. Это означает, что GPU достаточно мощ-

ный и успевает обрабатывать данные, поставляемые нашим очень эффективным пайплайном. Замена MobileNetV2 более тяжеловесной моделью ResNet-50 сразу же привела к увеличению загруженности GPU до 95 %.

## Используйте значения, кратные восьми

Большинство вычислений в глубоком обучении относятся к разряду умножения и сложения матриц. Это довольно дорогостоящие операции, но в последние несколько лет стала бурно развиваться разработка специализированного оборудования для оптимизации их производительности. Примерами могут служить создание тензорных процессоров (TPU) в Google и тензорных ядер в NVIDIA (которые можно найти в архитектурах Turing и Volta). GPU Turing имеют и тензорные ядра (для операций FP16 и INT8), и ядра CUDA (для операций FP32), причем тензорные ядра имеют значительно более высокую пропускную способность. Из-за особенностей своей специализации тензорные ядра требуют, чтобы определенные параметры в передаваемых им данных были кратны восьми. Таких параметров всего три:

- количество каналов в сверточном фильтре;
- количество нейронов и входов в полносвязанном слое;
- размер пакетов.

Если эти параметры не кратны восьми, для вычислений будут использоваться ядра CUDA. В эксперименте (<https://oreil.ly/KoEkM>), о котором писала NVIDIA, простое изменение размера пакета с 4095 до 4096 привело к увеличению пропускной способности в пять раз. Имейте в виду, что использование значений, кратных восьми (или 16 в случае операций INT8), в дополнение к автоматической смешанной точности — это минимальное требование для активации тензорных ядер. Для большей эффективности рекомендуется использовать значения, кратные 64 или 256. Точно так же для максимальной эффективности вычислений на тензорных процессорах Google рекомендует использовать значения, кратные 128.

## Определите оптимальную скорость обучения

Один из гиперпараметров, сильно влияющих на скорость сходимости (и точность), — это скорость обучения. Идеальный результат обучения — обнаружение глобального минимума, то есть точки наименьших потерь. При слишком большой скорости обучения модель может перешагнуть через глобальный минимум (как сильно раскачивающийся маятник) и никогда не сойтись. При слишком маленькой скорости на обучение может потребоваться очень много времени, потому что алгоритм будет приближаться к минимуму очень маленькими шагами. Выбор правильной начальной скорости обучения очень важен.

Простейший способ определить идеальную начальную скорость обучения — по пробовать несколько разных скоростей (например, 0,00001, 0,0001, 0,001, 0,01, 0,1) и выбрать ту, которая дает более быстрое сходжение. Еще лучше выполнить поиск по сетке с диапазоном значений. Но этот подход имеет два недостатка: 1) в зависимости от степени детализации он может найти приемлемое, но не самое оптимальное значение; и 2) обучение придется выполнить несколько раз, что может занять много времени.

В статье «Cyclical Learning Rates for Training Neural Networks», опубликованной в 2015 году, Лесли Смит (Leslie N. Smith) описывает более удачную стратегию поиска оптимальной скорости обучения:

1. Начните с реально низкой скорости обучения и постепенно увеличивайте ее до заранее определенного максимального значения.
2. На каждой скорости обучения наблюдайте за величиной потерь — сначала она будет оставаться неизменной, затем начнет снижаться, потом снова возрастет.
3. Вычислите скорость уменьшения потерь (первую производную) для каждой скорости обучения.
4. Выберите точку с наибольшей скоростью уменьшения потерь.

Этот алгоритм кажется довольно сложным, но, к счастью, его не нужно воплощать в коде. Библиотека keras\_lr\_finder ([https://oreil.ly/il\\_BI](https://oreil.ly/il_BI)) Павла Сурменока (Pavel Surmenok) предлагает подходящую функцию:

```
lr_finder = LRFinder(model)
lr_finder.find(x_train, y_train, start_lr=0.0001, end_lr=10, batch_size=512,
                epochs=5)
lr_finder.plot_loss(n_skip_beginning=20, n_skip_end=5)
```

На рис. 6.6 показан график зависимости величины потерь от скорости обучения. Скорость обучения  $10^{-4}$  или  $10^{-3}$  может быть слишком низкой (из-за того, что потери практически не уменьшаются), и точно так же скорость обучения выше 1 может быть слишком высокой (из-за быстрого увеличения потерь).

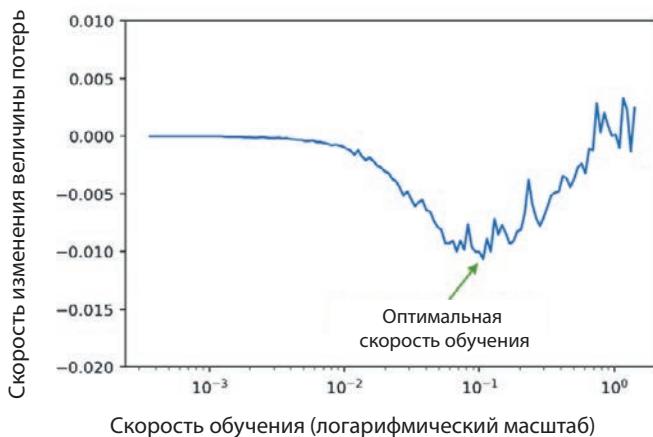
Нас интересует точка наибольшего уменьшения потерь. В конце концов, мы стремимся сократить время, которое тратится на минимизацию потерь во время обучения. На рис. 6.7 мы изобразили *скорость изменения* величины потерь — производную потерь в зависимости от скорости обучения:

```
# Показать простое скользящее среднее по 20 точкам для сглаживания графика
lr_finder.plot_loss_change(sma=20, n_skip_beginning=20, n_skip_end=5,
                            y_lim=(-0.01, 0.01))
```

Эти графики показывают, что быстрее всего величина потерь снижается при скорости обучения 0,1, поэтому ее можно считать оптимальной.



**Рис. 6.6.** График изменения величины потерь с увеличением скорости обучения



**Рис. 6.7.** График скорости изменения величины потерь с увеличением скорости обучения

## Используйте `tf.function`

Режим немедленного выполнения, по умолчанию включенный в TensorFlow 2.0, позволяет выполнять код построчно и сразу же видеть результаты, что очень удобно для разработки и отладки. В этом заключается существенное отличие от версии TensorFlow 1.x, которая требовала от пользователя создать законченный граф из операций, а затем запустить его, чтобы увидеть результаты. Такой подход превращал отладку в кошмар.

Но не влечет ли режим немедленного выполнения дополнительных накладных расходов? Увы, влечет. Обычно они невелики, порядка нескольких микросекунд, и в принципе их можно игнорировать, если речь идет об операциях с большим объемом вычислений, таких как обучение ResNet-50. Но если процесс состоит из множества мелких операций, немедленное выполнение даст значительное замедление.

Решить эту проблему можно двумя способами.

*Отключить режим немедленного выполнения*

Отключение немедленного выполнения в TensorFlow 1.x позволяет системе оптимизировать поток выполнения программы, построив оптимальный граф, и выполнить его быстрее.

## Использовать `tf.function`

В TensorFlow 2.x нет возможности отключить режим немедленного выполнения (да, в этой версии есть API поддержки совместимости, но его следует использовать только для миграции с TensorFlow 1.x). Зато любую функцию, которая может выиграть от ускорения за счет выполнения в режиме графа, можно аннотировать декоратором `@tf.function`. В этом случае любая функция, которая вызывается внутри аннотированной функции, также будет выполняться в режиме графа. Это позволяет ускорить выполнение за счет использования режима графа и при этом сохранить удобство отладки режима немедленного выполнения. Как правило, наибольшее ускорение достигается при выполнении коротких задач с интенсивными вычислениями:

```
print("With @tf.function:", timeit.timeit(lambda: tf_func(mat), number=10000),
      "seconds")  
  
=====Output=====  
Without @tf.function: 7.234016112051904 seconds  
With @tf.function: 0.7510978290811181 seconds
```

Этот искусственный пример показывает, что простое использование декоратора `@tf.function` дало ускорение в 10 раз, с 7,2 секунды до 0,7 секунды.

## Переобучите и научите обобщать

Переобучение модели считается вредным. Но мы покажем, что переобучение можно контролировать и использовать для ускорения обучения.

Как известно, «лучшее — враг хорошего». Наша цель не в том, чтобы сразу же получить идеальную сеть, и даже не в том, чтобы получить хотя бы хорошую сеть. Настоящая цель — быстро научить сеть *чему-либо*, пусть и не идеально, потому что тогда у нас появится хороший фундамент, который затем можно настроить и максимально раскрыть его потенциал. Следуя этим путем, можно быстрее добраться до конечной цели, чем при обычном обучении.



Идея переобучения с последующим обобщением становится понятна на примере с изучением языков. Допустим, вы решили выучить французский. Один из способов — изнурять себя чтением словарей и учебников в надежде запомнить все необходимое. Конечно, читая словари и учебники каждый день, вы, возможно, научитесь через пару лет немного говорить по-французски. Но это не лучший способ обучения.

А вот как подходят к этому процессу на курсах изучения языка. Сначала вас знакомят с небольшим набором слов и грамматических правил. Выучив их, вы сможете говорить на ломаном французском, например попросить чашку кофе в ресторане или узнать дорогу на автобусной остановке. На этом этапе вы будете постоянно знакомиться с более широким набором слов и правил и со временем усовершенствуете свое знание французского.

Постепенное обучение модели на все большем объеме данных выглядит примерно так же.

Но как заставить сеть учиться быстро и неидеально? Добейтесь ее переобучения на своих данных. Далее приводятся три стратегии, которые помогут в этом.

## Постепенно увеличивайте размер выборки

Один из способов подхода к реализации переобучения с последующим обобщением — постепенно вводить в модель все больше и больше данных из обучающей выборки. Вот как это выглядит:

1. Возьмите небольшую выборку из датасета (например, 10 %).
2. Обучите сеть до схождения, то есть пока она не достигнет хорошей точности на обучающей выборке.
3. Выполните обучение на большей выборке (или даже на всей обучающей выборке).

Обучаясь на небольшой выборке, сеть быстрее изучит закономерности, но только характерные для данной выборки, то есть будет испытывать тенденцию к переобучению, показывая на обучающей выборке более высокую точность, чем на контрольной. Обучение на всем датасете приведет к обобщению изученного, и в итоге точность на контрольной выборке увеличится.

### **Постепенно делайте аугментацию данных**

Другой способ: первоначально обучить на всем наборе данных без аугментации, а затем постепенно делать аугментацию набора, повторяя обучение. При многократном вводе набора изображений без аугментации сеть быстрее изучит закономерности, а при постепенном добавлении аугментации набора она будет повышать свою надежность.

### **Постепенно изменяйте разрешение данных**

Третий способ, предложенный прославленным Джереми Ховардом (Jeremy Howard) из fast.ai (который ведет бесплатные курсы по ИИ), заключается в постепенном увеличении разрешения данных. Ключевая идея состоит в том, чтобы сначала обучить сеть на изображениях с меньшим разрешением, а затем продолжить обучение, постепенно увеличивая их разрешение до исходного.

Изображения, уменьшенные вдвое по ширине и высоте, содержат на 75 % меньше пикселей, что теоретически может привести к увеличению скорости обучения в четыре раза по сравнению с обучением на исходных изображениях. Точно так же четырехкратное уменьшение размеров может привести к уменьшению продолжительности обучения в 16 раз (естественно, с уменьшением точности). На изображениях с меньшим разрешением меньше видимых деталей, что заставляет сеть изучать закономерности высокого уровня, включая общие формы и цвета. Последующее обучение с изображениями с более высоким разрешением поможет сети изучить более мелкие детали, а также повысить точность на контрольной выборке. Например, ребенка сначала учат высокоуровневым понятиям, а затем постепенно знакомят с более подробной информацией — та же идея применяется и к сверточной сети.



Попробуйте скомбинировать эти способы или даже изобрести свои собственные, например с обучением на узком круге классов с последующим обобщением на все имеющиеся классы.

## Установите оптимизированный программный стек для поддержки оборудования

Обычно бинарные файлы библиотеки с открытым исходным кодом создаются с учетом возможности выполнения в различных аппаратных и программных окружениях. Когда мы выполняем команду `pip install`, чтобы установить библиотеку, она загружает и устанавливает универсальный бинарный файл, подходящий для работы в большинстве окружений. Такая универсальность достигается за счет невозможности использовать некоторые особенности, предлагаемые конкретным аппаратным стеком. Эта проблема — одна из основных причин, почему следует избегать установки предварительно скомпилированных бинарных файлов и отдавать предпочтение сборке библиотек из исходного кода.

Например, Google компилирует и размещает TensorFlow в каталоге `pip` единой библиотекой, которая может работать и на старом ноутбуке Sandy Bridge (второе поколение Core i3), и на мощном 16-ядерном сервере Intel Xeon. Это удобно, но эта скомпилированная библиотека не использует преимущества мощного оборудования сервера Xeon. Поэтому разработчики из Google рекомендуют компилировать TensorFlow из исходного кода, чтобы оптимизировать ее для конкретной аппаратной архитектуры, когда обучение и инференс выполняются на CPU.

Один из способов сделать это вручную — задать флаги поддержки оборудования перед сборкой исходного кода. Например, чтобы включить поддержку наборов инструкций AVX2 и SSE 4.2, достаточно выполнить следующую команду сборки (обратите внимание на дополнительный символ `m` перед каждым набором инструкций в команде):

```
$ bazel build -c opt --copt=-mavx2 --copt=-msse4.2  
//tensorflow/tools/pip_package:build_pip_package
```

Как узнать, какие возможности поддерживает CPU? Используйте следующую команду (только в Linux):

```
$ lscpu | grep Flags
```

```
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36  
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm  
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid  
aperfmpf perf_pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16  
xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt  
tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch  
cpuid_fault epb cat_l3 cdp_l3 invpcid_single pt1 intel_ppin ssbd ibrs ibpb stibp  
tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2  
erms invpcid rtm cqmq rdta rdseed adx smap intel_pt xsaveopt cqmq_llc  
cqmq_occup_llc cqmq_mbmb_total cqmq_mbmb_local dtherm ida arat pln pts md_clear  
flush_lld
```

Сборка TensorFlow из исходного кода с соответствующим набором инструкций, который определяется флагами сборки, должна привести к значительному увеличению скорости. Обратной стороной является продолжительность сборки — минимум два часа. В качестве альтернативы используйте Anaconda для загрузки и установки оптимизированной версии TensorFlow, созданной в Intel для их библиотеки Math Kernel Library for Deep Neural Networks (MKL-DNN). Установка довольно проста. Сначала нужно установить диспетчер пакетов Anaconda (<https://anaconda.com/>), а затем выполнить команду:

```
# Для Linux и Mac  
$ conda install tensorflow  
  
# Для Windows  
$ conda install tensorflow-mkl
```

На процессорах Xeon библиотека MKL-DNN часто обеспечивает двукратное ускорение инференса.

А есть ли возможность оптимизации под GPU? NVIDIA абстрагирует различия между разными GPU с помощью своей библиотеки CUDA, поэтому обычно нет необходимости выполнять сборку из исходного кода. Достаточно просто установить версию TensorFlow с поддержкой GPU из pip (пакет `tensorflow-gpu`). Для удобства рекомендуем использовать установщик Lambda Stack (<https://oreil.ly/4AUxp>), который устанавливает эту библиотеку вместе с драйверами NVIDIA и библиотеками CUDA и cuDNN.

Для обучения и прогнозирования в облаке AWS Microsoft Azure и GCP предлагают образы машин с GPU и версией TensorFlow, оптимизированной для их оборудования. Вы можете быстро запустить в облаке несколько экземпляров и начать работу. Кроме того, NVIDIA предлагает контейнеры с поддержкой GPU для локальных и облачных окружений.

## Оптимизируйте количество потоков, выполняющихся на CPU параллельно

Сравните два примера:

```
# Пример 1  
X = tf.multiply(A, B)  
Y = tf.multiply(C, D)  
  
# Пример 2  
X = tf.multiply(A, B)  
Y = tf.multiply(X, C)
```

Здесь есть несколько мест, где было бы уместно параллельное выполнение.

### *Между операциями*

В примере 1 вычисление Y не зависит от вычисления X, потому что в этих вычислениях не участвуют общие данные, а значит, обе операции можно выполнять параллельно в двух разных потоках.

В примере 2, напротив, вторая операция (вычисление Y) зависит от первой (вычисление X), поэтому вторую операцию нельзя запустить до завершения первой.

Конфигурация для максимального количества потоков, которые используются для параллельных вычислений, задается так:

```
tf.config.threading.set_inter_op_parallelism_threads(num_threads)
```

Рекомендуемое количество потоков равно количеству процессоров (сокетов) на машине. Это значение можно получить с помощью команды `lscpu` (только в Linux).

### *Внутри операций*

Можно распараллелить сами операции, например умножение матриц.

На рис. 6.8 показана простая операция умножения матриц. Очевидно, что эту операцию можно разделить на четыре независимых вычисления. В конце концов, результат вычисления произведения между строкой одной матрицы и столбцом другой матрицы не зависит от вычисления произведений между другими строками и столбцами. Каждое из этих произведений можно вычислить в отдельном потоке, и все четыре потока могут выполняться одновременно.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

**Рис. 6.8.** Умножение матриц A × B с одной выделенной операцией произведения

Конфигурация для максимального количества потоков, которые могут использоваться для параллельных вычислений внутри операций, задается так:

```
tf.config.threading.set_intra_op_parallelism_threads(num_threads)
```

Рекомендуемое количество потоков равно количеству ядер процессора. Это значение можно получить с помощью команды `lscpu` (только в Linux).

## Используйте более производительное оборудование

Если вы уже оптимизировали производительность до максимума, а вам все еще требуется ускорить обучение, возможно, пришло время подумать о новом оборудовании. Замена вращающихся жестких дисков твердотельными накопителями может иметь большое значение, как и добавление одного или нескольких высокопроизводительных GPU. И не забывайте, что CPU тоже играет немаловажную роль.

Может так получиться, что вам не придется тратить много денег: общедоступные облака — AWS, Azure и GCP — предлагают возможность арендовать мощные конфигурации всего за несколько долларов в час. И самое замечательное, что они поставляются с предустановленным оптимизированным стеком TensorFlow.

Конечно, если у вас или у вашей компании есть деньги, то просто пропустите эту главу и купите NVIDIA DGX-2 с производительностью 2 петафлопс. При весе 163 кг и с 16 GPU V100 (всего 81 920 ядер CUDA) это устройство потребляет 10 кВт электроэнергии, что эквивалентно семи большим кондиционерам, и стоит 400 000 долларов!



**Рис. 6.9.** Система NVIDIA DGX-2 для глубокого обучения стоимостью 400 000 долларов США

## Используйте распределенное обучение

*«Горизонтальное масштабирование всего двумя строками!»*

Компьютер с единственным GPU позволяет зайти довольно далеко, но даже самый мощный графический процессор имеет предел вычислительных возможностей, и вертикальное масштабирование не поможет уйти слишком далеко. Поэтому мы отдаляем предпочтение горизонтальному масштабиро-

ванию — распределению вычислений между несколькими процессорами. Это могут быть несколько GPU, TPU или даже компьютеров. Фактически именно этим путем пошли исследователи из Google Brain еще в 2012 году, использовав 16 000 процессоров для нейронной сети, предназначеннной для поиска кошек на YouTube.

В начале 2010-х обучение сети на датасете ImageNet занимало от нескольких недель до месяцев. Использование нескольких GPU могло бы ускорить процесс, но тогда никто не имел опыта настройки таких конфигураций. Новичкам это было практически недоступно. К счастью, теперь благодаря TensorFlow 2.0, для настройки распределенного обучения достаточно всего двух строк кода:

```
mirrored_strategy = tf.distribute.MirroredStrategy()
with mirrored_strategy.scope():
    model = tf.keras.applications.ResNet50()
    model.compile(loss="mse", optimizer="sgd")
```

Скорость обучения увеличивается почти прямо пропорционально (90–95 %) с увеличением количества графических процессоров. Например, добавив четыре GPU (с одинаковой вычислительной мощностью), в идеале мы могли бы заметить увеличение скорости более чем в 3,6 раза.

Но одна система может поддерживать только ограниченное количество GPU. А можно ли вовлечь в обучение несколько нод, в каждой из которых несколько GPU? Да, можно. Подобно `MirroredStrategy` мы можем использовать `MultiWorkerMirroredStrategy`. Этот прием пригодится для создания кластера в облаке. В табл. 6.1 перечислена пара стратегий организации распределенного обучения для различных сценариев.

**Таблица 6.1.** Рекомендуемые стратегии распределенного обучения

Стратегия	Сценарий использования
<code>MirroredStrategy</code>	Единственная нода с несколькими GPU
<code>MultiWorkerMirroredStrategy</code>	Несколько нод, на каждой из которых имеется один или несколько GPU

Чтобы ноды кластера взаимодействовали друг с другом при использовании стратегии `MultiWorkerMirroredStrategy`, на каждой ноде нужно настроить переменную окружения `TF_CONFIG`, присвоив ей объект JSON с IP-адресами и портами всех других нод в кластере. Управляя такими настройками вручную, легко ошибиться, поэтому для этой цели лучше использовать фреймворки оркестровки, например Kubernetes.



Библиотека Horovod с открытым исходным кодом от Uber — еще один высокопроизводительный и простой в использовании фреймворк распределенных вычислений. Многие из рекордных показателей производительности, о которых пойдет речь в следующем разделе, требуют распределенного обучения на нескольких нодах, и высокая производительность Horovod помогла достичь такого уровня. Стоит отметить, что большая популярность Horovod обусловлена еще и тем, что организовать распределенное обучение на более ранних версиях TensorFlow было намного сложнее. Кроме того, Horovod работает со всеми основными библиотеками глубокого обучения и требует минимальных изменений в коде. Она легко настраивается через командную строку. Например, вот как можно запустить распределенную программу на четырех нодах, на каждой из которых установлено по четыре GPU:

```
$ horovodrun -np 16 -H  
server1:4,server2:4,server3:4,server4:4 python  
train.py
```

## Изучите отраслевые бенчмарки

В 1980-е годы популярны были три вещи: длинные волосы, плееры Walkman и бенчмаркинг баз данных. Как глубокое обучение сейчас, базы данных в свое время тоже прошли этап смелых обещаний, часть из которых оказались всего лишь рекламной шумихой. Для проверки этих обещаний были организованы бенчмарки, среди которых наиболее известным был Transaction Processing Council (TPC). Когда кто-то собирался приобрести ПО для баз данных, он мог обратиться к результатам этого общедоступного бенчмарка, чтобы принять обоснованное решение о том, на что потратить деньги своей компании. Состязание за более высокие показатели способствовало быстрому внедрению инноваций и увеличению скорости и производительности и продвигало отрасль вперед быстрее, чем предполагалось.

По образу и подобию TPC было создано несколько бенчмарков для оценки производительности систем машинного обучения.

### *DAWN Bench*

Стэнфордский бенчмарк DAWN Bench оценивает время и затраты на доведение модели до 93 % топ-5 точности на ImageNet. Кроме того, он проводит сравнительный анализ времени и финансовых затрат на инференс. Хотелось бы отметить, насколько быстро увеличивалась производительность обучения на таком огромном датасете. В сентябре 2017 года, когда был организован бенчмарк DAWN Bench, на обучение эталонной модели потребовалось 13 дней, а стоимость ее обучения составила 2323,39 доллара. Всего через полтора года самое дешевое обучение стоило всего 12 долларов, а самое быстрое занимало 2 минуты 43 секунды. Самое замечательное, что для большинства примеров в таблице DAWN Bench приводится исходный

код с реализацией процесса обучения и описываются использовавшиеся оптимизации, которые можно изучить и воспроизвести. Там же приводятся рекомендации по настройке гиперпараметров и то, как можно использовать облако для быстрого обучения и без больших затрат.

**Таблица 6.2.** Записи в DAWN Bench по состоянию на август 2019 года, отсортированные по наименьшей стоимости обучения модели до 93 % точности топ-5

Стоимость (в долларах США)	Время обучения	Модель	Оборудование	Фреймворк
12,60	2:44:31	ResNet-50 Google Cloud TPU	GCP n1-standard-2, Cloud TPU	TensorFlow 1.11
20,89	1:42:23	ResNet-50 Setu Chokshi (MS AI MVP)	Azure ND40s_v2	PyTorch 1.0
42,66	1:44:34	ResNet-50 v1 GE Healthcare (Мин Чанг (Min Zhang))	8*V100 (один экземпляр р3.16xlarge)	TensorFlow 1.11 + Horovod
48,48	0:29:43	ResNet-50 Эндрю Шоу (Andrew Shaw), Ярослав Булатов (Yaroslav Bulatov), Джереми Ховард (Jeremy Howard)	32 * V100 (4 экземпляра AWS р3.16xlarge)	Ncluster + PyTorch 0.5

### MLPerf

Как и DAWN Bench, бенчмарк MLPerf нацелен на предоставление точной информации о производительности систем ИИ. Хотя MLPerf появился позже DAWN Bench, он пользуется более широкой поддержкой и считается отраслевым стандартом, особенно в плане оборудования. Задачи обучения и инференса размещаются в двух разделах: открытом и закрытом. В закрытом разделе производится обучение одной и той же модели с использованием одних и тех же оптимизаторов, что позволяет сравнить производительность разного оборудования. В открытом разделе разрешается использовать более быстрые модели и оптимизаторы. Для сравнения с показателями в DAWN Bench (табл. 6.2) в табл. 6.3 приводятся лучшие показатели из MLPerf, которые могут быть недоступны для большинства из нас. Самый производительный комплекс оборудования NVIDIA DGX SuperPod, состоящий из 96 DGX-2H и 1536 GPU V100, стоит от 35 до 40 миллионов долларов США. С другой

стороны, 1024 Google TPU тоже стоят несколько миллионов, но их можно арендовать в облаке по цене 8 долларов за час каждый (по состоянию на август 2019 года), в результате чистая стоимость двухминутного обучения составит менее 275 долларов США.

**Таблица 6.3.** Записи в закрытом разделе MLPerf по состоянию на август 2019, показано время обучения модели ResNet-50 до 75,9 % топ-1 точности

Время (минуты)	Где установлено достижение	Оборудование	Ускоритель	Количество ускорителей
1,28	Google	TPUv3	TPUv3	1024
1,33	NVIDIA	96 x DGX-2H	Tesla V100	1536
8831,3	Эталон	Pascal P100	Pascal P100	1

В обоих вышеупомянутых бенчмарках оцениваются время обучения и инференса на особенно мощных устройствах, но есть и другие соревнования, оценивающие затраты на инференс на маломощных устройствах. Их цель — подтолкнуть исследователей максимизировать точность и скорость при одновременном снижении энергопотребления. Вот некоторые из соревнований, которые проводятся на ежегодных конференциях:

- LPIRC: Low-Power Image Recognition Challenge (конкурс распознавания изображений с низким энергопотреблением);
- EDLDC: Embedded Deep Learning Design Contest (конкурс архитектур глубокого обучения для встраиваемых систем);
- SDC at DAC: System Design Contest at Design Automation Conference (конкурс системных архитектур на конференции по автоматизации проектирования).

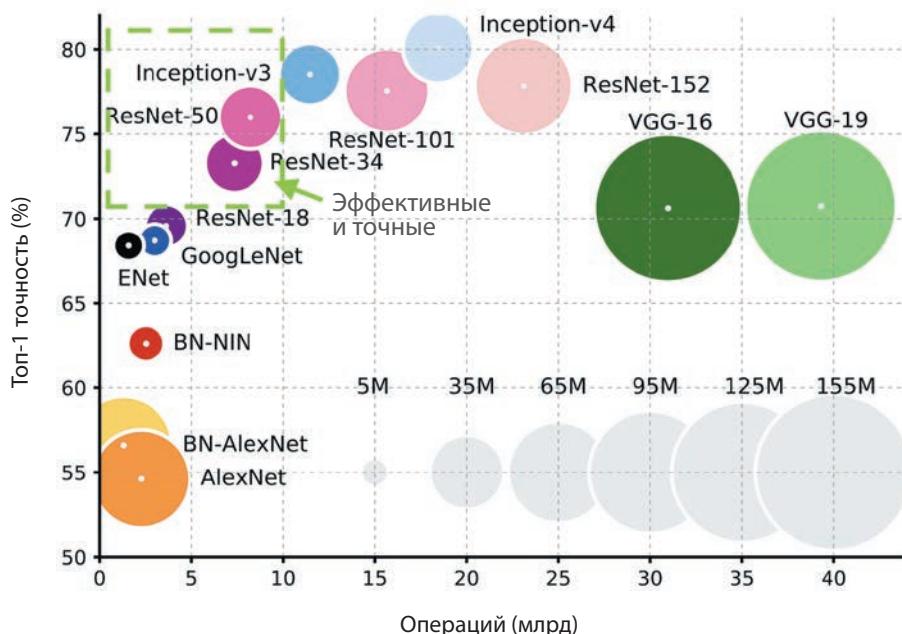
## Инференс

Обучение модели — это только полдела. В конце концов, модель должна представлять прогнозы пользователям. Советы ниже помогут вам сделать свой сервис более производительным.

### Используйте эффективную модель

Соревнования по глубокому обучению традиционно представляют собой гонку за тем, чтобы создать модель с наивысшей точностью, занять первое место в таблице лидеров и потом по праву хвастаться своими достижениями.

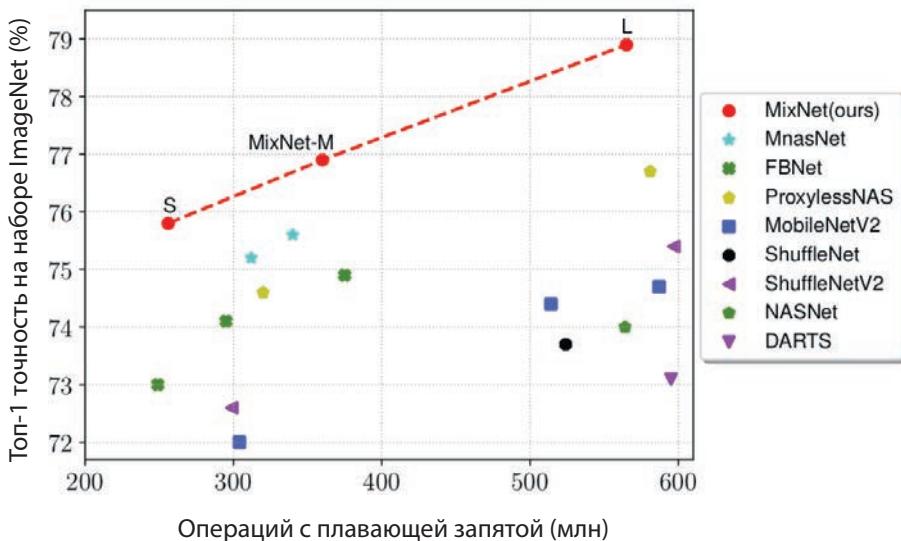
Но практики живут в другом мире — в мире быстрого и эффективного обслуживания пользователей. Для таких устройств, как смартфоны, краевые (edge) устройства и серверы, обслуживающие тысячи запросов в секунду, жизненно важна эффективность на всех направлениях (размер модели и продолжительность вычислений). В конце концов, не всякое устройство сможет обслуживать полугигабайтную модель VGG-16, которая выполняет 30 миллиардов операций даже при не самой высокой точности. Среди большого разнообразия доступных и предварительно обученных архитектур есть модели, обладающие высокой точностью, но и большей ресурсоемкостью, а также модели с умеренной точностью, зато намного более легковесные. Наша цель — выбрать архитектуру, которая обеспечит наивысшую точность с учетом доступной вычислительной мощности и объема памяти нашего устройства. Вверху слева на рис. 6.10 рамкой обведены модели, среди которых мы выбираем.



**Рис. 6.10.** Сравнение различных моделей по размеру, точности и количеству операций (изображение из статьи Альфредо Канциани (Alfredo Canziani), Адама Пашке (Adam Paszke) и Эудженио Кукурчелло (Eugenio Culurciello) «An Analysis of Deep Neural Network Models for Practical Applications»)

Наиболее эффективными для смартфонов являются модели из семейства MobileNet размером около 15 Мбайт, причем более свежие версии, такие как MobileNetV2 и MobileNetV3, обладают лучшими характеристиками, чем их

предшественницы. Кроме того, изменяя гиперпараметры моделей MobileNet, такие как множитель глубины (depth multiplier), можно дополнительно сократить объем вычислений, что делает их идеальным выбором для приложений реального времени. С 2017 года задача создания оптимальной архитектуры для максимальной точности была автоматизирована с помощью механизма NAS. Он помог обнаружить новые (довольно запутанные на вид) архитектуры, которые не раз ставили рекорды точности на наборе ImageNet. Например, модель FixResNeXt (на основе архитектуры PNASNet объемом 829 Мбайт) достигла колоссальных 86,4 % топ-1 точности. Таким образом, на закономерный вопрос «помогает ли NAS подобрать архитектуру с максимальной точностью, минимальным объемом вычислений и пригодную для мобильных устройств» был дан однозначный ответ: «Да». В результате были созданы еще более быстрые и точные модели, оптимизированные для имеющегося оборудования. Например, MixNet (июль 2019 г.) превосходит многие современные модели. Обратите внимание, что мы перешли от миллиардов операций с плавающей запятой к миллионам (рис. 6.10 и рис. 6.11).



**Рис. 6.11.** Сравнение нескольких моделей, оптимизированных для мобильных устройств (изображение заимствовано из статьи Миньхуна Тана (Mingxing Tan) и Куока В. Ле (Quoc V. Le) «MixNet: Mixed Depthwise Convolution Kernels»)

Где мы как практики можем найти современные модели? Лидеров по множеству задач ИИ можно найти по адресу [PapersWithCode.com/sota](http://PapersWithCode.com/sota). Здесь вы найдете сравнения и статьи с кодом моделей. Особый интерес представляют модели с небольшим числом параметров, обеспечивающие высокую точность. Например,

EfficientNet обладает потрясающими 84,4 % топ-1 точности с 66 миллионами параметров и может быть идеальным кандидатом для работы на серверах. Кроме того, показатели тестирования относятся к 1000 классам в ImageNet, тогда как вам может потребоваться классификация только по нескольким классам. В таких случаях часто достаточно модели гораздо меньшего размера. Модели, перечисленные в Keras Application (*tf.keras.applications*), TensorFlow Hub и TensorFlow Models, обычно имеют множество разновидностей (размеры исходных изображений, множители глубины, квантование и т. д.).



Вскоре после появления статьи исследователи из Google AI опубликовали модель, использованную в статье, в репозитории TensorFlow Models.

## Используйте квантование модели

*«Преобразуйте 32-битные веса в 8-битные целые числа и получите 2-кратное ускорение и 4-кратное уменьшение размеров».*

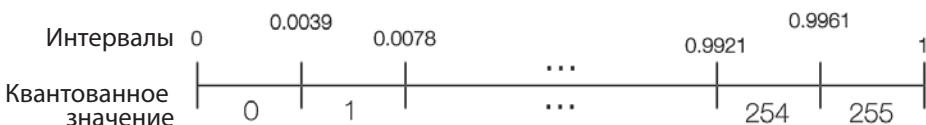
В работе нейронных сетей активно используется матричное умножение. Эта арифметика, как правило, довольно проста — небольшие отклонения в значениях не вызывают значительных колебаний в результатах. Это делает нейронные сети достаточно устойчивыми к шуму. В конце концов, для нас желательно, чтобы сеть правильно распознавала яблоко на картинке даже при неидеальном освещении. Используя квантование, мы, по сути, пользуемся преимуществами этой «прощающей» природы нейронных сетей.

Прежде чем перейти к обсуждению различных приемов квантования, попробуем сначала понять саму идею. Преобразуем 32-битные веса с плавающей запятой в INT8 (8-битные целые числа), используя линейное квантование. Очевидно, что FP32 может представлять  $2^{32}$  разных значений (и требует 4 байта для хранения), а INT8 может представлять  $2^8 = 256$  значений (1 байт). Для квантования:

1. Нужно найти в нейронной сети минимальное и максимальное значения весов в формате FP32.
2. Разделить полученный диапазон на 256 интервалов, каждый из которых соответствует значению INT8.
3. Вычислить коэффициент масштабирования, который преобразует INT8 (целое число) обратно в FP32. Например, если исходный диапазон составляет от 0 до 1, а числа INT8 — от 0 до 255, то коэффициент масштабирования будет равен  $1/256$ .
4. Заменить числа FP32 в каждом интервале значениями INT8. Дополнительно нужно сохранить коэффициент масштабирования для этапа инференса,

на котором значения INT8 будут преобразовываться обратно в значения FP32. Коэффициент масштабирования необходимо сохранить только один раз для всей группы квантованных значений.

5. Во время инференса нужно умножить значения INT8 на коэффициент масштабирования, чтобы преобразовать их обратно в представление с плавающей запятой. На рис. 6.12 показан пример линейного квантования для интервала  $[0, 1]$ .



**Рис. 6.12.** Квантование 32-битного диапазона с плавающей запятой  $[0, 1]$  в 8-битный целочисленный диапазон для уменьшения объема памяти

Есть несколько разных способов квантования моделей, и самый простой — уменьшение размеров представления весов с 32 до 16 бит или меньше. Как нетрудно догадаться, после преобразования 32-битных весов в 16-битные для хранения модели потребуется в два раза меньше памяти. Точно так же после преобразования в 8-битный формат для модели потребуется в четыре раза меньше памяти. Так почему бы не преобразовать все веса в 1-битное представление и получить 32-кратную экономию? Попробуем объяснить. Модели в какой-то степени прощаются ошибки, но с каждым снижением размера представления весов мы будем замечать падение точности. После определенного порога (особенно ниже 8 бит) точность будет падать экспоненциально. Чтобы уменьшить размер представления (например, до 1 бита) и по-прежнему иметь полезную рабочую модель, придется выполнить специальный процесс конвертирования модели в бинаризованную нейронную сеть. Начинающий проект по глубокому обучению XNOR.ai сумел внедрить этот прием в продакшен. Также в библиотеке Microsoft Embedded Learning Library (ELL) есть инструменты, важные для краевых устройств, таких как Raspberry Pi.

Квантование дает множество преимуществ.

#### *Экономное использование памяти*

При квантовании до 8-битного целочисленного представления (INT8) размер модели сокращается почти на 75 %. Это упрощает хранение модели и ее загрузку в память.

#### *Повышенная производительность*

Целочисленные операции выполняются быстрее операций с плавающей запятой. Кроме того, экономия памяти снижает необходимость выгружать

модель из ОЗУ во время выполнения, что также дает дополнительное преимущество в виде снижения энергопотребления.

### *Портативность*

Краевые устройства (например, IoT) могут не поддерживать арифметику с плавающей запятой, и в таких ситуациях нерационально сохранять веса модели в формате с плавающей запятой.

Большинство фреймворков для инференса: Apple Core ML Tools, NVIDIA TensorRT (для серверов) и TensorFlow Lite, а также Google TensorFlow Model Optimization Toolkit — предоставляют возможность квантования моделей. В TensorFlow Lite модели можно квантовать после обучения во время конвертирования (так называемое квантование после обучения). Чтобы еще больше минимизировать потери точности, можно использовать TensorFlow Model Optimization Toolkit во время обучения. Этот процесс называется *обучением с учетом квантования*.

Полезно оценить выгоды, обеспечиваемые квантованием. Результаты тестирования методов оптимизации в TensorFlow Lite (<https://oreil.ly/me4-I>; показаны в табл. 6.4) позволяют сравнить модели: 1) неквантованные, 2) квантованные после обучения и 3) обученные с учетом квантования. Производительность измерялась на устройстве Google Pixel 2.

**Таблица 6.4.** Результаты применения разных стратегий квантования (до 8 бит) моделей (источник: документация по оптимизации моделей в TensorFlow Lite)

<b>Модель</b>		<b>MobileNet</b>	<b>MobileNetV2</b>	<b>InceptionV3</b>
Топ-1 точности	Исходная	0,709	0,719	0,78
	Квантование после обучения	0,657	0,637	0,772
	Обучение с учетом квантования	0,7	0,709	0,775
Продолжительность вычисления прогноза (мс)	Исходная	124	89	1130
	Квантование после обучения	112	98	845
	Обучение с учетом квантования	64	54	543
Размер (Мбайт)	Исходная	16,9	14	95,7
	Квантование после обучения	4,3	3,6	23,9

Что означают эти числа? После квантования с использованием TensorFlow Lite до INT8 размер модели уменьшился почти в четыре раза, время прогнозирования

уменьшилось примерно в два раза, и при этом точность снизилась меньше чем на 1 %. Неплохо!

Более экстремальные формы квантования, такие как 1-битные бинаризованные нейронные сети (например, XNORNet), при тестировании с AlexNet в качестве эталона показывают колоссальное 58-кратное ускорение при примерно 32-кратном уменьшении размера и с потерей точности до 22 %.

## Прореживайте модели

Выберите число. Умножьте его на 0. Что получится? Ноль. А теперь умножьте выбранное число на небольшое значение, близкое к 0, например  $10^{-6}$ , и вы получите небольшое значение. Если заменить такие маленькие веса модели (близкие к нулю) нулем, это мало повлияет на прогнозы модели. Этот прием называется *прореживанием весов по величине* или просто прореживанием и представляет собой форму *сжатия модели*. По логике, обнуление веса между двумя вершинами узлов в полно связанным слое эквивалентно удалению ребра между ними. Это делает полно связанные слои модели более разреженными.

Как это часто бывает, многие веса в моделях близки к 0. Прореживание модели приведет к тому, что многие из этих весов обнулятся. Это, конечно, приведет к небольшой потере точности. Само по себе прореживание не экономит память, но вводит массу избыточности, которую можно использовать, когда придет время сохранить модель на диск, например сжать ее архиватором. (Стоит отметить, что алгоритмы сжатия особенно хорошо справляются с длинными последовательностями повторяющихся байтов. Чем больше повторений и чем они длиннее, тем выше степень сжатия.) В итоге модель часто можно сжать в четыре раза. Конечно, когда модель понадобится использовать, ее нужно будет распаковать.

Разработчики TensorFlow знают о потере точности, показанной в табл. 6.5, при прореживании моделей. Как и ожидалось, более эффективные прореживания модели, например MobileNet, демонстрируют более высокую (хотя и небольшую) потерю точности по сравнению с более крупными моделями вроде InceptionV3.

**Таблица 6.5.** Потеря точности модели в зависимости от степени прореживания

Модель	Разреженность	Потеря точности по сравнению с оригиналом
InceptionV3	50%	0,1%
InceptionV3	75%	2,5%
InceptionV3	87,5%	4,5%
MobileNet	50%	2%

Keras поддерживает возможность прореживания моделей. Этот процесс можно выполнять итеративно во время обучения. Обучите модель, как обычно, или выберите предварительно обученную модель. Затем периодически прореживайте модель и продолжайте обучение. При достаточном количестве эпох между периодическими прореживаниями модель будет успевать восстанавливаться после любых повреждений, обусловленных такой большой разреженностью. Величину разрежения и количество эпох между прореживаниями рассматривайте как гиперпараметры, которые нужно настроить.

Другой способ — использовать инструмент Tencent PocketFlow (<https://oreil.ly/JJms2>), предлагающий еще несколько стратегий прореживания, обнаруженных в результате недавних исследований.

## Используйте совмещенные операции

В любой серьезной сети CNN сверточный слой и слой пакетной нормализации часто размещаются друг за другом. Они вроде как Лелек и Болек среди слоев в CNN. По сути оба они представляют линейные операции. Из основ линейной алгебры мы знаем, что объединение двух или более линейных операций дает в результате одну линейную операцию. Объединив сверточные слои и слои пакетной нормализации, можно не только сократить количество вычислений, но и время, необходимое на передачу данных между CPU и GPU и между ОЗУ и регистрами или кэшем CPU. Их выполнение в виде одной операции сокращает число циклов. К счастью, большинство фреймворков инференса могут автоматически выполнять такое объединение или предоставляют средства конвертирования моделей (как, например, TensorFlow Lite), чтобы использовать эту оптимизацию при конвертировании модели в ходе ее подготовки для инференса.

## Включите сохранение состояния GPU

Загрузка и инициализация драйверов GPU требует времени. Это выражается в виде задержки, которую можно заметить в начале обучения или перед каждым запуском инференса. Когда модель часто используется, издержки могут быстро расти. Представьте программу классификации изображений, которая тратит на классификацию 10 секунд, из которых 9,9 секунды тратятся на загрузку драйвера. Нам нужно, чтобы драйвер графического процессора оставался инициализированным и готовым к выполнению заданий. В этом поможет NVIDIA GPU Persistence Daemon:

```
$ nvidia-persistenced --user {YOUR_USERNAME}
```

В этом случае GPU будет потреблять чуть больше электроэнергии во время простоя, зато останется готовым к работе при следующем запуске программы.

## Итоги

Мы рассмотрели способы повышения скорости и эффективности пайплайна глубокого обучения, от хранения и чтения данных до инференса. Медленный пайpline данных часто приводит к голоданию и простою GPU. Применив несколько простых оптимизаций, можно максимально эффективно использовать имеющееся оборудование. Чек-лист из этой главы послужит справочником. Распечатайте его и положите себе на стол (или повесьте на холодильник), чтобы он всегда был под рукой. Надеемся, что благодаря полученным знаниям вы сможете вписать свое имя в таблицу рекордов на MLPerf.

## ГЛАВА 7

---

# Практические инструменты, советы и приемы

В этой главе — ответы на вопросы, которые возникали у нас и во время повседневной работы, и в процессе написания этой книги. Обсуждаемые здесь темы трудно отнести к чему-то конкретному, что можно было бы включить в одну из глав. Это сведения, которые пригодятся практикующим специалистам по глубокому обучению в реальной работе при решении самых разных задач. Глава охватывает множество полезных практических рекомендаций по таким вопросам, как настройка окружения, обучение, взаимодействие моделей, сбор и разметка данных, качество кода, управление экспериментами, методы совместной работы в команде, конфиденциальность и т. д.

Из-за стремительных темпов развития в области ИИ эта глава представляет лишь небольшое подмножество «живого» документа, размещенного в репозитории книги (<https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке `code/chapter-7`, где он хранится и постоянно обновляется. Если у вас есть вопросы или, что еще лучше, ответы, которые помогут другим читателям, присылайте их в Twitter @PracticalDLBook (<https://www.twitter.com/PracticalDLBook>) или делайте пул-реквест в репозиторий.

## Установка

**Вопрос:** Я увидел интересный и полезный блокнот Jupyter на GitHub. Чтобы запустить его, нужно клонировать репозиторий, установить пакеты, настроить окружение и выполнить некоторые другие действия. А можно ли быстро запустить его в интерактивном режиме, минуя все эти шаги?

Просто введите URL-адрес репозитория Git в Binder (<http://mybinder.org/>) — и получите коллекцию интерактивных блокнотов. За кулисами Binder отыщет в корневом каталоге репозитория файл со списком зависимостей, например

*requirements.txt* или *environment.yml*, и использует его для создания образа Docker, с помощью которого можно интерактивно запускать блокноты в своем браузере.

**Вопрос:** *Как быстрее всего настроить окружение для глубокого обучения на новой машине с Ubuntu и NVIDIA GPU?*

Если бы команда `pip install tensorflow-gpu` решала все проблемы, жизнь была бы прекрасна. Но в реальности все намного сложнее. Перечисление всех шагов по настройке окружения на машине со свежеустановленным дистрибутивом Ubuntu займет не менее трех страниц, а для их выполнения, включая установку драйверов NVIDIA GPU, CUDA, cuDNN, Python, TensorFlow и других пакетов, потребуется больше часа. При этом нужно проверить совместимость версий CUDA, cuDNN и TensorFlow. Чаще всего это заканчивается разрушением системы. Мир полон боли!

Было бы здорово, если можно было бы решить все проблемы парой команд. Просите, и дано будет вам:

```
$ sudo apt update && sudo ubuntu-drivers autoinstall && sudo reboot
$ export LAMBDA_REPO=$(mktemp) \
&& wget -O${LAMBDA_REPO} \
https://lambdalabs.com/static/misc/lambda-stack-repo.deb \
&& sudo dpkg -i ${LAMBDA_REPO} && rm -f ${LAMBDA_REPO} \
&& sudo apt-get update && sudo apt-get install -y lambda-stack-cuda \
&& sudo reboot
```

Первая команда обновит все драйверы. Вторая команда обращается к Lambda Labs, поставщику оборудования для глубокого обучения и облачных вычислений из Сан-Франциско. Она установит комплект программного обеспечения Lambda Stack, в том числе TensorFlow, Keras, PyTorch, Caffe, Caffe2, Theano, CUDA, cuDNN и драйверы NVIDIA GPU. Поскольку компании приходится устанавливать одни и те же пакеты глубокого обучения на тысячи машин, она автоматизировала процесс, уместив его в единственную команду, а затем открыла исходный код, чтобы другие тоже могли его использовать.

**Вопрос:** *Как быстрее всего установить TensorFlow на компьютер с Windows?*

1. Установите Anaconda Python 3.7.
2. В командной строке выполните команду `conda install tensorflow-gpu`.
3. Если у вас нет GPU, выполните команду `conda install tensorflow`.

Еще одно преимущество пакета Conda, выполняющего вычисления на CPU: вместе с ним устанавливается версия TensorFlow с оптимизированной библиотекой Intel MKL, которая работает быстрее, чем версия, которую устанавливает `pip install tensorflow`.

**Вопрос:** У меня GPU от AMD. Могу ли я использовать его в TensorFlow в моей существующей системе?

Большая часть мира глубокого обучения использует GPU от NVIDIA, но постепенно растет сообщество людей, использующих оборудование от AMD с помощью стека ROCm. Все необходимое легко установить из командной строки:

1. `sudo apt install rocm-libs miopen-hip cxlactivitylogger`
2. `sudo apt install wget python3-pip`
3. `pip3 install --user tensorflow-rocm`

**Вопрос:** Где я могу получить контейнеры, подготовленные для глубокого обучения, чтобы не мучиться с установкой?

Docker является синонимом настройки окружения. Docker помогает запускать изолированные контейнеры, в которые упакованы все нужные инструменты, библиотеки и файлы конфигурации. Все основные поставщики услуг облачных вычислений (AWS, Microsoft Azure, GCP, Alibaba и др.) предлагают для установки на их виртуальные машины готовые контейнеры Docker с инструментами для глубокого обучения. Кроме того, NVIDIA предлагает бесплатные облачные контейнеры NVIDIA GPU Cloud — те же высокопроизводительные контейнеры, которые используются для оценки скорости обучения в состязаниях MLPerf. Вы даже можете запускать эти контейнеры на своем ПК.

## Обучение

**Вопрос:** Меня бесит, что надо постоянно смотреть на экран, чтобы увидеть, когда закончится обучение. Можно ли сделать отправку уведомления на смартфон?

Используйте Knock Knock (<https://oreil.ly/uX3qb>), библиотеку для Python. Как следует из названия<sup>1</sup>, по окончании обучения (или в случае аварии программы) вы получите уведомление по электронной почте, в Slack или даже в Telegram. Самое замечательное, что для этого в обучающий сценарий достаточно добавить всего две строки кода. И после этого не придется открывать окно программы снова и снова, чтобы проверить, закончилось ли обучение.

**Вопрос:** Я предпочитаю графический интерфейс текстовому. Могу ли я визуализировать процесс обучения в графическом интерфейсе в режиме реального времени?

Здесь поможет прогресс-бар FastProgress (первоначально разработанный Сильвеном Гуггером (Sylvain Gugger) для fast.ai).

---

<sup>1</sup> Knock Knock переводится как «тук-тук». — Примеч. пер.

**Вопрос:** Я часто повторяю эксперименты и забываю, что менялось между экспериментами и какое влияние оказывали эти изменения. Можно ли как-то упорядочить управление экспериментами?

Разрабатывая ПО, можно вести журнал истории изменений, используя систему управления версиями. К сожалению, раньше в машинном обучении не было такой роскоши. Но с появлением Weights and Biases и Comet.ml ситуация стала меняться к лучшему. Они позволяют фиксировать информацию об экспериментах, включая кривые обучения, гиперпараметры, выходные данные, использовавшиеся модели, примечания и многое другое, для чего достаточно добавить всего две строчки кода в сценарий обучения на Python. А благодаря поддержке облачных вычислений можно следить за экспериментами, даже находясь вдали от компьютера, и делиться результатами с другими.

**Вопрос:** Как проверить, использует ли фреймворк TensorFlow графический процессор на моем компьютере?

Выполните следующую команду:

```
tf.test.is_gpu_available()
```

**Вопрос:** На моем компьютере имеется несколько GPU. Я не хочу, чтобы мой обучающий сценарий использовал их все. Можно ли ограничить сценарий так, чтобы он использовал только определенный GPU?

Используйте переменную окружения `CUDA_VISIBLE_DEVICES=GPU_ID`. Просто добавьте ее инициализацию перед командой, запускающей сценарий, как показано ниже:

```
$ CUDA_VISIBLE_DEVICES=GPU_ID python train.py
```

Также можно добавить следующие строки в начало обучающего сценария:

```
import os  
os.environ["CUDA_VISIBLE_DEVICES"]="GPU_ID"
```

`GPU_ID` может иметь числовые значения, такие как 0, 1, 2 и т. д. Увидеть эти числовые идентификаторы (вместе с информацией о загруженности GPU) можно с помощью команды `nvidia-smi`. Чтобы разрешить использовать несколько GPU, перечислите числовые идентификаторы через запятую.

**Вопрос:** Иногда мне кажется, что у модели слишком много регуляторов, которыми нужно управлять во время обучения. Можно ли управлять ими автоматически, чтобы добиться максимальной точности?

Есть много инструментов для автоматической настройки гиперпараметров, включая версию Hyperas для Keras и Keras Tuner, а также более универсальные фреймворки, такие как Hyperopt и Bayesian, которые интенсивно эксперимен-

тируют, стараясь максимизировать заданную цель (точность в нашем случае), и действуют более интеллектуально, чем простой алгоритм перебора.

**Вопрос:** *ResNet и MobileNet достаточно хорошо соответствуют моим потребностям. А можно ли построить модель, которая обеспечит еще более высокую точность в моем сценарии?*

Запомните три ключевых слова: поиск нейронной архитектуры (Neural Architecture Search, NAS). Позвольте алгоритму подобрать для вас лучшую архитектуру. Поиск нейронной архитектуры можно реализовать с помощью Auto-Keras и AdaNet.

**Вопрос:** *Как отладить сценарий TensorFlow?*

Ответ скрыт в самом вопросе: используйте отладчик TensorFlow Debugger (`tfdbg`).

## Модель

**Вопрос:** *Я хочу быстро увидеть, как устроены слои ввода и вывода в моей модели, не погружаясь в код. Возможно ли это?*

Используйте Netron. Этот инструмент отобразит вашу модель в графическом интерфейсе. Щелкнув на любом слое в этом интерфейсе, вы получите подробную информацию о его архитектуре.

**Вопрос:** *Хочу опубликовать свою исследовательскую работу. С помощью какого инструмента можно нарисовать архитектуру моей органической безглютеновой модели?*

Конечно же, в MS Paint! Шутка. Мы фанаты NN-SVG и PlotNeuralNet, и используем эти инструменты для создания высококачественных диаграмм сверточных сетей.

**Вопрос:** *Есть ли какой-то зоопарк моделей?*

Конечно! Для начала загляните в [PapersWithCode.com](#), [ModelZoo.co](#) и [ModelDepot.io](#).

**Вопрос:** *Я закончил обучение своей модели. Как сделать ее доступной другим для использования?*

Можно выгрузить модель в репозиторий на GitHub, а затем сообщить об этом в зоопарках моделей, перечисленных в предыдущем ответе. Для еще более широкого распространения выгрузите ее в TensorFlow Hub (`tflib.dev`).

Кроме самой модели опубликуйте также «карточку модели» — небольшой отчет, содержащий информацию об авторе, показателях точности и датасетах,

на которых модель обучена. Также перечислите потенциальные предвзятости и упомяните возможность использования за пределами целевой области.

**Вопрос:** У меня есть модель, обученная в фреймворке X, но мне нужно использовать ее в фреймворке Y. Можно ли обойтись без повторного обучения в фреймворке Y?

Да. Но для этого нужно, чтобы модель хранилась в формате ONNX. Большинство библиотек глубокого обучения, не входящих в экосистему TensorFlow, поддерживают возможность сохранения обученных моделей в формате ONNX, который затем можно преобразовать в формат TensorFlow. В преобразовании может помочь набор утилит Microsoft MMdnn.

## Данные

**Вопрос:** Можно ли собрать сотни изображений по определенной тематике за несколько минут?

Да, с помощью расширения Fatkun Batch Download Image для Chrome. Выполните поиск по ключевому слову в своей любимой поисковой системе изображений, отфильтруйте изображения по правам использования (например, Public Domain) и щелкните на значке расширения Fatkun, чтобы загрузить все изображения. В главе 12 мы покажем, как его использовать, когда будем создавать приложение Not Hotdog.

Совет: чтобы загрузить изображения только с одного сайта, выполните поиск по ключевому слову, добавив после него `site:адрес_сайта`. Например: «`horse site:flickr.com`».

**Вопрос:** Как загрузить изображения из Google с помощью командной строки, без браузера?

```
$ pip install google_images_download  
$ googleimagesdownload -k=horse -l=50 -r=labeled-for-reuse
```

`-k`, `-l` и `-r` — это короткие версии длинных ключей `keyword` (ключевое слово), `limit` (предел, количество изображений) и `usage_rights` (права использования) соответственно. Это мощный инструмент с множеством параметров для управления и фильтрации изображений, загружаемых из результатов поиска в Google. Кроме того, вместо миниатюр, отображаемых в Google Images, он сохраняет исходные изображения, найденные поисковой системой. Чтобы получить возможность сохранять более 100 изображений, установите библиотеку `selenium` вместе с `chromedriver`.

**Вопрос:** Возможностей этих инструментов недостаточно для сбора изображений. Нужен более полный контроль. Какие еще инструменты можно использовать для загрузки данных другими способами, помимо поисковой системы?

Графический интерфейс (программирование не требуется):

### *ScrapeStorm.com*

Простой графический интерфейс для определения правил извлечения данных.

### *WebScraper.io*

Расширение Chrome для парсинга, особенно хорошо подходит для извлечения структурированного вывода с отдельных сайтов.

### *80legs.com*

Масштабируемый облачный парсер для параллельного выполнения больших задач.

Программные инструменты на основе Python:

### *Scrapy.org*

Один из самых известных инструментов извлечения данных из интернета, предлагающий обширные возможности настройки программным способом. Позволяет регулировать скорость загрузки по доменам, прокси-серверам и IP-адресам; может обрабатывать файлы *robots.txt*; предлагает гибкую обработку заголовков браузера и заботится о некоторых необычных крайних случаях. Все это было бы сложно реализовать в своем простом парсере для исследования сайтов.

### *InstaLooter*

Инструмент на Python для парсинга Instagram.

**Вопрос:** У меня есть изображения для целевых классов, но теперь нужны изображения для отрицательного класса (где отсутствует искомый элемент или изображен только фон). Есть ли быстрый способ создать большой набор изображений таких отрицательных классов?

ImageN (<https://oreil.ly/s4Nyk>) предлагает 1000 изображений — по 5 случайных изображений для 200 категорий, которые можно использовать в качестве отрицательного класса. Если понадобится больше изображений, загрузите случайные образцы из ImageNet, выбранные программно.

**Вопрос:** Как найти готовый датасет, соответствующий моим потребностям?

Попробуйте Google Dataset Search, *VisualData.io* и *DatasetList.com*.

**Вопрос:** При использовании таких датасетов, как ImageNet, скачивание данных, определение их формата и последующая загрузка для обучения занимает слишком много времени. Есть ли более простой способ чтения популярных датасетов?

TensorFlow Datasets — постоянно растущая коллекция датасетов, готовых к использованию с TensorFlow. Она включает, кроме всего прочего, наборы ImageNet,

COCO (37 Гбайт) и Open Images (565 Гбайт). Эти наборы данных представлены в формате `tf.data.Datasets` вместе с эффективным кодом для их передачи в пайпайн обучения.

**Вопрос:** *Обучение на миллионах изображений из ImageNet занимает много времени. Есть ли достаточно репрезентативный датасет меньшего размера, на котором можно быстро обучать свою модель, экспериментируя с настройками?*

Попробуйте Imagenette (<https://oreil.ly/NpYBe>). Этот датасет, созданный Джереми Ховардом (Jeremy Howard) из fast.ai, имеет объем 1,4 Гбайт и содержит всего 10 классов вместо 1000.

**Вопрос:** *Какие самые большие датасеты из доступных можно использовать для обучения?*

- Tencent ML Images: 17,7 миллиона изображений в 11 000 категорий.
- Open Images V4 (Google): 9 миллионов изображений в 19 700 категорий.
- BDD100K (Калифорнийский университет в Беркли): изображения из 100 000 видеороликов о вождении с суммарной продолжительностью более 1100 часов.
- YFCC100M (Yahoo): 99,2 миллиона изображений.

**Вопрос:** *Какие из доступных больших наборов видеоданных можно использовать?*

**Таблица 7.1.** Большие доступные наборы видеоданных

Название	Описание
YouTube-8M	6,1 миллиона видеороликов, 3862 класса, 2,6 миллиарда аудиовизуальных признаков. 3,0 метки на видеоролик. 1,53 Тбайт случайно выбранных видео
Something Something (от компании Twenty Billion Neurons)	221 000 видеороликов, 174 класса действий. Например, «Вы наливаете воду в бокал для вина, но промахиваетесь и проливаете мимо» Люди выполняют предопределенные действия с повседневными предметами
Jester (от компании Twenty Billion Neurons)	148 000 видеороликов, 27 классов. Например, «Увеличение двумя пальцами». Предопределенные жесты рук перед веб-камерой

**Вопрос:** *Эти наборы — самые большие размеченные наборы данных, которые когда-либо собирались?*

Нет! Facebook и Google создают собственные закрытые датасеты, которые намного больше общедоступных:

- Facebook: 3,5 миллиарда изображений в Instagram с нечеткими метками (впервые об этом было сообщено в 2018 году).
- Google — JFT-300M: 300 миллионов изображений с нечеткими метками (впервые об этом было сообщено в 2017 году).

К сожалению, если вы не сотрудник одной из этих компаний, то не можете получить доступ к этим датасетам. Надо сказать, хорошая тактика приема на работу.

**Вопрос:** *Как получить помощь в аннотировании данных?*

Есть несколько компаний, которые помогают с аннотированием. В их числе SamaSource, Digital Data Divide и iMerit. Они нанимают людей с ограниченными возможностями, что в итоге приводит к положительным социально-экономическим изменениям благодаря увеличению занятости среди незащищенных категорий населения.

**Вопрос:** *Есть ли инструменты управления версиями для датасетов вроде Git для программного кода?*

Qri и Quilt могут помочь в управлении версиями датасетов и способствовать повышению воспроизводимости экспериментов.

**Вопрос:** *Что делать, если у меня нет доступа к большому датасету, подходящему для моей уникальной задачи?*

Создайте синтетический датасет. Например, найдите реалистичную трехмерную модель интересующего объекта и поместите ее в реалистичное окружение, используя фреймворк трехмерной графики, например Unity. Отрегулируйте освещение и положение камеры, масштабирование и поворот и сделайте снимки этого объекта под разными углами. Так можно сгенерировать неисчерпаемый запас обучающих данных. AI.Reverie, CVEDIA, Neuromation, Cognata, Mostly.ai, DataGen Tech и другие компании предлагают услуги реалистичного моделирования для целей обучения. Одно из больших преимуществ синтезированных обучающих данных — включение разметки в процесс синтеза. В конце концов, вы знаете, что создаете. Такая автоматическая разметка позволит сэкономить много денег и усилий по сравнению с разметкой вручную.

## Защищенность

**Вопрос:** *Как создать модель, обеспечивающую конфиденциальность без криптографических ухищрений?*

Возможно, TensorFlow Encrypted — это то, что вы ищете. Этот фреймворк позволяет разрабатывать модели, работающие с зашифрованными данными, что особенно актуально, если вы работаете в облаке. Большой объем безопасных многосторонних вычислений и гомоморфного шифрования позволяют получить конфиденциальное машинное обучение.

**Вопрос:** *Можно ли сохранить модель в секрете от посторонних глаз?*

К сожалению, если вы не в облаке, то веса будут доступны и их можно будет реконструировать. Используйте библиотеку Fritz для защиты IP-адреса модели при развертывании на смартфонах.

## Обучение и исследования

**Вопрос:** *Хочу стать экспертом по искусственному интеллекту. Где еще, кроме этой книги, найти информацию по ИИ?*

В интернете есть ресурсы для более подробного знакомства с глубоким обучением. Настоятельно рекомендуем посмотреть видеолекции от лучших преподавателей, охватывающие самые разные прикладные области — от компьютерного зрения до обработки естественного языка.

- Fast.ai от Джереми Ховарда (Jeremy Howard) и Рэйчел Томас (Rachel Thomas) — это бесплатная серия из 14 лекций, где показан более практический подход с PyTorch. Вместе с курсами вы найдете экосистему инструментов и активное сообщество, опубликовавшее множество исследовательских работ и готового к использованию кода (например, три строки для обучения современной сети с использованием fast.ai).
- Deeplearning.ai от Эндрю Йна (Andrew Ng) включает пять курсов «Deep Learning Specialization». Курсы бесплатные (но может быть небольшая плата за сертификат), они помогут укрепить теоретический фундамент. Первый курс доктора Йна по машинному обучению на Coursera прослушали более двух миллионов студентов, и эта серия продолжает традицию доступного контента, любимого новичками и экспертами.
- Нельзя упомянуть и о платформе онлайн-обучения O'Reilly's Online Learning (<http://oreilly.com/>). Там вы найдете сотни книг, видеолекций, онлайн-тренингов и статей от ведущих практиков и теоретиков с конференций O'Reilly's, посвященных ИИ и данным. Этот ресурс помогает более чем двум миллионам пользователей продвигаться по карьерной лестнице.

**Вопрос:** *Где найти интересные блокноты с примерами машинного обучения?*

Google Seedbank – коллекция интерактивных примеров машинного обучения. Эти блокноты Jupyter, созданные на базе Google Colaboratory, можно запускать без установки дополнительного ПО. Вот несколько интересных примеров:

- генерация звука с помощью генеративно-состязательной сети;
- распознавание действий на видео;
- создание текста в стиле Шекспира;
- передача аудиостиля.

**Вопрос:** *Где узнать, как обстоят дела по конкретной теме?*

Учитывая, насколько быстро развиваются современные технологии ИИ, для поиска последних публикаций и моделей, датасетов, задач и многое другое используйте удобный инструмент командной строки SOTAWHAT. Например, чтобы найти последние результаты обучения на ImageNet, выполните команду `sotawhat imagenet`. На сайте [paperwithcode.com/sota](http://paperwithcode.com/sota) есть репозитории статей, исходного кода и моделей, а также интерактивная визуальная шкала бенчмарков.

**Вопрос:** *Я прочитал статью на Arxiv, и она мне очень понравилась. Придется ли мне писать код с нуля, чтобы реализовать описанную в ней модель?*

Конечно нет! Расширение ResearchCode для Chrome упростит поиск кода при просмотре [arxiv.org](https://arxiv.org) или Google Scholar. Для этого достаточно щелкнуть на значке расширения. Код можно найти и без установки расширения, заглянув на сайт [ResearchCode.com](http://ResearchCode.com).

**Вопрос:** *Я не хочу писать код, но хочу поэкспериментировать с моделью с помощью камеры в интерактивном режиме. Так можно?*

Runway ML (<https://runwayml.com/>) – простой в использовании и мощный инструмент с графическим интерфейсом, который позволяет загружать модели (из интернета или собственные) и использовать веб-камеру или другой источник данных, например видеофайлы, для интерактивной генерации результатов. С его помощью можно комбинировать и повторно микшировать результаты, сгенерированные моделями, для создания новых творений. И это в пару кликов мышью. Поэтому инструмент привлек обширное сообщество художников.

**Вопрос:** *Если я могу экспериментировать, значит, могу и обучать модели, не написав ни строчки кода?*

Мы подробно обсудим это в главе 8 (для веб-приложений) и в главе 12 (для обычных приложений). А пока лишь отметим, что такие инструменты, как Microsoft CustomVision.ai (<http://customvision.ai/>), Google Cloud AutoML Vision, Clarifai, Baidu EZDL и Apple Create ML, предлагают возможность обучения перетаскиванием мышью. Некоторым из этих инструментов достаточно нескольких секунд для обучения.

## Последний вопрос

**Вопрос:** Расскажите какой-нибудь прикол про глубокое обучение.

Распечатайте и повесьте рядом с куллером плакат с рис. 7.1 ([keras4kindergartners.com](http://keras4kindergartners.com)) и понаблюдайте за реакцией людей.



**Рис. 7.1.** Сатирический плакат о состоянии ИИ с сайта [keras4kindergartners.com](http://keras4kindergartners.com) (Текст на плакате: Вы любите своего ребенка? Тогда познакомьте его с машинным обучением. Устроить ребенка в Гарвард нынче непросто. Но есть отличный способ выделиться на общем фоне — стать творцом, идеяным лидером и инфлюэнсером глубокого обучения. Мы разработали революционную программу глубокого обучения, нацеленную на ваш самый ценный актив. Всего за четыре часа в неделю мы научим вашего ребенка основам, которые позволят ему подняться до небывалых высот в мире глубокого обучения. Классы заполняются быстро, действуйте прямо сейчас!)

## ГЛАВА 8

# Облачные API для компьютерного зрения: установка и запуск за 15 минут

Из-за неоднократных инцидентов и аварий на городской атомной электростанции в библиотеке Спрингфилда (название штата<sup>1</sup> указать мы не можем) решили, что хранить ценные архивы в физическом виде слишком рискованно. Узнав, что в библиотеке, находящейся в городе-сопернике Шелбивилле, начали оцифровывать свой фонд, спрингфилдцы тоже решили поучаствовать в игре. В конце концов, их коллекция статей, таких как «Старик кричит на облако» и «Местный житель считает, что рестлеры дерутся по-настоящему», а также столетние культовые фотографии Каньона и статуя основателя города Джебедайя Спрингфилда незаменимы. Помимо устойчивости к авариям, архивы станут доступными для поиска и исследований. И конечно же, жители Спрингфилда получат доступ ко всем материалам, не вставая с диванов.

Первый шаг в оцифровке документов — это, разумеется, сканирование. Это самая легкая часть. Затем начинается настоящая работа — обработка и классификация визуальных образов. У команды Спрингфилда было несколько вариантов.

- Вручную классифицировать каждую отдельную страницу и каждую фотографию. Учитывая более чем 200-летнюю богатую историю города, такая работа займет очень много времени, будет подвержена ошибкам и обойдется довольно дорого. Классификация вручную всех полученных материалов была бы настоящим испытанием.
- Нанять команду специалистов по обработке данных для построения системы классификации изображений. Это был бы отличный план, но в нем есть одна заминка. Для библиотеки, которая работает на пожертвования, наем

<sup>1</sup> Авторы имеют в виду вымышленный город Спрингфилд из мультсериала «Симпсоны» и далее описывают персонажей из сериала. — Примеч. ред.

команды специалистов по анализу данных означал бы быстрое исчерпание ее бюджета. Специалист по обработке данных легко может оказаться самым высокооплачиваемым не только среди сотрудников библиотеки, но и во всем Спрингфилде (за исключением богатого промышленника Монтгомери Бернса).

- Найти кого-нибудь, кто смог бы написать сценарий, использующий интеллект готовых к использованию библиотек машинного зрения.

Они выбрали третий вариант — самый быстрый и недорогой. Им повезло. Мартин Принс, прилежный ученик четвертого класса начальной школы Спрингфилда, который, как оказалось, немного умел программировать, вызвался написать для них программу. Мартин плохо разбирался в глубоком обучении (в конце концов, ему всего 10 лет), но обладал базовыми навыками программирования и знал, как посылать запросы к REST API из сценариев на Python. И это действительно все, что ему нужно было знать. Фактически ему потребовалось чуть меньше 15 минут, чтобы отправить первый запрос.

Мартин решил использовать простой *порядок действий*: отправить отсканированное изображение в облачный API, получить прогноз и сохранить его в базе данных для последующего использования, а затем повторить все то же самое для каждого следующего изображения, принадлежащего библиотеке. Ему просто нужно было выбрать правильный инструмент.

Все крупные компании — Amazon, Google, IBM, Microsoft — предоставляют похожие API компьютерного зрения, которые классифицируют изображения, обнаруживают и распознают лица и знаменитостей, идентифицируют похожие изображения, читают текст и иногда распознают почерк. Некоторые даже предлагают возможность обучить свой классификатор, не написав ни единой строчки кода. Очень удобно!

Эти компании постоянно работают над улучшением своих механизмов компьютерного зрения. Они тратят миллионы на сбор и разметку датасетов с детальной таксономией, намного превосходящих датасет ImageNet. Их API полны потом, кровью и слезами исследователей (и приправлены счетами за электричество).

Простота использования и адаптации, разнообразие функций, разнообразие тегов и конкурентоспособные цены делают облачные API весьма привлекательными. И при этом нет нужды нанимать команду дорогостоящих специалистов по анализу данных. В главах 5 и 6 мы обсуждали приемы оптимизации для достижения более высокой точности и производительности соответственно, а в этой главе поговорим об оптимизации человеческих ресурсов.

Мы рассмотрим несколько облачных API распознавания образов. Сравним их как качественно, так и количественно. Надеемся, что это упростит вам выбор API, наиболее подходящих для ваших приложений. А если они вдруг окажутся

непригодными для ваших целей, мы покажем, как обучить свой классификатор всего в несколько кликов.

(Для полноты понимания отметим, что некоторые из авторов этой книги работали в Microsoft, чьи предложения обсуждаются здесь. Мы постарались, чтобы этот опыт не повлиял на наши результаты, построили воспроизводимые эксперименты и обосновали методологию.)

## Ландшафт API распознавания образов

Рассмотрим некоторые API распознавания образов.

### Clarifai

Компания Clarifai (рис. 8.1) стала победителем в конкурсе классификаторов ILSVRC 2013 года. Основанная Мэттью Зайлером (Matthew Zeiler), аспирантом Нью-Йоркского университета, она оказалась в числе первых компаний, занимающихся распознаванием образов.



Забавный факт: при разработке классификатора для поиска изображений с предупреждением NSFW (Not Safe For Work — небезопасно для работы) в какой-то момент потребовалось понять, что узнала сверточная сеть, чтобы отладить ее и уменьшить количество ложных срабатываний. Это привело к тому, что в Clarifai изобрели метод визуализации, позволяющий узнать, какие изображения оказывают стимулирующее воздействие на карты признаков в любом слое сверточной сети. Как говорится, необходимость — мать изобретательства.

The screenshot shows the Clarifai API interface. At the top, there are tabs for GENERAL, FACE, NSFW, and COLOR. On the right, there's a 'MORE MODELS' dropdown and a 'VIEW DOCS' link. The main area displays a photograph of a young child in a high chair feeding a dog. To the right of the image is a table of predicted concepts and their probabilities:

PREDICTED CONCEPT	PROBABILITY
child	0.987
family	0.979
indoors	0.976
cute	0.972
people	0.937
little	0.934
dog	0.925
one	0.989
sit	0.987

At the bottom, there's a blue button labeled 'TRY YOUR OWN IMAGE OR VIDEO' with a camera icon.

**Рис. 8.1.** Пример результатов Clarifai

## Уникальные особенности этого API

API предлагает многоязычные теги на более чем 23 языках, возможность поиска визуального сходства среди ранее выгруженных фотографий, классификацию национальностей по лицам, эстетическую оценку фотографий, оценку фокуса и создание векторных представлений (эмбеддингов), чтобы помочь в разработке своих инструментов поиска по изображениям. Кроме того, есть возможность распознавания в узкоспециализированных областях, включая одежду и моду, путешествия и гостеприимство, а также свадьбы. Через свой общедоступный API инструмент разметки изображений поддерживает 11 000 понятий.

## Microsoft Cognitive Services

С созданием ResNet-152 в 2015 году Microsoft смогла выиграть семь конкурсов на ILSVRC, в конкурсе COCO Image Captioning Challenge, а также Emotion Recognition in the Wild, начиная с классификации и обнаружения (локализации) и заканчивая описанием изображений. И большую часть своих разработок компания перевела в облачные API. Первоначально проект получил название Project Oxford в Microsoft Research, а в 2016 году был переименован в Cognitive Services. Этот исчерпывающий набор включает более 50 API, в том числе компьютерное зрение, обработку текстов на естественных языках, распознавание речи, поиск, связывание графов знаний и многое другое. Исторически многие из библиотек использовались в командах Xbox и Bing, а теперь доступны и для сторонних разработчиков. В числе популярных приложений, демонстрирующих творческие способы использования этих API, можно назвать *how-old.net*<sup>1</sup> (на сколько лет я выгляжу?), Mimicker Alarm (где надо сделать определенное выражение лица, чтобы отключить будильник) и *CaptionBot.ai*.

## Уникальные особенности этого API

Как показано на рис. 8.2, API предлагает возможность создания подписей к изображениям, распознавание почерка и головных уборов. Из-за большого количества корпоративных клиентов Cognitive Services не использует изображения клиентов для улучшения своих услуг.

## Google Cloud Vision

В 2014 году Google победила в конкурсе ILSVRC со своей 22-слойной сетью GoogLeNet, которая в итоге положила начало семейству архитектур Inception. В дополнение к моделям Inception в декабре 2015 года Google выпустила набор

---

<sup>1</sup> Приложение можно найти по адресу <https://www.microsoft.com/ru-ru/p/microsoft-how-old/9nblggh2wtwg>. — Примеч. науч. ред.

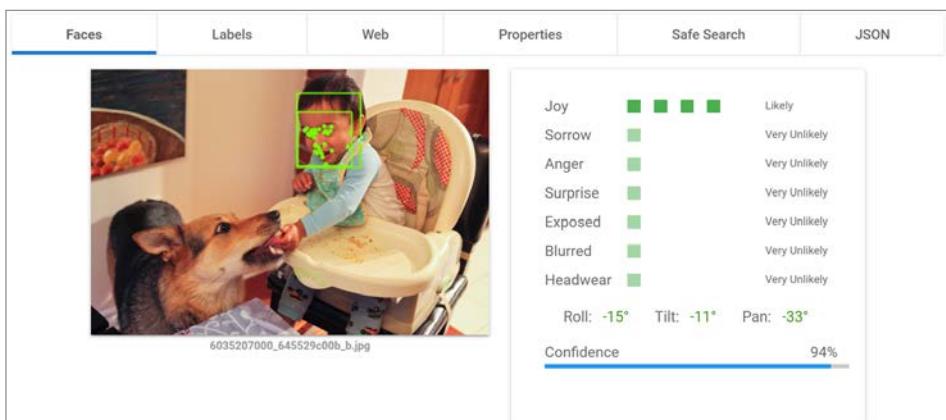
Vision API. Наличие больших объемов данных — это большое преимущество в мире глубокого обучения, а у Google есть очень много данных о потребителях. Так, обучив сеть на Google Street View, можно ожидать от нее хорошей производительности в реальных задачах извлечения текста, например, с рекламных щитов.



**Рис. 8.2.** Пример результатов Microsoft Cognitive Services

### Уникальные особенности этого API

При анализе изображений человеческих лиц этот API может выделить ключевые точки (рис. 8.3) и определить другие параметры, например поворот и наклон для точной локализации черт лица. Кроме того, API может выполнять в интернете поиск похожих изображений. Простой способ протестировать систему без написания кода — выгрузить фотографии в Google Photos и сделать поиск по тегам.



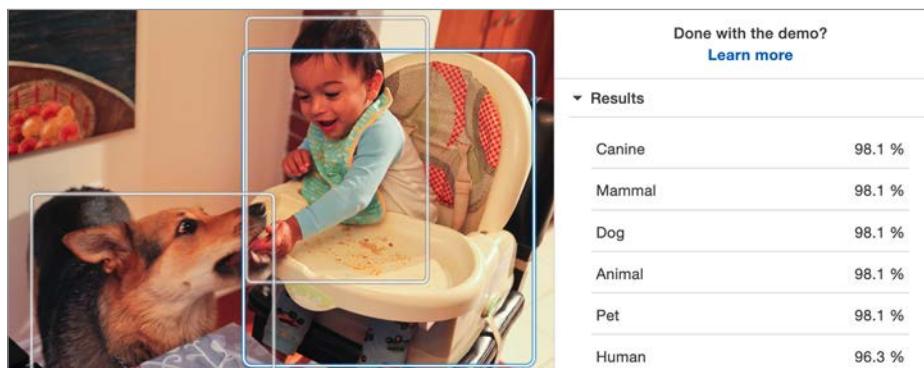
**Рис. 8.3.** Пример результатов Google Cloud Vision

## Amazon Rekognition

Нет, это не опечатка. Amazon Rekognition API (рис. 8.4) в значительной степени основан на разработках Orbeus, стартапа из Саннивейла, штат Калифорния, которая была приобретена Amazon в конце 2015 года. Ее главный научный сотрудник выиграл конкурс ILSVRC 2014 по обнаружению. Те же API использовались в PhotoTime, известном приложении для организации фотографий. Сервисы API доступны как часть предложений AWS. Учитывая, что большинство компаний уже предлагают API для анализа фотографий, Amazon удвоила ставки, предложив также возможность распознавания по видео.

### Уникальные особенности этого API

Распознавание номерных знаков, API распознавания по видео и лучшие примеры сквозной интеграции Rekognition API с такими предложениями AWS, как Kinesis Video Streams, Lambda и др. Кроме того, Amazon API – единственный, способный определить, открыты или закрыты глаза объекта на изображении.



**Рис. 8.4.** Пример результатов Amazon Rekognition

## IBM Watson Visual Recognition

Предложение IBM Visual Recognition под брендом Watson появилось в начале 2015 года. После покупки AlchemyAPI, стартапа из Денвера, в IBM использовали их разработку AlchemyVision для поддержки Visual Recognition API (рис. 8.5). Как и другие компании, IBM также предлагает возможность обучения своего классификатора. Удивительно, но пока Watson не поддерживает оптическое распознавание символов.

**Watson sees...**



**Classes**

Classes	Score
favorite	0.69
person	0.72
highchair	0.52
chair	0.52
seat	0.52
furniture	0.52
animal breeding	0.52
caretaker	0.50
dog	0.60
Animal Tending	0.60
light brown color	0.77
reddish orange color	0.56

Type Hierarchy

- /person/favorite
- /furniture/seat/chair/highchair
- /person/caretaker

*Did We Wow You?*  Yes  No

**Food**

Food	Score
cake mix	0.77
ready-mix	0.77
food mix	0.77
food mixture	0.77
pottage	0.50
soup	0.50

*Did We Wow You?*  Yes  No

**Faces**

Faces	Score
age 18 - 24	0.47
female	0.82

*Did We Wow You?*  Yes  No

**Рис. 8.5.** Пример результатов IBM Watson Visual Recognition

## Algorithmia

Algorithmia — это площадка для размещения алгоритмов в виде API в облаке. Стартап Algorithmia из Санта-Клары был основан в 2013 году и предлагает собственные алгоритмы и алгоритмы, созданные другими (в этом случае создатели получают доход в зависимости от количества запросов). По нашему опыту, у этого API самое большое время отклика.

### Уникальные особенности этого API

Раскрашивание черно-белых фотографий (рис. 8.6), стилизация изображений, определение сходства изображений и возможность запускать эти службы локально или у любого провайдера облачных услуг.



Deep-Style.io

**Рис. 8.6.** Пример результатов Algorithmia

При таком большом количестве предложений выбор подходящего может стать непосильной задачей. Есть много причин, по которым можно предпочесть одно предложение другому. Очевидно, что самыми важными факторами для большинства разработчиков будут точность и цена. Точность — важное обещание, которое дает революцию в области глубокого обучения, и многие приложения требуют высокой точности постоянно. Стоимость услуги — еще

один важный фактор. Мы можем выбрать определенного провайдера услуг, потому что у нашей компании уже есть с ним контракт, а для интеграции со службами другого провайдера могут потребоваться дополнительные усилия. Еще один фактор — скорость ответа API, особенно если на другом конце пользователь ожидает ответа. Поскольку многие из обращений к API можно абстрагировать, организовать переключение между разными провайдерами в принципе несложно.

## Сравнение API для распознавания образов

Сравним эти API между собой, чтобы вы могли принять решение. В этом разделе рассмотрим предлагаемые возможности, стоимость и точность каждого из них.

### Предлагаемые услуги

В табл. 8.1 перечислены услуги, предлагаемые каждым облачным провайдером.

**Таблица 8.1.** Сравнительный анализ провайдеров API для распознавания образов (по состоянию на август 2019)

	Algorithmia	Amazon Rekognition	Clarifai	Microsoft Cognitive Services	Google Cloud Vision	IBM Watson Visual Recognition
Классификация изображений	✓	✓	✓	✓	✓	✓
Определение изображений	✓	✓		✓	✓	
Оптическое распознавание символов	✓	✓		✓	✓	
Распознавание лиц	✓	✓		✓		
Распознавание эмоций	✓		✓	✓	✓	
Распознавание логотипов			✓	✓	✓	
Распознавание достопримечательностей			✓	✓	✓	✓
Распознавание знаменитостей	✓	✓	✓	✓	✓	✓

Таблица 8.1 (окончание)

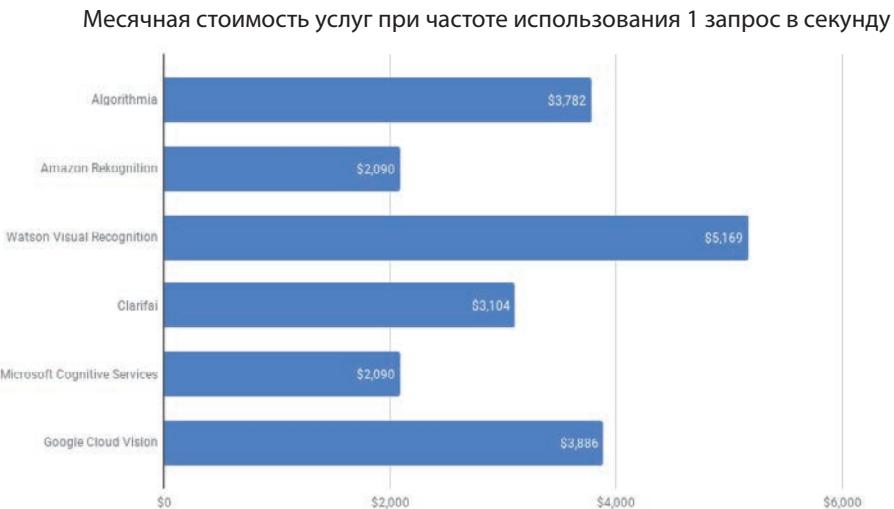
	Algorithmia	Amazon Rekognition	Clarifai	Microsoft Cognitive Services	Google Cloud Vision	IBM Watson Visual Recognition
Многоязычные теги			✓			
Подписи к изображениям				✓		
Распознавание рукописного текста				✓	✓	
Создание миниатюр	✓			✓	✓	
Модерирование контента	✓	✓	✓	✓	✓	
Обучение своего классификатора			✓	✓	✓	✓
Обучение своего детектора				✓	✓	
Обучение своих мобильных моделей			✓	✓	✓	
Бесплатное обслуживание	5000 запросов в месяц	1000 запросов в месяц	7500 запросов в месяц			

Это целый набор сервисов, которые уже запущены и готовы к использованию в нашем приложении. Поскольку числа и достоверные данные упрощают принятие решения, проанализируем эти сервисы по двум факторам: стоимости и точности.

## Стоимость

Поскольку деньги не растут на деревьях (пока), важно проанализировать экономическую сторону использования готовых API. Для примера возьмем вариант интенсивного использования этих услуг с частотой около 1 запроса в секунду на протяжении одного полного месяца (всего за месяц будет выполнено 2,6 миллиона запросов) и сравним предполагаемые затраты, как показано на рис. 8.7 (по состоянию на август 2019 года).

Для большинства разработчиков это экстремальный сценарий, но для крупных корпораций — вполне реальная нагрузка. Впоследствии мы сравним эти



**Рис. 8.7.** Сравнение стоимости разных облачных сервисов распознавания образов

стоимости со стоимостью запуска своего собственного сервиса в облаке, чтобы убедиться, что действительно получаем максимальную отдачу от вложенных средств в соответствии со сценарием.

Но для многих разработчиков плата может оказаться чисто символической, учитывая, что все провайдеры облачных услуг, которых мы здесь рассматриваем, предлагают бесплатную обработку до 5000 запросов в месяц (за исключением Google Vision, который бесплатно обслуживает всего 1000 запросов в месяц), а затем взимают примерно 1 доллар за 1000 запросов.

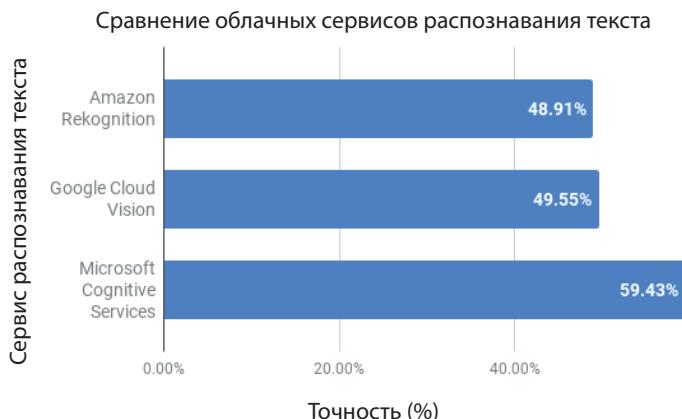
## Точность

В мире, где балом правят отделы маркетинга, заявляющие, что их компании — это лидеры рынка, трудно понять, кто же на самом деле лучший. Нужны универсальные показатели для сравнения провайдеров услуг, опирающиеся на некоторые внешние датасеты.

Чтобы показать, как создавать воспроизводимые бенчмарки, мы оценим качество извлечения текста с помощью датасета COCO-Text, который является подмножеством датасета MS COCO. Этот набор включает 63 686 изображений с повседневными предметами, содержащими текст, такими как баннеры, дорожные знаки, номера на автобусах, ценники в продуктовых магазинах, рубашки и т. д. Эта обыденность делает изображения относительно сложными для анализа. В качестве контрольной метрики используем процент ошибок на уровне слов

(Word Error Rate, WER). Чтобы не усложнять задачу, проигнорируем положение слов и сосредоточимся только на их присутствии. Совпадение засчитывается, если все слово распознано правильно.

В проверочный датасет COCO-Text отбираем все изображения с одним или несколькими образцами разборчивого текста (полнотекстовые последовательности без разрывов) и сравниваем образцы текста с длиной более одного символа. Затем отправляем эти изображения в различные облачные API распознавания образов. Результаты показаны на рис. 8.8.



**Рис. 8.8.** Значение WER для разных API извлечения текста по состоянию на август 2019 года

Учитывая сложность датасета, результаты выглядят впечатляюще. Большинство лучших инструментов извлечения текста, созданных в 2001–2010 годах, не могли преодолеть отметку 10 %. Это лишний раз показывает мощь глубокого обучения. Кроме того, нельзя не отметить улучшение качества некоторых из этих API, по сравнению с прошлым годом, на контрольном подмножестве изображений, отобранных вручную, что является еще одним преимуществом облачных API.

Как обычно, весь код реализации эксперимента размещен в репозитории GitHub (<https://github.com/PracticalDL/Practical-Deep-Learning-Book>).

Результаты нашего анализа в значительной степени обусловлены использованным датасетом и метриками. В зависимости от используемого датасета (который, в свою очередь, зависит от конкретного сценария) и минимальных метрик качества, результаты могут варьироваться. Кроме того, провайдеры услуг постоянно совершенствуют свои сервисы. И эти результаты не высечены в граните и со временем улучшаются. Вы можете сами воспроизвести эти результаты на любом датасете, воспользовавшись программными сценариями на GitHub.

## Предвзятость

В главе 1 мы видели, как в датасеты может закрасться предвзятость и какие последствия это может иметь. Облачные сервисы из этой главы не исключение. Джой Буоламвини (Joy Buolamwini), исследовательница из MIT Media Lab, обнаружила, что ни один из сервисов Microsoft, IBM и Megvii (также известный как Face++) не смог точно определить ее лицо и пол. Задавшись вопросом о том, какие уникальные черты лица делают ее незаметной для этих сервисов, Джой (вместе с Тимнит Гебру (Timnit Gebru)) собрала лица членов законодательных органов шести стран с высоким представительством женщин, создав датасет Pilot Parliaments Benchmark (PPB; см. рис. 8.9). Она выбрала три африканских и три европейских страны, чтобы проверить, как сервисы работают с разным цветом кожи. Скорее всего, вы понимаете, к чему мы клоним.

Она отметила, что API в целом показывают довольно высокую точность, от 85 % до 95 %. И только начав делить данные на разные категории, она заметила огромную разницу в точности. Сначала исследовательница заметила, что есть значительная разница между точностью классификации мужчин и женщин. Затем заметила еще большую разницу в точности в зависимости от цвета кожи. Наконец, в задачах классификации по полу и цвету кожи выявились болезненно резкие различия между группами людей, распознаваемыми хуже всего и лучше всего — темнокожими женщинами и белыми мужчинами соответственно. Например, API из IBM показал точность распознавания темнокожих женщин всего 65,3 %, в то время как тот же API распознавал белых мужчин с точностью 99,7 %. Колossalная разница в 34,4 %! Учитывая, что многие из этих API используются правоохранительными органами, последствия предвзятости могут стоить жизни.



**Рис. 8.9.** Усредненные лица разного пола и цвета кожи по данным Pilot Parliaments Benchmark (PPB)

Вот некоторые выводы, полученные в этом исследовании:

- Качество алгоритма зависит от данных, на которых он обучен. Это говорит о важности разнообразия в наборе обучающих данных.

- Часто общие цифры не отражают истинную картину. Предвзятость в датасете очевидна только при его разделении на разные подгруппы.
- Предвзятость не является особенностью какой-либо конкретной компании; скорее это общегородское явление.
- Результаты не высечены в граните и зависят только от времени проведения эксперимента. Об этом говорит резкое изменение цифр между исследованиями, проводившимися в 2017 (рис. 8.10) и в 2018 (рис. 8.11) годах. Все компании серьезно относятся к устранению предвзятости в своих датасетах.
- Исследователи, проверяющие коммерческие компании с помощью общедоступных тестов, способствуют совершенствованию отрасли в целом (даже если усовершенствования вызваны опасениями представить компанию в плохом свете).

Есть ли предвзятость в API, генерирующих теги для изображений? В Facebook AI Research изучили вопрос «Однокаково ли хорошо распознаются все объекты?»

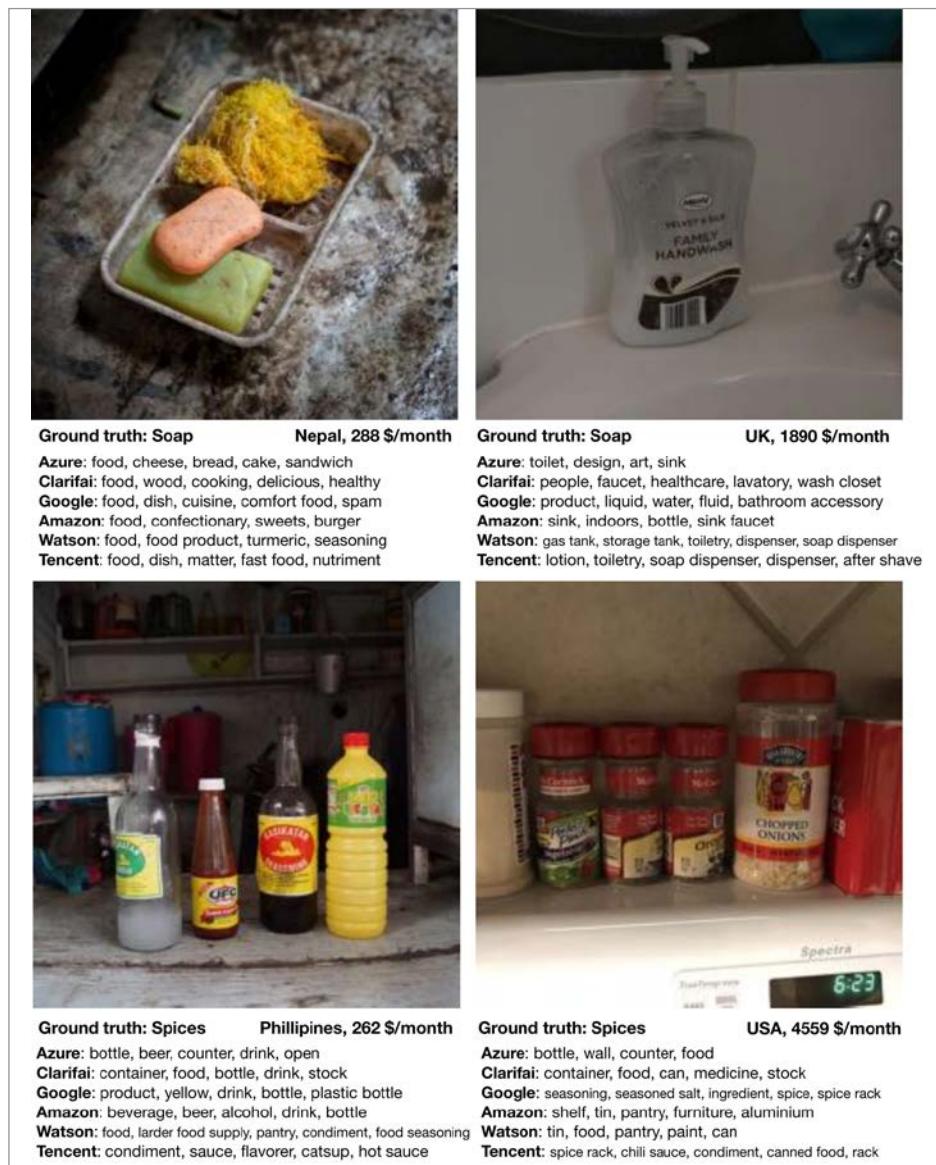


**Рис. 8.10.** Сравнение качества распознавания лиц в датасете PPB разными API в апреле и мае 2017 года



**Рис. 8.11.** Сравнение качества распознавания лиц в датасете PPB разными API в августе 2018 года (проводила Иниолува Деборха Раджи (Inioluwa Deborah Raji) с коллегами)

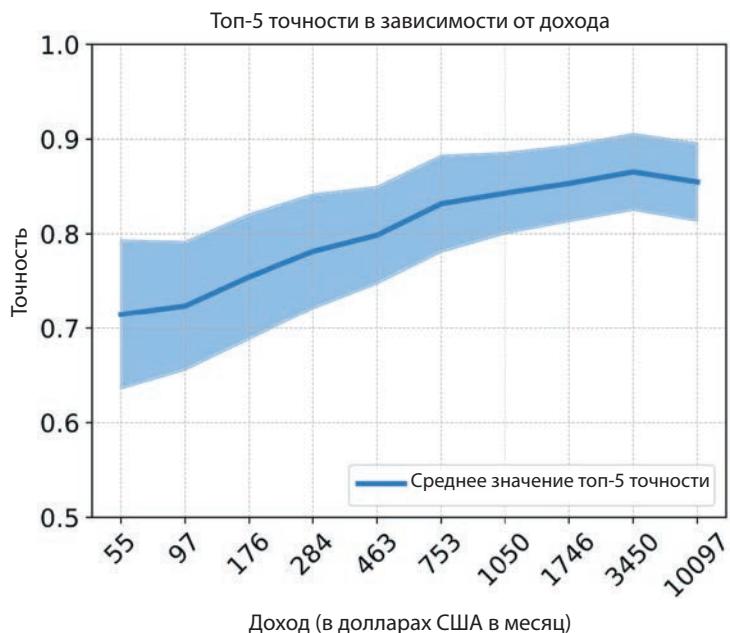
в одноименной статье Терренс ДеВрис и др. (Terrance DeVries et al.). Исследователи протестирували несколько облачных API в феврале 2019 года на датасете Dollar Street — разнообразной коллекции изображений предметов домашнего обихода из 264 разных домов в 50 странах (рис. 8.12).



**Рис. 8.12.** Качество генерирования тегов для изображений разными API на географически разнообразном датасете Dollar Street

Вот некоторые выводы, полученные в этом исследовании.

- Точность классификации объектов значительно ниже для изображений из регионов с более низким уровнем дохода, как показано на рис. 8.13.
- В ImageNet, COCO и OpenImages ощущается острая нехватка изображений из Африки, Индии, Китая и Юго-Восточной Азии, что приводит к снижению качества классификации изображений из незападного мира.
- В большинстве датасетов изображения подписыны на английском языке и нет изображений тех же объектов с подписями на других языках.



**Рис. 8.13.** Зависимость средней точности (и стандартного отклонения) шести облачных API от дохода семей в странах, откуда были взяты изображения

В зависимости от сценария использования облачных API следует создавать собственные бенчмарки и периодически тестировать их, чтобы оценить, подходят ли эти API для конкретного случая использования.

## Подготовка и использование облачных API

Для использования облачных сервисов требуется минимум кода. Если говорить в общих чертах, то нужно получить ключ API, загрузить изображение, обозначить

намерение, отправить запрос POST в правильной кодировке (например, base64 для изображения) и получить результаты. Большинство провайдеров поставляют наборы средств для разработки ПО (Software Development Kit, SDK) и образцы кода, в которых показаны правильные способы обращения к их сервисам. Дополнительно они предоставляют пакеты Python, которые устанавливаются с помощью pip, чтобы еще больше упростить доступ к сервисам. Если вы используете Amazon Rekognition, настоятельно рекомендуем использовать соответствующий пакет pip.

А теперь снова воспользуемся нашим захватывающим изображением и протестируем эти сервисы.

Первым протестируем Microsoft Cognitive Services.

Получите ключ API и замените его в следующем коде (первые 5000 запросов предоставляются бесплатно — этого более чем достаточно для наших экспериментов):

```
cognitive_services_tagimage('DogAndBaby.jpg')
```

Результаты:

```
{  
    "description": {  
        "tags": ["person", "indoor", "sitting", "food", "table", "little",  
        "small", "dog", "child", "looking", "eating", "baby", "young", "front",  
        "feeding", "holding", "playing", "plate", "boy", "girl", "cake", "bowl",  
        "woman", "kitchen", "standing", "birthday", "man", "pizza"],  
        "captions": [  
            {"text": "a little girl sitting at a table with a dog",  
            "confidence": 0.84265453815486435  
        ]  
    },  
    "requestId": "1a32c16f-fda2-4adf-99b3-9c4bf9e11a60",  
    "metadata": {  
        "height": 427,  
        "width": 640,  
        "format": "Jpeg"  
    }  
}
```



Теги в датасете ImageNet в основном состоят из существительных, но многие из этих сервисов возвращают описания, включающие глаголы, а также прилагательные. Эти глаголы и прилагательные могут оказаться излишними для нашего приложения, и возникнет желание отфильтровать их. Для этого можно воспользоваться моделью WordNet из Принстона и с ее помощью проверить лингвистический тип каждого слова в подписи. Эта модель доступна в Python в составе набора инструментов обработки естественного языка Natural Language Processing Toolkit (NLTK). Аналогично можно отфильтровать такие слова, как «внутри» и «снаружи» (часто добавляемые службами Clarifai и Cognitive Services).

Сгенерированное описание «A little girl sitting at a table with a dog»<sup>1</sup> очень близко к истине. Есть и другие способы получения более подробных результатов, включая вероятность для каждого тега.

А теперь посмотрим, какую подпись генерирует Google Vision API для того же изображения. Получите ключ API на веб-сайте службы и используйте его в следующем коде (и радуйтесь, потому что обработка первых 1000 запросов предоставляется бесплатно):

```
google_cloud_tagimage('DogAndBaby.jpg')
```

Результаты:

```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/0bt9lr",
          "description": "dog",
          "score": 0.951077,
          "topicality": 0.951077
        },
        {
          "mid": "/m/06z04",
          "description": "skin",
          "score": 0.9230451,
          "topicality": 0.9230451
        },
        {
          "mid": "/m/01z5f",
          "description": "dog like mammal",
          "score": 0.88359463,
          "topicality": 0.88359463
        },
        {
          "mid": "/m/01f5gx",
          "description": "eating",
          "score": 0.7258142,
          "topicality": 0.7258142
        }
      ]
    }
  ]
}
```

Что может быть проще? Для использования этих API не требуется докторская степень, и можно получить самые точные результаты всего за 15 минут!

---

<sup>1</sup> Маленькая девочка сидит за столом рядом с собакой. — Примеч. пер.



Сервисы возвращают теги и подписи к изображениям, сопровождая их соответствующими вероятностями, поэтому разработчик должен определить порог. Обычно хорошими пороговыми значениями для тегов и подписей являются 60 % и 40 % соответственно.

Эти вероятности важно сообщить конечному пользователю. Например, если достоверность результата превышает 80 %, мы могли бы предварить теги префиксом «Это изображение *содержит...*». Если достоверность меньше 80 %, можно использовать префикс «Это изображение *может содержать...*», чтобы подчеркнуть меньшую уверенность в результате.

## Обучение собственного классификатора

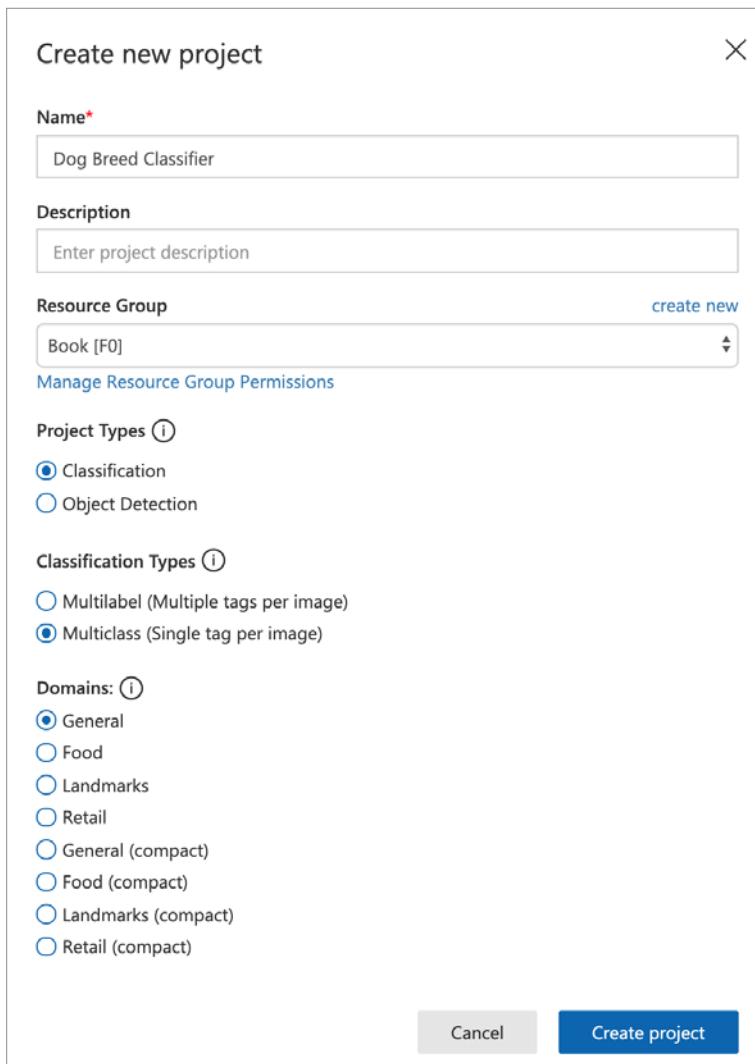
В некоторых случаях возможностей облачных сервисов может оказаться недостаточно для нашего сценария использования. Например, представьте, что один сервис отметил нашу фотографию тегом «собака», а нас интересует еще и порода собаки. Конечно, можно последовать указаниям из главы 3 и обучить свой классификатор в Keras. Но лучше, если бы не понадобилось писать код. И это возможно!

Некоторые из облачных провайдеров позволяют обучить свой классификатор простым перетаскиванием изображений мышью. Их удобные UI используют под капотом прием переноса обучения. Такую возможность поддерживают Cognitive Services Custom Vision, Google AutoML, Clarifai и IBM Watson. Кроме того, некоторые из них позволяют создавать собственные детекторы, которые определяют местоположение объектов и рисуют ограничительные рамки. Процесс обучения в этих сервисах выглядит примерно одинаково:

1. Выгрузить изображения.
2. Подписать их.
3. Обучить модель.
4. Оценить модель.
5. Опубликовать модель для использования через REST API.
6. Бонус: скачать мобильную модель для использования на смартфонах и в краевых (edge) устройствах.

Рассмотрим пошаговый пример использования Microsoft Custom Vision (<https://www.customvision.ai/>).

1. *Создайте проект* (рис. 8.14): выберите предметную область (**Domains**), которая точнее соответствует вашему сценарию использования. В большинстве случаев оптимальным вариантом будет **General** (Общая). Для специализированных случаев выберите более узкую предметную область.

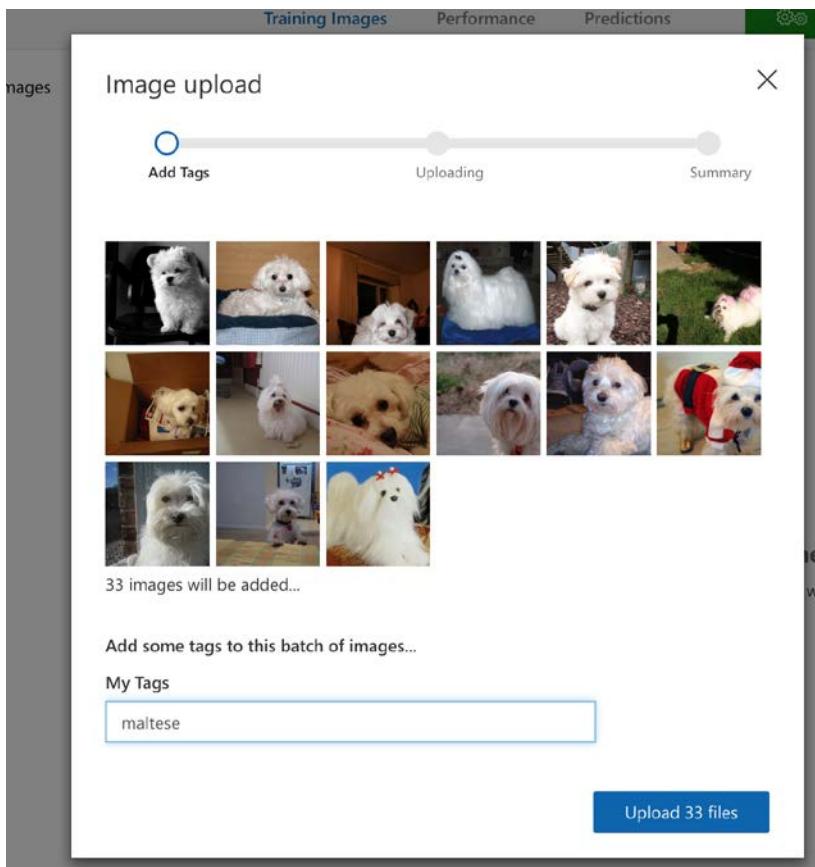


**Рис. 8.14.** Создание нового проекта в Custom Vision

Например, если у вас сайт интернет-магазина с фотографиями продуктов на чистом белом фоне, то выберите предметную область **Retail** (Розничная торговля). Если вы намерены использовать модель на мобильном телефоне, то следует выбрать «компактную» (compact) версию. Она имеет меньший размер и немного проигрывает в точности.

2. *Выгрузите изображения* (рис. 8.15): выгрузите изображения для каждой категории и подпишите их. Важно выгрузить для каждой категории не ме-

нее 30 фотографий. Для нашего теста мы выгрузили более 30 изображений мальтийских болонок и подписали их.



**Рис. 8.15.** Выгрузка изображений в CustomVision.ai

3. *Обучите модель* (рис. 8.16): щелкните на кнопке Train (Обучить) и примерно через три минуты ваш новый классификатор будет готов.



**Рис. 8.16.** Кнопка Train (Обучить) в правом верхнем углу на странице CustomVision.ai

4. *Проанализируйте качество модели*: проверьте точность и полноту модели. По умолчанию система устанавливает доверительную вероятность 90 %

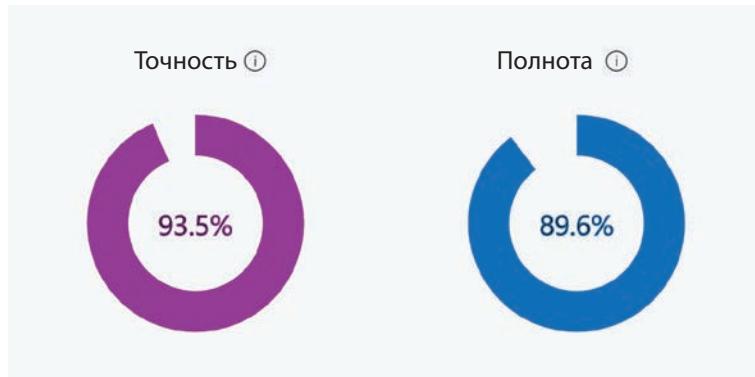
и сообщает метрики точности и полноты для этого значения. Для большей точности увеличьте доверительную вероятность, но при этом может уменьшиться полнота. На рис. 8.17 показан пример вывода.

5. *Модель готова к использованию:* теперь у вас есть готовая к использованию конечная точка API, к которой можно обратиться из любого приложения.

Чтобы показать, как объем обучающих данных влияет на качество модели, обучим классификатор пород собак. Для этого воспользуемся датасетом Stanford Dogs — коллекцией, включающей фотографии собак более чем 100 пород. Для простоты мы случайным образом выбрали 10 пород, для каждой из которых доступно более 200 изображений. Для случая с 10 классами случайный классификатор будет показывать 10%-ную точность идентификации изображений. Мы должны без труда преодолеть эту планку. В табл. 8.2 показано влияние объема набора обучающих данных на точность и полноту модели.

**Таблица 8.2.** Влияние количества обучающих изображений на точность и полноту

	30 обучающих изображений на класс	200 обучающих изображений на класс
Точность	91,2%	93,5%
Полнота	85,3%	89,6%



**Рис. 8.17.** Относительная точность и полнота испытуемой модели после обучения на наборе с 200 изображениями на класс

Поскольку мы не выгружали контрольный датасет, приведенные здесь показатели точности и полноты относятся к полному датасету и получены с использованием общепринятой методики перекрестной проверки по  $k$ -блокам. Согласно этой методике, данные случайным образом делятся на  $k$  частей, затем

$(k - 1)$  частей используются для обучения, а оставшаяся часть — для контроля. Этот процесс повторяется несколько раз и всегда со случайным подмножеством изображений. Здесь представлены усредненные результаты.

Самое интересное, что даже с тридцатью изображениями на класс точность классификатора превысила 90 %, как показано на рис. 8.18. И, что удивительно, обучение длилось чуть меньше 30 секунд.

Мало того, можно копнуть глубже и получить характеристики качества для каждого класса. Классы с высокой точностью могут заметно отличаться друг от друга, тогда как классы с низкой точностью могут быть похожими друг на друга.

### Характеристики по тегам

Тег	Точность	Полнота
afghan_hound	87.5%	92.0%
airedale	96.0%	92.5%
basenji	97.4%	93.0%
bernese_mountain_dog	91.3%	91.0%
entlebucher	97.2%	87.5%
great_pyrenees	87.7%	85.0%
irish_wolfhound	87.8%	85.0%
leonberg	98.9%	87.0%
maltese_dogs	96.4%	91.5%
pomeranian	97.4%	91.5%

**Рис. 8.18.** Некоторые из возможных тегов, возвращаемых моделью

Этот простой и удобный способ обучения своего классификатора не лишен недостатков, о которых мы поговорим в следующем разделе. А сейчас рассмотрим стратегии, которые помогут получить дополнительные выгоды от этого полезного инструмента.

## Основные причины плохой работы классификатора

Есть несколько причин, по которым качество работы собственного классификатора может оказаться неудовлетворительным. Вот некоторые из них:

### *Недостаточно данных*

Если обнаружится, что собственный классификатор имеет недостаточную точность, можно попробовать обучить модель на большем количестве данных. Тридцать изображений на класс — это лишь необходимый минимум. Для высокого качества желательно иметь больше изображений. Обычно рекомендуется 200 изображений на класс.

### *Нерепрезентативные обучающие данные*

Часто изображения из интернета слишком чистые, получены при студийном освещении с чистым фоном и с объектом съемки в центре кадра. Изображения, которые наше приложение будет видеть ежедневно, могут быть не такими. Поэтому, чтобы добиться максимального качества, очень важно обучить свой классификатор на реальных изображениях.

### *Неправильно выбранная предметная область*

Под капотом Custom Vision использует прием переноса обучения. По этой причине при создании проекта важно правильно выбрать предметную область. Например, если приложение должно классифицировать рентгеновские снимки, то перенос обучения с использованием модели, обученной на наборе ImageNet, может не дать достаточной точности. В таких случаях лучше вручную обучить классификатор с помощью Keras, как показано в главе 3 (хотя это займет больше трех минут).

### *Использование классификатора для регрессии*

В машинном обучении есть две большие категории задач: классификация и регрессия. Классификация — это предсказание принадлежности входных данных к одному или нескольким классам. Регрессия же, напротив, — прогнозирование числового значения по входным данным, например цен на жилье. Custom Vision — это, прежде всего, система классификации. Использование ее для подсчета объектов путем указания их количества в подписи — неправильный подход, который обязательно приведет к неудовлетворительным результатам.

Подсчет объектов — это задача регрессии. Его можно реализовать, выделив каждый экземпляр объекта на изображении (методом распознавания) и подсчитав их. Другой пример задачи регрессии — прогнозирование возраста человека на основе его фотографии в профиле. Мы рассмотрим обе эти задачи в следующих главах.

### Классы слишком похожи

Если классы слишком похожи друг на друга и отличаются лишь мелкими деталями, модель может плохо различать их. Например, пяти- и двадцатидолларовые банкноты очень похожи в общих чертах и различаются только мелкими деталями. Другой пример: вы легко отличите чихуахуа от сибирского хаски, но вот отличить аляскинского маламута от сибирского хаски уже будет не так просто. Полностью заново обученная сверточная сеть, как показано в главе 3, должна лучше справиться с этой задачей, чем система на основе Custom Vision.



Отличительная особенность Custom Vision: если модель не уверена в выборе категории для какого-либо изображения, полученного через конечную точку API, веб-интерфейс покажет это изображение для его разметки вручную. Мы можем периодически просматривать и вручную размечать новые изображения, постоянно улучшая качество модели. Такие изображения, как правило, больше всего способствуют улучшению классификатора, потому что, во-первых, представляют реальное использование, а во-вторых, что особенно важно, они оказывают большее влияние на модель, чем изображения, с классификацией которых модель справляется без труда. Это называется обучением с частичным привлечением учителя.

В этом разделе мы обсудили несколько способов повышения точности нашей модели. Но в реальном мире точность — не единственный показатель, который учитывается пользователем. Не менее важно быстро ответить на запрос. В следующем разделе рассмотрим несколько способов повышения скорости без ущерба для качества.

## Сравнение качества работы собственных классификаторов в разных API

Как вы могли заметить на протяжении всей книги, мы довольно категорично относимся к качеству результатов. Если уж мы собираемся потратить хорошие деньги на сервис, то хотелось бы получить максимальную отдачу от вложенных средств. Пришло время проверить рекламные обещания.

Наши собственные облачные классификаторы хорошо справляются с широким кругом задач классификации. Но чтобы по-настоящему проверить их возможности, нужно что-то посложнее. Давайте возьмем самый сложный датасет с изображениями собак — Stanford Dogs, обучим классификаторы и сравним полученные результаты.

Использование всего датасета может упростить задачу для этих классификаторов (в конце концов, в ImageNet довольно много изображений собак разных

пород), поэтому поднимем планку. С этой целью мы обучили собственный классификатор Keras на всем датасете и создали небольшой датасет с 34 хуже всего распознаваемыми классами (каждый из которых содержит не менее 140 изображений). Причина низкой точности их распознавания состоит в том, что эти породы собак очень похожи друг на друга. Чтобы повысить качество, классификаторы должны научиться распознавать мелкие признаки. Мы случайным образом выбрали по 100 изображений из каждого класса для обучающей выборки и по 40 изображений для контрольной выборки. Чтобы избежать дисбаланса классов, который может повлиять на прогнозы, мы отобрали из каждого класса одинаковое количество обучающих и контрольных изображений.

Наконец, мы выбрали пороговое значение 0,5, так как, на наш взгляд, оно обеспечивает хороший баланс между точностью и полнотой для всех служб. При высоком пороге достоверности, например 0,99, классификатор может быть очень точным, но при этом будет распознавать лишь небольшую часть изображений; иначе говоря, будет иметь низкий уровень полноты. Низкий порог, такой как 0,01, напротив, приведет к тому, что достоверными будут признаны предсказания почти для всех изображений, однако многие из этих результатов в действительности будут ошибочными.

Кроме того, вместо точности и полноты мы использовали *оценку F1* (известную также как *F-мера*) — гибридную метрику, сочетающую оба этих значения:

$$\text{Оценка F1} = \frac{2 \times \text{точность} \times \text{полнота}}{\text{точность} + \text{полнота}}.$$

Дополнительно мы измерили время, затраченное на обучение, как показано на рис. 8.19. Кроме облачных сервисов мы также обучили свою модель с помощью



**Рис. 8.19.** Оценки F1 обучения облачных классификаторов по состоянию на август 2019 года (чем выше, тем лучше)

инструмента Apple Create ML на MacBook Pro с аугментацией данных и без нее (поворот, обрезка и отражение).

Google и Microsoft позволяют настроить продолжительность обучения. Google Auto ML – в диапазоне от 1 до 24 часов. Microsoft предлагает бесплатный вариант «Fast Training» (быстрое обучение) и платный вариант «Advanced Training» (продвинутое обучение, аналогичное предложению Google), при выборе которого можно отрегулировать продолжительность обучения от 1 до 24 часов.

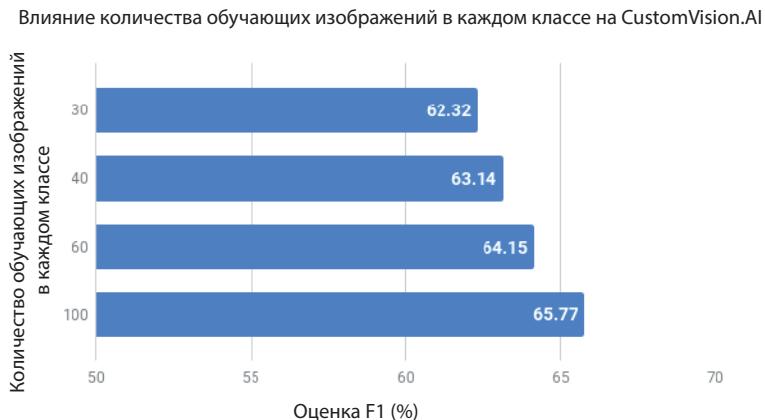
Вот несколько интересных выводов из этого эксперимента:

- Clarifai и Microsoft позволяют практически мгновенно обучить классификатор на 3400 обучающих изображениях.
- Платный вариант «Advanced Training» (продвинутое обучение) в Microsoft показал чуть более высокий результат (примерно на один процентный пункт), чем вариант «Fast Training» (быстрое обучение), за счет более продолжительного обучения в течение часа. Поскольку «быстрое обучение» заняло менее 15 секунд, делаем вывод, что база классификатора уже хорошо подготовлена к извлечению мелких деталей.
- Как это ни удивительно, но Apple Create ML ухудшила результат после аугментации данных, несмотря на то что обучение шло на два часа дольше и большая часть из этого времени была потрачена на аугментацию. Все операции выполнялись на топовом Mac Book Pro и со 100%-ной загрузкой GPU, которая контролировалась с помощью Activity Monitor.

Кроме того, для проверки границ возможностей мы меняли объем обучающих данных, передаваемых сервису (рис. 8.20). Поскольку сервис Microsoft тратил на обучение менее 15 секунд, экспериментировать с ним было легко (и недорого!). Мы попробовали несколько вариантов обучения на разном количестве изображений, от 30 до 100 на класс, сохранив те же 40 изображений на класс в контрольной выборке.

Несмотря на то что Microsoft рекомендует использовать не менее 50 изображений для каждого класса, превышение этого лимита не оказывает существенного влияния на качество классификации. Тот факт, что оценка F1 изменилась не так сильно, как можно было бы ожидать, показывает ценность переноса обучения (позволяющего использовать меньше данных для создания классификаторов) и наличия хорошей основы, способной обучиться более тонкой классификации.

Отметим еще раз, что этот эксперимент был намеренно затруднен для стресс-тестирования классификаторов. В среднем результаты были бы намного лучше, если бы обучение выполнялось на полном датасете Stanford Dogs.



**Рис. 8.20.** Влияние количества обучающих изображений в каждом классе на оценку F1 (чем выше, тем лучше)

## Настройка производительности облачных API

Современные смартфоны делают фотографии разрешением до  $4000 \times 3000$  пикселей и размером более 4 Мбайт. В зависимости от качества сетевого подключения выгрузка каждого такого изображения в сервис занимает до нескольких секунд, и каждая дополнительная секунда может расстраивать наших пользователей все больше. Можно ли как-то ускорить процесс?

Есть два способа уменьшить размер изображения.

### *Изменить разрешение*

Большинство сверточных сетей принимает на входе изображения размером  $224 \times 224$  или  $448 \times 448$  пикселей. То есть сверточные сети не нуждаются в столь высоком разрешении, которое дают мобильные телефоны. Поэтому вместо отправки больших изображений на сервер и уменьшения их там было бы разумнее уменьшить размеры изображений до передачи в Сеть.

### *Сжатие*

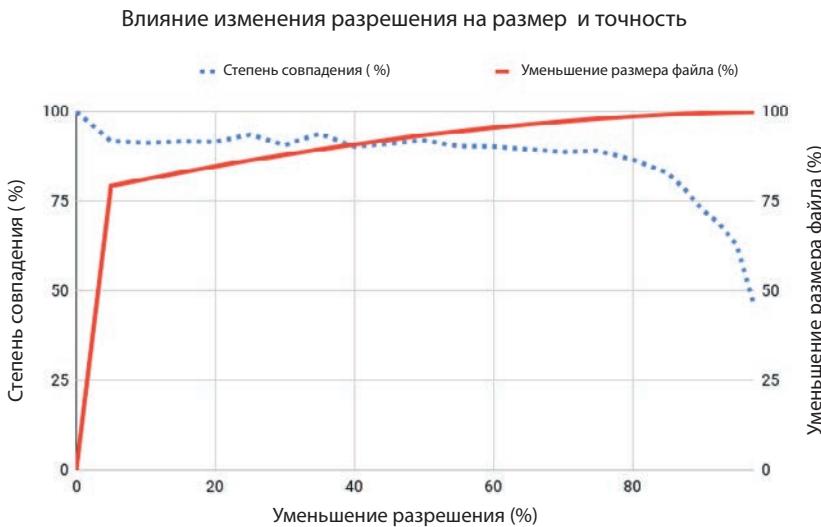
Большинство библиотек для работы с изображениями поддерживают сжатие с потерями при сохранении файла. Даже небольшая степень сжатия может значительно уменьшить размер изображения, оставив минимальное влияние на его качество. Сжатие действительно вносит шум, но сверточные сети обычно достаточно устойчивы, чтобы справиться с ним.

## Влияние изменения разрешения на API разметки изображений

Мы провели эксперимент, взяв более сотни различных фотографий, снятых на iPhone с разрешением по умолчанию ( $4032 \times 3024$ ), и отправили их в Google Cloud Vision API, чтобы получить метки для каждого из них. Затем уменьшили размеры исходных изображений с шагом 5 % (5 %, 10 %, 15 %, ..., 95 %), собрали полученные результаты и для каждого изображения вычислили степень совпадения меток по следующей формуле:

$$\text{Степень совпадения (\%)} = \frac{\text{Количество меток эталонного изображения, присвоенных также испытуемым изображениям}}{\text{Количество меток эталонного изображения}} \times 100.$$

На рис. 8.21 показаны результаты этого эксперимента. Сплошная линия показывает, как уменьшался размер файла, а пунктирная — степень совпадения. Наш главный вывод из эксперимента: уменьшение разрешения на 60 % привело к уменьшению размера файла на 95 %, при этом точность по сравнению с исходными изображениями пострадала незначительно.

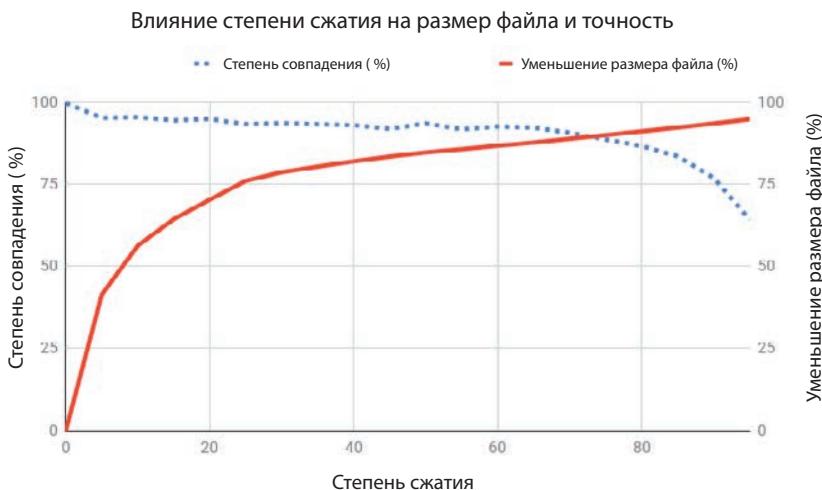


**Рис. 8.21.** Влияние изменения разрешения изображения на степень совпадения и размер файла

## Влияние сжатия на API разметки изображений

Мы повторили тот же эксперимент, но вместо разрешения постепенно изменяли коэффициент сжатия изображений. На рис. 8.22 сплошная линия показывает уменьшение размера файла, а пунктирная линия — степень совпадения. Главный

вывод: степень сжатия 60 % (или сохранение 40 % качества) привела к уменьшению размера файла на 85 %, при этом точность по сравнению с исходными изображениями пострадала незначительно.



**Рис. 8.22.** Влияние изменения степени сжатия изображения на степень совпадения и размер файла

## Влияние сжатия на API оптического распознавания символов

Мы сфотографировали документ, содержащий более 300 слов, на iPhone с разрешением по умолчанию ( $4032 \times 3024$ ) и отправили его в Microsoft Cognitive Services API, чтобы проверить, насколько хорошо распознается текст. Затем несколько раз сжали изображение с шагом 5 %, отправили сжатые изображения в тот же API и сравнили результаты с эталоном, вычислив процент ошибок на уровне слов (Word Error Rate, WER). Мы заметили, что даже сжатие с коэффициентом 95 % (то есть с сохранением 5 % качества исходного изображения) не повлияло на качество результатов.

## Влияние изменения разрешения на API оптического распознавания символов

Мы повторили предыдущий эксперимент, но на этот раз вместо степени сжатия меняли разрешение каждого изображения. В некоторый момент значение WER подскочило с нуля почти до 100 % и почти все слова были классифицированы

неправильно. Повторное тестирование с другим документом, в котором все слова напечатаны шрифтом разного размера, показало, что все слова с определенным размером шрифта классифицируются с ошибками. Для эффективного распознавания текста необходимо, чтобы текст был больше минимальной высоты (как было подмечено на практике — больше 20 пикселей). То есть чем выше разрешение, тем выше точность.

### Что мы узнали?

- В случае распознавания символов сжатие изображений текстовых документов меньше влияет на точность, чем уменьшение разрешения.
- В случае разметки изображений умеренное уменьшение разрешения (скажем, на 50 %) и умеренная степень сжатия (скажем, 30 %) приведут к значительному уменьшению размера файла (и ускорению их обработки) без ущерба для качества результатов.
- В зависимости от сферы применения ваше приложение может получать уже сжатые изображения или изображения с уменьшенным разрешением. Любое дополнительное изменение изображений может влиять на результаты этих API, поэтому постарайтесь их минимизировать.



Получив изображения, облачные API изменяют их размеры в соответствии со своими требованиями. Для нас это означает, что есть два уровня изменения размера: сначала мы уменьшаем размеры изображений перед отправкой в облачный API, а затем он дополнительно изменяет размеры. Уменьшение размеров изображений приводит к искажениям, которые более очевидны при более низком разрешении. Мы можем минимизировать искажения, уменьшая размеры изображений до значений, которые в несколько раз больше конечного. Например, изменение разрешения  $3024 \times 3024$  (исходное)  $\rightarrow 302 \times 302$  (перед отправкой в облако)  $\rightarrow 224 \times 224$  (внутри API) приведет к гораздо большим искажениям в конечном изображении, чем последовательность  $3024 \times 3024 \rightarrow 896 \times 896 \rightarrow 224 \times 224$ . То есть перед отправкой изображений желательно попробовать найти подходящий промежуточный размер. Кроме того, настройка дополнительных параметров интерполяции, таких как BICUBIC и LANCZOS, поможет получить более точное представление исходного изображения в уменьшенной версии.

## Примеры

Говорят, что самые прекрасные творения рождаются в муках. Мы же хотим доказать обратное. В следующем разделе мы покажем, как некоторые титаны технологической индустрии используют облачные API искусственного интеллекта в своей работе.

## New York Times

Возможно, вы решили, что сценарий, представленный в начале главы, был взят из мульти сериала, но в действительности нечто похожее было в *New York Times* (NYT). За более чем 160-летнюю историю в редакции NYT сформировалась целая сокровищница фотографий. Многие из них хранятся в подвале здания редакции, метко названном «моргом», на три этажа ниже уровня земли. Это бесценная коллекция. В 2015 году из-за протечки водопровода часть подвала оказалась затоплена и пострадали некоторые из архивных документов. К счастью, ущерб был минимальным. Но это побудило NYT задуматься о возможности их оцифровки, чтобы защититься от новых катастроф.

Фотографии были отсканированы с высоким качеством и сохранены. Но на самих фотографиях не было идентифицирующей информации. На многих из них на обратной стороне были пометки. В NYT использовали Google Vision API для распознавания этого текста и добавления полученной информации к соответствующим изображениям. Кроме того, в пайплайн оцифровки были включены многие другие возможности извлечения метаданных из фотографий, в том числе распознавание достопримечательностей, распознавание знаменитостей и т. д. Эти вновь добавленные теги используются системой поиска, благодаря чему любой внутри компании и за ее пределами может изучать галерею и делать поиск по ключевым словам, датам и другим параметрам, не спускаясь в «морг» на третьем подземном этаже.

## Uber

Uber использует Microsoft Cognitive Services для идентификации любого из семи с лишним миллионов водителей за пару миллисекунд. Вообразите масштабы использования новой функции Uber под названием «Проверка личности в реальном времени». Эта функция проверяет, действительно ли водитель зарегистрирован, предлагая ему сделать селфи через случайные интервалы времени или перед каждой передачей заказа. Снимок сравнивается с фотографией водителя в профиле, и только если модели лиц совпадают, водителю разрешается использовать систему Uber. Эта функция помогает повысить ответственность, обеспечивает безопасность пассажиров и охраняет учетную запись водителя от взлома. Эта функция способна обнаруживать на фотографии вновь появившиеся детали, включая шляпу, бороду, солнцезащитные очки и многое другое, и предлагать водителю сделать снимок без шляпы или очков.

## Giphy

Еще в 1976 году, когда доктор Ричард Докинз придумал термин «мем», он не подозревал, что через четыре десятилетия термин заживет своей жизнью. В наше

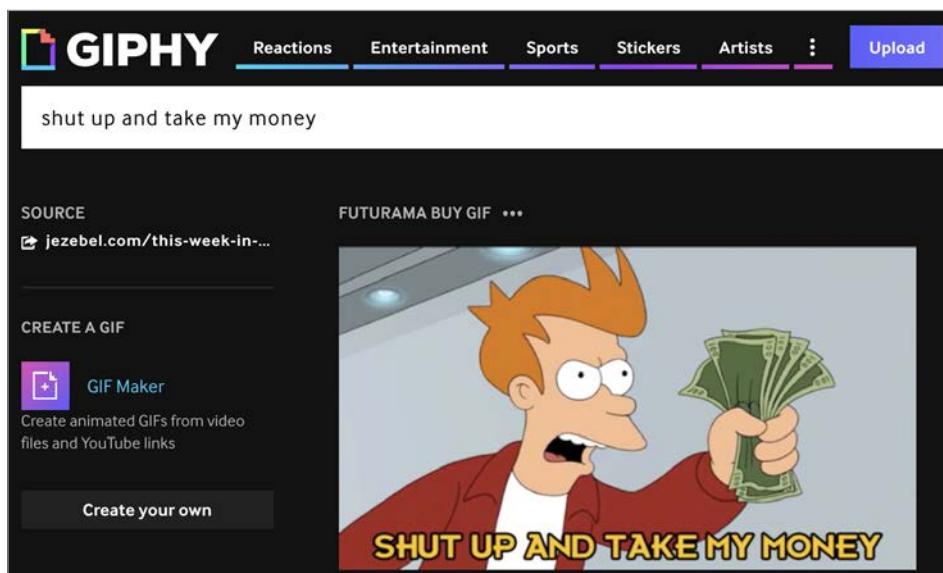


**Рис. 8.23.** Приложение Uber Drivers предлагает водителю сделать селфи для проверки его личности (изображение взято по адресу <https://oreil.ly/lw1Ho>)

время большинство чат-приложений предлагают вместо простого текстового ответа отправить анимированное изображение в формате GIF, соответствующее контексту. Tenor, мессенджер Facebook, Swype и Swiftkey поддерживают возможность поиска конкретных мемов и изображений GIF. Большинство из них выполняет поиск с помощью Giphy (рис. 8.24), крупнейшей в мире поисковой системы анимированных мемов, обычно в формате GIF.

Изображения GIF часто содержат наложенный текст (например, диалоги), и иногда бывает нужно найти GIF с конкретным диалогом из фильма или телешоу. Например, изображение на рис. 8.24 — это эпизод из мультсериала «Futurama» 2010 года, часто используемый для выражения восхищения продуктом или идеей. Доступность текста упрощает поиск GIF-файлов. Giphy использует службу Google Vision API для распознавания текста и объектов на изображении, что помогает находить идеальные изображения GIF.

Очевидно, что разметка GIF-файлов — сложная задача, потому что человек должен просмотреть миллионы таких анимаций и вручную аннотировать их кадр за кадром. В 2017 году в Giphy придумали два решения для автоматизации этого процесса. Первое заключается в обнаружении текста внутри изображения. Второе — в создании тегов с названиями объектов, присутствующих на изображении, которые дополняют метаданные, используемые для поиска. Хранение и поиск этих метаданных реализованы с помощью Elasticsearch — масштабируемой поисковой системы. Чтобы проверить, действительно ли GIF содержит текст, компания передавала первый кадр из каждого файла GIF



**Рис. 8.24.** Giphy извлекает текст из анимированных изображений и использует его как метаданные для поиска

сервису оптического распознавания символов (Optical Character Recognition, OCR) в Google Vision API. Если сервис возвращал утвердительный ответ, Giphy отправляла следующие кадры, получала тексты, обнаруженные с помощью OCR, и выявляла различия между ними; например, был ли текст статическим (оставался неизменным на протяжении всей анимации) или динамическим (разный текст в разных кадрах). Для создания меток классов, соответствующих объектам на изображении, инженеры имели на выбор возможности определения меток и веб-сущностей, доступные в Google Vision API. Определение меток, как следует из названия, возвращает фактическое имя класса объекта. Определение веб-сущностей возвращает идентификатор объекта (ссылку в графе знаний Google Knowledge Graph) — уникальный URL, ссылающийся на идентичное или похожее изображение, которое можно увидеть где-либо в интернете. Использование этих дополнительных аннотаций дало новой системе увеличение отношения числа переходов к числу показов (Click Through Rate, CTR) на 32 %. Наибольшую пользу из этого извлекли поисковые запросы со средним и длинным хвостом (то есть не очень частые). Для них стал возвращаться более богатый релевантный контент, потому что извлеченные метаданные помогали отыскать ранее неаннотированные файлы GIF, которые в противном случае остались бы незаметными для поисковой системы. Кроме того, эти метаданные и поведение пользователей при переходе по ссылкам предоставляют данные для создания определения подобия и исключения избыточных результатов.

## OmniEarth

OmniEarth — компания из Вирджинии, специализирующаяся на сборе, анализе и объединении спутниковых и аэрофотоснимков с другими датасетами для мониторинга использования водных ресурсов по всей стране с возможностью масштабирования и с высокой скоростью. Компания способна просканировать 144 миллиона земельных участков по всей территории США за считанные часы. Для классификации изображений земельных участков и получения ценной информации, например степени озеленения, она использует IBM Watson Visual Recognition API. Сочетая результаты классификации с другими данными, такими как температура и количество осадков, OmniEarth может предсказать, сколько воды было использовано для орошения полей.

Она способна по изображениям определять наличие бассейнов, подсчитывать количество деревьев и оценивать площади орошаемых угодий, и на основе этой информации прогнозировать объем водопользования. Компания даже может предсказать, где вода тратится впустую из-за чрезмерного полива или утечек. OmniEarth помогла правительству штата Калифорния понять структуру и объемы потребления воды, проанализировав более 150 000 земельных участков, а затем разработать эффективную стратегию по сокращению потерь воды.

## Photobucket

Photobucket — это популярное онлайн-сообщество, размещающее изображения и видео, куда ежедневно выгружается более двух миллионов изображений. Используя модели Clarifai NSFW, Photobucket автоматически выявляет нежелательный или оскорбительный контент, созданный пользователями, и отправляет его на дальнейшее рассмотрение группе модераторов. Раньше команда модераторов могла проверить только около 1% входящего контента, причем недопустимыми признавались до 70 % изображений. Автоматизация этой задачи позволила Photobucket выявить в 700 раз больше нежелательного контента, очистить от него сайт и повысить репутацию у пользователей. Она также помогла вскрыть две учетные записи, распространяющие детскую порнографию, о чем было сообщено в ФБР.

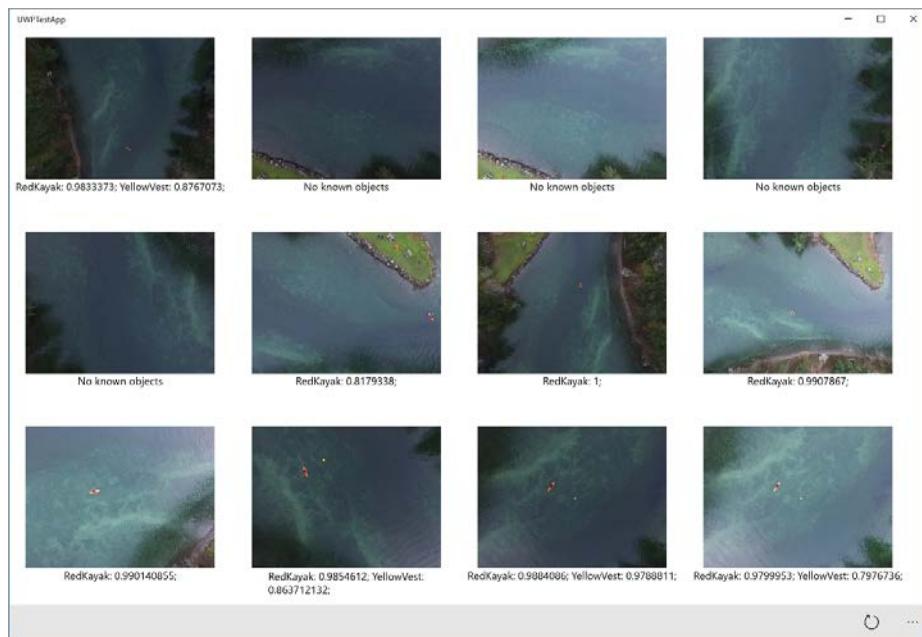
## Staples

Интернет-магазины часто полагаются на органический поисковый трафик для увеличения продаж. Один из способов занять высокие позиции в рейтинге поисковых систем — добавить описательные теги в текстовое поле ALT для изображения. Компания Staples Europe, обслуживающая на 12 разных языках, обнаружила, что добавление тегов к изображениям продуктов и перевод ключевых слов — довольно дорогостоящая услуга, которая традиционно передается на аутсорсинг

агентствам, нанимающим для этого людей. К счастью, Clarifai предоставляет теги на 20 языках по гораздо более низкой цене, что позволило Staples сократить расходы до пятизначных цифр. Использование релевантных ключевых слов привело к увеличению трафика и в итоге к увеличению продаж через интернет-магазин за счет увеличения количества посетителей на страницах продуктов.

## InDro Robotics

Эта канадская компания по производству дронов использует Microsoft Cognitive Services не только для проведения поисково-спасательных операций во время стихийных бедствий, но и для предупреждения чрезвычайных ситуаций. Компания использует Custom Vision для обучения моделей, специализирующихся на выявлении таких объектов, как лодки и спасательные жилеты на воде (рис. 8.25), и использует эту информацию для уведомления постов наблюдения. Дроны могут самостоятельно сканировать гораздо большие участки океана, чем спасатели. Эта автоматизированная система предупреждает спасателей о чрезвычайных ситуациях и повышает скорость обнаружения и вероятность спасения жизни.



**Рис. 8.25.** Аварийные ситуации, обнаруженные системой InDro Robotics

Австралия начала использовать дроны других компаний с надувными капсулами, чтобы люди могли удерживаться на плаву, пока не подойдет помощь. Вскоре после развертывания такие капсулы спасли жизнь двум подросткам, оказавшимся

в океане (рис. 8.26). Австралия использует дроны для обнаружения акул и оповещения купающихся на пляжах. Нетрудно понять, насколько ценными могут быть подобные автоматизированные сервисы обучения собственных моделей.



**Рис. 8.26.** Дрон обнаружил двух пловцов и сбросил надувную капсулу, чтобы помочь им удержаться на плаву (изображение взято по адресу <https://oreil.ly/dPxBv>)

## Итоги

В этой главе мы изучили различные облачные API компьютерного зрения, сначала сравнив спектр предлагаемых услуг, а затем количественно оценив их точность и стоимость. Мы также рассмотрели потенциальные источники предвзятости, которые могут появиться в результатах. Мы увидели, что можем начать использовать эти API уже через 15 минут, написав всего несколько строк кода. Поскольку иногда стандартных моделей может оказаться недостаточно, мы обучили свой классификатор, воспользовавшись простым веб-интерфейсом, а затем сравнили несколько API между собой. Наконец, мы обсудили рекомендации по сжатию и изменению разрешения изображений для ускорения их передачи и их влияние на различные задачи. В завершение мы увидели, как разные компании используют облачные API для создания реальных приложений.

Поздравляем, вы забрались очень далеко! В следующей главе мы покажем, как развернуть собственный сервер прогнозирования для нестандартных сценариев.

## ГЛАВА 9

---

# Масштабируемый инференс в облаке с помощью TensorFlow Serving и KubeFlow

Представьте, что вы только что создали первоклассный классификатор. Ваша цель, как гласит девиз Кремниевой долины, — «сделать мир лучше», что вы и собираетесь сделать... с помощью великолепного классификатора собак и кошек. У вас есть надежный бизнес-план, и вы с нетерпением ждете, как на следующей неделе представите свой волшебный классификатор инвесторам. Вы знаете, что они будут спрашивать об облачной стратегии, поэтому нужно убедить их в выгодности предприятия, чтобы они задумались об инвестициях в вашу разработку. Как это сделать лучше всего? Создание модели — это лишь полдела, следующая задача — ее обслуживание — часто более серьезная. Фактически на обучение модели ушло всего несколько недель, но чтобы передать ее дальше, предстоит многомесячная битва, в которую будут вовлечены бэкенд-инженеры и команды DevOps.

В этой главе ответим на несколько вопросов, которые обычно возникают в контексте размещения и обслуживания моделей.

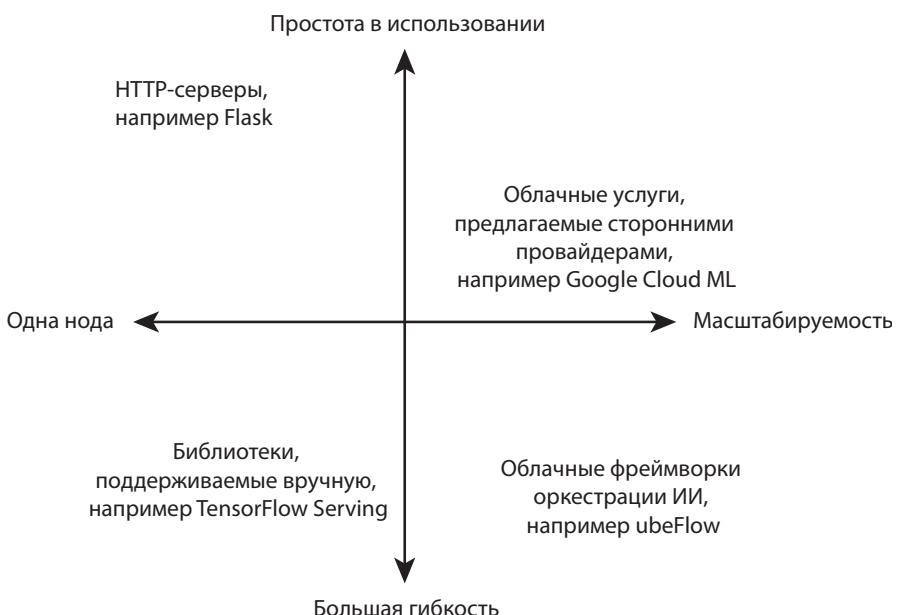
- Как разместить модель на своем личном сервере, чтобы мои коллеги могли ее использовать?
- Я не специалист по бэкенду и инфраструктуре, но хочу сделать свою модель доступной, чтобы она могла обслуживать тысячи (или даже миллионы) пользователей. Как реализовать это за разумные деньги, не беспокоясь о проблемах масштабируемости и надежности?
- Есть причины (стоимость, требования законодательства, конфиденциальность и т. д.), по которым я не могу разместить свою модель в облаке — только локально (в рабочей сети). Можно ли в таком случае обеспечить масштабность и надежность прогнозирования?

- Можно ли реализовать вычисление прогнозов на GPU?
- Сколько придется заплатить за каждый из возможных вариантов?
- Можно ли масштабировать обучение и работу моей модели с привлечением нескольких провайдеров облачных услуг?
- Сколько времени и знаний потребуется, чтобы запустить проект?

Что ж, начнем наше путешествие с общего обзора доступных инструментов.

## Ландшафт услуг прогнозирования с помощью ИИ

Есть много инструментов, библиотек и облачных служб для создания обученных моделей искусственного интеллекта для получения прогнозов. На рис. 9.1 показано, как можно разделить их на четыре категории.



**Рис. 9.1.** Сравнение вариантов инференс-сервисов

Выбор зависит от конкретного сценария инференса. Плюсы и минусы этих вариантов представлены в табл. 9.1.

**Таблица 9.1.** Инструменты для обслуживания моделей глубокого обучения по сети

Категория и примеры	Ожидаемое время до первого предсказания	Плюсы и минусы
<b>HTTP-серверы</b> • Flask • Django • Apache OpenWhisk • Python <code>http.server</code>	< 5 минут	+ Простота запуска + Выполняет самый свежий код на Python – Невысокая скорость – Не оптимизировано для ИИ
<b>Облачные сервисы сторонних провайдеров</b> • Google Cloud ML • Azure ML • Amazon Sage Maker • Algorithmia	< 15 минут	+ Простой интерфейс, графический или командной строки + Высокая масштабируемость + Полностью управляемый поставщиком услуги, что снижает потребность в создании команды для поддержки – Обычно ограничивается инференсом на CPU, из-за чего большие модели могут работать медленно – Время «прогрева» может быть большим
<b>Библиотеки, поддерживаемые вручную</b> • TensorFlow Serving • NVIDIA TensorRT • DeepDetect • MXNet Model Serving • Skymind Intelligence Layer с DeepLearning4J • Seldon • DeepStack AI Server	<15 минут	+ Высокая производительность + Позволяет вручную управлять оптимизацией, пакетной обработкой и т. д. + Возможность инференса на GPU – Более сложная настройка – Масштабирование с использованием нескольких нод обычно требует дополнительных усилий
<b>Облачные фреймворки оркестрации ИИ</b> • KubeFlow	~1 час	+ Упрощает масштабирование этапов обучения и прогнозирования + Переносимость между поставщиками облачных услуг + Единообразие окружений для разработки и эксплуатации + Интеграция с инструментами, хорошо знакомыми специалистам по обработке данных, например Jupyter Notebook

Категория и примеры	Ожидаемое время до первого предсказания	Плюсы и минусы
		<ul style="list-style-type: none"> <li>+ Позволяет конструировать условные пайплайны для автоматизации тестирования, создавать каскадные модели</li> <li>+ Использует реальные библиотеки, обслуживаемые вручную</li> <li>- Все еще развивается</li> <li>- Ориентирован на начинающих, размещается в управляемых облачных окружениях, упрощает освоение</li> </ul>

В этой главе мы рассмотрим различные инструменты и сценарии. Некоторые из этих вариантов просты в использовании, но имеют ограниченную функциональность. Другие предлагают более полный контроль и высокую производительность, но требуют приложить больше усилий для настройки. Рассмотрим по одному примеру в каждой категории и остановимся на некоторых подробностях. Затем представим анализ стоимости различных решений, а также примеры из практики, в которых подробно расскажем, как некоторые из этих решений работают сегодня.

## Flask: создание собственного сервера

Начнем с самого простого варианта создания собственного сервера на примере Flask.

### Создание REST API с помощью Flask

Flask — фреймворк для веб-приложений на Python. Выпущенный в 2010 году и набравший более 46 000 звезд на GitHub, он постоянно развивается. Он легко и быстро настраивается и очень удобен для создания прототипов. Его часто выбирают практикующие специалисты по обработке данных, когда хотят без особых хлопот представить свои модели ограниченному кругу пользователей (например, коллегам в корпоративной сети).

Flask устанавливается с помощью pip:

```
$ pip install flask
```

После установки можно запустить «Hello World»:

```
from flask import Flask
app = Flask(__name__)
```

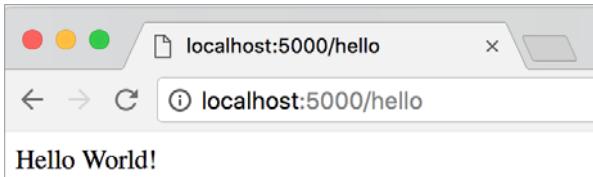
```
@app.route("/hello")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Следующая команда запускает программу «Hello World»:

```
$ python hello.py
* Running on http://127.0.0.1:5000/ (Press Ctrl+C to quit)
```

По умолчанию Flask прослушивает порт 5000. Если теперь в браузере ввести URL *http://localhost:5000/hello*, то вы увидите слова «Hello World!», как показано на рис. 9.2.



**Рис. 9.2.** Введите в адресной строке браузера URL *http://localhost:5000/hello*, чтобы увидеть веб-страницу с текстом «Hello World!»

Для создания и запуска простого веб-приложения требуется написать всего несколько строк. Одна из самых важных строк в этом сценарии — декоратор `@app.route("/hello")`. Он сообщает, что путь `/hello` после имени хоста будет обслуживаться методом, следующим непосредственно за ним. В нашем случае этот метод просто возвращает строку «Hello World!». Далее мы посмотрим, как развернуть модель Keras на сервере Flask и создать маршрут, который будет получать прогнозы от нашей модели и возвращать их пользователям.

## Развертывание модели Keras в Flask

Первым делом загрузим модель Keras. Следующий код загружает модель из файла *.h5*. Код и все необходимые файлы для этой главы вы найдете в репозитории книги (<https://github.com/PracticalDL/Practical-Deep-Learning-Book>), в папке *code/chapter-9*:

```
from tf.keras.models import load_model
model = load_model("dogcat.h5")
```

Теперь определим маршрут */infer* для получения прогнозируемого класса изображения. Отправка изображений будет осуществляться методом POST:

```
@app.route('/infer', methods=[POST])
def infer():
    file = request.files['file']
    image = Image.open(file)
    image = preprocess(image)

    predictions = model.predict(image)
    max_index = numpy.argmax(predictions)
    # Нам известны метки, возвращаемые моделью, которую мы обучили раньше
    if max_index == 0:
        return "Cat"
    else:
        return "Dog"
```

Чтобы проверить результат, воспользуемся командой `curl`, передав ей образец изображения с собакой:

```
$ curl -X POST -F image=@dog.jpg 'http://localhost:5000/infer'  
{"predictions": [{"label": "dog", "probability": 0.8525022864341736}]}]
```

Мы получили ожидаемый прогноз «dog» (собака). Пока все хорошо. Сейчас Flask работает локально, то есть никто другой в сети не сможет послать запрос этому серверу. Чтобы открыть доступ к Flask для других, достаточно просто изменить `app.run()`, как показано ниже:

```
app.run(host="0.0.0.0")
```

Теперь любой в нашей сети сможет обратиться к модели. Следующий вопрос: можно ли сделать что-то, чтобы открыть доступ к модели широкой публике? Ответ на этот вопрос — категорическое нет! На сайте проекта Flask можно увидеть предупреждение: «*WARNING: Do not use the development server in a production environment.*» («**ВНИМАНИЕ:** не используйте сервер разработки в производственном окружении»). Flask на самом деле не поддерживает из коробки работу в производственном окружении, и для этого потребуется написать специальный код. В следующих разделах мы покажем, как размещать модели в системах, предназначенных для производственной эксплуатации. А пока отметим некоторые плюсы и минусы Flask.

## Плюсы использования Flask

Flask предлагает ряд преимуществ:

- быстрая установка и создание прототипов;
  - быстрый цикл разработки;
  - нетребовательность в отношении ресурсов;
  - популярность в сообществе Python.

## Минусы использования Flask

В то же время Flask может быть не лучшим выбором:

- невозможность масштабирования — по умолчанию Flask не предназначен для производственных нагрузок, он может обслуживать запросы только по одному;
- нет поддержки управления версиями моделей из коробки;
- нет поддержки пакетной обработки запросов из коробки.

## Желаемые качества системы производственного уровня

Любой общедоступный облачный сервис имеет определенные атрибуты, которые следует учитывать при выборе решения. В контексте машинного обучения есть дополнительные качества, на которые мы должны обращать внимание при создании инференс-сервисов. В этом разделе рассмотрим некоторые из них.

### Высокая доступность

Чтобы пользователи доверяли сервису, он должен быть практически всегда доступен. Многие серьезные игроки оценивают показатель доступности *количество девяток*. Если компания заявляет, что сервис доступен на уровне четырех девяток, это означает, что система работает и доступна 99,99 % времени. Число 99 % выглядит впечатляюще, но, как показано в табл. 9.2, сервис с таким показателем доступности допускает большое суммарное время простоя в год.

**Таблица 9.2.** Суммарное время простоя в год сервисов с разными уровнями доступности

Уровень доступности (%)	Суммарное время простоя в год
99% («две девятки»)	3,65 дня
99,9% («три девятки»)	8,77 часа
99,99% («четыре девятки»)	52,6 минуты
99,999% («пять девяток»)	5,26 минуты

Представьте, насколько нелепой выглядела бы ситуация, если бы Amazon заявил уровень доступности всего 99,9 % и был готов терять миллионы из-за невозмож-

ности обслуживать пользователей в течение восьми с лишним часов простоя. Пять девяток считается Святым Граалем. Любой уровень меньше трех девяток обычно считается не подходящим для высококачественной производственной системы.

## Масштабируемость

Трафик, обрабатываемый производственными сервисами, почти никогда не бывает однородным в течение длительного периода времени. Например, трафик New York Times значительно выше утром, тогда как у Netflix всплеск трафика обычно наблюдается в вечерние часы, когда люди отдыхают после работы. На трафик также влияют сезонные факторы. Трафик Amazon увеличивается на порядок в период распродаж и перед Рождеством.

Более высокий спрос требует большего количества вычислительных ресурсов для его обслуживания, иначе доступность системы окажется под угрозой. Простейший способ добиться этого — оценить самый высокий объем трафика, который система когда-либо будет обслуживать, определить необходимый для этого объем ресурсов, а затем выделить этот объем на неограниченный срок. У этого подхода есть две проблемы: 1) если вы верно определили максимально необходимый объем ресурсов, то все эти ресурсы большую часть времени будут оставаться невостребованными и приводить к напрасной трате денег; и 2) если вы ошиблись в оценке максимально необходимого объема ресурсов, то это повлияет на доступность сервиса, и в итоге вы столкнетесь с гораздо более серьезной проблемой — потерей доверия клиентов и их кошельков.

Разумнее следить за объемом поступающего трафика и динамически выделять и освобождать ресурсы для обслуживания. Это гарантирует, что увеличенный объем трафика будет обрабатываться без потерь, и минимизирует эксплуатационные расходы в периоды низкого трафика.

При освобождении ресурсов есть вероятность, что ресурс, подлежащий освобождению, в этот момент будет обрабатывать трафик. Поэтому перед отключением ресурса необходимо убедиться, что он обработал все полученные запросы. Что очень важно, ресурс не должен принимать новые запросы. Это называется *режимом стока* (draining). Режим стока критичен, когда машины останавливаются для планового обслуживания и/или модернизации.

## Низкая задержка

Обратимся к фактам. В 2008 году компания Amazon опубликовала исследование, где утверждалось, что увеличение задержки в работе ее розничного сайта на каждые 100 мс приводит к потере прибыли на 1 %. Задержка в одну секунду в периоды пиковой нагрузки на сайте привела к потере дохода в 1,6 миллиарда

долларов! Компания Google обнаружила, что задержка в 500 мс на мобильных сайтах приводит к падению трафика на 20%, а это уменьшение количества показов рекламных объявлений на 20%. И это касается не только гигантов индустрии. Если веб-страница загружается на смартфоне дольше трех секунд, 53 % пользователей покидают ее (согласно исследованию Google за 2017 год). Наглядное доказательство пословицы «время — деньги».

Измерение средней задержки может вводить в заблуждение, потому что рисует более радужную картину, чем есть в реальности. Все равно что сказать, будто если в комнате находится Билл Гейтс, то в среднем все находящиеся в ней являются миллиардерами. Вместо средней задержки обычно измеряется процентная задержка. Например, сервис может сообщать о задержке 987 мс в 99 процентиле. Это означает, что 99 % запросов были обслужены за время 987 мс или меньше. Та же самая система может иметь среднюю задержку 20 мс. Конечно, с увеличением трафика к сервису задержка может увеличиться, если сервис не масштабируется в адекватной степени. Таким образом, задержка, высокая доступность и масштабируемость взаимосвязаны.

## Географическая доступность

Расстояние между Нью-Йорком и Сиднеем почти 16 000 км. Скорость света в вакууме составляет примерно 300 000 км/с. Кварцевое стекло (используемое в волоконно-оптических кабелях) уменьшает скорость света примерно на 30 %, до 210 000 км/с. На отрезке оптоволоконного кабеля, проложенного по прямой линии между этими двумя городами, время передачи одного запроса туда и обратно составляет почти 152 мс. Имейте в виду, что при этом не учитывается время, необходимое на обработку запроса на сервере, а также переключения в маршрутизаторах, стоящих на пути. Такой уровень обслуживания был бы неприемлемым для многих приложений.



Хотите узнать, сколько времени потребуется для передачи запросов с вашего компьютера в центр обработки данных конкретного провайдера? В табл. 9.3 перечислены некоторые удобные инструменты для браузеров, которые помогут в этом.

**Таблица 9.3.** Инструменты измерения задержки для разных провайдеров

Сервис	Провайдер облачных услуг
<i>AzureSpeed.com</i>	Microsoft Azure
<i>CloudPing.info</i>	Amazon Web Services
<i>GCPing.com</i>	Google Cloud Platform

Кроме того, для получения более реалистичной информации о задержках при переходе из одного местоположения в другое *CloudPing.co* позволяет измерить межрегиональную задержку при передаче данных между более чем 16 центрами обработки данных AWS в США.

Сервисы, которые предполагается использовать по всему миру, должны географически располагаться так, чтобы снижать задержку для пользователей из разных регионов. Кроме того, объем доступных ресурсов можно динамически увеличивать или уменьшать в зависимости от локального трафика, что дает более полный контроль. Основные провайдеры присутствуют как минимум на пяти континентах (пингвины, простите!).

## Обработка сбоев

Есть старая поговорка, что в жизни неизбежны две вещи: смерть и налоги. В XXI веке эта поговорка применима не только к людям, но и к компьютерному оборудованию. Машины время от времени выходят из строя. Вопрос не в том, *выйдет ли* машина из строя, а в том, *когда* это случится. Одно из важных качеств надежного обслуживания — способность элегантно обрабатывать сбои. Если машина выйдет из строя, нужно быстро переключить трафик на другую машину и продолжить обслуживание. Если из строя выйдет весь центр обработки данных, нужно перенаправить трафик в другой центр обработки данных, чтобы пользователи даже не заметили, что что-то произошло.

## Мониторинг

Если вы не можете что-то измерить, то вы не сможете это улучшить. Хуже того, вы даже не узнаете, что это что-то существует! Мониторинг количества поступающих запросов, доступности, задержки, использования ресурсов, количества нод, распределения трафика и местоположения пользователей жизненно важен для понимания условий работы сервисов, его совершенствования и, что особенно важно, оптимизации расходов на поддержку. Большинство провайдеров уже имеют встроенные панели мониторинга, предоставляющие все эти показатели.

Кроме того, запись аналитики для конкретных задач, такой как время, потраченное моделью для вычисления прогноза, предварительной обработки и т. д., может добавить еще один уровень понимания.

## Управление версиями модели

В этой книге вы узнали (и будете узнавать дальше), что машинное обучение циклично. В частности, в реальном мире постоянно генерируются новые

данные, на которых модель может обучаться. Более того, распределение входных данных может со временем измениться по сравнению с распределением данных, использовавшихся для обучения, что повлечет снижение точности прогнозов (это называется *дрейфом данных*). Чтобы не разочаровывать своих пользователей, следует постоянно совершенствовать свои модели. Каждый раз, обучив модель с использованием новых данных, чтобы еще больше повысить ее точность, мы как можно быстрее должны сделать ее доступной для наших пользователей. Любая хорошая система прогнозирования производственного уровня должна обеспечивать возможность поддержки различных версий модели, в том числе возможность в любой момент заменить действующую версию модели другой.

## A/B-тестирование

Помимо поддержки нескольких версий модели желательно иметь возможность одновременно предоставлять доступ к разным версиям в зависимости от географического положения пользователя, демографических данных или просто в результате случайного выбора.

*A/B-тестирование* — особенно ценный инструмент для совершенствования модели. В конце концов, мы предпочли бы, чтобы наша новая мощная модель была доступна только ограниченному кругу пользователей, пока мы продолжаем выискивать возможные ошибки. Кроме того, если модель хорошо зарекомендовала себя, обслуживая этот узкий круг пользователей, мы можем быть уверены, что она успешно преодолела этап испытаний, и доступ к ней можно смело открыть для всех пользователей.

## Поддержка нескольких библиотек машинного обучения

И последнее, но не менее важное: мы не хотим оставаться привязанными к какой-то одной библиотеке машинного обучения. Одни специалисты по данным могут обучать модели с использованием PyTorch, другие — с TensorFlow или, может быть, с scikit-Learn, достаточной для задач, не связанных с глубоким обучением. Гибкость поддержки нескольких библиотек была бы приятным бонусом.

## Google Cloud ML Engine: управляемый стек облачных услуг ИИ

Учитывая все желательные качества, о которых шла речь в предыдущем разделе, использовать Flask для обслуживания пользователей в производственной

среде — не очень хорошая идея. Если у вас нет отдельной команды по инфраструктуре и вы хотите тратить больше времени на создание более совершенных моделей, чем на их развертывание, правильным выбором будет использовать управляемое облачное решение. Сейчас на рынке есть несколько облачных решений Inference-as-a-Service (инференс как услуга). Для подробного рассмотрения мы выбрали Google Cloud ML Engine, отчасти из-за удобной интеграции с TensorFlow, а отчасти потому, что этот сервис хорошо сочетается с набором инструментов ML Kit, о котором рассказывается в главе 13.

## Плюсы использования Cloud ML Engine

- Простота развертывания моделей в производственном окружении с помощью графического веб-интерфейса.
- Мощность и простота масштабирования для обслуживания миллионов пользователей.
- Возможность получения подробной информации об использовании модели.
- Поддержка нескольких версий моделей.

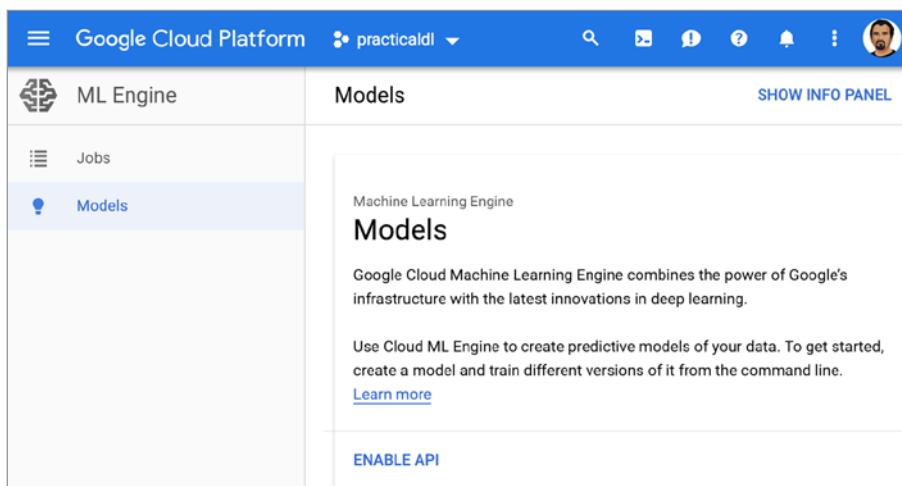
## Минусы использования Cloud ML Engine

- Высокая задержка, инференс только на CPU (по состоянию на август 2019 г.).
- Не подходит для сценариев, связанных с юридическими ограничениями и ограничениями, обусловленными требованиями к конфиденциальности данных, когда данные не должны покидать сеть.
- Накладывает ограничения на архитектуру сложных приложений.

## Создание API классификации

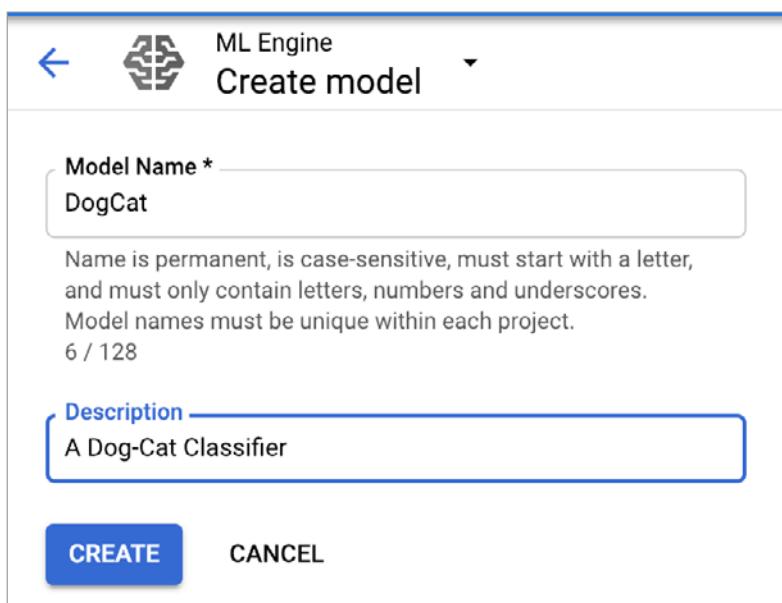
Ниже мы покажем, как загрузить и развернуть нашу модель классификатора кошек и собак в Google Cloud ML Engine:

1. Создайте модель в дашборде Google Cloud ML Engine по адресу <https://console.cloud.google.com/mlengine/models>. При первом использовании дашборда нужно щелкнуть на ссылке **ENABLE API** (Включить API), как показано на рис. 9.3.
2. Дайте модели имя и добавьте описание (рис. 9.4).
3. После создания модели она появится на странице со списком (рис. 9.5).



The screenshot shows the Google Cloud Platform interface for the ML Engine. The top navigation bar includes the 'Google Cloud Platform' logo, a dropdown for the project ('practicaldl'), and various icons for search, refresh, and notifications. On the left, there's a sidebar with 'ML Engine' at the top, followed by 'Jobs' and 'Models'. The 'Models' option is selected and highlighted in blue. The main content area is titled 'Machine Learning Engine Models'. It contains a brief description: 'Google Cloud Machine Learning Engine combines the power of Google's infrastructure with the latest innovations in deep learning.' Below this is a section titled 'Use Cloud ML Engine to create predictive models of your data. To get started, create a model and train different versions of it from the command line.' with a 'Learn more' link. At the bottom of this section is a button labeled 'ENABLE API'. A 'SHOW INFO PANEL' link is located in the top right corner of the main content area.

**Рис. 9.3.** Страница в дашборде Google Cloud ML Engine со списком моделей машинного обучения



The screenshot shows a 'Create model' form within the Google Cloud ML Engine interface. At the top, there's a back arrow icon, the 'ML Engine' logo, and the title 'Create model'. Below this is a 'Model Name \*' field containing the value 'DogCat'. A note below the field specifies: 'Name is permanent, is case-sensitive, must start with a letter, and must only contain letters, numbers and underscores. Model names must be unique within each project.' It also shows a character count of '6 / 128'. Below the name field is a 'Description' field containing the text 'A Dog-Cat Classifier'. At the bottom of the form are two buttons: a blue 'CREATE' button on the left and a 'CANCEL' button on the right.

**Рис. 9.4.** Страница создания модели в Google Cloud ML Engine

The screenshot shows the 'Models' section of the Google Cloud ML Engine interface. At the top, there is a search bar and a 'NEW MODEL' button. Below it, a filter bar says 'Filter by prefix...'. A table lists one model entry:

	Name	Default version	Description	Labels
<input type="checkbox"/>	DogCat	-	A Dog-Cat Classifier	

Рис. 9.5. Страница со списком моделей в Google Cloud ML Engine

4. Щелкните на имени модели, чтобы перейти на страницу с подробной информацией (рис. 9.6), и добавьте новую версию.

The screenshot shows the 'Model Details' page for the 'DogCat' model. It includes a back arrow, a title 'Model Details', and a 'NEW VERSION' button. The 'Name' field is pre-filled with 'DogCat'. Below that, under 'Versions', it says 'This model has no versions yet. Create at least one version to start using your model.' and has a link 'Create a version'.

Рис. 9.6. Страница с подробной информацией о только что созданном классификаторе Dog/Cat

5. Заполните поля необходимой информацией, чтобы создать новую версию. В последнем поле, в самом низу, нужно выгрузить модель в Google Cloud Storage; только после этого вы сможете ею воспользоваться. Щелкните на кнопке **Browse** (Обзор), чтобы выделить место для хранения модели (рис. 9.7).

[←](#) Create version

To create a new version of your model, make necessary adjustments to your saved model file before exporting and store your exported model in Cloud Storage. [Learn more](#)

**Name**  
v1

Name cannot be changed, is case sensitive, must start with a letter, and may only contain letters, numbers, and underscores. 2 / 128

**Description**  
Version 1

**Python version**  
3.5

Select the Python version you used to train the model

**⚠ Model version with Python 3.0 and beyond can't be used for batch prediction jobs. Online prediction still works.**

**Framework**  
TensorFlow

**Framework version**  
1.14.0

**ML runtime version**  
1.14

**Machine type**  
Single core CPU

**Model URI \***  
gs:// Model URI [BROWSE](#)

Cloud Storage path to the entire SavedModel directory. [Learn more](#)

**Рис. 9.7.** Создание новой версии модели машинного обучения

6. Создайте новый раздел (bucket) с уникальным именем, укажите класс хранения и регион. После создания раздела перейдите по адресу <https://console.cloud.google.com/storage/browser> (в другой вкладке, сохранив текущую вкладку открытой), найдите вновь созданный раздел и выгрузите туда модель (рис. 9.8).

**Create a bucket**

Name \* practicaldl-neuralnetworks ?

Must be unique across Cloud Storage. Privacy: Do not include sensitive information in your bucket name. Others can discover your bucket name if it matches a name they're trying to use.

Default storage class ?

Multi-Regional  
Use to stream videos and host hot web content.  
Best for data accessed frequently around the world.

Regional  
Use to store data and run data analytics.  
Best for data accessed frequently in one part of the world.

Nearline  
Use to store rarely accessed documents.  
Best for data accessed less than once a month.

Coldline  
Use to store very rarely accessed documents.  
Best for data accessed less than once every few months.

Region \* us-central1 ▼

Redundant within a single region.

**CREATE** **CANCEL**

**Рис. 9.8.** Создание нового раздела в Google Cloud Storage на странице создания версии модели ML

7. Наша модель классификатора Dog/Cat — это файл с расширением *.h5*. Однако Google Cloud ожидает получить файл в формате SavedModel. Сценарий для преобразования файлов из формата *.h5* в формат SavedModel вы найдете в репозитории книги (<https://github.com/PracticalDL/Practical-Deep-Learning-Book>): *code/chapter-9/scripts/h5\_to\_tf.ipynb*. Просто загрузите модель и запустите остальную часть блокнота.
8. На странице Google Cloud Storage Browser, открытой во второй вкладке, выгрузите преобразованную модель (рис. 9.9) в раздел, созданный на шаге 6.

**Bucket details**

**practicaldl-neuralnetworks**

**Objects**   Overview   Permissions   Bucket Lock

Upload files   Upload folder   Create folder   Manage holds   Delete

Filter by prefix...

Buckets / practicaldl-neuralnetworks / dogcat\_v1

Name	Size	Type	Storage class	Last modified	Public access
saved_model.pb	1.93 MB	application/octet-stream	Regional	2/10/19, 5:48:45 PM UTC-8	Not public
variables/	—	Folder	—	—	Per object

**Рис. 9.9.** Страница Google Cloud Storage Browser после выгрузки модели классификатора Dog/Cat в формате TensorFlow

9. На странице создания версии укажите URI только что выгруженной версии модели (рис. 9.10).

Model URI \*

gs:// practicaldl-neuralnetworks/dogcat\_v1/ BROWSE

Enter the Google Cloud Storage path where you uploaded the model. Select a framework above for file format instructions.

**Рис. 9.10.** Добавление URI модели, выгруженной в Google Cloud Storage

10. Щелкните на кнопке **Save** (Сохранить) и дождитесь, когда завершится создание версии модели. После этого можно использовать ее для получения прогнозов.
11. Загрузите Google Cloud SDK с сайта <https://cloud.google.com/sdk/install> и установите на своем компьютере, если вы этого еще не сделали.
12. Для отправки запросов можно использовать Cloud ML Engine REST API. Но для быстроты воспользуйтесь инструментами командной строки, входящими в состав Cloud SDK. Сначала преобразуйте изображение в файл *request.json* с помощью сценария *image-to-json.py*, находящегося в папке *code/chapter-9*:

```
$ python image-to-json.py --input dog.jpg --output request.json
```

13. Затем передайте только что созданный файл *request.json* вашей модели:

```
$ time cloud ai-platform predict --model DogCat --version v1  
                               --json-instances request.json  
request.json  
  
SCORES  
[0.14749771356, 0.8525022864]  
  
real 0m3.370s  
user 0m0.811s  
sys 0m0.182s
```

Как видите, мы получили те же результаты, что и при использовании сервера Flask, — прогноз «dog» (собака) с достоверностью 85 %.



Перед первым использованием команды **gcloud** выполните следующую команду, чтобы привязать инструмент командной строки к вашей учетной записи Google:

```
$ gcloud auth login
```

Затем выберите проект, выполнив следующую команду:

```
$ gcloud config set project {имя_проекта}
```

Здорово, правда? В нашем примере мы использовали Google Cloud SDK для отправки запроса на получение прогноза. В производственном окружении тот же запрос можно выполнить, используя конечные точки Google API, либо создав HTTP-запрос, либо прибегнув к помощи клиентских библиотек. Более полную информацию ищите в документации Google Cloud Docs.

Сейчас модель готова обслужить любого пользователя, находящегося в любой точке мира и использующего веб-приложение, приложение на мобильном или

встраиваемом устройстве, на ПК или в облаке. Использование управляемого стека — жизнеспособный вариант для независимых разработчиков и компаний, которым нужна гибкость и надежность облачной инфраструктуры при минимальных затратах на настройку и обслуживание.

Но иногда сторонние решения по размещению — не лучший подход. Причины могут быть разными: дороговизна, требования к конфиденциальности данных, юридические ограничения, технические ограничения, проблема доверия или договорные обязательства. В таких случаях предпочтительнее локальное решение.



Для обработки сразу нескольких изображений можно изменить сценарий *image-to-json.py* и реализовать в нем создание файла *request.json*, содержащего массив входных данных.

## TensorFlow Serving

TensorFlow Serving — это библиотека с открытым исходным кодом в экосистеме TensorFlow, предназначенная для обслуживания моделей машинного обучения. В отличие от Flask, она оптимизирована для повышения производительности, имеет низкие накладные расходы и предназначена для использования в продакшене. TensorFlow Serving широко используется крупными компаниями для обслуживания своих моделей в сервисах прогнозирования. Это один из неотъемлемых компонентов TensorFlow Extended (TFX) — сквозного пайплайна глубокого обучения в экосистеме TensorFlow.

В соответствии с набором желаемых качеств производственной системы, который мы рассмотрели выше, TensorFlow Serving обеспечивает низкую задержку, обработку сбоев, высокую пропускную способность и поддерживает управление версиями моделей. Еще одно преимущество — поддержка нескольких моделей в рамках одного сервиса. В ней реализовано несколько методов ускорения обслуживания:

- Во время запуска сервера запускается пул потоков выполнения для быстрой загрузки модели.
- Использует отдельные пулы потоков для загрузки моделей и вычисления прогнозов, при этом потоки из пула вычисления прогнозов получают более высокий приоритет. Это уменьшает задержки при обработке запроса.
- Создает мини-пакеты входящих асинхронных запросов за короткие периоды времени. Как мы уже видели, пакетная обработка данных на GPU во время обучения помогает добиться большей производительности; аналогичную цель преследует объединение в пакеты асинхронных запросов. На-

пример, в один пакет можно объединить запросы, поступившие в течение последних 500 мс. Конечно, такое решение задержит обработку первого запроса в пакете на 500 мс, но снизит среднюю задержку по всему пакету и максимизирует использование оборудования.



Библиотека TensorFlow Serving дает полный контроль над развертыванием модели. С ее помощью в одном процессе можно обслуживать разные модели или разные версии одной и той же модели. От вас требуется только указать имя и местоположение версии, которую вы хотите удалить или запустить в эксплуатацию.

## Установка

Установить TensorFlow Serving можно несколькими способами:

- собрать из исходного кода;
- загрузить и установить с помощью APT;
- развернуть образ Docker.

Любителям приключений посоветуем сборку из исходного кода. Но если вы хотите быстро приступить к работе, рекомендуем развернуть образ Docker — он требует минимальной настройки для запуска системы. Вы спросите, что такое Docker? Docker обеспечивает виртуальную среду для выполнения приложений в Linux и гарантирует изоляцию ресурсов, предлагая по сути «чистый лист» для настройки окружения, в котором может работать приложение. Как правило, приложение и все его зависимости упаковываются в один контейнер Docker, который затем можно развертывать по мере необходимости. Поскольку приложение настроено на выполнение в чистом окружении, это снижает вероятность ошибок конфигурации и развертывания. Все это делает Docker очень привлекательным решением для запуска приложений в продакшене.

Большим преимуществом Docker является устранение «ада зависимостей», потому что все необходимые зависимости упаковываются в контейнер. Еще одно преимущество — процесс настройки приложения остается более или менее одинаковым для разных платформ, неважно, используете ли вы Windows, Linux или Mac.

Инструкции по установке Docker на разных plataформах ищите на странице Docker. Установка займет не более нескольких минут, потому что настройка довольно проста. После установки Docker выполните следующую команду, чтобы настроить TensorFlow Serving для вычислений на CPU:

```
$ docker run -p 8501:8501 \
--mount type=bind,source=/path/to/dogcat/,target=/models/dogcat \
-e MODEL_NAME=dogcat -t tensorflow/serving
```

На компьютерах с GPU можно выполнить такую команду:

```
$ docker run -p 8501:8501 --runtime=nvidia \
--mount type=bind,source=/path/to/dogcat/,target=/models/dogcat \
-e MODEL_NAME=dogcat -t tensorflow/serving
```

В обоих случаях, если все прошло успешно, будет запущен REST API, прослушивающий локальный порт 8501 и обслуживающий наш классификатор Dog/Cat.



Общая задержка обработки любого запроса складывается из времени выполнения нескольких этапов процесса, в том числе время на передачу данных туда и обратно, время сериализации и десериализации объектов в запросе и ответе и, конечно же, фактическое время вычисления прогноза. Еще один компонент, который увеличивает накладные расходы, — это обслуживающая инфраструктура, то есть TensorFlow Serving. В Google утверждают, что накладные расходы библиотеки TensorFlow Serving минимальны. В своих экспериментах специалисты Google выяснили, что сама TensorFlow Serving способна обрабатывать до 100 000 запросов в секунду на каждое ядро. Эксперименты проводились на машине с 16 vCPU Intel Xeon E5 с тактовой частотой 2,6 ГГц. Поскольку это оценка накладных расходов библиотеки, в нее не входят затраты на вызов удаленных процедур (RPC) и вычисление прогноза в TensorFlow.

Библиотека TensorFlow Serving — отличный выбор для инференса на единственной машине, но она не имеет встроенной поддержки горизонтального масштабирования. Эта библиотека предназначалась для использования в сочетании с другими системами, способными дополнить TensorFlow Serving динамическим масштабированием. Одно из таких решений мы рассмотрим в следующем разделе.

## KubeFlow

В книге мы рассматривали разные этапы пайплайна глубокого обучения, от приема данных, анализа, распределенного обучения (включая настройку гиперпараметров), планирования экспериментов, развертывания и, наконец, до обслуживания запросов. Все эти этапы сложны сами по себе со своими наборами инструментов, экосистемами и областями знаний. Люди посвящают жизнь развитию экспертизы только в одной из этих областей. Это не простая прогулка. Комбинаторный взрыв необходимых знаний с учетом разработки ПО, проектирования оборудования и инфраструктуры, управления зависимостями, эксплуатации и сопровождения, обеспечения отказоустойчивости и решения других инженерных задач может потребовать от компании нанять высокооплачиваемых специалистов.

Как мы видели в предыдущем разделе, Docker избавляет от забот по управлению зависимостями, позволяя использовать переносимые контейнеры. Это помогает развертывать TensorFlow Serving на разных платформах без необходимости выполнять сборку из исходного кода или устанавливать зависимости вручную. Круто! Но у Docker до сих пор нет ответа на многие другие вопросы. Как масштабировать контейнеры, чтобы удовлетворить растущий спрос? Как эффективно распределить трафик между контейнерами? Как сделать так, чтобы контейнеры видели друг друга и могли взаимодействовать?

На эти вопросы отвечает *Kubernetes*. Kubernetes — это среда оркестрации, позволяющая автоматически развертывать, масштабировать и управлять контейнерами (например, контейнеры Docker). Поскольку Docker предлагает преимущества переносимости, можно использовать Kubernetes для развертывания контейнеров на ноутбуках разработчиков и в кластерах из тысяч машин практически идентичным образом. Это помогает поддерживать согласованность в различных окружениях и дает дополнительное преимущество масштабируемости. Стоит отметить, что Kubernetes не является специализированным решением для машинного обучения (как и Docker). Скорее это универсальное решение многих проблем, возникающих при разработке ПО, которое используется в контексте глубокого обучения.

Но не будем забегать вперед. В конце концов, если бы Kubernetes была действительно универсальным решением, ее название появилось бы в названии главы. Специалист по машинному обучению, использующий Kubernetes, еще должен собрать все нужные наборы контейнеров (для обучения, развертывания, мониторинга, управления API и т. д.) и организовать их вместе, чтобы создать действующий сквозной пайплайн. К сожалению, многие специалисты по данным именно это и пытаются сделать у себя, заново изобретая пайплайны для машинного обучения. А можно ли создать единое решение на основе Kubernetes для сценариев машинного обучения?

Такую амбициозную цель поставили перед собой разработчики *KubeFlow*. Они обещают автоматизировать большую часть инженерных задач и скрыть сложность запуска распределенной масштабируемой сквозной системы глубокого обучения за фасадом веб-интерфейса и инструментов командной строки. Это больше чем простой инференс-сервис. Это огромная экосистема инструментов, способных беспрепятственно взаимодействовать между собой и, что более важно, масштабироваться по мере необходимости. Фреймворк KubeFlow изначально создавался для работы в облаке. Причем не в каком-то одном облаке — фреймворк совместим со всеми основными облачными провайдерами. Это дает большие выгоды в плане стоимости. Поскольку фреймворк не привязан к конкретному провайдеру, мы в любой момент можем перенести все наши операции, если конкурирующий провайдер предложит более низкие цены. Наглядный пример того, как конкуренция приносит пользу потребителям.

Фреймворк KubeFlow поддерживает разные аппаратные инфраструктуры, от ноутбуков разработчиков и локальных центров обработки данных до общедоступных облачных сервисов. А поскольку в его основе Docker и Kubernetes, можно быть увереными, что окружения будут идентичными, независимо от того, развернуты ли они на ноутбуке или в большом кластере в центре обработки данных. Любые отличия в настройках у разработчика и в производственном окружении могут привести к потере работоспособности, поэтому очень важно иметь такую согласованность между окружениями.

В табл. 9.4 приводится краткий список инструментов, доступных в экосистеме KubeFlow.

**Таблица 9.4.** Инструменты, доступные в экосистеме KubeFlow

Инструмент	Назначение
Jupyter Hub	Окружение для блокнотов Jupyter
TFJob	Обучение моделей TensorFlow
TensorFlow Serving	Обслуживание моделей TensorFlow
Seldon	Обслуживание моделей
NVIDIA TensorRT	Обслуживание моделей
Intel OpenVINO	Обслуживание моделей
KFServing	Абстракция для обслуживания моделей Tensorflow, XGBoost, scikit-learn, PyTorch и ONNX
Katib	Настройка гиперпараметров и выбор архитектуры нейронной сети
Kubebench	Запуск заданий бенчмаркинга
PyTorch	Обучение моделей PyTorch
Istio	Обслуживание API, аутентификация, A/B-тестирование, развертывание, показатели работы
Locust	Нагрузочное тестирование
Pipelines	Управление экспериментами, заданиями и прогонами, планирование процессов машинного обучения

Как говорят в сообществе, с таким количеством готовых технологий KubeFlow наконец-то сделал язык соискателей и рекрутеров совместимым.

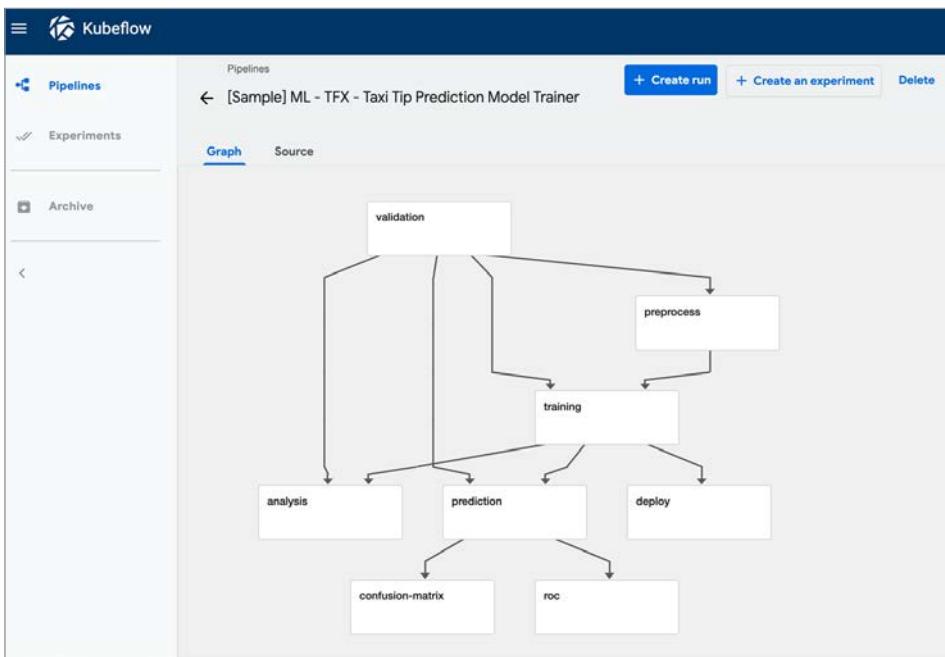
KubeFlow делится на две основные части, которые делают его уникальным: пайплайны и инструменты управления.



Многие считают, что KubeFlow – это комбинация Kubernetes и TensorFlow, но это далеко не так. Кроме этих двух компонентов во фреймворк входит и многое другое.

## Пайплайны

Пайплайны дают возможность объединять этапы машинного обучения для планирования сложных рабочих процессов. На рис. 9.11 показан пример пайплайна. Возможность заглянуть в пайpline с помощью инструмента с графическим интерфейсом помогает понять его всем стейкхолдерам (не только инженерам, которые его построили).



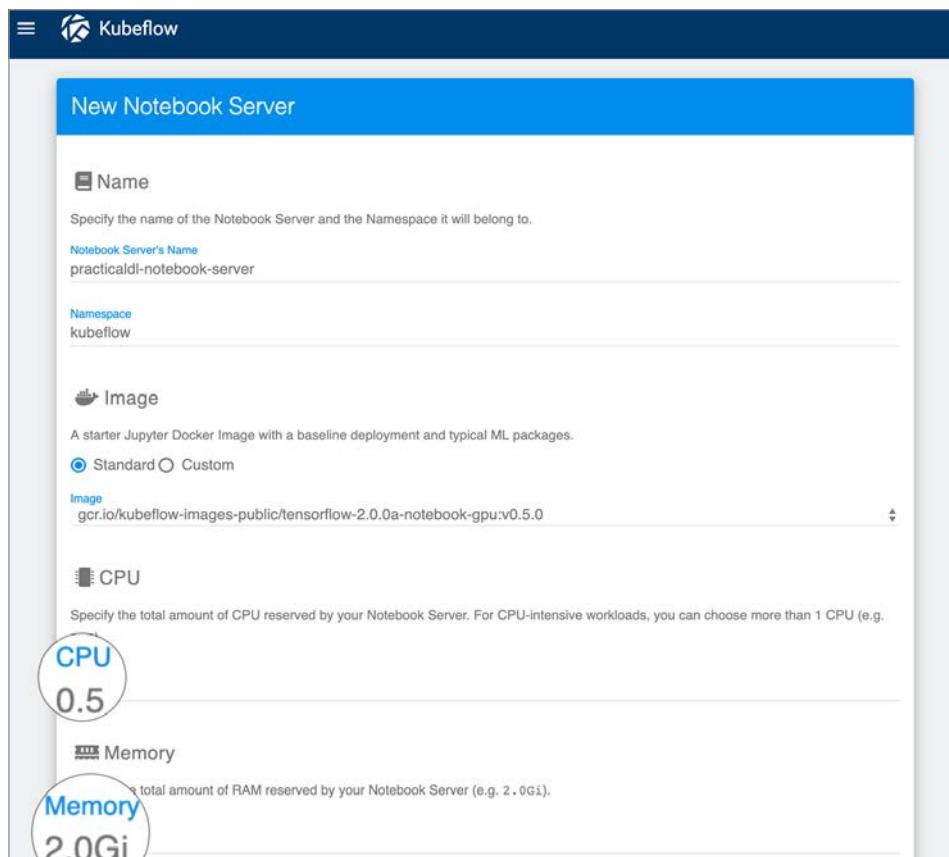
**Рис. 9.11.** Сквозной пайpline в веб-интерфейсе KubeFlow

## Инструменты управления

Инструменты управления позволяют управлять всем жизненным циклом сборки, обучения и развертывания непосредственно через блокноты Jupyter. На рис. 9.12 показано, как запустить новый сервер блокнотов, на котором можно разместить все блокноты Jupyter, запустить в них обучение и развернуть

получившиеся модели в Google Cloud, используя всего несколько строк кода, не теряя при этом комфорта знакомой среды Jupyter:

```
from fairing.deployers.gcp.gcpserving import GCPoServingDeployer
GCPoServingDeployer().deploy(model_dir, model_name, version_name)
```



**Рис. 9.12.** Создание нового сервера блокнотов Jupyter Notebook в KubeFlow

## Установка

Развертывание KubeFlow — довольно простой процесс, который подробно описан на сайте KubeFlow. Можно развернуть и настроить KubeFlow в GCP с помощью браузера. Для развертывания KubeFlow в GCP, AWS и Microsoft Azure можно также воспользоваться инструментом командной строки. На рис. 9.13 показано развертывание KubeFlow в GCP с помощью браузера.

Create a Kubeflow deployment

Project\*  
practicaldl

Deployment name\*  
kubeflow

Choose how to connect to kubeflow service:\*

Login with GCP Iap

- An endpoint protected by GCP IAP will be created for accessing kubeflow. Follow these [instructions](#) to create an OAuth client and then enter as IAP OAuth Client ID and Secret

IAP OAuth client ID\*  
[REDACTED].apps.googleusercontent.com

IAP OAuth client secret\*

GKE zone:\*

us-central1-a

Kubeflow version:\*

v0.5.0

Create Permanent Storage    Share Anonymous Usage Report

**Create Deployment**   Kubeflow Service Endpoint   View YAML

Рис. 9.13. Развёртывание KubeFlow в GCP с помощью браузера

На момент написания этих строк фреймворк KubeFlow продолжал активно разрабатываться, и никаких признаков близкого окончания этого процесса не было видно. Компании Red Hat, Cisco, Dell, Uber и Alibaba — одни из активных его участников помимо облачных гигантов Microsoft, Google и IBM. Простота и доступность для решения сложных задач привлекают людей к любым платформам, а KubeFlow именно таков.

## Соотношение цены и производительности

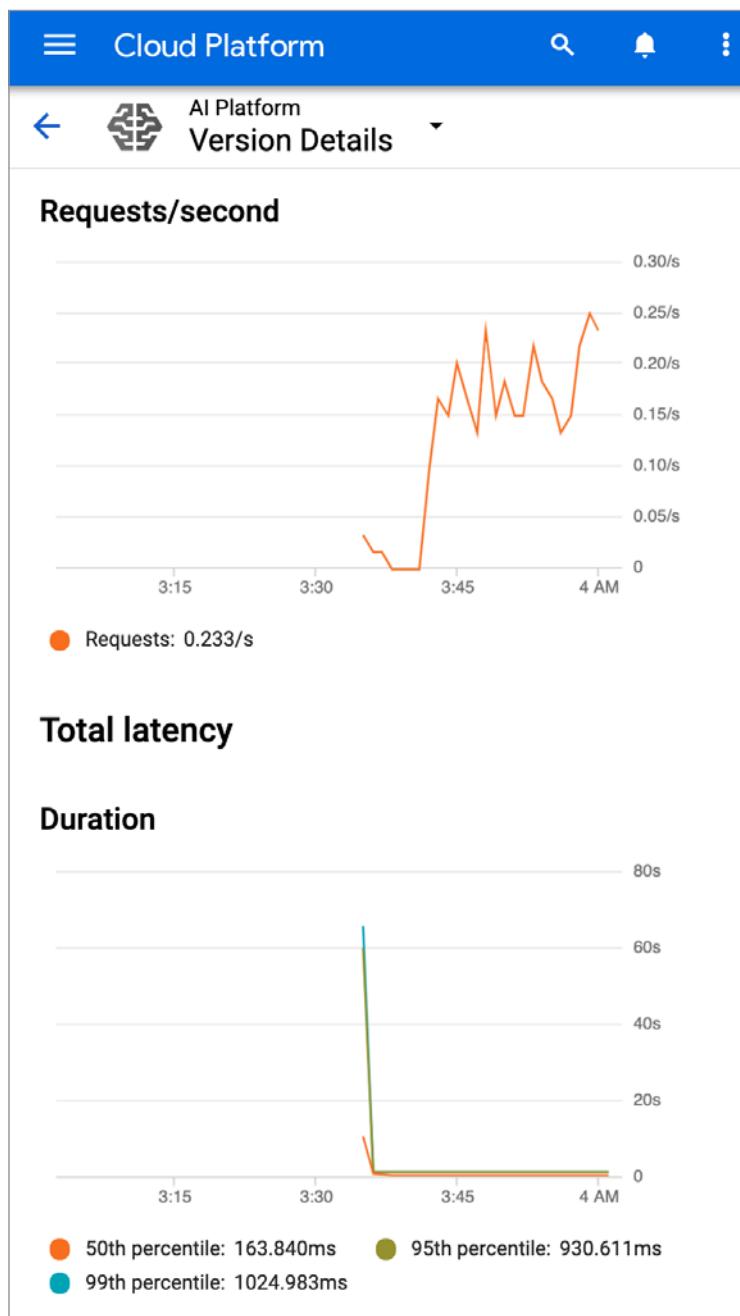
В главе 6 мы рассмотрели вопросы повышения производительности наших моделей при инференсе (на смартфонах или на сервере). Теперь рассмотрим соотношение производительности оборудования и стоимости услуги.

Часто при создании производственной системы для нас важна гибкость в выборе подходящего оборудования, чтобы можно было обеспечить баланс между производительностью, масштабом и ценой. Допустим, надо разработать приложение, использующее облачные услуги для инференса. Можно настроить свой стек вручную (используя Flask, TensorFlow Serving или KubeFlow) или использовать управляемый стек Inference-as-a-Service (например, Google Cloud ML Engine). Сколько это будет стоить, если предположить, что наш сервис стал популярен?

### Анализ затрат на управляемый стек Inference-as-a-Service

По состоянию на август 2019 года услуга Google Cloud ML Engine в Северной Америке стоила недорого — 0,0401 доллара за час комбинированного времени на машине с одноядерным процессором. Есть вариант машины с четырехъядерным процессором, но в реальности для большинства применений достаточно одного ядра. Обработка нескольких запросов с небольшими изображениями размером 12 Кбайт занимала в среднем 3,5 секунды, как показано на рис. 9.14. Это довольно медленно, отчасти потому, что инференс выполнялся на сравнительно медленной машине и, что более важно, исключительно на CPU. Стоит отметить, что этот тест проводится на «разогретой» машине, которая недавно обработала аналогичный запрос и, следовательно, модель уже загружена в память. Для сравнения: обработка самого первого запроса на этой машине занимает от 30 до 60 секунд. Это показывает важность постоянной загрузки сервиса работой или периодической отправки запросов для разогрева. Причина в том, что движок Google Cloud ML отключает модель, если замечает длительный простой.

Если бы запросы поступали каждую секунду в течение всего месяца, всего было бы обработано  $60 \times 60 \times 24 \times 30 = 2\,592\,000$  запросов. Если предположить, что на обработку каждого запроса требуется 3,5 секунды, то одной ноды будет явно недостаточно. Сервис быстро заметит это и в ответ на увеличение трафика подключит три дополнительные машины для его обработки. В целом работа четырех машин в течение месяца по цене 0,0401 доллара США за машино-час будет стоить 115,48 доллара США. Для сравнения: обработка двух миллионов запросов обойдется в такую же сумму, что и покупка одной чашки кофе в день в Starbucks в течение месяца. И давайте не будем забывать, что обработка выполняется автоматически, без участия многочисленной команды DevOps, время которой стоит дорого. Если взять гипотетический сценарий сервиса, похожего на Yelp, пользователи которой выгружают фотографии еды со средней



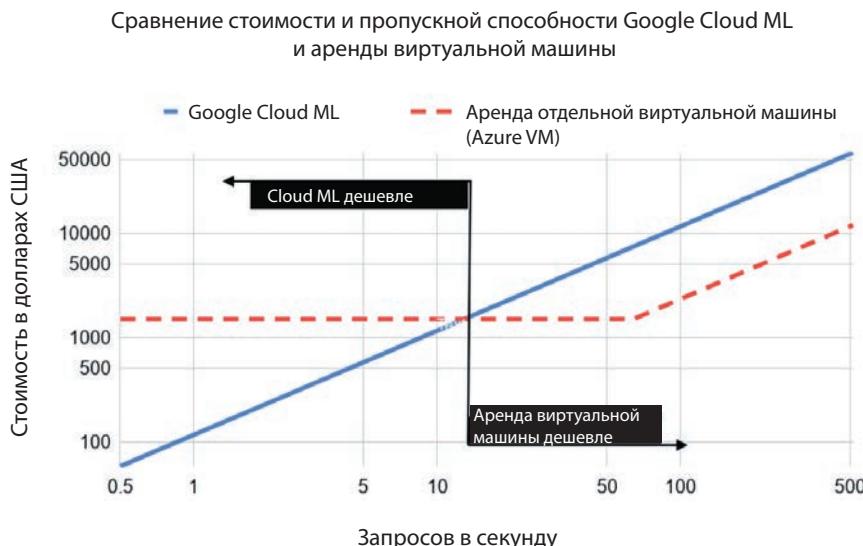
**Рис. 9.14.** Google Cloud ML Engine показывает входящие запросы и время их обработки; полная задержка, измеренная на стороне пользователя, составляет около 3,5 секунды

скоростью 64 запроса в секунду, вычисление прогнозов по ним с использованием модели классификации будет стоить всего 7390 долларов.

## Анализ затрат на создание собственного стека

Меньше затрат и высокая масштабируемость — вот выигрышная комбинация. Но есть один недостаток — суммарная задержка на обработку каждого запроса. Взяв дело в свои руки, развернув виртуальную машину со скромным GPU в облаке и настроив масштабируемый пайплайн (с использованием KubeFlow или TensorFlow Serving и встроенных функций балансировки нагрузки в облаке), можно обрабатывать запросы за миллисекунды или настроить объединение нескольких запросов (скажем, за каждые 500 мс). В качестве примера рассмотрим виртуальную машину в Azure: за 2,07 доллара в час мы получим машину ND6 с графическим процессором NVIDIA P40 и 112 Гбайт ОЗУ. Объединяя входящие запросы в пакеты за 500–1000 мс, этот компьютер может обрабатывать по 64 запроса в секунду всего за 1490 долларов в месяц, причем быстрее, чем Google Cloud ML Engine.

Подводя итог, можно сказать, что экономия затрат и увеличение производительности за счет аренды виртуальной машины в облачном окружении очень быстро проявит себя в сценариях с большим количеством запросов в секунду, как показано на рис. 9.15.



**Рис. 9.15.** Сравнение затрат на инфраструктуру как услугу (Google Cloud ML Engine) и создание собственного стека на виртуальных машинах (Azure VM) (затраты подсчитаны по состоянию на август 2019 года)



При тестировании часто возникает вопрос: с какой предельной нагрузкой справится моя система? Ответить на него поможет JMeter (<https://jmeter.apache.org/>). JMeter – это инструмент нагрузочного тестирования, который позволяет выполнить стресс-тест системы с помощью простого в использовании графического интерфейса. Он позволяет создавать повторно используемые конфигурации для моделирования различных сценариев использования.

## Итоги

В этой главе мы ответили на вопрос, который задают многие инженеры и разработчики: как масштабировать обслуживание запросов на вычисление прогнозов в реальном мире? Мы изучили четыре разных метода обслуживания модели распознавания изображений: с помощью Flask, Google Cloud ML, TensorFlow Serving и KubeFlow. В зависимости от масштаба, требований к задержке и уровня квалификации одни решения могут быть более привлекательными, чем другие. Наконец, мы изучили вопрос рентабельности различных стеков. Теперь, когда мы можем показать миру нашу потрясающую модель, осталось только сделать нашу работу вирусной!

## ГЛАВА 10

---

# ИИ в браузере с TensorFlow.js и ml5.js

Написана совместно с Заидом Аляфеи (Zaid Alyafeai)

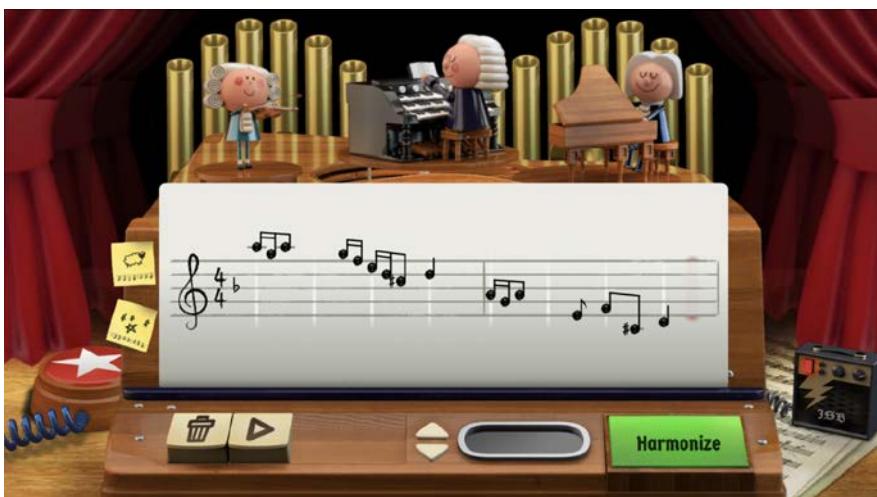
Вы разработчик, мечтающий о чем-то большом. У вас есть классная модель ИИ, и вам хочется, чтобы ее попробовало как можно больше людей. А «как можно больше» — это сколько? Десять тысяч? Миллион? Конечно нет! Вы мечтаете о большем. Как насчет 100 миллионов? Красивое круглое число. Вот только убедить 100 миллионов человек загрузить и установить приложение на смартфоны — непростая задача. А что, если мы скажем вам, что у всех этих людей уже установлено приложение, предназначенное только для вас. Без загрузок. Без установок. Без магазинов приложений. Что еще за черная магия, скажете вы. Это не магия, это браузер. И он также работает на вашем компьютере.

Именно так поступили в Google со своей домашней страницей, когда решили представить свой первый в истории тематический ИИ миллиардам пользователей (рис. 10.1). И что может быть лучше, чем музыка И. С. Баха. (Родители Баха, решив назвать его И. С. Бахом<sup>1</sup> за 310 лет до появления JavaScript, были необычайно дальновидны!)

Если не вдаваться в подробности, то это приложение ИИ позволяет любому написать пару тактов из случайных нот (одноголосную мелодию) с помощью мыши. Когда пользователь щелкнет на кнопке с надписью **Harmonize** (Обработать), входные данные сопоставляются с сотнями музыкальных произведений, написанных Бахом, содержащих от двух до четырех голосов. Система определяет, какие ноты будут лучше всего сочетаться с вводом пользователя, чтобы на выходе получился музыкальный фрагмент с более богатым звучанием, похожий на творчество Баха. Весь процесс выполняется в браузере, поэтому Google не пришлось расширять свою инфраструктуру машинного обучения.

---

<sup>1</sup> Здесь игра слов: на английском языке инициалы И. С. записываются как J.S.; примерно так же в сокращенной форме записывается название языка JavaScript. — Примеч. пер.



**Рис. 10.1.** Преобразователь музыки в стиле Баха, созданный в Google (<https://oreil.ly/BYFfg>)

Кроме экономии средств и возможности работать на любой платформе, с помощью браузера мы можем дать пользователям более богатый интерактивный опыт, потому что в этом случае сетевые задержки перестают играть важную роль. И поскольку после загрузки модель может работать локально, сама собой отпадает проблема сохранения конфиденциальности данных конечного пользователя.

Учитывая, что браузеры говорят на языке JavaScript, будет полезно познакомиться с библиотеками глубокого обучения на этом языке, способными запускать обученные модели в браузерах пользователей. Именно этим мы сейчас и займемся.

Мы сосредоточимся на реализации моделей глубокого обучения в браузере. Сначала рассмотрим краткую историю различных фреймворков глубокого обучения на JavaScript, затем познакомимся с библиотекой TensorFlow.js и, наконец, с более высокоуровневой абстракцией под названием ml5.js. Мы также рассмотрим несколько сложных приложений, выполняющихся в браузере, — определение позы человека или преобразование нарисованного от руки наброска в фотографию (с использованием генеративно-состязательной сети). А в заключение дадим некоторые практические соображения и примеры из реальной жизни.

## Библиотеки машинного обучения на JavaScript: краткая история

С момента прорыва в сфере глубокого обучения было предпринято множество попыток сделать ИИ доступным для широкого круга людей в форме веб-

библиотек. В табл. 10.1 вы найдете краткий обзор различных библиотек в порядке их появления.

**Таблица 10.1.** Исторический обзор библиотек глубокого обучения на JavaScript (по состоянию на август 2019 г.)

Библиотека	Годы развития	Звезд на GitHub	Отличительные особенности
brain.js	2015 — по настоящее время	9856	Нейронные сети, рекуррентные нейронные сети (RNN), механизм долгой краткосрочной памяти (LSTM), управляемые рекуррентные блоки (GRU)
ConvNetJS	2014–2016	9735	Нейронные сети, сверточные нейронные сети (CNN)
Synaptic	2014 — по настоящее время	6571	Нейронные сети, механизм долгой краткосрочной памяти (LSTM)
MXNetJS	2015–2017	420	Обслуживание моделей MXNet
Keras.js	2016–2017	4562	Обслуживание моделей Keras
CaffeJS	2016–2017	115	Обслуживание моделей Caffe
TensorFlow.js прежде известная как deeplearn.js)	2017 — по настоящее время	11282	Обслуживание моделей TensorFlow на GPU
ml5.js	2017 — по настоящее время	2818	Простота использования поверх TF.js
ONNX.js	2018 — по настоящее время	853	Высокая скорость, обслуживание моделей ONNX

Познакомимся с некоторыми поближе и посмотрим, как они развивались.

## ConvNetJS

ConvNetJS (<https://oreil.ly/URdv9>) — эта библиотека была разработана в 2014 году Андреем Карпатым (Andrej Karpathy), когда он учился в аспирантуре Стэнфордского университета. Он обучал сверточные сети в браузере, что выглядело весьма захватывающе, особенно в 2014 году. В ту пору как раз начала набирать силу шумиха вокруг ИИ, а его библиотека избавляла разработчиков от необходимости проходить через сложный и болезненный процесс настройки перед началом работы. ConvNetJS — первая библиотека, которая познакомила огромное количество людей с искусственным интел-

лектом с помощью интерактивных обучающих демонстрационных примеров, выполняющихся в браузере.



Фактически, когда Лекс Фридман (Lex Fridman) — ученый из МИТ — читал свой популярный курс автоматизации управления в 2017 году, он предлагал студентам со всего мира обучить симулятор беспилотного автомобиля с помощью ConvNetJS, используя прием обучения с подкреплением, как показано на рис. 10.2.

```

54 brain = new deepqlearn.brain(num_inputs, num_actions, opt);
55
56 learn = function (state, lastReward) {
57   brain.backward(lastReward);
58   var action = brain.forward(state);
59
60   draw_net();
61   draw_stats();
62
63   return action;
64 }

```

**Рис. 10.2.** Приложение DeepTraffic для обучения модели автомобиля с помощью ConvNetJS и использованием приема обучения с подкреплением

## Keras.js

Библиотека Keras.js была создана в 2016 году Леоном Ченом (Leon Chen). Это была версия Keras, написанная на JavaScript и адаптированная для работы в браузере. Для выполнения вычислений на GPU Keras.js использовала WebGL. Для

вычисления прогнозов она использовала шейдеры (специальные операции для отображения пикселей), благодаря чему скорость работы была намного выше, чем при использовании только CPU. Кроме того, Keras.js может работать на сервере в окружении Node.js, что позволяет организовать вычисления на сервере. В Keras.js реализовано несколько видов слоев: сверточные, полносвязные, объединяющие, слои активации и рекуррентные. В настоящее время библиотека не развивается.

## ONNX.js

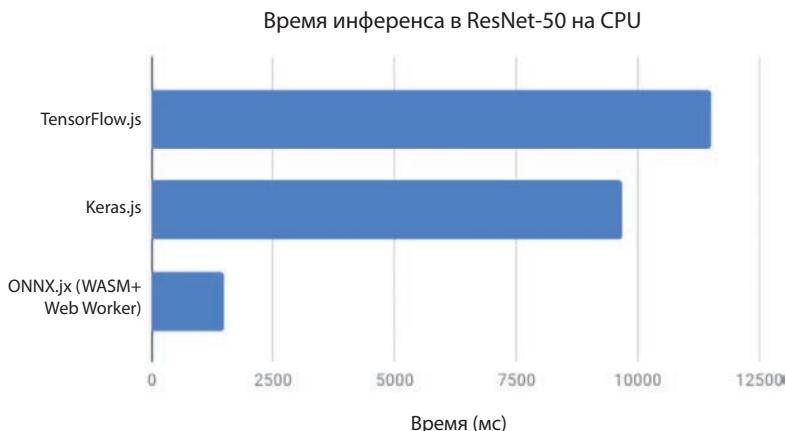
ONNX.js – библиотека, созданная Microsoft в 2018 году для обслуживания моделей ONNX в браузерах и в окружении Node.js. ONNX – это открытый стандарт представления моделей машинного обучения, созданный в сотрудничестве несколькими компаниями: Microsoft, Facebook, Amazon и др. Библиотека ONNX.js на удивление быстрая. Фактически она работает быстрее, чем TensorFlow.js (о чём поговорим в следующем разделе) в ранних бенчмарках, как показано на рис. 10.3 и 10.4. Причины этому могут быть такие:

- ONNX.js использует механизм WebAssembly (разработан в Mozilla) для выполнения на CPU и WebGL – на GPU;
- WebAssembly позволяет запускать программы на C/C++ и Rust в браузерах, обеспечивая при этом производительность, близкую к производительности машинного кода;
- WebGL обеспечивает возможность переноса на GPU таких вычислений, как обработка изображений в браузере;
- подавляющее большинство сценариев выполняется в браузерах в одном потоке, но ONNX.js использует механизм Web Workers для организации многопоточного окружения и параллельного выполнения операций с данными.

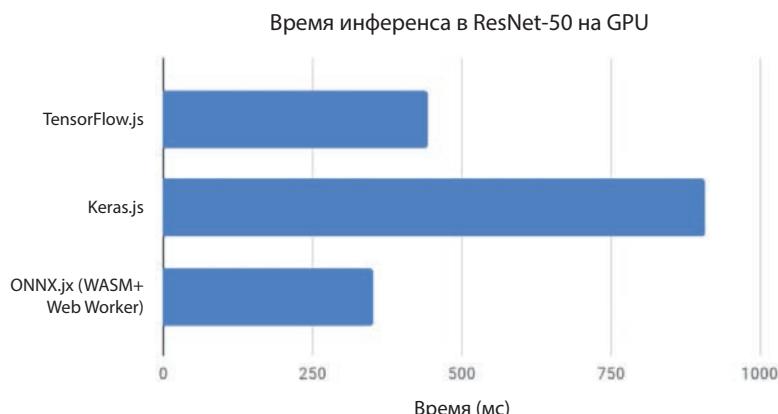
## TensorFlow.js

Одни библиотеки предлагали возможность обучения в браузере (например, ConvNetJS), другие имели невероятно высокую производительность (например, несуществующая ныне TensorFire). Библиотека deeplearn.js, созданная в Google, была первой, поддерживавшей вычисления на GPU с использованием WebGL, а также дававшей возможность определять, обучать и вычислять прогнозы в браузере. Она обеспечивала возможность как немедленного (для инференса), так и отложенного выполнения моделей (для обучения, как в TensorFlow 1.x). Появившийся в 2017 году, этот проект стал ядром библиотеки TensorFlow.js (увидела свет в 2018 году). Он считается неотъемлемой частью экосистемы TensorFlow и, как следствие, сейчас это одна из наиболее активно разрабатываемых

емых библиотек глубокого обучения на JavaScript. В этой главе мы сосредоточимся на TensorFlow.js, а также познакомимся с библиотекой ml5.js, еще больше упрощающей работу с TensorFlow.js. Она построена на основе TensorFlow.js и предоставляет более простой API с готовыми к использованию моделями от генеративно-состязательных сетей до PoseNet.



**Рис. 10.3.** Сравнение производительности модели ResNet-50 при выполнении на CPU с использованием различных библиотек JavaScript (данные взяты с сайта <https://github.com/microsoft/onnxjs>)



**Рис. 10.4.** Сравнение производительности модели ResNet-50 при выполнении на GPU с использованием различных библиотек JavaScript (данные взяты с сайта <https://github.com/microsoft/onnxjs>)

### От создателя

Предшественница TensorFlow.js — библиотека deeplearn.js — возникла в результате попытки Google создать интуитивно понятную и интерактивную визуализацию, чтобы научить людей обучать нейронные сети. Эта визуализация, известная как TensorFlow Playground и доступная по адресу <https://playground.tensorflow.org>, использовала раннюю версию deeplearn.js для обучения многослойной нейронной сети в браузере. При создании TensorFlow Playground инженеры были впечатлены возможностью использования WebGL для ускорения обучения моделей глубокого обучения и вычисления прогнозов в браузере на стороне клиента. В результате в Google была создана команда инженеров для реализации этого подхода. Это дало начало TensorFlow.js — полноценной библиотеке глубокого обучения, поддерживающей сотни операций и десятки слоев в нейронных сетях и работающей в различных окружениях от браузера до Node.js, от мобильных до кроссплатформенных настольных приложений.

Шаньцин Цай (Shanqing Cai),  
старший инженер-программист в Google  
и автор книги «Deep Learning with JavaScript» (Manning)

### От создателя

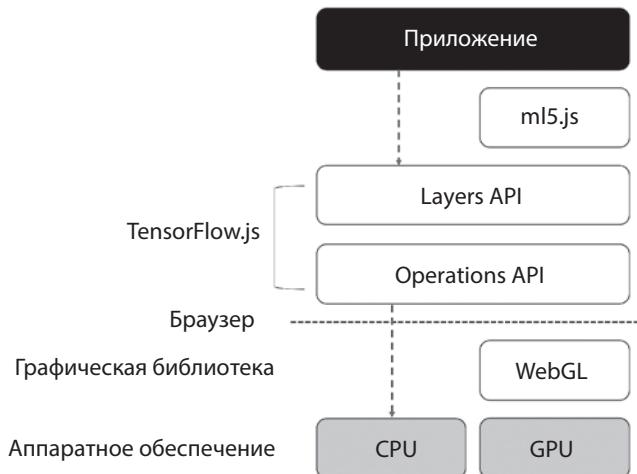
До появления TensorFlow.js мы с Нихилом Торатом (Nikhil Thorat) работали над инструментами интерпретации нейронных сетей в браузере. Чтобы обеспечить настоящую интерактивность, мы хотели организовать инференс и вычисление градиентов прямо в браузере, не отправляя данные на сервер. Это привело к созданию библиотеки deeplearn.js (вы все еще можете увидеть пакет в npm), которую мы выпустили в августе 2017 года. Проект получил дополнительный импульс, когда появились творческие программисты, которые начали его использовать. Благодаря этому наша команда быстро выросла и через шесть месяцев мы выпустили TensorFlow.js.

Даниэль Смилков (Daniel Smilkov), инженер-программист  
в Google Brain и соавтор библиотеки TensorFlow.js

## Архитектура TensorFlow.js

В общих чертах рассмотрим архитектуру TensorFlow.js (рис. 10.5). TensorFlow.js действует непосредственно в браузерах на настольных компьютерах и мобильных устройствах. Она использует WebGL для вычислений на GPU, но также может производить вычисления на CPU в среде выполнения браузера.

Библиотека состоит из двух API: Operations API (API операций) и Layers API (API слоев). Operations API обеспечивает доступ к операциям низкого уровня — тензорной арифметике и другим математическим операциям. Layers API опирается на Operations API и реализует сверточные слои, функции активации (например, ReLU) и т. д.



**Рис. 10.5.** Обобщенная архитектура TensorFlow.js и экосистемы ml5.js

Кроме браузера TensorFlow.js также может работать на сервере Node.js. В добавок к этому библиотека ml5.js реализует более высокоровневый API для доступа к TensorFlow.js и предлагает несколько готовых моделей. Доступ ко всем этим API на разных уровнях абстракции позволяет создавать веб-приложения не только для простого вычисления прогнозов, но и для обучения моделей непосредственно в браузере.

Ниже перечислены типичные вопросы, возникающие в течение жизненного цикла разработки ИИ на основе браузера:

- Как запустить предварительно обученные модели в браузере? Можно ли использовать веб-камеру для интерактивного взаимодействия в масштабе реального времени?
- Как создать модели для браузера из обученных моделей TensorFlow?
- Можно ли обучить модель в браузере?
- Как различное оборудование и браузеры влияют на производительность?

Мы ответим на каждый из этих вопросов, сначала сосредоточившись на возможностях TensorFlow.js, а потом ml5.js. Попутно изучим некоторые интерес-

ные возможности, воплощенные сообществом ml5.js, для реализации которых непосредственно на TensorFlow.js потребовались бы масса усилий и богатый опыт. Рассмотрим и подходы к сравнительному тестированию, а потом изучим несколько любопытных примеров.

А пока посмотрим, как использовать предварительно обученные модели для вычисления прогнозов в браузере.

## Использование предварительно обученных моделей с TensorFlow.js

TensorFlow.js предлагает на выбор множество предварительно обученных моделей, которые можно использовать прямо в браузере, в том числе MobileNet, SSD и PoseNet. Следующий пример показывает, как загрузить предварительно обученную модель MobileNet. Полный код можно найти в репозитории книги на GitHub (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке `code/chapter-10/mobilenet-example/`.

Импортируем последнюю версию библиотеки:

```
<script
  src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js">
</script>
```

Затем импортируем модель MobileNet:

```
<script
  src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0">
</script>
```

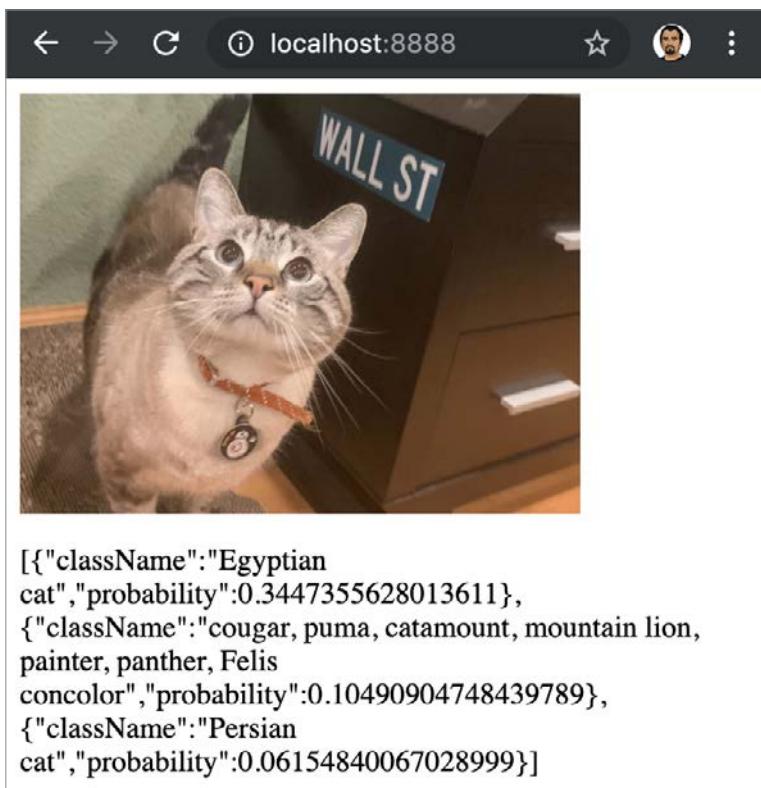
Вычислим прогноз:

```

<p id="prediction_output">Loading predictions...</p>
<script>
  const image = document.getElementById("image");
  const predictionOutput = document.getElementById("prediction_output");

  // Загрузить модель.
  mobilenet.load().then(model => {
    // Классифицировать изображение и вывести прогноз
    model.classify(image).then(predictions => {
      predictionOutput.innerText = predictions[0].className;
    });
  });
</script>
```

На рис. 10.6 показан пример вывода.



**Рис. 10.6.** Вывод результатов классификации в браузере

Модель можно также загрузить как файл JSON:

```
const path =  
  'https://storage.googleapis.com/tfjsmodels/tfjs/mobilenet_v1_1.0_224/  
  model.json';  
  
model = tf.loadLayersModel(path).then(model => {  
  // Загрузить модель и вывести прогноз  
});
```

Файл JSON содержит саму модель, имена параметров модели и пути к файлам сегментов с весами. Сегментирование позволяет хранить веса в относительно небольших файлах, чтобы браузеры могли их кэшировать и тем самым ускорять загрузку модели, когда это понадобится.

## Преобразование модели для использования в браузере

В предыдущем разделе вы увидели, как загрузить предварительно обученную модель, которая уже хранится в формате JSON. Сейчас мы покажем, как преобразовать предварительно обученную модель Keras (в формате *.h5*) в файл JSON, совместимый с TensorFlow.js. Для этого с помощью `pip` установим инструмент преобразования:

```
$ pip install tensorflowjs
```

Если предположить, что обученная модель Keras хранится в папке *keras\_model*, то ее можно преобразовать следующей командой:

```
$ tensorflowjs_converter --input_format keras keras_model/model.h5 web_model/
```

После этого в каталоге *web\_model* появятся файлы *.json* и *.shard*, которые можно загрузить с помощью метода `tf.loadLayersModel`:

```
$ ls web_model  
group1-shard1of4 group1-shard3of4 model.json group1-shard2of4 group1-shard4of4
```

Вот и все! Как оказалось, перенести обученную модель в браузер совсем несложно. В случаях, когда нет обученной модели, TensorFlow.js позволяет обучить ее прямо в браузере. В следующем разделе мы покажем, как это сделать, на исчерпывающем примере обучения модели с помощью веб-камеры.



Для загрузки модели в локальной системе необходимо запустить веб-сервер. Сделать это можно множеством способов, начиная от установки стека LAMP (Linux, Apache, MySQL, PHP) и заканчивая установкой модуля `http-server` с помощью `npm` и даже запуском Internet Information Services (IIS) в Windows. Даже в Python 3 есть возможность запустить простенький веб-сервер:

```
$ python3 -m http.server 8080
```

## Обучение в браузере

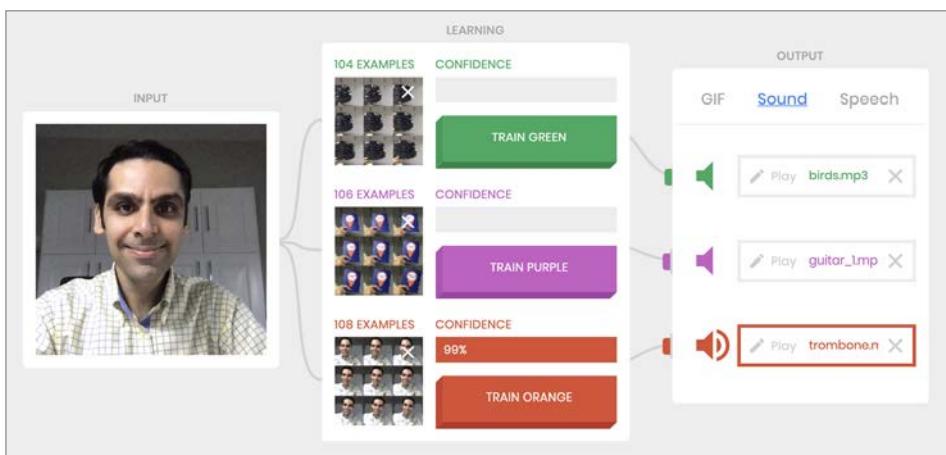
В предыдущем примере мы использовали предварительно обученную модель. Теперь поднимемся на ступеньку выше и обучим собственную модель прямо в браузере, получив данные с веб-камеры. Как и в предыдущих главах, рассмотрим простой пример, где используем перенос обучения, чтобы ускорить процесс.

Позаимствовав решение из Google Teachable Machine, используем перенос обучения, чтобы создать простую модель бинарной классификации, которая будет

обучаться с помощью веб-камеры. Для этого понадобится извлечь признаки (преобразовать входные изображения в признаки, а точнее в эмбеддинги), а затем подключить сеть, которая преобразует эти признаки в прогноз. После этого мы сможем обучить модель, подавая на вход изображения с веб-камеры. Полный код ищите в репозитории книги на GitHub (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке `code/chapter-10/teachable-machine`.



В Google Creative Lab был создан забавный интерактивный сайт Teachable Machine (<https://oreil.ly/jkM6W>), где пользователь может обучить модель классификации на три класса, просто показывая соответствующие объекты перед веб-камерой. Этим трем классам присвоены простые цветовые обозначения: зеленый, фиолетовый и оранжевый. На этапе прогнозирования вместо вывода текста на веб-странице (или, что еще хуже, в консоли) с вероятностями классов Teachable Machine показывает GIF-изображения с мимики животными или воспроизводит разные звуки в зависимости от класса, получившего наибольшую вероятность. Этот сайт мог бы стать забавным и увлекательным занятием для школьников и прекрасным инструментом для их знакомства с ИИ.



**Рис. 10.7.** Обучение Teachable Machine в браузере

## Извлечение признаков

Как мы знаем из первых глав книги, обучение большой модели с нуля — довольно долгий процесс. Намного проще и быстрее использовать предварительно обученную модель и применить методику переноса обучения для адаптации модели под конкретный случай использования. Согласно этой методике, предварительно обученная модель извлекает высокоуровневые признаки (эмбеддинги) из входных изображений и использует их для обучения.

Для извлечения признаков загрузим и используем предварительно обученную модель MobileNet:

```
const path =
  'https://storage.googleapis.com/tfjsmodels/tfjs/mobilenet_v1_1.0_224/
    model.json';
const mobilenet = await tf.loadLayersModel(path);
```

Проверим входы и выходы модели. Мы знаем, что модель была обучена на датасете ImageNet, и последний ее слой предсказывает вероятность каждого из 1000 классов:

```
const inputLayerShape = mobilenet.inputs[0].shape; // [null, 224, 224, 3]
const outputLayerShape = mobilenet.outputs[0].shape; // [null, 1000]
const numLayers = mobilenet.layers.length; // 88
```

Чтобы извлечь признаки, выберем слой, ближайший к выходу, в данном случае слой conv\_pw\_13\_relu, и сделаем его выходным слоем модели, то есть удалим полно связные слои в конце. Созданная нами модель называется моделью извлечения признаков:

```
// Получить конкретный слой
const layer = mobilenet.getLayer('conv_pw_13_relu');

// Создать новую модель для извлечения признаков
featureExtractionModel = tf.model({inputs: mobilenet.inputs,
                                     outputs: layer.output});

featureExtractionModel.layers.length; // 82
```

В процессе обучения мы не будем изменять модель извлечения признаков, а добавим поверх нее набор обучаемых слоев, которые образуют классификатор:

```
const trainableModel = tf.sequential({
  layers: [
    tf.layers.flatten({inputShape: [7, 7, 1024]}),
    tf.layers.dense({
      units: 64,
      activation: 'relu',
      kernelInitializer: 'varianceScaling',
      useBias: true
    }),
    tf.layers.dense({
      units: 2,
      kernelInitializer: 'varianceScaling',
      useBias: false,
      activation: 'softmax'
    })
  ]
});
```

## Сбор данных

Здесь мы получим изображения с помощью веб-камеры и извлечем из них признаки. Функция `capture()` в Teachable Machine отвечает за настройку `webcamImage` для хранения в памяти изображений, полученных с веб-камеры. Теперь обработаем их, чтобы передать в модель извлечения признаков:

```
function capture() {
    return tf.tidy(() => {
        // преобразовать в тензор
        const webcamImage = tf.fromPixels(webcam);
        // Обрезать до размеров 224x224
        const croppedImage = cropImage(webcamImage);
        // создать пакет и нормализовать
        const batchedImage = croppedImage.expandDims(0);

        return batchedImage.toFloat().div(tf.scalar(127)).sub(tf.scalar(1));
    });
}
```

После захвата изображений добавим к ним метки и сохраним в обучающем датасете:

```
function addTrainingExample(img, label) {
    // Извлечь признаки.
    const data = featureExtractionModel.predict(img);
    // Представить метку с помощью унитарного (one-hot) кодирования.
    const oneHotLabel = tf.tidy(() =>
        tf.oneHot(tf.tensor1d([label], 'int32'), 2));
    // Добавить метку и данные в обучающий набор.
}
```

## Обучение

Следующий шаг — обучение модели — во многом напоминает процесс обучения из главы 3. Так же как в Keras и TensorFlow, добавим оптимизатор и определим функцию потерь:

```
const optimizer = tf.train.adam(learningRate);
model.compile({ optimizer: optimizer, loss: 'categoricalCrossentropy' });
model.fit(data, label, {
    batchSize,
    epochs: 5,
    callbacks: {
        onBatchEnd: async (batch, logs) => {
            await tf.nextFrame();
        }
    }
})
```



Имейте в виду, что память в GPU, выделенная TensorFlow.js, не освобождается, когда объект `tf.tensor` выходит из области видимости. Одно из решений — вызвать метод `dispose()` для каждого созданного объекта. Но это сделало бы код более трудным для чтения, особенно при использовании цепочек вызовов. Рассмотрим следующий пример:

```
const result = a.add(b).square().neg();
return result;
```

Чтобы потом освободить память, код нужно разбить на части:

```
const sum = a.add(b);
const square = sum.square();
const result = square.neg();
sum.dispose();
square.dispose();
return result;
```

Вместо этого можно просто воспользоваться методом `tf.tidy()` и поручить ему все заботы по управлению памятью, сохранив код простым и легкочитаемым. Для этого достаточно заключить первую строку в блок `tf.tidy()`, как показано ниже:

```
const result = tf.tidy(() => {
  return a.add(b).square().neg();
});
```

При выполнении вычислений на CPU объекты автоматически удаляются сборщиком мусора браузера. Вызов `.dispose()` в этом случае не даст никакого эффекта.

Для простоты обучим модель классификации эмоций. Для этого нужно собрать обучающие примеры, относящиеся к одному из двух классов: счастливый и нейтральный. Используя эти данные, можно приступить к обучению. На рис. 10.8 показан окончательный результат после обучения модели на 30 изображениях в каждом классе.



Нейтральный



Счастливый

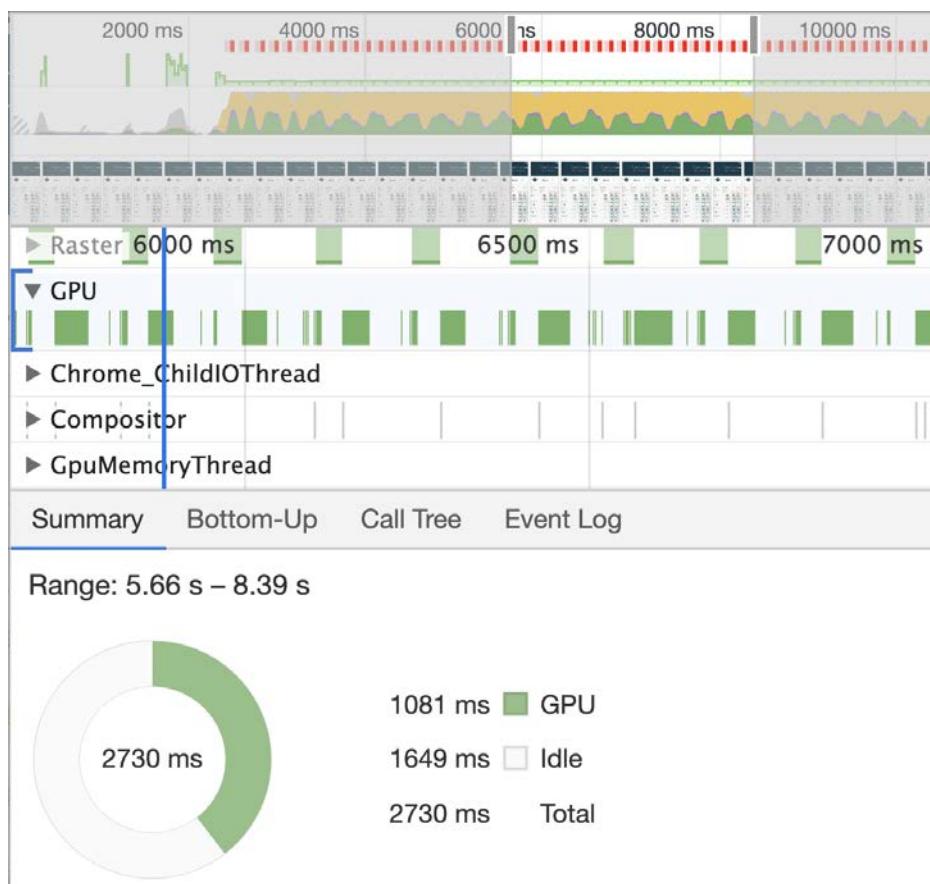
**Рис. 10.8.** Результаты классификации изображений нашей моделью, полученных с веб-камеры



Обычно, когда для прогнозирования используется веб-камера, UI имеет тенденцию зависать. Это связано с тем, что вычисления происходят в том же потоке, что и отображение UI. Вызов `await tf.nextFrame()` освободит поток UI, что сделает веб-страницу отзывчивой и предотвратит зависание вкладки/браузера.

## Нагрузка на GPU

Увидеть уровень нагрузки на CPU или GPU во время обучения и инференса можно с помощью профилировщика Chrome. В предыдущем примере мы наблюдали за нагрузкой на GPU и фиксировали ее в течение 30 секунд. Как показано на рис. 10.9, в течение этого времени GPU использовался на четверть.



**Рис. 10.9.** Периоды использования GPU в профилировщике Google Chrome

До сих пор мы обсуждали, как реализовать все необходимое с нуля, включая загрузку модели, захват видео с камеры, сбор обучающих данных, обучение модели и запуск инференса. Но было бы гораздо удобнее, если бы все эти этапы выполнялись автоматически, а нам оставалось бы только сосредоточиться на том, что делать с результатами прогнозирования. В следующем разделе обсудим, как этого достичь с помощью ml5.js.

## ml5.js

ml5.js — это библиотека высокогоуровневой абстракции TensorFlow.js, которая упрощает использование предварительно обученных моделей глубокого обучения, предлагая единообразное решение, для реализации которого требуется писать минимум строк кода. В пакет входит широкий спектр готовых моделей, от сегментации изображений до классификации звука и генерации текста, как показано в табл. 10.2. Кроме того, ml5.js сокращает количество шагов для предварительной обработки, постобработки и т. д., что позволяет сконцентрироваться на создании приложений, использующих эти модели. Все функциональные возможности ml5.js сопровождаются демонстрационным (<https://ml5js.org/reference>) и справочным кодом.

**Таблица 10.2.** Список моделей, входящих в состав ml5.js, демонстрирует широкий спектр возможностей для работы с текстом, изображениями и звуком

Модель	Описание
PoseNet	Определяет местоположения суставов человека
U-Net	Сегментирует объекты, например удаляет фоновые объекты
Style Transfer	Переносит стиль из одного изображения в другое
Pix2Pix	Преобразует изображения, например черно-белые в цветные
Sketch RNN	Заканчивает начатый набросок
YOLO	Определяет объекты, например выделяет лицо рамкой
Sound Classifier	Распознает звуки, например свист, хлопок в ладоши, «один», «стоп» и т. д.
Pitch Detector	Оценивает высоту звука
Char RNN	Генерирует новые тексты после обучения на объемном тексте
Sentiment Classifier	Классифицирует эмоциональную окраску текстовых предложений
Word2Vec	Генерирует эмбеддинги слов для выявления отношений между словами

Модель	Описание
Feature Extractor	Генерирует признаки или эмбеддинги из входных данных
kNN Classifier	Создает быстрые классификаторы с использованием метода $k$ ближайших соседей

Посмотрим, как она работает. Импортируем последнюю версию библиотеки ml5.js, как мы делали это с TensorFlow.js:

```
<script src="https://unpkg.com/ml5@latest/dist/ml5.min.js"
type="text/javascript"></script>
```

Обратите внимание, что теперь не нужно импортировать ничего, связанного с библиотекой TensorFlow.js, потому что она уже включена в состав ml5.js. Реализуем тот же сценарий с использованием MobileNet, что и выше:

```
// Инициализировать классификатор изображений на основе MobileNet
const classifier = ml5.imageClassifier('MobileNet', modelLoaded);

// Сгенерировать прогноз для выбранного изображения
classifier.predict(document.getElementById('image'), function(err, results) {
  console.log(results);
});
```

Готово! Потребовалось написать всего три строки кода, чтобы запустить в браузере предварительно обученную модель. Теперь откроем консоль браузера, где должен появиться результат, изображенный на рис. 10.10.

```
▼ (3) [...]
  ▶ 0: Object { label: "Labrador retriever", confidence: 0.4080851674079895 }
  ▶ 1: Object { label: "Ibizan hound, Ibizan Podenco", confidence: 0.4062190651893616 }
  ▶ 2: Object { label: "Chesapeake Bay retriever", confidence: 0.04214375838637352 }
    length: 3
  ▶ <prototype>: Array []
```

**Рис. 10.10.** Наиболее вероятные классы с соответствующими вероятностями



Для тех, кто никогда не пользовался консолью браузера: ее можно открыть, щелкнув правой кнопкой мыши в любом месте в окне браузера и в контекстном меню выбрать пункт **Inspect element** (Изучить элемент). После этого откроется отдельное окно с консолью внутри.

Полный код примера можно найти в папке *code/chapter-10/ml5js*.

Обратите внимание, что для работы с моделями, выполняющимися асинхронно, в ml5.js используются обратные вызовы. Обратный вызов — это функция, которая вызывается автоматически после завершения вызова, инициировавшего

операцию. Например, в последнем фрагменте кода после загрузки модели будет вызвана функция `modelLoaded`, таким способом `ml5.js` сообщает, что модель загружена в память.



`p5.js` — это еще одна библиотека, прекрасно работающая в паре с `ml5.js` и позволяющая с помощью моделей вычислять прогнозы в масштабе реального времени на основе видеопотока. Фрагмент кода, показывающий возможности `p5.js`, можно найти в папке `code/chapter-10/p5js-webcam/`.

`ml5.js` изначально поддерживает элементы и объекты `p5.js`. Можно использовать элементы `p5.js` для рисования объектов, захвата изображений с веб-камеры и многое другое, и затем использовать эти элементы в качестве входных данных в функциях обратного вызова `ml5.js`.

## PoseNet

До сих пор в этой книге мы ограничивались преимущественно классификацией изображений. В следующих главах мы поближе познакомимся с задачами распознавания и сегментации объектов. Задачи такого типа широко освещаются в литературе по компьютерному зрению. Но в этом разделе мы решили отклониться в сторону и рассмотреть еще один вид задач: распознавание ключевых точек. Решение таких задач играет важную роль в здравоохранении, спорте, безопасности, играх, дополненной реальности и робототехнике. Например, для популяризации здорового образа жизни с помощью физических упражнений в Мексике установили киоски, в которых ИИ распознает позу приседания и предлагает бесплатные билеты в метро пассажирам, которые смогут сделать не менее 10 приседаний. В этом разделе мы покажем, как запустить что-то столь же мощное в скромном веб-браузере.

Модель PoseNet действует в браузере и способна распознавать позы в реальном времени. «Поза» определяется взаимным расположением различных ключевых точек (включая суставы) человеческого тела — верхняя часть головы, глаза, нос, шея, запястья, локти, колени, лодыжки, плечи и бедра. С помощью PoseNet можно распознать одну или несколько поз, присутствующих в одном кадре.

Создадим пример с PoseNet, который легко реализовать с помощью `ml5.js` и который будет распознавать и рисовать ключевые точки (с помощью `p5.js`).

Код для определения ключевых точек на неподвижном изображении можно найти в папке `code/chapter-10/posenet/single.html`:

```
<script src="http://p5js.org/assets/js/p5.min.js"></script>
<script src="http://p5js.org/assets/js/p5.dom.min.js"></script>
<script src="https://unpkg.com/ml5@latest/dist/ml5.min.js"></script>

<script>
```



**Рис. 10.11.** Ключевые точки на фотографии президента США Барака Обамы, играющего в снежки, нарисованные моделью PoseNet

```
function setup() {
    // Здесь выполняется настройка камеры

    // Вызов модели PoseNet
    const poseNet = ml5.poseNet(video, modelReady);

    // Функция обратного вызова, через которую PoseNet возвращает результат
    poseNet.on('pose', function (results) {
        const poses = results;
    });
}
```

Также можно запустить похожий сценарий, использующий веб-камеру (*code/chapter-10/posenet/webcam.html*).

Теперь рассмотрим другой пример использования библиотеки ml5.js.

## От создателя

Создание библиотеки ml5.js началось с эксперимента в рамках программы интерактивных телекоммуникаций (Interactive Telecommunications Program, ITP) Нью-Йоркского университета, цель которого состояла в том, чтобы оценить перспективы использования машинного обучения в браузерах для творчества. Первоначальная идея, принадлежащая Processing и p5.js, заключалась в проведении серии экспериментов с deeplearn.js (ныне TensorFlow.js) и p5.js. В августе 2017 года, через несколько дней после выхода deeplearn.js, я создал репозиторий «p5deeplearn: deeplearn.js meets p5» для первой попытки. Он стал родоначальником ml5.js, где цифра «5» отдает дань уважения p5.js и самой философии Processing.

Под руководством Дэна Шиффмана (Dan Shiffman) и при поддержке Google Faculty Research Award в сотрудничестве с командой TensorFlow.js мы организовали еженедельные встречи для обсуждения развития самой библиотеки и пригласили художников и исследователей поучаствовать в ее разработке. Основная идея ml5.js всегда заключалась в создании доступного и простого в использовании библиотечного пространства и сообщества, где мы могли бы поощрять эксперименты и творчество с ИИ в браузере.

Кристобаль Валенсуэла (Cristobal Valenzuela),  
соучредитель Runway и участник проекта ml5.js

## От создателя

Несмотря на простоту использования моделей машинного обучения в браузере с помощью TensorFlow.js, начинающие экспериментаторы, желающие написать свой код, по-прежнему сталкиваются с проблемой непонимания математических выкладок и необходимости писать низкоуровневый технический код. Фреймворки машинного обучения, как правило, ориентированы на людей, обладающих знаниями в области численных методов, линейной алгебры, статистики, науки о данных и имеющими немалый опыт программирования на Python или C++. Конечно, все это важно для исследования и разработки новых моделей и архитектур машинного обучения, но столь высокие начальные требования могут отпугнуть новичков. Вместо размышлений о творческом применении машинного обучения как художественной платформы новички могут столкнуться с препятствиями, для преодоления которых требуется понимание тонких различий между скалярами, векторами, матрицами, операциями, входными и выходными слоями и т. д.

Библиотека ml5.js может им помочь. Ее цель — обеспечить дружелюбный подход к созданию и изучению ИИ в браузере. Цифра «5» в названии «ml5» — это дань уважения p5.js, библиотеке JavaScript, которая служит основным источником вдохновения и моделью для ml5. Библиотека p5.js поддерживается

Processing Foundation, цель которой состоит в том, чтобы «дать возможность людям с разными интересами и опытом научиться программировать и творчески экспериментировать с кодом, особенно тем, кто не имеет иной возможности использовать эти инструменты и ресурсы».

Благодаря усилиям Processing художники могут изобретать свои инструменты, не полагаясь на те, которые созданы и поддерживаются другими. В настоящее время «компьютерная грамотность», пожалуй, важнее, чем когда-либо. Алгоритмы влияют на нашу жизнь так, как мы не могли даже представить раньше, а бурный рост исследований в области МО еще больше расширил охватываемые ими сферы. Теперь мы взаимодействуем не только друг с другом, но также с беспилотными транспортными средствами, голосовыми помощниками и камерами, которые определяют наши лица и позы. Без доступа к моделям машинного обучения, базовым и выходным данным, управляющим ПО, и без их понимания мы не сможем осмысленно взаимодействовать со всем этим, задавать вопросы и предлагать альтернативы. Машинное обучение столкнулось с той же проблемой доступности, что и простое обучение программированию более 15 лет назад.

Разработка ml5.js финансируется за счет Google Faculty Research Award в ITP. ITP — это двухгодичная программа для выпускников школы искусств Тиш при Нью-Йоркском университете, цель которой — исследовать творческое использование коммуникационных технологий, как они могут разнообразить и улучшить жизнь людей и привнести в нее радость творчества. Каждую неделю группа из 10–15 студентов программы ITP, а также сторонние участники и приглашенные художники встречаются вместе, чтобы обсудить решения в API, поделиться творческими проектами и помочь друг другу внести свой вклад в открытый исходный код. На сегодняшний день более 50 участников выпустили 17 версий ml5.js.

Дэниел Шиффман (Daniel Shiffman), доцент программы интерактивных телекоммуникаций (Interactive Telecommunications Program, ITP) школы искусств Тиш в Нью-Йоркском университете и директор The Processing Foundation

## pix2pix

«Hasta la vista, baby!»

Это одна из самых известных фраз в истории кино. И произнесена она была киборгом в классическом фильме 1991 года «Терминатор 2: Судный день». Кстати, она переводится как «Прощай, крошка!». С тех пор технология языкового перевода прошла долгий путь. Раньше языковой перевод строился на правилах подстановки фраз. А теперь его заменяют гораздо более эффективные системы глубокого обучения, которые рассматривают контекст предложения, чтобы преобразовать его в предложение с аналогичным значением на целевом языке.

Возникает интересная мысль: если можно перевести предложение с одного языка на другой, то можно ли преобразовать изображение, изменив его характеристики? Например:

- Преобразовать изображение с низким разрешением в изображение с более высоким разрешением.
- Преобразовать черно-белое изображение в цветное.
- Преобразовать дневное изображение в ночное.
- Преобразовать спутниковый снимок Земли в карту.
- Преобразовать эскиз, нарисованный рукой человека, в фотографию.

Что ж, все это — теперь не научная фантастика. В 2017 году Филип Изола (Philip Isola) с коллегами (<https://oreil.ly/g5R60>) разработали способ преобразования изображений и назвали его pix2pix. Модель pix2pix, обученная на нескольких парах изображений «до» и «после», может создавать весьма реалистичные визуализации на основе входного изображения. Например, как показано на рис. 10.12, из карандашного наброска сумки она может создать фотографию этой сумки. Кроме того, она способна сегментировать изображения, синтезировать художественные произведения и многое другое.



**Рис. 10.12.** Примеры преобразования изображений с помощью pix2pix

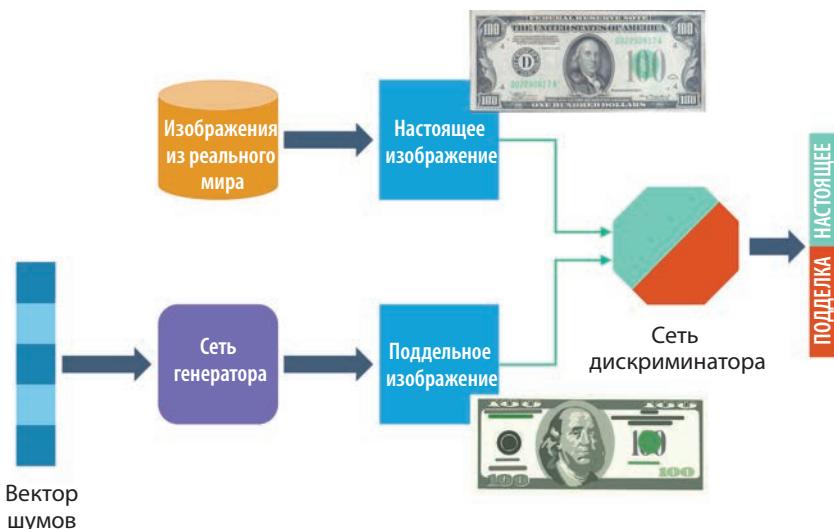


Представьте себе кассира в банке и фальшивомонетчика. Задача кассира — выявить и отбраковать фальшивые банкноты, а цель фальшивомонетчика — так хорошо подделать банкноты, чтобы кассиру банка было как можно сложнее выявлять фальшивки. Они находятся в постоянном противоборстве. Каждый раз, когда кассир выявляет фальшивые купюры, фальшивомонет-

чик узнает о своей ошибке, воспринимает эту ситуацию как возможность усовершенствовать свое мастерство (установка на личностный рост) и пытается еще больше усложнить задачу кассиру банка. В результате такого противостояния кассир тоже совершенствуется в распознавании подделок. Этот цикл обратной связи заставляет их обоих нарабатывать опыт и умения. Это — основной принцип работы генеративно-состязательных сетей.

Как показано на рис. 10.13, GAN состоят из двух сетей, генератора и дискриминатора, которые находятся в таком же противостоянии, как фальшивомонетчик и кассир. Задача генератора — сгенерировать реалистичный результат, очень похожий на обучающие данные. Задача дискриминатора — определить, какие данные были переданы генератором — настоящие или фальшивые. Результат дискриминатора возвращается в генератор и запускает следующий цикл. Каждый раз, когда дискриминатор правильно классифицирует сгенерированный результат как подделку, он заставляет генератор повысить качество своей работы в следующем цикле.

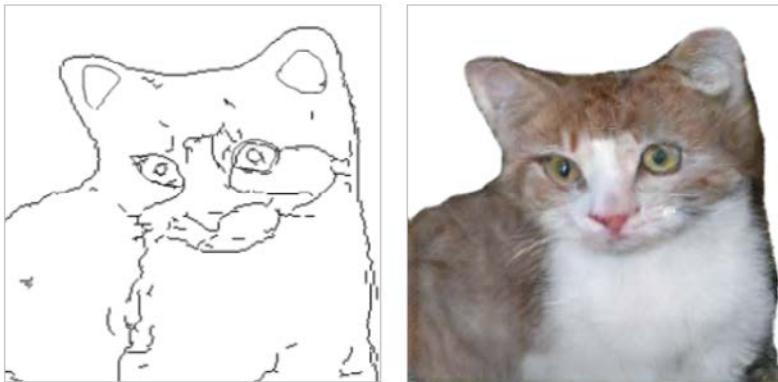
Стоит отметить, что обычно GAN не управляют данными, которые нужно сгенерировать. Но есть такие варианты, как *условные генеративные состязательные сети*, поддерживающие метки во входных данных и тем самым обеспечивающие более полный контроль над генерированием выходных данных, то есть приведением их к соответствию обязательным условиям. pix2pix — это пример условной генеративно-состязательной сети.



**Рис. 10.13.** Схема GAN

Мы использовали pix2pix для создания простого приложения, работающего в браузере и преобразующего эскизы в фотoreалистичные изображения. Сге-

нерированные изображения действительно выглядят интересно. Рассмотрим примеры на рис. 10.14 и 10.15.



**Рис. 10.14.** Пример преобразования эскиза в фотореалистичное изображение

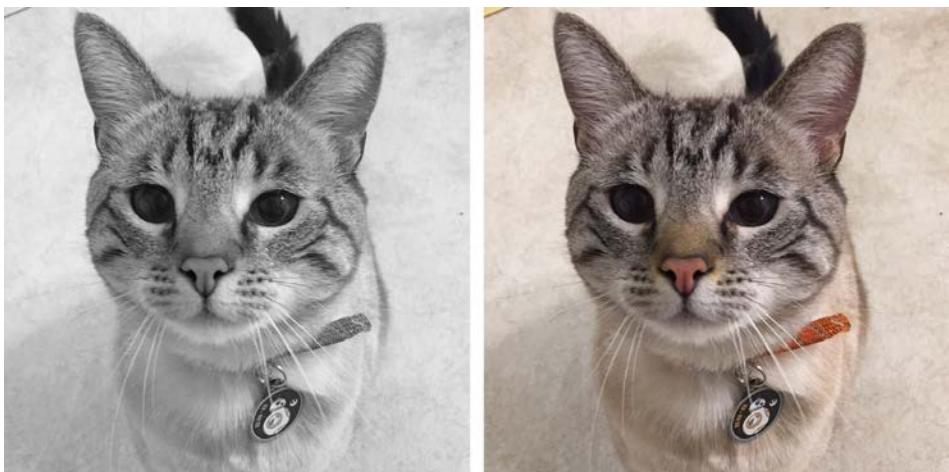


**Рис. 10.15.** Мы создаем цветной схематический набросок (слева), а pix2pix преобразует его в фотореалистичное изображение человеческого лица (справа)



Забавный факт: идея GAN пришла в голову Яну Гудфеллоу (Ian Goodfellow) в баре. Еще один пункт к списку изобретений, организаций и компаний, идеи которых возникли за бокалом, включая создание алгоритма RSA, Southwest Airlines и игру в квиддич.

Pix2pix обучается на парах изображений. Изображение слева на рис. 10.16 – это исходное изображение, или условный вход. Изображение справа – целевое изображение, фотореалистичный результат, который требуется получить (в печатной версии книги изображение справа – черно-белое).



**Рис. 10.16.** Обучение модели pix2pix на паре изображений: черно-белое изображение слева и его цветной оригинал справа

Далее мы воспользуемся одной из простейших реализаций pix2pix, предназначенней для самостоятельного обучения. Ее создал Кристофер Хессе (Christopher Hesse) на основе TensorFlow, использовав простой сценарий:

```
python pix2pix.py \
--mode train \
--output_dir facades_train \
--max_epochs 200 \
--input_dir facades/train \
--which_direction BtoA
```

По окончании обучения готовую модель можно сохранить командой:

```
python tools/export-checkpoint.py --checkpoint ../export \ --output_file
models/MY_MODEL_BtoA.pict
```

После этого можно использовать следующий простой код, чтобы загрузить сохраненные веса в ml5.js. Обратите внимание на вызов функции `transfer`, она используется для вывода результатов в canvas из HTML5:

```
// Создать модель pix2pix из предварительно обученной сети
const pix2pix = ml5.pix2pix('models/customModel.pict', modelLoaded);

// Перенести результат в canvas
pix2pix.transfer(canvas, function(err, result) {
  console.log(result);
});
```

Теперь можно рисовать наброски в масштабе реального времени и преобразовывать их в полноцветные изображения. На рис. 10.17 показан пример с Пикачу.



**Рис. 10.17.** pix2pix: преобразование карандашного наброска с изображением Пикачу (<https://oreil.ly/HlaSy>), пример реализации с использованием ml5.js представил Инин Ши (Yining Shi)

### От создателя

С помощью ml5.js появилось множество интересных проектов и примеров: художники использовали эту библиотеку для создания произведений искусства, а также для иллюстраций во время выступлений. Никита Хаггинс (Nikita Huggins) и Айодамола Окунсеинде (Ayodamola Okunseinde), два художника из Нью-Йорка, изучают способы увеличения разнообразия моделей, которые сейчас поддерживает библиотека, используя приложения, созданные студентами и разработчиками в ходе всевозможных соревнований. Но самое интересное, что она входит в учебные программы по всему миру для показа возможностей машинного обучения в форме, доступной для широкого круга художников, программистов и студентов.

Кристобаль Валенсуэла, соучредитель Runway и участник проекта ml5.js

## Сравнительный анализ и практические рекомендации

Для нас очень важно, как конечные пользователи воспринимают наш продукт, поэтому следует учитывать их мнения. Два критерия, имеющих значение для пользователей, — размер модели и время вычисления результата. Рассмотрим каждый из них подробнее.

## Размер модели

Типичная модель MobileNet имеет размер 16 Мбайт. Ее загрузка из обычной домашней или офисной сети может занять всего несколько секунд. Загрузка той же модели из мобильной сети займет больше времени. Если на загрузку потребуется слишком много времени, пользователь может потерять терпение, причем еще до того, как модель вернет результат. Ожидание загрузки больших моделей оказывает более сильное отрицательное влияние на впечатления пользователя, чем время их выполнения, особенно если скорость интернет-соединения не так высока, как, например, в Сингапуре — районе широкополосного доступа. Вот несколько стратегий, которые могут помочь:

### *Выберите для работы самую маленькую модель*

Среди предварительно обученных сетей наименьший размер имеют (в порядке уменьшения точности) EfficientNet, MobileNet или SqueezeNet.

### *Примените прием квантования модели*

Уменьшите размер модели с помощью TensorFlow Model Optimization Toolkit перед экспортом в TensorFlow.js.

### *Создайте свою архитектуру минимального размера*

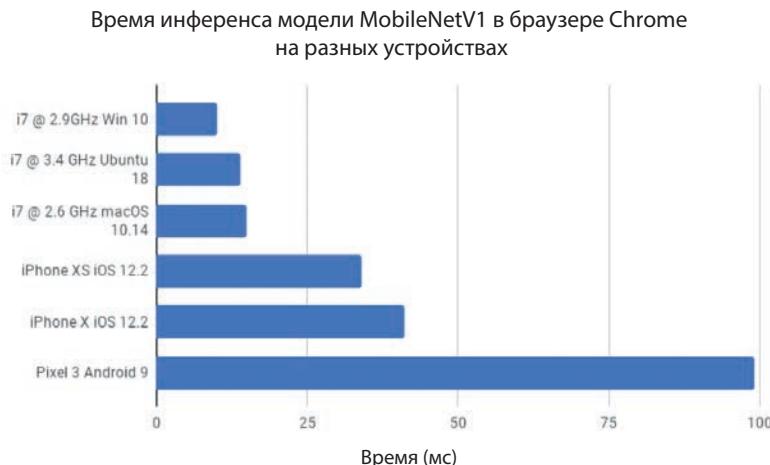
Если конечный продукт не требует мощных возможностей классификации на уровне ImageNet, то попробуйте создать свою модель меньшего размера. Когда в Google создали модель для сочинения музыки в стиле Баха, она имела размер всего 400 Кбайт и загружалась практически мгновенно.

## Время инференса

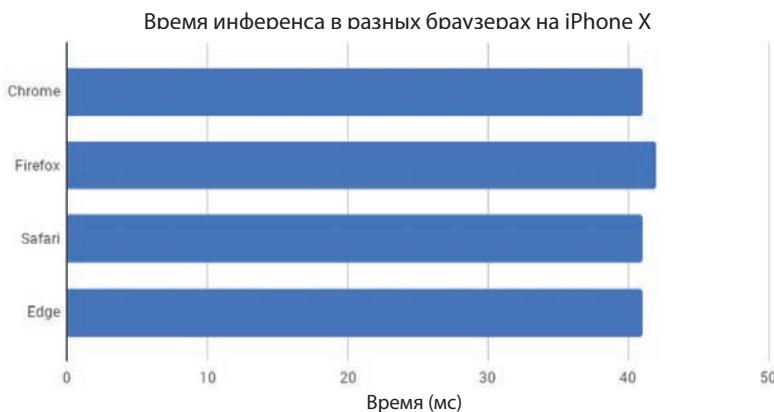
Учитывая, что модель должна работать в браузере, выполняющемся на настольном компьютере или смартфоне, нужно уделить особое внимание пользовательскому интерфейсу, ориентируясь на самое медленное оборудование. В процессе тестирования мы запускали *chapter10/code/benchmark.html* в разных браузерах на разных устройствах. Результаты экспериментов показаны на рис. 10.18.

Как показывают результаты на рис. 10.18, чем быстрее оборудование, тем быстрее работает модель. Судя по этим результатам, производительность устройств Apple с GPU превосходит производительность устройств на Android, хотя такое сравнение не совсем корректно.

Было бы любопытно посмотреть, отличается ли время инференса в разных браузерах на одном и том же устройстве? Мы выбрали для этого эксперимента iPhone X и получили результаты, показанные на рис. 10.19.



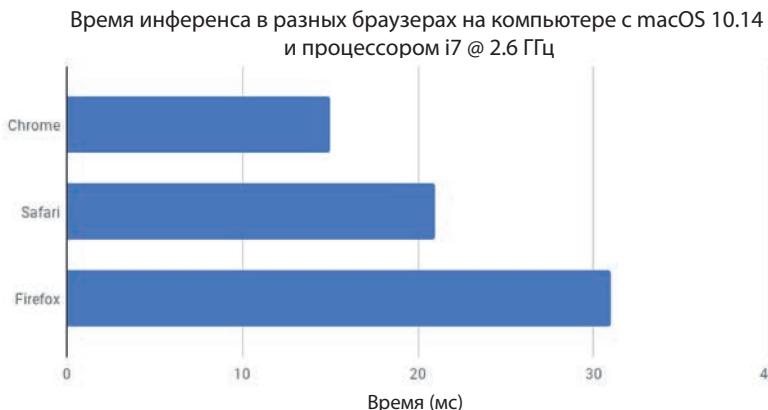
**Рис. 10.18.** Время инференса модели MobileNetV1 в браузере Chrome на разных устройствах



**Рис. 10.19.** Время инференса в разных браузерах на iPhone X

Судя по результатам на рис. 10.19, все браузеры на iPhone обеспечивают примерно одинаковую скорость работы модели. И это неудивительно, потому что все эти браузеры используют встроенный в iPhone компонент управления на основе WebKit под названием WKWebView. А если провести тот же эксперимент на MacBook Pro? Взгляните на рис. 10.20.

Кому-то эти результаты могут показаться неожиданными. В этом примере скорость инференса в Chrome почти вдвое выше, чем в Firefox. Почему? Открыв монитор использования GPU, мы выяснили, что при выполнении модели в Chrome загруженность GPU намного выше, чем при выполнении в Firefox,



**Рис. 10.20.** Время инференса в разных браузерах на компьютере с macOS 10.14 и процессором i7 @ 2.6 ГГц

и немного выше, чем в Safari. Чем интенсивнее используется GPU, тем быстрее инференс. Это означает, что браузеры в зависимости от операционной системы могут по-разному оптимизировать инференс с использованием GPU, что дает разное время выполнения.

Отметим ключевой момент: эти эксперименты проводились на топовых устройствах, которые могут быть не у каждого пользователя. Продолжительность работы модели также важна для потребления заряда батареи. Поэтому мы должны соответствовать ожиданиям пользователей в отношении производительности, особенно при взаимодействии в масштабе реального времени.

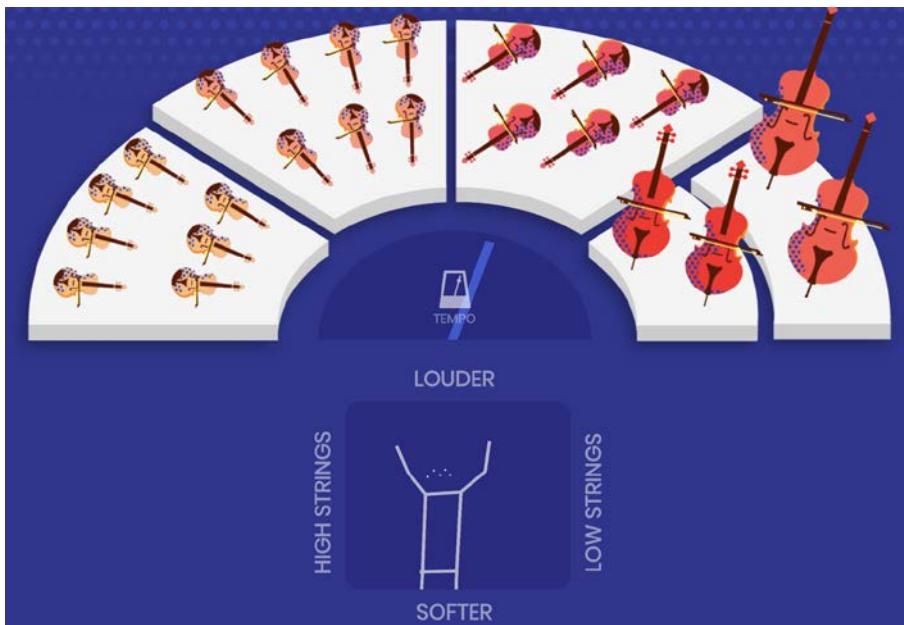
## Примеры

Теперь, познакомившись со всеми ингредиентами, необходимыми для реализации глубокого обучения в браузере, посмотрим, что готовит индустрия.

### Дирижер

Вы когда-нибудь мечтали подирижировать Нью-Йоркским филармоническим оркестром? Веб-приложение Semi-Conductor (<https://oreil.ly/sNOFg>) готово частично осуществить эту мечту. Откройте приложение, встаньте перед веб-камерой, помашите руками и понаблюдайте, как под вашим руководством оркестр исполняет произведение Моцарта «Маленькая ночная серенада». Как вы уже, наверное, догадались, приложение использует модель PoseNet для распознавания движений рук и выбора темпа и громкости исполнения, а также секции

инструментов (рис. 10.21), включая скрипки, альты, виолончели и контрабасы. Приложение, созданное в Google Creative Lab в Сиднее, использует заранее записанное музыкальное произведение, разбитое на крошечные фрагменты, и каждый фрагмент воспроизводится с определенной скоростью и громкостью в зависимости от движений рук. Высота подъема рук управляет громкостью, скорость движения — темпом исполнения. Такое интерактивное управление звучанием стало возможным только потому, что инференс PoseNet работает со скоростью до нескольких кадров в секунду (на обычном ноутбуке).

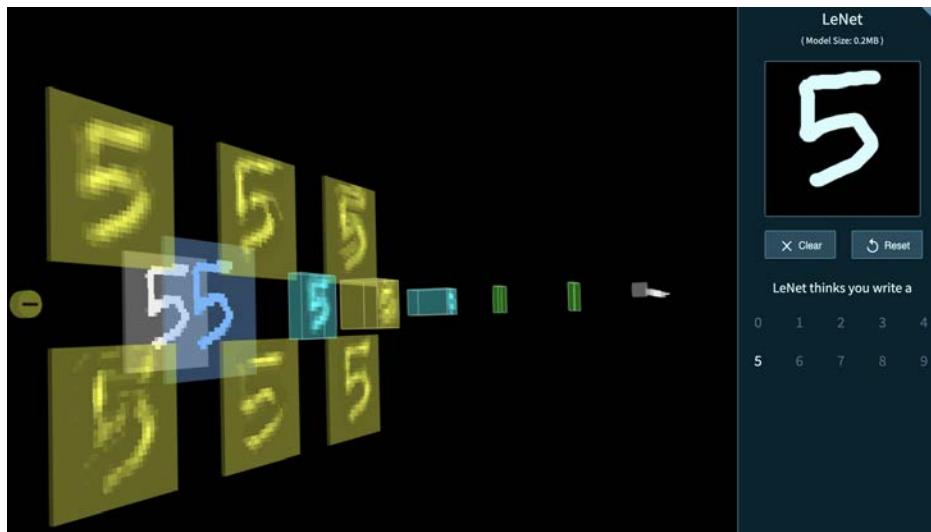


**Рис. 10.21.** Дирижирование оркестром в веб-приложении Semi-Conductor (Louder — громче; Softer — тише; High strings — высокие струнные; Low strings — низкие струнные)

## TensorSpace

Многие сверточные нейронные сети выглядят чересчур запутанными. В них трудно разобраться, и поэтому часто их воспринимают как черные ящики. Как выглядят фильтры? Что их активирует? Почему они выдали тот или иной прогноз? Все это тайна, покрытая мраком. Но как часто бывает со сложными механизмами, наглядная визуализация помогает открыть этот черный ящик и понять его устройство. Для изучения устройства сверточных сетей можно использовать TensorSpace (<https://tensorspace.org/>) — библиотеку, которая «представляет тензоры в пространстве».

Она позволяет загружать модели в трехмерное пространство, изучать их структуру в браузере, масштабировать и поворачивать, вводить исходные данные и смотреть порядок обработки изображения слой за слоем на всем пути до получения окончательного результата. Наконец, фильтры можно открыть и «потрогать» вручную без необходимости устанавливать что-то еще. И, как показано на рис. 10.22, опытные пользователи могут даже отобразить процесс в виртуальной реальности на любом фоне.



**Рис. 10.22.** Визуализация модели LeNet с помощью TensorSpace

## Metacar

Беспилотные автомобили — сложные устройства. А обучение с подкреплением для их подготовки может потребовать много времени, денег и сил (не считая аварий, которые неизбежно будут происходить первое время). А можно ли обучать их в самом браузере? Этую задачу решает приложение Metacar (<https://metacar.scottletcher.guru>), предоставляя искусственное двумерное окружение для обучения игрушечных машинок с помощью методики обучения с подкреплением, и все это в браузере (рис. 10.23).

Как и в видеоиграх, где игрок переходит на все более сложные уровни, Metacar позволяет создать несколько уровней для совершенствования автомобиля. Библиотека TensorFlow.js помогает сделать обучение с подкреплением более доступным (подробнее об этом поговорим в главе 17, когда будем обсуждать создание системы управления для небольшого беспилотного автомобиля).



**Рис. 10.23.** Окружение Metacar для обучения моделей с использованием методики обучения с подкреплением

## Классификация фотографий в Airbnb

Airbnb — компания по аренде недвижимости. Она требует от арендодателей и арендаторов выгружать фотографии в профилях. К сожалению, некоторые люди выбирают фотографии с водительского удостоверения или паспорта, переснимая их на телефон, из-за чего в кадр попадает часть документа. Учитывая конфиденциальный характер информации, Airbnb использует нейронную сеть, реализованную с использованием TensorFlow.js, для обнаружения фотографий с конфиденциальной информацией и предотвращения их выгрузки на сервер.

## GAN Lab

Как и TensorFlow Playground (<https://oreil.ly/vTrptu>, инструмент визуализации нейронной сети в браузере), GAN Lab (<https://oreil.ly/aQgga>, рис. 10.24) представляет собой удобный инструмент для визуализации и изучения генеративно-состязательных сетей, использующий TensorFlow.js. Визуализация GAN — сложный процесс, поэтому для его упрощения GAN Lab пытается изучить простые распределения и визуализирует выходные данные генератора и дискриминатора. Например, ис-

тинное распределение точек может иметь форму круга в двумерном пространстве. Генератор начинает со случайного гауссова распределения и постепенно пытается приблизиться к истинному распределению. Этот проект является результатом сотрудничества Georgia Tech и Google Brain/PAIR (People + AI Research).

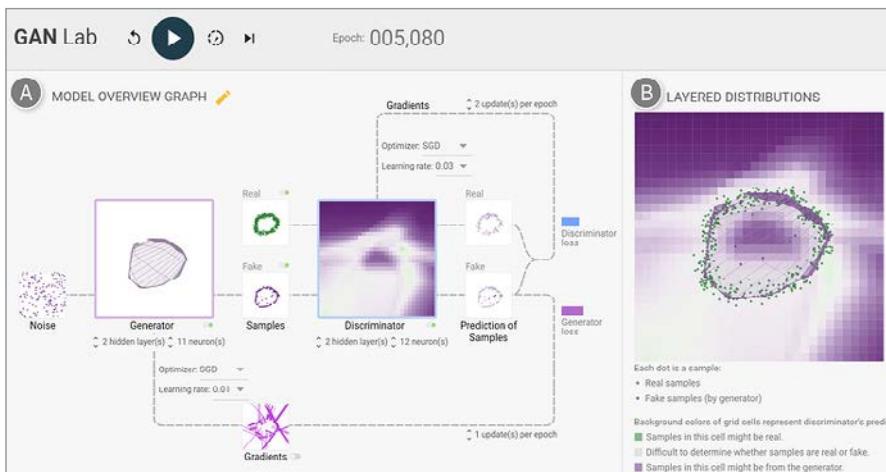


Рис. 10.24. Скриншот GAN Lab

## Итоги

Мы познакомились с историей развития библиотек глубокого обучения на JavaScript и выбрали TensorFlow.js для более близкого знакомства. Изучили предварительно обученные модели, обрабатывающие ввод с веб-камеры в масштабе реального времени, а затем узнали, что модели можно обучать даже в браузере. Такие инструменты, как профилировщик Chrome, помогают получить представление об использовании GPU. Затем мы познакомились с библиотекой ml5.js, которая еще больше упрощает разработку и позволяет создавать демонстрации применения PoseNet и pix2pix, написав всего несколько строк кода. Наконец, мы оценили производительность этих моделей и библиотек и перечислили несколько интересных примеров из практики.

Одним из огромных преимуществ поддержки нейронных сетей в браузере является широкий охват аудитории по сравнению с любой другой платформой. Их дополнительное преимущество в том, что пользователю не нужно устанавливать сторонние приложения. Браузеры также обеспечивают платформу для быстрого прототипирования, позволяющую оперативно проверить идею, прежде чем вкладывать время и деньги в создание полноценного приложения. Библиотеки TensorFlow.js и ml5.js ускорили процесс внедрения ИИ в браузеры и сделали его доступным широким массам.

## ГЛАВА 11

---

# Классификация объектов в реальном времени в iOS с Core ML

До сих пор мы наблюдали за работой наших моделей глубокого обучения на десктопных компьютерах в облаке и в браузере. Такие конфигурации имеют свои преимущества, но подходят не для всех сценариев. В этой главе узнаем, как организовать вычисление прогнозов с помощью моделей глубокого обучения на мобильных устройствах.

Перенос вычислений на устройство пользователя выгодно по многим причинам:

### *Задержка и интерактивность*

Отправка изображения, его обработка в облаке и возврат результата могут занять несколько секунд, в зависимости от качества сети и объема передаваемых данных. Это путь к плохому UX и негативному пользовательскому опыту. Десятки лет исследований в области UX, в том числе выводы Якоба Нильсена (Jakob Nielsen), опубликованные в 1993 году в книге «Usability Engineering», показали следующее:

- 0,1 секунды — это примерный предел, когда у пользователя создается ощущение, что система реагирует мгновенно;
- 1 секунда — это предел, чтобы поток мыслей пользователя не прерывался;
- 10 секунд — это предел удержания внимания пользователя.

Примерно два десятилетия спустя Google опубликовала данные, согласно которым половина пользователей мобильных браузеров покидает страницу, если она загружается дольше трех секунд. Но забудьте о трех секундах, увеличение задержки отклика сайта Amazon всего на 100 мс приводит к снижению объема продаж на 1%. Это большой упущеный доход. Устранение задержки за счет мгновенной обработки на устройстве пользователя может обеспечить отзывчивость и интерактивность UI. Запуск моделей глубокого обучения

в режиме реального времени, как, например, Snapchat Lenses, может повысить вовлеченность пользователей.

### *Доступность 24/7 и снижение затрат на облачные услуги*

Чем меньше данных отправляется в облако, тем меньше денег разработчики тратят на обслуживание и тем больше экономия. При этом снижаются затраты на масштабирование, когда приложение становится популярным и пользователей становится больше. Вычисления на самом устройстве несут выгоду и самим пользователям, потому что у них меньше причин для беспокойства о тарифном плане. Кроме того, поддержка вычислений на локальном устройстве означает, что они доступны 24/7 независимо от наличия интернет-соединения.

### *Конфиденциальность*

Локальные вычисления обеспечивают конфиденциальность пользовательских данных, потому что данные, которые потенциально могут использоваться для получения информации о пользователе, никуда не передаются и избавляют разработчиков от лишних забот по обеспечению защиты персональной информации. В связи с принятием в ЕС общего регламента по защите данных (General Data Protection Regulation, GDPR) и законов о защите персональных данных в других странах этот аспект становится важен.

Надеемся, все это убедительно доказывает важность переноса моделей ИИ на мобильные устройства. Тем, кто создает какие-либо серьезные приложения, нужно подумать:

- Как преобразовать модель для работы на смартфоне?
- Сможет ли модель работать на других платформах?
- Как обеспечить быструю работу модели?
- Как уменьшить размер приложения?
- Как гарантировать невысокое потребление приложением заряда аккумулятора?
- Как обновить модель, минуя примерно двухдневный процесс проверки приложения?
- Как проводить А/В-тестирование моделей?
- Можно ли обучить модель на устройстве?
- Как защитить интеллектуальную собственность (например, модель) от кражи?

В следующих трех главах мы расскажем, как запускать алгоритмы глубокого обучения на смартфонах с помощью различных фреймворков, и ответим на все эти вопросы по мере их возникновения.

В этой главе углубимся в мир мобильного ИИ на устройствах iOS. Сначала рассмотрим общий жизненный цикл ПО (показанный на рис. 11.1) и то, как сочетаются его различные части. Затем изучим экосистему Core ML, ее историю и возможности. Потом развернем приложение для классификации объектов в масштабе реального времени на устройстве iOS и рассмотрим приемы оптимизации производительности. И наконец, проанализируем некоторые реальные приложения, построенные на Core ML.

Взглянем на картину в целом.

## Жизненный цикл разработки искусственного интеллекта для мобильных устройств

На рис. 11.1 изображен типичный жизненный цикл разработки ИИ для мобильных устройств.



**Рис. 11.1.** Жизненный цикл разработки ИИ для мобильных устройств

Рассмотрим все этапы.

1. *Сбор данных*: данные должны отражать контекст, в котором будет использоваться приложение. Снимки, сделанные реальными пользователями на

смартфоны, как правило, лучше подходят для обучения, чем снимки профессиональных фотографов. У нас может не быть этих данных в начале разработки, но мы можем постепенно их накапливать по мере использования приложения. Во многих случаях хорошей отправной точкой является загрузка изображений из поисковых систем.

2. *Разметка данных*: образцы данных должны сопровождаться метками (названиями категорий), которые должна прогнозировать модель. Качественная (то есть правильная) разметка — непременное условие для получения хорошей модели.
3. *Обучение модели*: на этом этапе фактически создается нейронная сеть, обладающая максимально возможной точностью, с использованием имеющихся данных и меток.
4. *Преобразование модели*: модель экспортируется из среды обучения в среду, совместимую с мобильными устройствами.
5. *Оптимизация производительности*: из-за ограниченности ресурсов мобильных устройств крайне важно сделать модель максимально эффективной с точки зрения использования памяти, заряда аккумулятора и потребления процессора.
6. *Развертывание*: модель добавляется в приложение и передается пользователям.
7. *Мониторинг*: на этом этапе мы наблюдаем, как приложение используется в реальном мире, и ищем пути его совершенствования. Мы также собираем образцы данных у пользователей, давших согласие на участие в этом жизненном цикле, а затем возвращаемся к шагу 1.

В первой части книги мы рассматривали первые три этапа и немного поговорили об улучшении производительности. В этой главе сосредоточимся на этапах 4, 5 и 6. В следующих главах рассмотрим все эти этапы в контексте разработки мобильных приложений.



Перед тем как передать приложение в руки пользователей (которые могут его возненавидеть, если приложение работает не так, как они ожидали), принято оценивать его качество, выпуская версию для внутреннего использования. Как говорится: ешьте сами свою еду. Этот этап предполагает наличие внутреннего круга лояльных пользователей, которые тестируют первые версии и выявляют ошибки еще до того, как приложение будет опубликовано. Для приложений ИИ этот внутренний круг может также стать источником первых данных и позволит оценить успех модели ИИ в реальном мире. Круг внутренних пользователей можно постепенно расширять и затем, наконец, развернуть приложение для всех желающих.

Итак, начнем!

## Краткая история Core ML

Фреймворк Core ML — один из простейших способов запустить глубокую нейронную сеть на устройстве Apple — iPhone, iPad, MacBook, Apple TV и Apple Watch. Помимо простоты использования он также оптимизирован для работы на базовой аппаратной архитектуре. За последние несколько лет появились более удачные альтернативные фреймворки, но еще ни одному из них не удалось превзойти простоту и производительность Core ML.

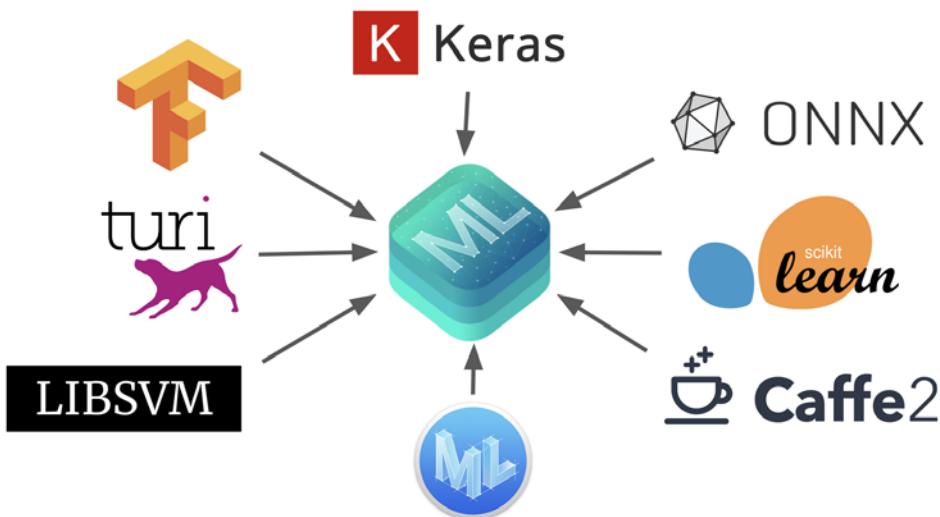
Традиционно, чтобы запустить сверточную сеть на устройстве Apple, разработчикам *приходилось* писать код, использующий Metal — библиотеку, позволявшую разработчикам игр оптимально использовать GPU. К сожалению, разработка с Metal была сродни написанию кода на ассемблере или CUDA для NVIDIA GPU. Это было утомительно, а код было трудно отладить. Мало кто решался пойти по этому пути. Набор инструментов DeepLearningKit, созданный (в декабре 2015 г.) Амундом Твейтом (Amund Tveit), стал одной из попыток создать абстракцию над Metal для развертывания сверточной сети (CNN).

На Всемирной конференции разработчиков Apple (Worldwide Developers Conference, WWDC) в 2016 году компания анонсировала выход фреймворка Metal Performance Shaders (MPS), созданного на основе Metal, — высокопроизводительной библиотеки для оптимизации графики и некоторых вычислительных операций. Он скрывал множество низкоуровневых деталей, давая базовые строительные блоки: Convolution (свертку), Pooling (объединение) и ReLU (функции активации). Это позволило разработчикам писать глубокие нейронные сети, комбинируя операции с блоками в коде. Для пришедших из мира Keras это было знакомой и довольно простой задачей. К сожалению, при использовании MPS требовалось писать массу шаблонного кода (boilerplate code), так как нужно было вручную контролировать входные и выходные параметры на каждом этапе процесса. Например, фрагмент кода от Apple, использующий модель InceptionV3 для классификации объектов по 1000 категорий, был длиной более 2000 строк, причем большую его часть составляло определение сети. А теперь представьте, что нужно немного изменить модель, — в этом случае придется перелопатить все 2000 строк, чтобы внести изменения в код для iOS. В апреле 2017-го Маттийс Холлеманс (Matthijs Hollemans) выпустил библиотеку Forge с целью упростить разработку с использованием MPS за счет уменьшения объема шаблонного кода, необходимого для запуска модели.

Все эти трудности исчезли с появлением фреймворка Core ML, выход которого был анонсирован компанией Apple на конференции WWDC в 2017 году. Он включал механизм вычисления прогнозов на iOS и библиотеку с открытым исходным кодом для Python под названием Core ML Tools, обеспечивающую возможность сериализации моделей CNN, созданных с использованием других фреймворков, таких как Keras и Caffe. В общем случае процесс создания при-

ложения включал следующие этапы: обучение модели с использованием других библиотек, преобразование ее в файл *.mlmodel* и развертывание в приложении iOS, работающем на платформе Core ML.

Фреймворк Core ML поддерживает импорт широкого спектра моделей машинного обучения, созданных с использованием собственных и сторонних фреймворков и форматов файлов. На рис. 11.2 показаны некоторые из них (по часовой стрелке, начиная с верхнего левого угла): TensorFlow, Keras, ONNX, scikit-learn, Caffe2, Apple Create ML, LIBSVM и TuriCreate (также создан в Apple). ONNX, в свою очередь, тоже поддерживает большое разнообразие фреймворков, включая PyTorch (Facebook), MXNet (Amazon), Cognitive Toolkit (Microsoft), PaddlePaddle (Baidu) и др., тем самым обеспечивая совместимость с любым широко известным фреймворком.



**Рис. 11.2.** Фреймворки, совместимые с Core ML по форматам моделей (по состоянию на 2019 год)

## Альтернативы фреймворку Core ML

Есть несколько вариантов получения прогнозов в масштабе реального времени. В том числе универсальные фреймворки: Core ML (Apple), TensorFlow Lite (Google), ML Kit (Google) и Fritz, а также фреймворки, специализированные для конкретных аппаратных окружений, включая Snapdragon Neural Processing Engine (Qualcomm) и мобильную вычислительную платформу Huawei AI Mobile Computing Platform (для нейронного процессора Huawei Neural Processing Unit). В табл. 11.1 приводятся сравнительные характеристики фреймворков.

**Таблица 11.1.** Сравнительные характеристики фреймворков ИИ для мобильных устройств

Фреймворк	Доступность для iOS	Доступность для Android	Динамическое обновление	А/В-тестирование	Обучение на устройстве	Шифрование модели
Core ML	✓	–	✓	–	✓	–
TensorFlow Lite	✓	✓	–	–	В версиях, вышедших в конце 2019 года	–
ML Kit	✓	✓	✓	✓	–	–
Fritz	✓	✓	✓	✓	–	✓

## TensorFlow Lite

В ноябре 2017 года Google анонсировала выход инструмента для инференса на мобильных устройствах TensorFlow Lite, цель которого заключалась в расширении экосистемы TensorFlow за пределы серверов и персональных компьютеров. До этого в экосистеме TensorFlow доступные варианты ограничивались полной версией библиотеки TensorFlow для iOS (которая была тяжеловесной и медленной) и слегка урезанной версией под названием TensorFlow Mobile (которая появилась чуть позже и все еще была довольно громоздкой).

Механизм TensorFlow Lite был переписан с нуля специально для мобильных и краевых (edge) устройств и оптимизирован для увеличения скорости работы, уменьшения размеров моделей и интерпретатора, а также энергопотребления. В нем появилась поддержка делегатов для работы с GPU, а это означает, что при наличии GPU TensorFlow Lite может использовать его для ускорения вычислений. В iOS в роли такого делегата GPU используется Metal. Более подробно мы обсудим TensorFlow Lite в главе 13.

## ML Kit

ML Kit — это высокоуровневая библиотека от Google, которая предлагает множество функций для реализации моделей компьютерного зрения, обработки естественного языка и ИИ без настройки, включая возможность использования моделей TensorFlow Lite. В числе поддерживаемых возможностей — нахождение лиц, сканирование штрих-кодов, интеллектуальный выбор ответов, перевод с одного языка на другой на самом устройстве и определение языка. Но главное преимущество ML Kit — это интеграция со службой Google Firebase, которая поддерживает динамическое обновление моделей, А/В-тестирование и удален-

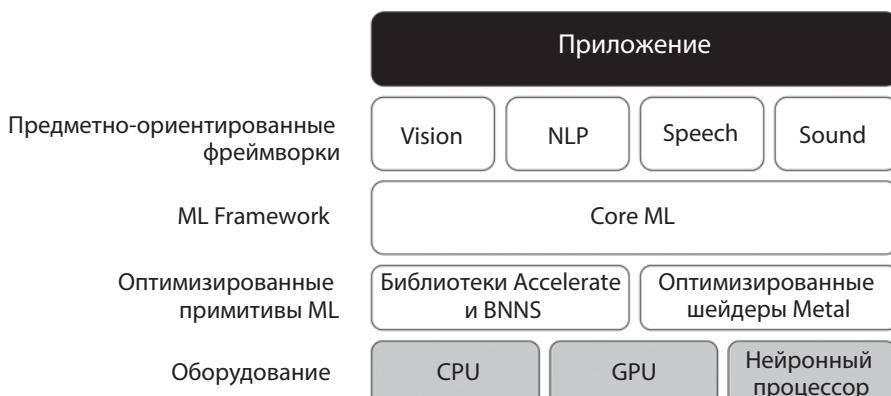
ный выбор динамической модели на основе конфигурации (так замысловато называется функция выбора используемой модели в зависимости от клиента). Более подробно об ML Kit в главе 13.

## Fritz

Fritz — это стартап, основанный для упрощения процесса прогнозирования на мобильных устройствах. Этот фреймворк сокращает разрыв между практиками и разработчиками МО, предоставляя простые в использовании инструменты командной строки. С одной стороны, он интегрирует обучение с использованием Keras непосредственно в пайплайн развертывания, благодаря чему разработчик МО может добавить одну строку с обратным вызовом Keras, чтобы развернуть модель для пользователей сразу после завершения обучения. С другой стороны, разработчик мобильных приложений может протестировать модель без развертывания на физическом устройстве, смоделировав ее работу в виртуальном окружении, оценить совместимость модели Keras с Core ML и получить аналитику для каждой модели. Одно из уникальных преимуществ Fritz — функция защиты модели методом обfuscации, которая в случае взлома телефона запутывает исходный код и не позволяет его изучить.

## Архитектура машинного обучения Apple

Чтобы лучше понять экосистему Core ML, рассмотрим в общих чертах все API, которые предлагает Apple, а также их соответствие друг другу. На рис. 11.3 показаны компоненты архитектуры машинного обучения Apple.



**Рис. 11.3.** Различные API, предлагаемые компанией Apple для разработчиков приложений

## Предметно-ориентированные фреймворки

Чтобы упростить решение наиболее типичных задач машинного обучения без необходимости обладать экспертными знаниями в предметной области, Apple предлагает множество готовых API в таких областях, как компьютерное зрение, обработка естественного языка, распознавание речи и анализ звуков. Функции, доступные в операционных системах Apple, перечислены в табл. 11.2.

**Таблица 11.2.** Функции, доступные в операционных системах Apple

Vision	NLP	Другие
Определение характерных точек на лицах	Лексемизация	Распознавание речи (на устройствах и с помощью облачных служб)
Определение сходства изображений	Определение языка	Классификация звуков
Определение объектов внимания	Определение частей речи	
Оптическое распознавание символов		
Выделение прямоугольных областей		
Нахождение лиц		
Классификация объектов		
Чтение штрих-кодов		
Определение горизонта		
Распознавание людей и животных		
Трассировка объектов (на видео)		

## ML Framework

Core ML поддерживает возможность инференса с использованием моделей глубокого обучения и машинного обучения.

## Оптимизированные примитивы ML

Вот некоторые примитивы машинного обучения, которые можно найти в стеке Apple:

*Оптимизированные шейдеры Metal (Metal Performance Shaders, MPS)*

Представляет низкоуровневые и высокопроизводительные примитивы, использующие GPU для ускорения вычислений в большинстве сетей CNN.

Если Core ML не поддерживает какую-то модель, MPS предоставляет все нужное для реализации такой поддержки. Кроме того, MPS можно использовать для улучшения модели вручную, чтобы добиться более высокой производительности (например, гарантировать выполнение вычислений на GPU).

### *Accelerate и Basic Neural Network Subroutine (BNNS)*

Accelerate – это реализация библиотеки Apple Basic Linear Algebra Subprogram (BLAS). Она обеспечивает основу для высокопроизводительных крупномасштабных математических вычислений и обработки изображений, как и библиотека Basic Neural Network Subroutine (BNNS), которая помогает реализовать и запускать нейронные сети.

Теперь, после знакомства с общей архитектурой фреймворка Core ML и составляющими его предметно-ориентированными API, посмотрим, что нужно для запуска модели в приложении iOS, которая использует Core ML и Vision.



Apple предлагает несколько загружаемых моделей (рис. 11.4) для различных задач компьютерного зрения, от классификации до обнаружения объектов (с ограничивающими рамками), сегментации (идентификации пикселей), оценки глубины и т. д. Они доступны по адресу <https://developer.apple.com/machine-learning/models>.

На сайте Apple Machine Learning есть множество предварительно обученных моделей Core ML для классификации, включая MobileNet, SqueezeNet, ResNet-50 и VGG16.

## Приложение для распознавания объектов в реальном времени

Мы не собираемся учить вас разрабатывать приложения для iOS, а хотим показать, как запустить модель распознавания объектов, которая делает выбор из 1000 категорий в датасете ImageNet в режиме реального времени.

Рассмотрим минимальный объем кода, необходимый для запуска этого приложения. Вот общая схема действий:

1. Перетащите файл *.mlmodel* в проект Xcode.
2. Загрузите модель в контейнер Vision (*VNCoreMLModel*).
3. Создайте запрос (*VNCoreMLRequest*) на основе этого контейнера и укажите функцию, которую следует вызвать после обработки запроса.
4. Создайте обработчик запроса, принимающий указанное изображение.
5. Обработайте запрос и выведите результаты.

**Machine Learning**

Overview Core ML Create ML Models Resources

Images Text



**Resnet50**  
Image Classification

A Residual Neural Network that will classify the dominant object in a camera frame or image.

[View models](#)



**SqueezeNet**  
Image Classification

A small Deep Neural Network architecture that classifies the dominant object in a camera frame or image.

[View models](#)



**DeeplabV3**  
Image Segmentation

Segment the pixels of a camera frame or image into a predefined set of classes.

[View models](#)



**YOLOv3**  
Object Detection

Locate and classify 80 different types of objects present in a camera frame or image.

[View models](#)



**YOLOv3-Tiny**  
(Real-time) Object Detection

Locate and classify 80 different types of objects present in a camera frame or image.

[View models](#)

**Рис. 11.4.** Готовые к использованию модели на сайте Apple Machine Learning

Вот как эта схема выглядит в коде:

```
import CoreML
import Vision

// загрузить модель
let model = try? VNCoreMLModel(for: Resnet50().model)!
// создать запрос с функцией обратного вызова
let classificationRequest = VNCoreMLRequest(model: model) {
    (request, error) in
    // вывести результаты по завершении обработки запроса
    if let observations = request.results as? [VNClassificationObservation] {
        let results = observations
            .map{"\"$0.identifier) - \"\$0.confidence\""}
            .joined(separator: "\n")
        print(results)
    }
}
```

```
// создать обработчик запроса, принимающий аргумент с изображением
let requestHandler = VNImageRequestHandler(cgImage: cgImage)
// обработать запрос
try? requestHandler.perform([classificationRequest])
```

В этом коде `cgImage` может быть любым изображением. Это может быть фотография из фотоальбома или из интернета.

Изображения также можно получать в реальном времени из видеопотока камеры и передавать отдельные кадры в эту функцию. Камера в iPhone может снимать до 60 кадров в секунду.

По умолчанию Core ML обрезает изображение по длинному краю. Иначе говоря, если модель требует ввода квадратных изображений, то Core ML извлечет самый большой квадрат в центре изображения. Это может вызывать путаницу у разработчиков, которые обнаружили, что верхняя и нижняя кромки изображения игнорируются при прогнозировании. В зависимости от целей можно использовать параметры `.centerCrop`, `.scaleFit` или `.scaleFill`, как показано на рис. 11.5, например:

```
classificationRequest.imageCropAndScaleOption = .scaleFill
```

После знакомства с кодом хотелось бы увидеть что-нибудь поинтереснее. Как насчет запуска кода на телефоне? Мы создали полноценное приложение и сделали его доступным в репозитории книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке `code/chapter-11`. Владельцы iPhone или iPad могут развернуть его и поэкспериментировать, даже ничего не зная о разработке приложений на iOS. Вот что нужно сделать (примечание: для этого нужен компьютер Mac):

1. Загрузите Xcode с сайта разработчиков Apple или из Mac App Store.
2. Подключите устройство iOS к компьютеру. В процессе развертывания телефон должен оставаться разблокированным.
3. Перейдите в каталог *CameraApp*:

```
$ cd code/chapter-11/CameraApp
```

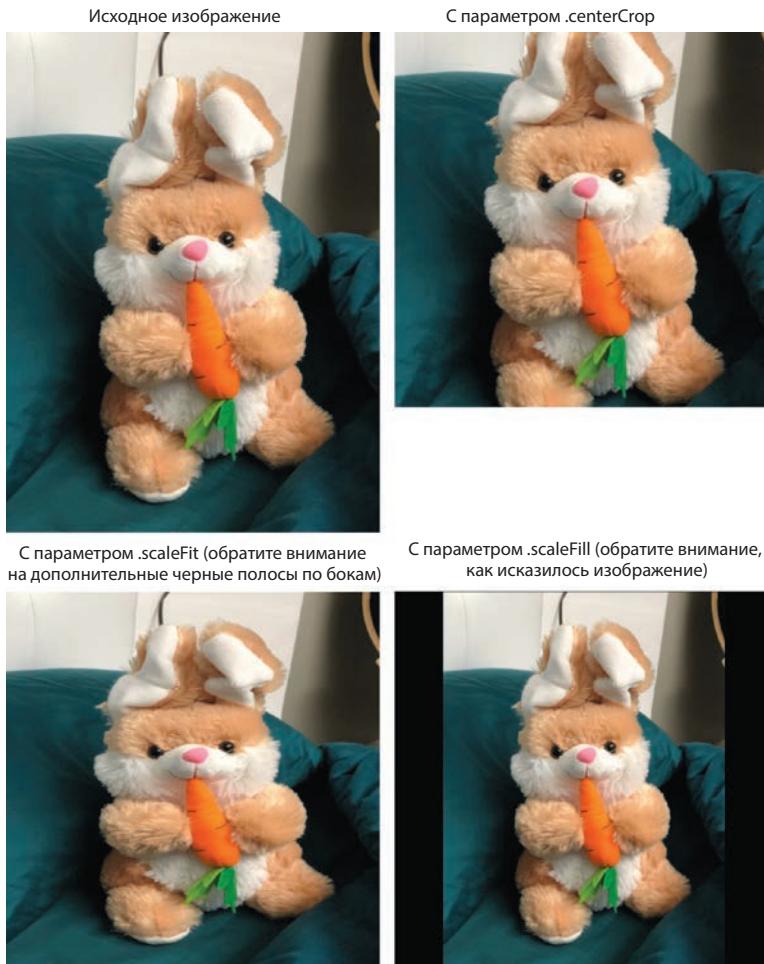
4. Загрузите модели Core ML с веб-сайта Apple с помощью сценария на Bash:

```
$ ./download-coreml-models.sh
```

5. Откройте проект Xcode:

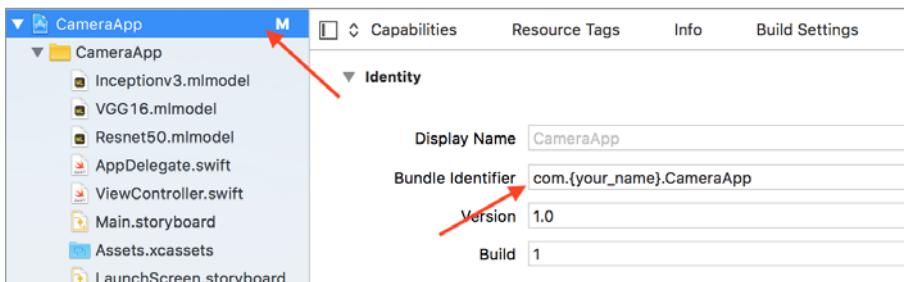
```
$ open CameraApp.xcodeproj
```

6. В Project Hierarchy Navigator (Навигатор иерархии проектов) щелкните на проекте *CameraApp* в верхнем левом углу, как показано на рис. 11.6, чтобы открыть представление Project Information (Информация о проекте).

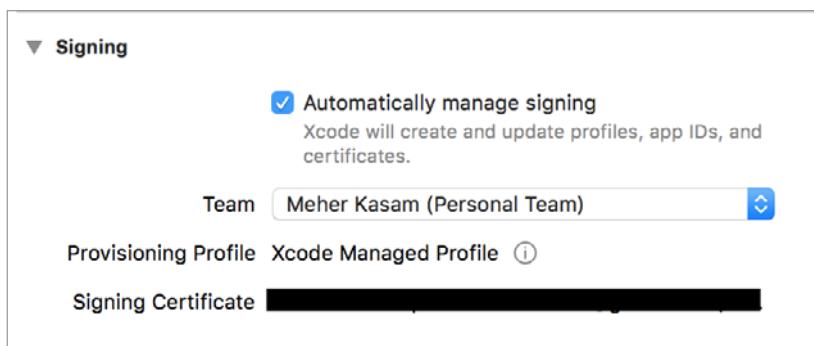


**Рис. 11.5.** Как различные параметры масштабирования изменяют изображение, подаваемое на вход моделей Core ML

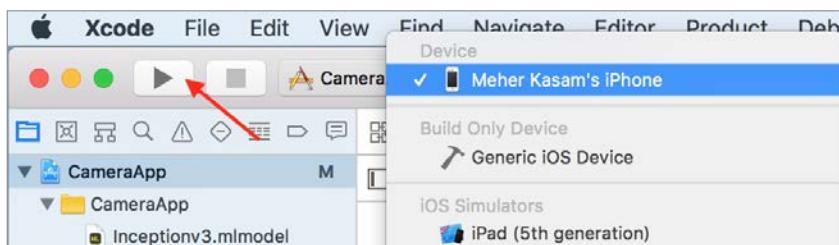
7. Поскольку Xcode резервирует уникальный идентификатор пакета, используйте уникальное имя для идентификации проекта.
8. Войдите в учетную запись Apple, чтобы Xcode подписал приложение и развернул его на устройстве. Выберите название команды для подписи, как показано на рис. 11.7.
9. Щелкните на кнопке **Build and Run** (Собрать и запустить; треугольник, указывающий вправо), чтобы развернуть приложение на устройстве, как показано на рис. 11.8. Обычно это занимает от 30 до 60 секунд.



**Рис. 11.6.** Информация о проекте в Xcode



**Рис. 11.7.** Выберите название команды и позвольте Xcode автоматически управлять подписью исполняемого кода



**Рис. 11.8.** Выберите устройство и щелкните на кнопке Build and Run (Собрать и запустить), чтобы развернуть приложение

10. Устройство не запустит приложение сразу, потому что не доверяет ему. На устройстве перейдите в **Settings > General > Profiles and Device Management** (Настройки > Основные > Профили и управление устройствами) и выберите строку с вашей информацией, а затем коснитесь ссылки **Trust {ваш\_идентификатор\_учетной\_записи}** (Доверять {ваш\_идентификатор\_учетной\_записи}), как показано на рис. 11.9.



**Рис. 11.9.** Экран Profiles and Device Management  
(Профили и управление устройствами)



**Рис. 11.10.** Скриншот приложения

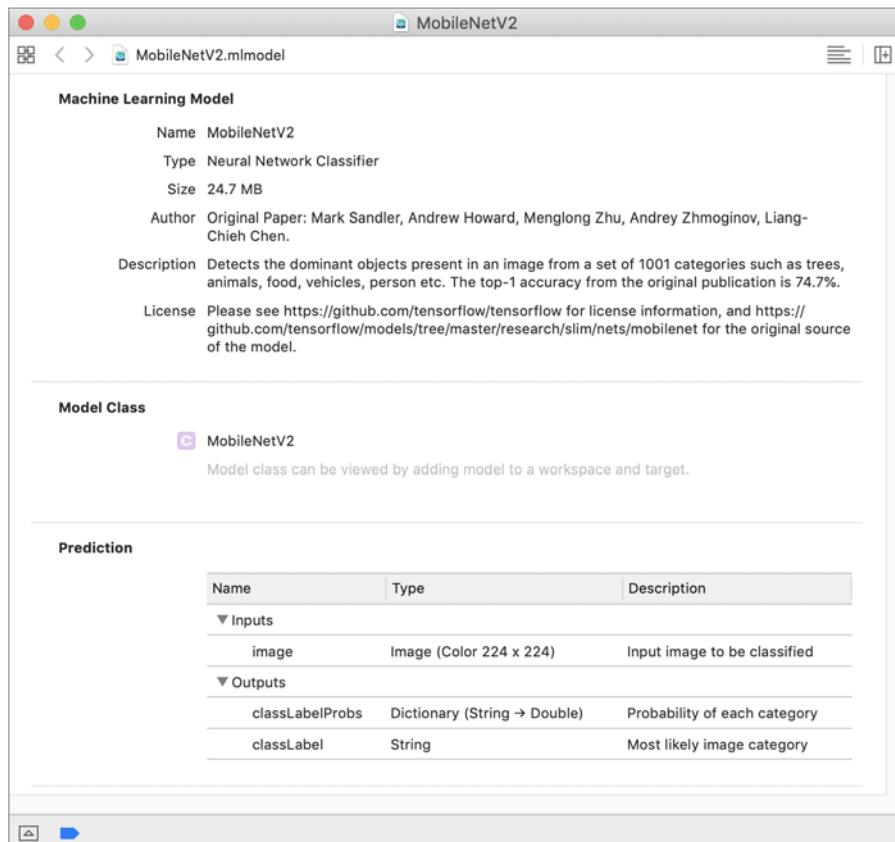
11. На главном экране найдите приложение CameraApp и запустите его.

Из скриншота видно, что приложение классифицировало изображение как «notebook, notebook computer» (ноутбук, портативный компьютер) с достоверностью 84 % и как «notebook, laptop computer» (ноутбук, дорожный компьютер) с достоверностью 11 %.

Надеемся, вам понравилось! А теперь перейдем к более серьезному делу: преобразованию моделей, созданных в разных фреймворках, в модели Core ML.



Xcode обладает замечательной особенностью: отображает входные и выходные параметры модели при загрузке файла `.mlmodel` в Xcode, как показано на рис. 11.11. Это очень полезно, когда мы не собираемся обучать модель самостоятельно и не хотим писать код (например, `model.summary()`) для изучения архитектуры модели.



**Рис. 11.11.** Сведения о входных и выходных параметрах модели MobileNetV2 в инспекторе моделей в Xcode

## Конвертация моделей в формат Core ML

В коде предыдущего примера был файл *Inceptionv3.mlmodel*. Вы задумывались над тем, как он появился? Модель Inception была обучена в Google с помощью TensorFlow. Затем получившийся файл *.pb* был преобразован в формат Core ML. Точно так же нам может понадобиться преобразовать модели в формате Keras, Caffe или любом другом в формат Core ML. Ниже даны несколько инструментов для конвертации моделей в формат Core ML.

- Core ML Tools (Apple): из форматов Keras (*.h5*), Caffe (*.caffemodel*), а также библиотек LIBSVM, scikit-learn и XGBoost;
- tf-coreml (Google): из формата TensorFlow (*.pb*);
- onnx-coreml (ONNX): из формата ONNX (*.onnx*).

Рассмотрим подробнее первые два инструмента.

### Конвертация из формата Keras

Core ML Tools помогает преобразовать модели из форматов Keras, ONNX и других в формат Core ML (*.mlmodel*). Установите фреймворк *coremltools* с помощью *pip*:

```
$ pip install --upgrade coremltools
```

Теперь посмотрим, как можно конвертировать модель Keras в формат Core ML. Конвертация — это всего лишь односторонняя команда, после выполнения которой преобразованную модель можно сохранить, как показано ниже:

```
from tensorflow.keras.applications.resnet50 import ResNet50
model = ResNet50()

import coremltools
coreml_model = coremltools.converters.keras.convert(model)
coreml_model.save("resnet50.mlmodel")
```

Вот и все! Проще простого. В главе 12 мы обсудим порядок конвертации моделей с неподдерживаемыми слоями (таких, как MobileNet).

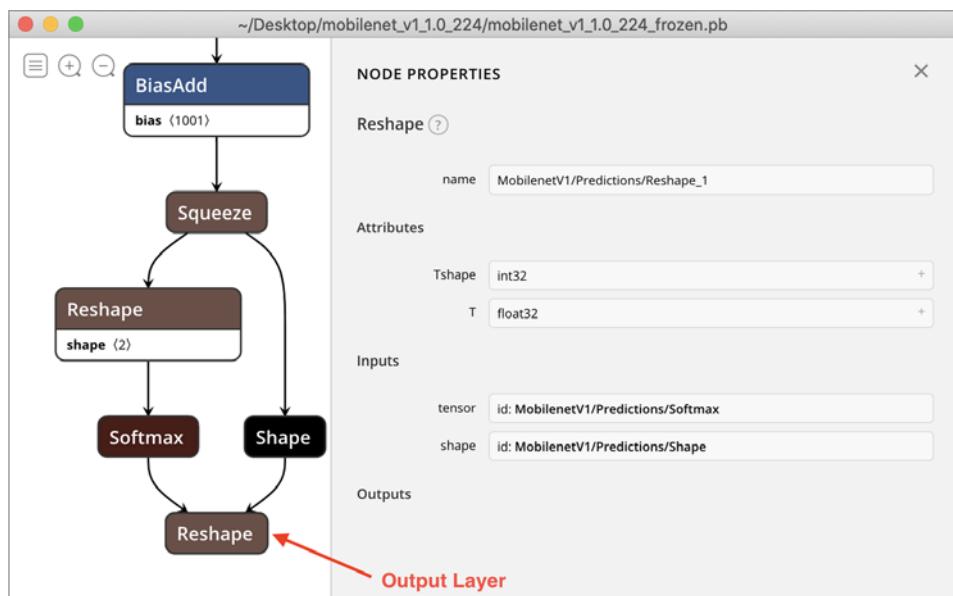
### Конвертация из формата TensorFlow

Для конвертации моделей TensorFlow компания Apple рекомендует использовать пакет *tf-coreml* (созданный в Google). Далее мы преобразуем предварительно обученную модель TensorFlow в формат Core ML. Этот процесс немного сложнее и требует написать больше строк кода в отличие от предыдущего примера.

Сначала установим `tfcoreml` с помощью `pip`:

```
$ pip install tfcoreml --user --upgrade
```

Для преобразования нужно узнать параметры первого и последнего слоев модели. Сделать это можно с помощью инструмента визуализации архитектуры модели Netron (<https://oreil.ly/hJoly>). После загрузки модели MobileNet (файла `.pb`) в Netron можно увидеть устройство модели в графическом интерфейсе. На рис. 11.12 показана небольшая часть модели MobileNet — ее выходной слой.



**Рис. 11.12.** Выходной слой модели MobileNet в интерфейсе Netron

Теперь вставим описание слоя в код на Python и запустим его:

```
import tfcoreml as tf_converter
tf_converter.convert(tf_model_path = "input_model.pb",
    mlmodel_path = "output_model.mlmodel",
    output_feature_names = ["MobilenetV1/Predictions/Reshape_1:0"])
```

В процессе выполнения сценария вы увидите, как каждая операция в каждом слое преобразуется в соответствующие эквиваленты Core ML. По окончании в рабочем каталоге появится конвертированная модель `.mlmodel`.

Теперь модель Core ML готова. Перетащите ее в Xcode для проверки.

## Развертывание динамической модели

Как разработчики мы, вероятно, захотим и дальше совершенствовать свою модель и как можно скорее открыть доступ к ее последней версии для своих пользователей. Один из способов — отправлять обновления в App Store каждый раз, когда мы решим развернуть новую модель. Но это не лучший подход, потому что каждый раз придется ждать пару дней, пока Apple утвердит обновление, а это долго.

Другое решение — организовать в приложении динамическую загрузку файла *.mlmodel* и его компиляцию на устройстве пользователя. Есть несколько причин, оправдывающих такой подход:

- Возможность регулярно обновлять модель, независимо от частоты выпуска новых версий в App Store.
- Уменьшение размера загружаемого приложения. Пользователям потребуется загрузить только нужные им модели. Это уменьшит объем пространства, занимаемого данными в хранилище, и затраты на их загрузку. Apple требует, чтобы объем данных, загружаемых из App Store через мобильные сети, не превышал 200 Мбайт, и важно следить за этим лимитом, чтобы не потерять потенциальных пользователей.
- Возможность А/В-тестирования моделей на разных группах пользователей.
- Возможность использовать разные модели для разных групп пользователей, стран и регионов.

Реализовать это довольно просто: разместите *.mlmodel* на сервере и добавьте в приложение динамическую загрузку файла. После того как модель загрузится на устройство пользователя, ее можно скомпилировать вызовом `MLModel.compileModel` и получить скомпилированную версию модели:

```
let compiledModelUrl = try MLModel.compileModel(at: downloadedModelUrl)
let model = try MLModel(contentsOf: compiledModelUrl)
```

Имейте в виду, что `compiledModelUrl` — это адрес во временном хранилище. Если нужно, чтобы модель сохранилась на устройстве до следующего сеанса, переместите ее в постоянное хранилище.



Конечно, можно организовать прямое управление моделями вручную с помощью Core ML, но для этого надо писать большой объем шаблонного кода, выполняющегося на стороне сервера и на устройстве. Придется вручную управлять версиями каждой модели, хранилищем файлов и конфигурациями, а также обрабатывать любые ошибки, возникающие в процессе. Именно с этой точки зрения ML Kit и Fritz выглядят особенно привлекательно, предоставляя все нужное прямо из коробки. Более подробно об этом поговорим в главе 13.

## Обучение на устройстве

До сих пор мы рассматривали ситуации, с которыми прекрасно справляются нейронные сети «одного размера для всех». Но иногда без персонализации моделей просто не обойтись. Например, приложение, которое организует фотогалерею, используя распознавание лиц людей на фотографиях. Учитывая, что телефон обычного пользователя в основном заполнен фотографиями друзей и членов семьи, универсальная модель, обученная на лицах Дэнни ДеВито и Тома Хэнкса, едва ли будет полезна пользователям (если, конечно, они сами не члены семей ДеВито или Хэнкса).

В качестве примера из реальной жизни можно привести системную клавиатуру iOS, которая постепенно изучает языковые шаблоны пользователя и начинает предлагать ему все более актуальные подсказки. Это становится особенно очевидным, когда человек использует сленг, псевдонимы, термины и т. д., которые могут не входить в общий языковой словарь. Персонализированные предложения, основанные на таких данных, будут бесполезны для других пользователей.

В таких случаях желательно было бы собрать данные этого пользователя и обучить модель, персонализированную только для него. Один из способов — собрать данные и отправить их в облако, обучить там новую модель и вернуть обновленную модель пользователю. Но у этого подхода есть проблемы с масштабируемостью, стоимостью и конфиденциальностью.

Вместо этого в Core ML есть функциональность для обучения на устройстве, так что данные пользователя никогда не должны покидать устройство. Модель Core ML будет доступна для обновления, если ее свойству `isUpdatable` присвоить значение `true`. Кроме того, слои, которые нужно переобучить (обычно ближе к концу сети), тоже должны быть `true` в этом же свойстве. Также можно установить дополнительные параметры обучения, такие как скорость обучения и используемые оптимизации.

Конечно, для обучения может потребоваться GPU и Neural Processor Unit (NPU — нейронный процессор, подробнее о нем в главе 13). Но запуск обучения можно запланировать в фоновом режиме (с использованием фреймворка `BackgroundTasks`), и даже когда устройство находится в режиме ожидания и заряжается, обычно ночью, чтобы не вызывать раздражения у пользователя.

Чтобы запустить обучение на устройстве, можно вызвать функцию `MLUpdateTask` и передать ей новые данные для дообучения модели и путь для сохранения обновленной версии. После завершения обучения обновленная модель будет готова к работе:

```
let modelUrl = bundle.url(forResource: "MyClassifier",
                           withExtension: "mlmodelc")!
```

```
let updatedModelUrl = bundle.url(forResource: "MyClassifierUpdated",
                                  withExtension: "mlmodelc")!
let task = try MLUpdateTask(
    forModelAt: modelUrl,
    trainingData: trainData,
    configuration: nil,
    completionHandler: { [weak self] (updateContext) in
        self?.model = updateContext.model
        updateContext.model.write(to: updatedModelUrl)
    })
task.resume()
```

## Федеративное обучение

Обучение на устройстве — это прекрасно, но у него есть недостаток: общая глобальная модель при этом не улучшается. Было бы замечательно, если бы разработчик мог как-то использовать данные, генерируемые на устройствах пользователей, для улучшения глобальной модели без передачи персональных данных за пределы устройства. И такая возможность есть — это *федеративное обучение*.

Федеративное обучение — это распределенный процесс обучения на коллективных данных. По сути, обучение на устройстве делает один маленький шаг вперед, а его результаты — инкрементальные обновления (персонализированных моделей на устройствах пользователей) — пересылаются в облако, где объединяются и обогащают глобальную модель, доступную всем. Обратите внимание, что при этом не передается никакой персональной информации и реконструировать пользовательские данные из агрегированного набора признаков нельзя. Такой подход гарантирует конфиденциальность пользовательских данных и одновременно приносит всем пользу.

Обучение на устройстве — жизненно важный этап для федеративного обучения. Мы еще не достигли этого, но отрасль неуклонно движется в этом направлении. Со временем можно ожидать появления более полноценной поддержки федеративного обучения.

TensorFlow Federated — одна из реализаций федеративного обучения, обеспечивающая обучение моделей для приложения клавиатуры Google GBoard. Обучение на пользовательских устройствах протекает в фоновом режиме, пока они заряжаются.

## Анализ качества моделей

Создание прототипов — это одно. Создание приложений, готовых к широкому использованию, — совершенно другое. Есть несколько факторов, влияющих на восприятие приложения пользователем, и очень важно знать и понимать

все компромиссы. К ним можно отнести список поддерживаемых моделей устройств, минимальную версию операционной системы, частоту кадров обработки и выбор модели глубокого обучения. В этом разделе мы посмотрим, какое влияние могут оказывать некоторые из этих факторов на качество и производительность продукта.

## Бенчмарк моделей на iPhone

Как принято в бенчмаркинге, эксперименты лучше всего проводить на общедоступных моделях, которые можно легко загрузить и опробовать.

Для начала рассмотрим наше приложение классификации объектов в реальном времени на нескольких моделях iPhone, выпускавшихся с 2013 по 2018 год. Результаты представлены в табл. 11.3.

**Таблица 11.3.** Бенчмаркинг времени вывода для разных моделей на разных версиях iPhone

Модель устройства			iPhone 5s	iPhone 6	iPhone 6s	iPhone 7+	iPhone X	iPhone XS	iPhone 11 Pro
Год выпуска			2013	2014	2015	2016	2017	2018	2019
Объем ОЗУ			1 Гбайт	1 Гбайт	2 Гбайт	2 Гбайт	2 Гбайт	4 Гбайт	4 Гбайт
Процессор			A7	A8	A9	A10	A11	A12	A13
Модель	Точность (%)	Размер (Мбайт)	FPS	FPS	FPS	FPS	FPS	FPS	FPS
VGG-16	71	553	0,1	0,2	4,2	5,5	6,9	27,8	34,5
InceptionV3	78	95	1,4	1,5	9	11,1	12,8	35,7	41,7
ResNet-50	75	103	1,9	1,7	11,6	13,5	14,1	38,5	50
MobileNet	71	17	7,8	9	19,6	28,6	28,6	55,6	71,4
SqueezeNet	57	5	13,3	12,4	29,4	33,3	34,5	66,7	76,9

Особенно разительна разница между устройствами 2014 и 2015 годов. Что случилось в 2015 году? Если вы подумали о GPU, то оказались правы. В iPhone 6S впервые появился отдельный графический процессор, поддерживающий такие функции, как «Привет, Сири»<sup>1</sup>.

<sup>1</sup> Сири (англ. Siri) — голосовой помощник в устройствах Apple.

## Методология

Чтобы провести бенчмаркинг с воспроизводимыми результатами, мы взяли код примера приложения из этой главы и просто заменили файл *.mlmodel* в Xcode другими предварительно обученными моделями, доступными на сайте Apple. Мы запускали приложение с каждым типом модели на фиксированное время и собирали параметры выполнения в консоли отладки. После запуска на разных iPhone мы усреднили результаты по прогнозам между 6-й и 20-й попытками в каждом прогоне и свели их в таблицу.

А теперь посмотрим, как распределялся рынок iPhone в США на момент появления iPhone XS в сентябре 2018 года (табл. 11.4). Обратите внимание, что новые устройства обычно выпускались в сентябре каждого года.

**Таблица 11.4.** Распределение рынка устройств iPhone в США по состоянию на сентябрь 2018 года (месяц выпуска iPhone XS; данные взяты с сайта Flurry Analytics (<https://oreil.ly/L47c0>) и из них исключены доли iPhone XS, XS Plus и XS Max)

Год выпуска	Модель iPhone	Процент
2017	8 Plus	10,8%
2017	X	10,3%
2017	8	8,1%
2016	7	15,6%
2016	7 Plus	12,9%
2016	SE	4,2%
2015	6S	12,5%
2015	6S Plus	6,1%
2014	6	10,7%
2014	6 Plus	3,3%
2013	5S	3,4%
2013	5C	0,8%
2012	5	0,8%
2011	4S	0,4%
2010	4	0,2%

В табл. 11.5 показан накопленный процент, полученный из табл. 11.4. Эта таблица помогает оценить потенциальную долю рынка по самому старому устройству, которое мы выберем для поддержки. Например, iPhone, выпущенные после сентября 2016 года (за два года до сентября 2018 года), занимают существенную долю рынка — 61,9 %.

**Таблица 11.5.** Накопленная доля рынка устройств iPhone по годам

Лет тому назад	Накопленный процент
1	29,2%
2	61,9%
3	80,5%
4	94,5%
5	98,7%
6	99,5%

Объединение результатов анализа качества и доли рынка дает несколько вариантов действий:

#### *Использование более быстрой модели*

На iPhone 6 модель VGG-16 работает примерно в 40 раз медленнее, чем MobileNet. На iPhone XS она же работает примерно в два раза медленнее. Простой выбор более эффективной модели может сильно повлиять на производительность, часто не требуя снижать точность. Стоит отметить, что MobileNetV2 и EfficientNet предлагают еще лучшее сочетание скорости и точности.

#### *Определение минимальной поддерживаемой частоты кадров (FPS)*

Для применения к видеопотоку фильтров в масштабе реального времени должна быть возможность обрабатывать как можно больше кадров, поступающих с камеры каждую секунду, чтобы обеспечить плавность просмотра. Напротив, в приложении, которое обрабатывает изображения по одному, почти не придется беспокоиться о производительности. Многие приложения занимают промежуточное положение между этими крайностями. Важно определить минимально требуемую частоту кадров и выполнить сравнительный анализ по аналогии с приведенным в табл. 11.5, чтобы выбрать лучшую модель (или модели) и поколения iPhone для поддержки.

#### *Пакетная обработка*

Графические процессоры особенно хорошо подходят для параллельной обработки. Поэтому неудивительно, что пакеты данных они обрабатывают

намного эффективнее, чем отдельные образцы. Фреймворк Core ML использует этот факт, предлагая API для пакетной обработки. Некоторые пользовательские интерфейсы, особенно асинхронные или интенсивно использующие память, могут значительно выиграть от использования таких API. Например, любая массовая обработка фотографий в фотогалерее. Вместо обработки изображений по одному можно отправить множество изображений в пакетный API, чтобы фреймворк Core ML мог оптимально использовать возможности GPU.

### *Динамический выбор модели*

В зависимости от варианта использования иногда можно поступиться высокой точностью ради плавности взаимодействий (при некотором минимальном пороге FPS). В этих случаях на медленном и старом устройстве можно выбрать более легковесную и менее точную модель, а на быстром и современном устройстве — более крупную и точную. Модели для выбора можно развернуть в облаке, чтобы не увеличивать размер приложения.

### *Использование приема «шерлокинга» в своих интересах*

«Шерлокинг» (Sherlocking) — это когда производитель (в частности, Apple) делает стороннее ПО ненужным, реализуя его функции в своем продукте. Например, реализация фонарика в iPhone сделала ненужными все сторонние приложения-фонарики (платные или бесплатные). В качестве иллюстрации возьмем гипотетическое приложение с функцией распознавания лиц, выпущенное в 2017 году. Год спустя Apple добавила более точный API распознавания лиц в Core ML для iOS 12. Поскольку нейронные сети для распознавания лиц теперь встроены в операционную систему, разработчики могут обновить свой код и использовать встроенный API. Но поскольку системный API будет по-прежнему недоступен на устройствах с iOS 11, приложение может использовать гибридный подход и обеспечить обратную совместимость за счет сохранения старой ветки в коде. Также разработчики приложения могут отказаться от встраивания нейронных сетей в приложение, уменьшив его размер на несколько мегабайт, и динамически загружать старую модель для пользователей iOS 11.

### *Интеллектуальное сужение возможностей*

Иногда старый телефон просто не способен работать с новыми моделями глубокого обучения из-за высоких требований к производительности. Это не редкость, особенно для топовых приложений. Тогда допустимо использовать один из двух подходов. Первый — перенести вычисления в облако. Очевидно, это влечет ухудшение интерактивности и конфиденциальности. Другой вариант — отключение части функций и отображение сообщения о причинах их недоступности.

## Оценка энергопотребления

В предыдущих главах мы уделяли основное внимание размещению классификаторов на серверах. Конечно, есть определенные факторы, влияющие на проектные решения, но энергопотребление часто не входит в их число. Но на стороне клиента емкость аккумулятора часто весьма ограничена, поэтому минимизация энергопотребления становится приоритетом. Восприятие продукта пользователями во многом зависит от того, сколько энергии он потребляет. Вспомните старые добрые времена, когда модуль GPS сильно разряжал аккумулятор. Многие приложения, которым требовался доступ к информации о местоположении, получали массу оценок в одну звезду за слишком большой расход заряда аккумулятора. А нам этого точно не нужно.



Люди довольно щедры на плохие отзывы, когда замечают, что приложение разряжает аккумулятор быстрее, чем ожидалось, как показано на рис. 11.13.



25/03/2019

### **Glitches & Battery!!!**

Recently, I haven't been able to watch videos properly at ALL. The video glitches within the first few seconds, freezing. Then the only way for me to actually get the video to get out of my screen is the skip thee video. If this app's "a way to keep connected with friends", then at least let me see what my friends are saying!! Also, it's a mega battery drainer and using it while I'm on trips usually ends up being a pain for me.



31/10/2017

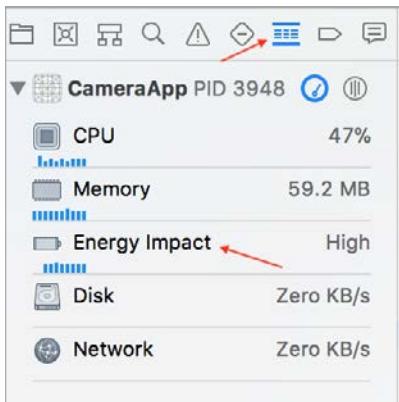
### **Burning through my battery**

Your space-time continuum update broke the app. Watching videos burns through the battery and the phone gets very hot. Lost 20% battery watching a 10min video on my iPhone 7 Plus. Other video streaming apps work as usual.

**Рис. 11.13.** Отзывы в App Store на приложения YouTube и Snapchat, в которых пользователи жалуются на слишком большое энергопотребление<sup>1</sup>

<sup>1</sup> Вы можете найти множество отрицательных отзывов с аналогичными жалобами на русском языке. — Примеч. ред.

Вот пример выбора вкладки Energy Impact (Энергопотребление; рис. 11.14) в Xcode Debug Navigator, чтобы сгенерировать рис. 11.15.



**Рис. 11.14.** Вкладки в Xcode Debug Navigator



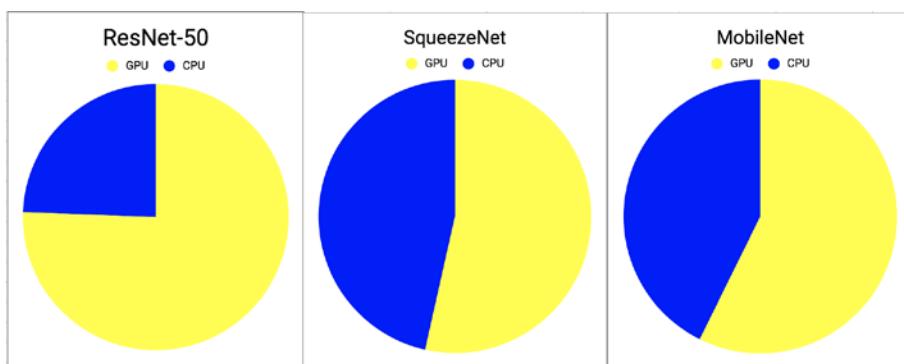
**Рис. 11.15.** График энергопотребления на iPad Pro 2017 (примечание: этот скриншот был сделан в другое время по сравнению со скриншотом на рис. 11.14, поэтому цифры отличаются)

На рис. 11.15 показано, что запущенный процесс на iPad Pro 2017 имеет высокое энергопотребление. Основная причина в том, что в нашем примере кода мы

обрабатывали каждый кадр. Каждый кадр, захваченный камерой, отправлялся на GPU для обработки, что и явилось причиной высокого потребления энергии. Во многих приложениях нет необходимости классифицировать каждый кадр. Даже обработка каждого второго кадра может привести к значительной экономии энергии, не вызывая при этом негативных эмоций у пользователей. В следующем разделе мы исследуем связь между частотой обработки кадров и энергопотреблением.



Соотношение использования CPU и GPU во многом зависит от архитектуры модели, в частности от количества операций свертки. Здесь мы наблюдаем среднюю загрузку CPU и GPU для разных моделей (рис. 11.16). Эти числа можно извлечь из диаграммы Energy Impact (Энергопотребление) в Xcode. Обратите внимание, что в идеале для повышения производительности мы предпочли бы модели, более интенсивно использующие GPU.



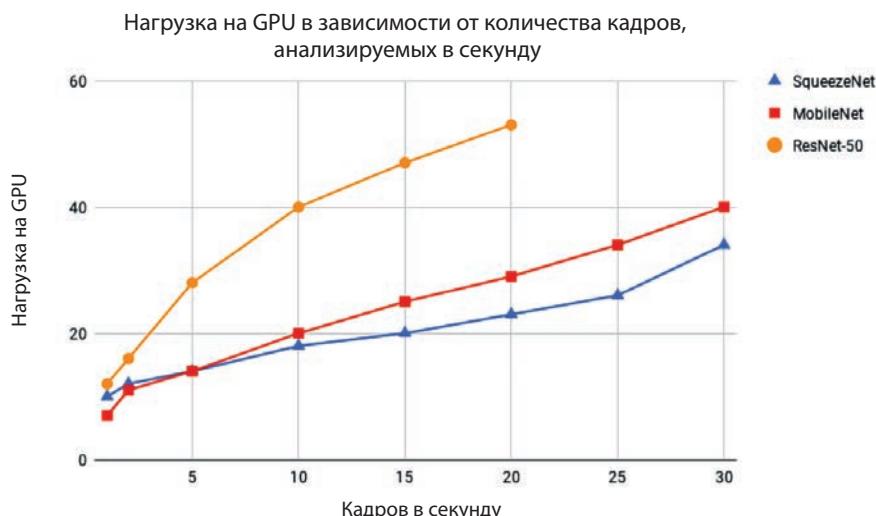
**Рис. 11.16.** Сравнение использования CPU и GPU разными моделями в iOS 11

## Оценка нагрузки

Как и следовало ожидать, применение сверточных моделей к каждому кадру в реальном времени влечет увеличение нагрузки на GPU и CPU, и поэтому телефон быстро разряжается и нагревается.

А что, если анализировать не все кадры, а с некоторым интервалом? На анализ одного кадра модель MobileNet тратит 20 мс на iPad Pro 2017. То есть она способна классифицировать около 50 кадров в секунду. Если применять модель с частотой 1 кадр в секунду, нагрузка на GPU снизится с 42 % до 7 %, то есть больше чем на 83%! Для многих приложений частоты обработки 1 кадр в секунду вполне достаточно, при этом устройство не нагревается. Например, для камеры видеонаблюдения вполне достаточно обрабатывать 1 кадр за две секунды.

Меняя количество кадров, анализируемых в секунду, и измеряя уровень нагрузки на GPU, мы заметили интересную тенденцию. Как показано на графике на рис. 11.17, чем выше частота обработки кадров, тем выше нагрузка на GPU. Это существенно влияет на потребление энергии. Для приложений, которые должны работать долго, может быть полезно уменьшить количество кадров, анализируемых каждую секунду.



**Рис. 11.17.** Зависимость нагрузки на GPU от количества кадров, анализируемых в секунду, на iPad Pro 2017

Значения, показанные на графике, были получены с помощью Core Animation из набора инструментов Xcode. Вот как это сделать:

1. В Xcode в меню **Product** (Продукт) выберите пункт **Profile** (Профилировать).
2. После появления окна **Instruments** (Инструменты) выберите инструмент **Core Animation** (Базовая анимация), как показано на рис. 11.18.
3. Щелкните на кнопке **Record** (Запись), чтобы запустить приложение в режиме профилирования.
4. Подождите несколько секунд, пока начнется вывод собранных данных.
5. Оцените значения в колонке **GPU Hardware Utilization** (Использование GPU; рис. 11.19).

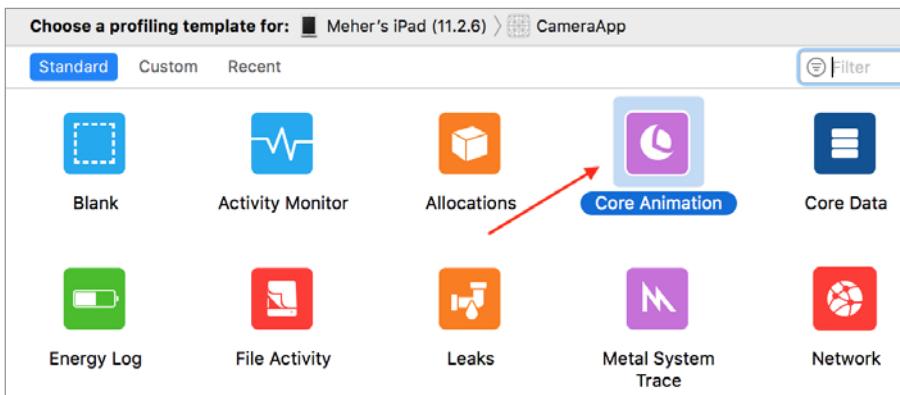


Рис. 11.18. Окно Instruments (Инструменты) в Xcode

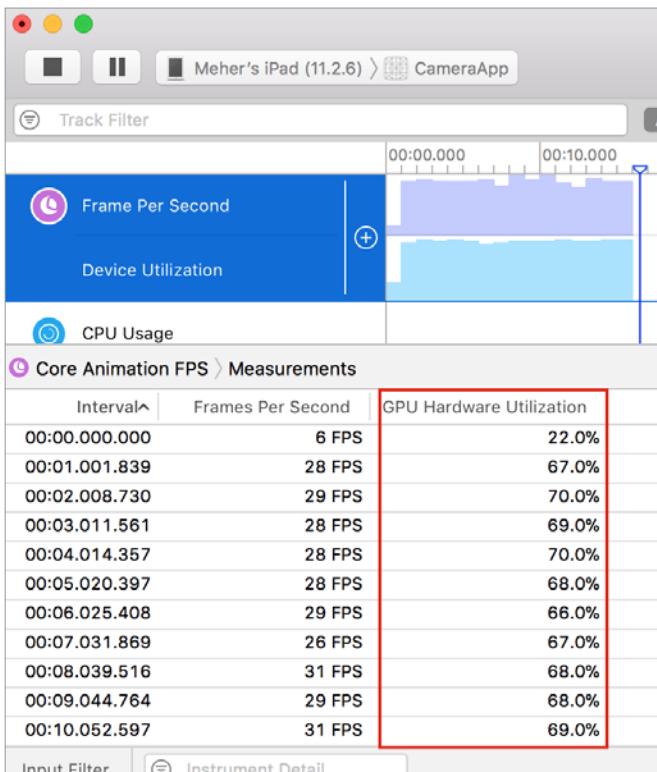


Рис. 11.19. Процесс профилирования приложения в инструменте Core Animation (базовая анимация)

## Уменьшение размера приложения

Размер приложения иногда бывает очень важен. Например, Apple не позволяет загружать через мобильную сеть приложения весом более 200 Мбайт. Размер становится важным параметром для приложений, которые часто используются в дороге, например Uber и Lyft. Отметим, что компании Uber пришлось внедрить несколько действительно жестких оптимизаций и значительно сократить использование компонентов Swift (упрощающих сопровождение кода) — Optionals, Structs и Protocols, чтобы уменьшить размер бинарного файла приложения и привести его в соответствие с требованиями App Store (на тот момент 150 Мбайт). Без этого компания потеряла бы много новых пользователей.

Очевидно, что любая новая добавляемая функция ИИ увеличивает размер приложения. Есть несколько стратегий, которые можно использовать, чтобы снизить остроту проблемы.



В компании Segment из Сан-Франциско, занимающейся анализом данных, решили выяснить, как размер приложения влияет на количество установок. Для экспериментов компания купила приложение ипотечного калькулятора, показывающее устойчивое количество скачиваний (примерно 50 раз в день). Начав с первоначального размера 3 Мбайт, они постепенно увеличивали его, добавляя картинки из альбомов Тейлор Свифт (все во имя науки и ради экспериментов). Инженеры отметили, что с ростом размера приложения количество установок значительное уменьшилось. Когда размер превысил 100 Мбайт (тогдашний предел для загрузок из App Store через мобильную сеть), количество ежедневных установок упало на колоссальные 44%! Кроме того, приложение получило несколько отрицательных отзывов, связанных с размером.

Мораль в том, что размер приложения намного важнее, чем мы думаем. И стоит помнить об этом, когда мы готовим свои приложения к релизу.

## Не внедряйте модели в приложение

По возможности не включайте модели в бинарный файл, публикуемый в App Store. Пользователю так или иначе придется загрузить один и тот же объем данных, поэтому, если это не влияет на пользовательский опыт, загрузку модели лучше отложить до того момента, когда она действительно понадобится. Кроме того, желательно организовать загрузку моделей только при подключении к Wi-Fi, чтобы сэкономить мобильный трафик. Такой подход, например, реализуют Microsoft Translator и Google Translate, которые по умолчанию выполняют перевод только в облаке. Но понимая, что путешественники часто используют эти приложения в дороге (причем там, где может не быть хорошего доступа в интернет), у них есть и автономный режим, в котором нужные языковые модели загружаются по запросу пользователя в фоновом режиме.

## Используйте квантование

Из главы 6 мы знаем, что квантование — хорошая стратегия для резкого уменьшения размера модели при умеренной потере точности. По сути, квантование заключается в замене 32-битных весов (чисел с плавающей запятой) 16-битными числами с плавающей запятой, 8-битными целыми и даже 1-битными значениями. Но не рекомендуем использовать значения с размерностью меньше 8 бит из-за сильной потери точности. Квантование моделей Keras делается с помощью Core ML Tools за пару строк кода:

```
import coremltools

model_spec = coremltools.utils.load_spec("MyModel.mlmodel")

# преобразование в 16-битные значения
model_fp16_spec =
coremltools.utils.convert_neural_network_spec_weights_to_fp16(model_spec)
coremltools.utils.save_spec(model_fp16_spec, "MyModel_FP16.mlmodel")

# преобразование в 8-битные значения
num_bits = 8
model_quant_spec =
coremltools.models.neural_network.quantization_utils.quantize_weights(model_
spec,
num_bits, "linear")
coremltools.utils.save_spec(model_quant_spec, "MyModel_Quant.mlmodel")
```

Мы покажем вам влияние квантования на сложную модель, небольшие модификации которой могут привести к радикальным изменениям. С помощью Keras мы создали классификатор по 102 категориям на датасете с картинками цветов (получившийся классификатор был примерно 14 Мбайт), затем сделали квантование до различных представлений и измерили снижение точности и уменьшение размера.

Чтобы оценить изменение точности классификации, мы сравнили процент совпадений при использовании полной и квантованной модели. Мы проверили три метода квантования.

- Простое линейное (`linear`) квантование, которое было описано в главе 6. В этой стратегии интервалы распределяются равномерно.
- Линейное квантование с использованием таблицы поиска (`linear_lut`). В этом случае интервалы распределяются неравномерно: более плотные области делятся на большее число коротких интервалов, а более разреженные области — на меньшее число длинных интервалов. Так как эти интервалы распределены неравномерно, информацию о них нужно хранить в таблице поиска, а не вычислять напрямую.

- Таблица поиска, созданная методом  $k$ -средних (`kmeans_lut`), которая часто используется в классификаторах ближайших соседей.

Результаты экспериментов показаны в табл. 11.6.

**Таблица 11.6.** Результаты квантования весов до представлений с разными размерами и использованием разных методов квантования

Квантование до	Процент уменьшения размера (примерно)	Процент совпадений в сравнении с версией с 32-битными весами		
		linear	linear_lut	kmeans_lut
16 бит	50%	100%	100%	100%
8 бит	75%	88,37%	80,62%	98,45%
4 бита	88%	0%	0%	81,4%
2 бита	94%	0%	0%	10,08%
1 бит	97%	0%	0%	7,75%

Изучив результаты эксперимента, мы пришли к следующим выводам.

- Уменьшение размерности представления весов до 16-битного никак не повлияло на точность. То есть размер модели можно уменьшить вдвое без заметной потери точности.
- Применение метода  $k$ -средних для построения таблицы поиска дает увеличение точности по сравнению с линейными методами деления. Даже при уменьшении размерности до 4 бит этот метод обеспечивает потерю всего 20% точности, что удивительно.
- Уменьшение размерности представления весов до 8-битного приводит к уменьшению размера модели в 4 раза и небольшой потере точности (особенно небольшой при использовании метода `kmeans_lut`).
- Квантование до представлений с размерами меньше 8 бит приводит к резкому падению точности, особенно при использовании линейных методов.

## Используйте Create ML

Create ML — это инструмент, созданный в Apple и предназначенный для обучения моделей перетаскиванием данных мышью в графическом интерфейсе на Mac. В нем есть несколько шаблонов, включая классификацию и распо-

знавание объектов на изображениях, классификацию звука и классификацию текста, что позволяет обучать ИИ даже новичкам, не требуя каких-либо специальных знаний в данной области. Он использует метод переноса обучения для настройки лишь нескольких слоев, которые нужны для конкретной задачи. Благодаря этому обучение обычно занимает всего несколько минут. Большая часть слоев (которые могут использоваться для решения широкого круга задач) уже входит в состав iOS, а специализированные слои могут поставляться как часть приложения. В результате экспортированная модель оказывается очень маленькой (всего 17 Кбайт, как мы вскоре увидим). Поближе с Create ML мы познакомимся в следующей главе.

До сих пор в этой главе мы исследовали методы создания приложений с использованием Core ML. В следующем разделе мы обсудим некоторые примеры реальных приложений.

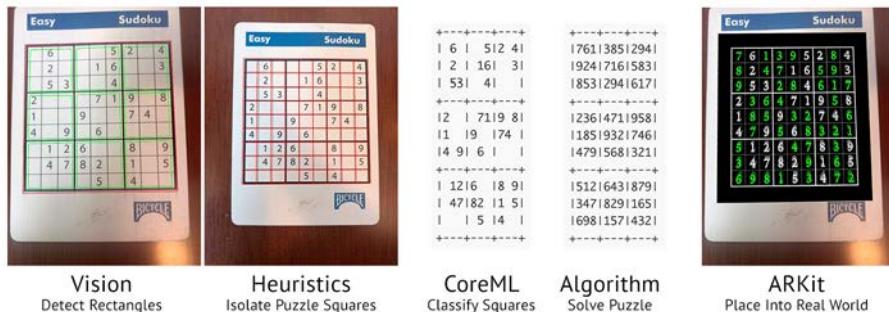
## Примеры приложений

Рассмотрим несколько примеров использования Core ML в мобильных приложениях.

### Magic Sudoku

Вскоре после выхода ARKit для iOS (<https://developer.apple.com/arkit>) в 2017 году стартап Hatchlings, занимающийся разработкой игр и мобильных приложений, выпустил приложение Magic Sudoku (<https://magicsudoku.com/>), быстро ставшее популярным. Просто наведите камеру телефона на судоку, и приложение покажет решение прямо на снимке. Как нетрудно догадаться, для распознавания цифр приложение использует сверточную сеть на Core ML. Для этого оно выполняет следующие шаги:

1. С помощью ARKit получает кадр с камеры.
2. С помощью iOS Vision Framework распознает прямоугольники.
3. Определяет, является ли изображение судоку.
4. Извлекает из изображения 81 квадрат.
5. С помощью Core ML распознает цифры в каждом квадрате.
6. Использует решающий модуль для судоку, чтобы заполнить пустые квадраты.
7. Проецирует полученное решение на исходное изображение с помощью ARKit, как показано на рис. 11.20.



**Рис. 11.20.** Шаги решения судоку с помощью ARKit (изображения взяты с сайта <https://oreil.ly/gzmb9>)

Сперва команда Hatchlings использовала модель распознавания цифр, обученную на датасете MNIST, состоящем в основном из картинок рукописных цифр. Но она показала низкую надежность при распознавании печатных шрифтов. Затем команда сфотографировала тысячи страниц книг с судоку, извлекая квадраты с помощью пайплайна, пока не получила большое количество изображений отдельных цифр, напечатанных разными шрифтами. Для разметки этих данных команда предложила своим фанатам классифицировать каждое изображение и снабдить их метками от «0» до «9» и «пусто». В течение суток команда отсканировала 600 000 цифр. Следующим шагом было обучение собственной сверточной сети, которая должна была работать достаточно быстро, потому что приложению требовалось классифицировать 81 квадратное изображение. Модель была развернута с помощью Core ML, а получившееся приложение стало хитом.

Приток новых пользователей привел к открытию новых неожиданных случаев использования. Поскольку у большинства пользователей не было напечатанных на бумаге судоку, они часто искали такие головоломки в интернете и выводили их на экраны компьютеров, но модель испытывала сложности с распознаванием головоломки на экране. Кроме того, из-за ограничения фиксированного фокусного расстояния в ARKit изображение получалось немного размытым. Тогда команда Hatchlings собрала дополнительные примеры фотографий экранов компьютеров с головоломками, слегка размыла их и обучила новую сверточную сеть с дополнительными данными. Обновленное приложение стало намного удобнее.

Таким образом, поддерживая приложение, разработчик должен постоянно выявлять новые сценарии использования, которые ранее не ожидались.

## Seeing AI

Seeing AI (<https://oreil.ly/hJxUE>) — это «говорящая камера», созданная в Microsoft Research для слепых и слабовидящих. Она использует механизмы компьютер-

ного зрения для описания людей, текста, почерка, предметов, банкнот и многое другого, генерируя речь. Основная обработка изображений выполняется непосредственно на устройстве с использованием Core ML. Одна из основных нужд пользователей — распознавание товаров в магазине. Обычно такую возможность можно реализовать, сканируя штрих-коды. Но слепой пользователь не знает, где находится штрих-код, поэтому приложение для чтения штрих-кодов не годится. Чтобы решить эту проблему, команда создала свою сверточную сеть, обучила ее на фотографиях штрих-кодов, снятых под разными углами, с разного расстояния, с разным освещением и ориентацией. Пользователю остается только покрутить объект перед камерой iPhone, и когда приложение определит наличие штрих-кода (оно работает в масштабе реального времени, анализируя кадр за кадром), то подаст серию звуковых сигналов. Частота сигнала напрямую зависит от положения штрих-кода относительно камеры. С приближением штрих-кода к камере сигналы начинают следовать чаще. Когда штрих-код оказывается достаточно близко, чтобы библиотека чтения штрих-кода могла четко распознать его, приложение расшифровывает универсальный код товара и произносит его название. В прошлом слепым пользователям приходилось покупать и носить с собой громоздкие лазерные сканеры штрих-кода, стоимость которых нередко превышала 1300 долларов. Благотворительный фонд собрал миллионы, чтобы купить эти аппаратные считыватели штрих-кодов и передать тем, кто в них нуждался. Теперь глубокое обучение может решить проблему бесплатно. Это хороший пример сочетания компьютерного зрения и пользовательского опыта.

## HomeCourt

В любом деле, будь то сочинение рассказов, игра на музыкальном инструменте или приготовление пищи, для успеха важна регулярная практика. При этом качество практики гораздо важнее количества. Использование данных для поддержки практических навыков и наблюдения за их развитием творит чудеса, помогая нарабатывать навыки намного быстрее. Именно с этой целью стартап NEX Team из Сан-Хосе, штат Калифорния, создал приложение HomeCourt, предназначенное для организации тренировок баскетболистов. Пользоваться приложением очень просто: установите устройство на земле или на штативе, наведите камеру на баскетбольную площадку и попробуйте побить рекорд.

Приложение распознает объекты в режиме реального времени с помощью Core ML, определяет людей, мяч и баскетбольное кольцо. В игре часто возникает вопрос: кто забросил мяч? Когда мяч приближается к кольцу, приложение перематывает видео, чтобы определить игрока, который бросил мяч. Затем оно оценивает позу игрока и отслеживает его движения. Мало того, приложение использует геометрические преобразования, чтобы превратить трехмерную сцену площадки в двумерную карту (как показано на рис. 11.21 в правом верхнем углу), чтобы определить место, откуда игрок бросил мяч. Отметим, что в приложении

одновременно работает несколько моделей. Анализируя объекты, положение тела, суставов и т. д., приложение выводит статистику и показывает высоту, угол, начальную точку и время каждого броска, скорость игрока и т. д. Эти характеристики важны, так как тесно связаны с успехом попытки. Игровки могут записать всю игру, а затем по дороге домой проанализировать каждый бросок, чтобы выявить недостатки и устраниить их.

Менее чем за два года стартап привлек сотни тысяч пользователей, от любителей до профессионалов. Даже NBA начала сотрудничать с NEX Team, чтобы помочь своим игрокам улучшить результаты. Это стало возможным потому, что в NEX Team заметили неудовлетворенную потребность и предложили творческое решение, основанное на глубоком обучении.



**Рис. 11.21.** Приложение HomeCourt следит за бросками игроков

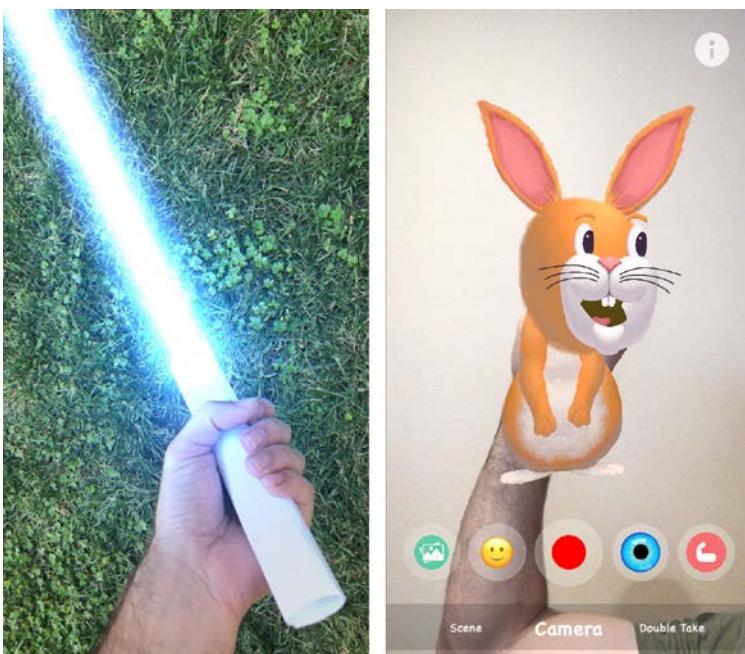
## InstaSaber + YoPuppet

Знаете, какой самый большой источник прибыли во франшизе «Звездные войны»? Это не продажа билетов в кино и уж точно не продажа DVD-дисков. Это торговля товарами с символикой. В Lucas Film (а теперь и Disney) заработали много денег, продавая игрушечных дроидов-астромехаников R2D2 и костюмы Чубакки. Но самым популярным по-прежнему остается световой меч. Но игрушечные световые мечи, сделанные в основном из пластика, выглядят не так круто, как в фильмах, и, откровенно говоря, имеют совсем не фантастический вид. К тому же, махая им, вы рискуете случайно ударить кого-нибудь.

Харт Вулери (Hart Woolery), основатель 2020CV, решил изменить ситуацию, добавив визуальные эффекты голливудского уровня в телефон с помощью приложения InstaSaber. Просто сверните лист бумаги в трубку, возьмите ее в руку, направьте на нее камеру телефона и смотрите, как трубка превращается в яркий световой меч (рис. 11.22). Взмахните рукой, и вы увидите, насколько реалистично приложение отображает взмах мечом и воспроизводит звуковые эффекты, как во время битвы Люка со своим отцом (внимание, спойлер!) Дартом Вейдером.

Подняв магию отслеживания на новый уровень, Харт создал приложение YoPuppet, определяющее положение суставов рук в режиме реального времени и рисующее виртуальных кукол, которые двигаются синхронно с движениями руки. Такие имитации выглядят правдиво только в том случае, если двигаются без задержек, точно и реалистично.

Эти приложения с потрясающей реалистичностью могут доставить немало приятных минут. Неудивительно, что ролики с InstaSaber стали хитом в Instagram с миллионами просмотров. Потенциал ИИ даже заставил миллиардера-инвестора Марка Кьюбана (Mark Cuban) сказать: «Сделка ваша», — и инвестировать в 2020CV.



**Рис. 11.22.** Скриншоты приложений InstaSaber и YoPuppet

## От создателя

С момента создания компании 2020CV несколько лет тому назад я разработал несколько приложений, которые пытаются раздвинуть границы спецэффектов. Расскажу о двух из них: InstaSaber и YoPuppet. InstaSaber превращает свернутый лист бумаги в виртуальный световой меч, а YoPuppet — руку пользователя в перчаточную куклу. В обоих случаях я перехватываю кадры с камеры в реальном времени, передаю их на вход моделей и получаю нормализованные координаты X-Y местоположений ключевых элементов этих изображений. Идея InstaSaber пришла мне в голову после попытки отследить перемещение рук. Первоначальная задача — как это обычно бывает с большинством моделей машинного обучения — сбор большого объема данных, достаточного для создания обобщенного решения, и их разметка.

Сколько размеченных изображений нужно для модели машинного обучения, которая надежно декодирует объекты, отличающиеся разными характеристиками — освещением, оттенком кожи и наличием перекрытий? Думаю, ответ во многом зависит от задачи, но обычно не менее 10 000. Если предположить, что для правильной разметки каждого изображения нужно 30 секунд или даже больше, то становится очевидно, что этот процесс может затянуться надолго. Простое обходное решение — аугментация данных (то есть применение аффинных преобразований, кадрирование и манипуляции с цветом и освещением). Да, это позволяет превратить конечный объем данных в почти бесконечный набор вариантов, но ваша модель все равно будет ограничена качеством первоначального набора. Работая над InstaSaber, я вручную разметил все изображения, но когда приступил к созданию YoPuppet, то обнаружил, что намного эффективнее искусственно генерировать изображения рук в различных положениях и при разной освещенности (рис. 11.23). Следующая задача после обучения модели — заставить ее хорошо работать в ограниченных условиях мобильного устройства.

С появлением Core ML использование моделей МО на мобильных устройствах стало простой задачей. Но заставить модель работать в режиме реального времени, быстро и без задержек, не расплавив при этом телефон пользователя, очень непросто. В YoPuppet и InstaSaber требовалось определить начальное местоположение объекта и следить за ним в реальном времени, чтобы можно было вырезать квадрат из полного изображения, полученного с камеры, уменьшить его и передать в модель. Один из приемов, которые я использовал, чтобы обеспечить плавное чередование кадров, — буферизация изображений, чтобы CPU и GPU могли выполнять предварительную обработку и выполнять инференс в tandemе.

Часто после публикации приложения появляются отзывы реальных пользователей, изучая которые вы обнаружите, что в вашей модели есть множество



**Рис. 11.23.** Определения положения ключевых точек на руке в режиме реального времени в YoRipret

проблем, о которых вы, скорее всего, никогда не задумывались (предвзятость — реальна!). Например, один пользователь отметил, что накрашенные ногти вызывают проблемы с распознаванием рук. К счастью, эта обратная связь помогает еще больше обобщить модель и сделать ее более устойчивой к вариациям объектов. Улучшение приложения — это бесконечный процесс, но главная награда заключается в том, что вы станете пионером в новой области.

Харт Вулери (Hart Woolery),  
основатель и генеральный директор 2020CV

## Итоги

В этой главе мы совершили захватывающее путешествие по миру Core ML — механизму поддержки алгоритмов машинного и глубокого обучения на устройствах с iOS. Проанализировав минимальный код, нужный для использования сверточной сети, мы создали приложение, которое может распознавать объ-

екты в режиме реального времени и классифицировать их по 1000 категорий в ImageNet. Попутно мы обсудили некоторые полезные приемы конвертации моделей. Перейдя на следующий уровень, мы познакомились с практическими методами динамического развертывания моделей и обучения на устройстве, а также провели сравнительный анализ различных моделей глубокого обучения на разных устройствах с iOS, что помогло лучше понять ограничения, связанные с аккумулятором и другими ресурсами. Мы также увидели, как оптимизировать размер приложения с помощью квантования модели. А для вдохновения мы познакомились с некоторыми реальными приложениями, использующими Core ML.

В следующей главе мы создадим полноценное приложение, обучив модель классификатора с помощью Create ML (и некоторых других инструментов), развернув ее и подготовив с помощью настраиваемого обученного классификатора и его запуска с использованием Core ML.

## ГЛАВА 12

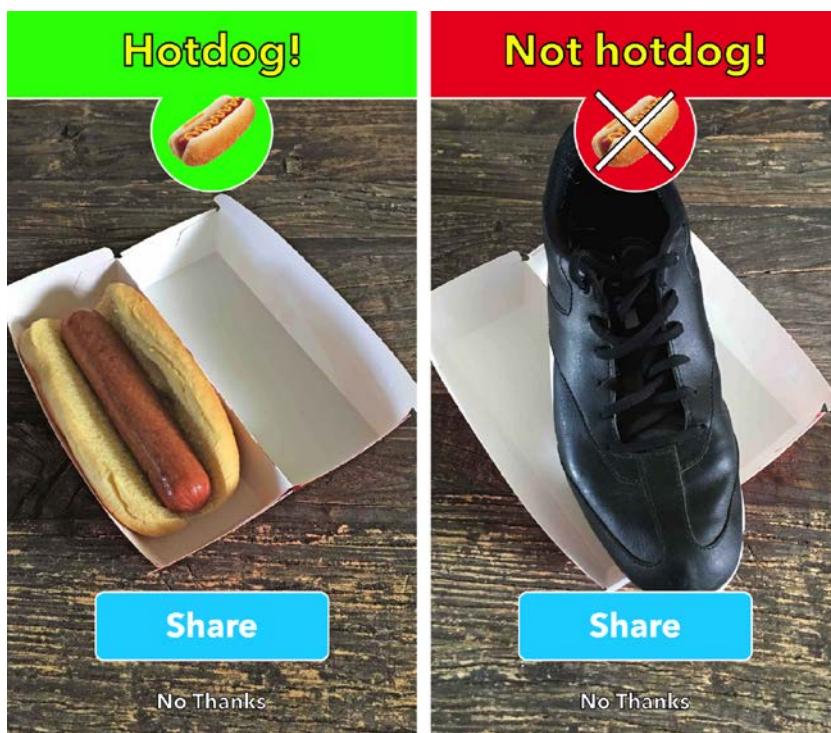
# Not Hotdog на iOS с Core ML и Create ML

«Я богат», — сказал Джин Ян, новоиспеченный миллионер, в интервью агентству Bloomberg (рис. 12.1). Как он этого добился? Создал приложение Not Hotdog (рис. 12.2) и «сделал мир лучше».



**Рис. 12.1.** Джин Ян дает интервью агентству Bloomberg News после того, как компания Periscope приобрела его технологию «Not Hotdog» (источник: сериал «Кремниевая долина» HBO)

Для тех, кто не в курсе (включая треть авторов этой книги), сообщаем, что «Кремниевая долина» — это сериал телекомпании HBO, в котором одному из персонажей поручено создать приложение SeeFood — «Шазам для еды», классифицирующее изображения продуктов и предоставляющее рецепты и информацию об их пищевой ценности. Как ни странно, приложение научилось



**Рис. 12.2.** Приложение Not Hotdog в действии (источник изображения: описание приложения Not Hotdog в Apple App Store)

распознавать только хот-доги. Все остальные блюда оно классифицировало как «не хот-дог».

Мы решили упомянуть это вымышленное приложение по нескольким причинам. Во-первых, это часть поп-культуры, и многие поймут суть без труда. Во-вторых, это достаточно простое приложение и вместе с тем достаточно мощное, чтобы на его примере можно было продемонстрировать магию глубокого обучения. Кроме того, его очень легко обобщить до возможности распознавать более одного класса предметов.

Рассмотрим несколько подходов к созданию приложения Not Hotdog. Общая схема такая:

1. Собрать соответствующие данные.
2. Обучить модель.
3. Преобразовать модель в формат Core ML.
4. Собрать приложение для iOS.

В табл. 12.1 перечислены варианты выполнения шагов с 1-го по 3-й. Каждый из них подробно рассмотрим ниже.

**Таблица 12.1.** Подходы к созданию модели для использования на мобильных устройствах

Сбор данных	Обучение модели	Преобразование модели
Найти готовый или создать свой датасет	Через веб-интерфейс: CustomVision.ai, IBM Watson, Clarifai, Google AutoML	С помощью Create ML, CustomVision.ai или других инструментов, генерирующих файлы в формате <i>.mlmodel</i>
С помощью расширения Fatkun для браузера Chrome	С помощью Create ML	Преобразование модели Keras с помощью инструментов Core ML Tools
С помощью Bing Image Search API	Тонкая настройка с использованием любого фреймворка, например Keras	Преобразование обученной модели TensorFlow с помощью <i>tf-coreml</i>

Итак, приступим!

## Сбор данных

Прежде чем приступить к решению любой задачи компьютерного зрения с использованием глубокого обучения, нужно собрать набор изображений для обучения. В этом разделе рассмотрим три разных подхода к сбору изображений соответствующих категорий в порядке возрастания требуемого времени, от нескольких минут до нескольких дней.

### Подход 1: поиск готового или создание своего датасета

Самое простое решение — получить готовый датасет. В мире есть множество общедоступных датасетов, разбитых на категории, подходящие для нашей задачи. Например, набор Food-101 (<https://www.tensorflow.org/datasets/catalog/food101>) от ETH Zurich включает класс хот-догов. Кроме того, набор ImageNet содержит 1257 изображений хот-догов. А для класса «не хот-дог» мы можем создать выборку случайных изображений.

Загрузить изображения из определенной категории можно с помощью инструмента ImageNet-Utils (<https://oreil.ly/ftyOU>):

1. Найдите соответствующую категорию на сайте ImageNet, например «Hot dog».

2. Обратите внимание на параметр `wnid` (WordNet ID — идентификатор WordNet) в URL: <http://wordnet-rdf.princeton.edu/pwn30/07697537-n>.
3. Клонируйте репозиторий ImageNet-Utils:

```
$ git clone --recursive  
https://github.com/tzutalin/ImageNet_Utils.git
```

Загрузите изображения из конкретной категории, указав идентификатор `wnid`:

```
$ ./downloadutils.py --downloadImages --wnid n07697537
```

Если готовый датасет найти не удалось, можно создать свой, сделав фотографии на смартфон. Снимки нужно делать с учетом того, как будет потом использоваться наше приложение. Или можно попросить родных, друзей и коллег помочь в создании разнообразного датасета. Еще один подход, используемый крупными компаниями, — нанять подрядчиков и поручить им собрать необходимые изображения. Например, Google Allo реализовала функцию преобразования автопортретов в стикеры. Для ее реализации они наняли команду художников, которые создавали стикеры, соответствующие фотографиям, чтобы потом обучить модель на этих парах изображений.



Обязательно прочтайте лицензию, на условиях которой выпущены изображения в датасете. Лучше использовать изображения, выпущенные по разрешительным лицензиям, например Creative Commons.

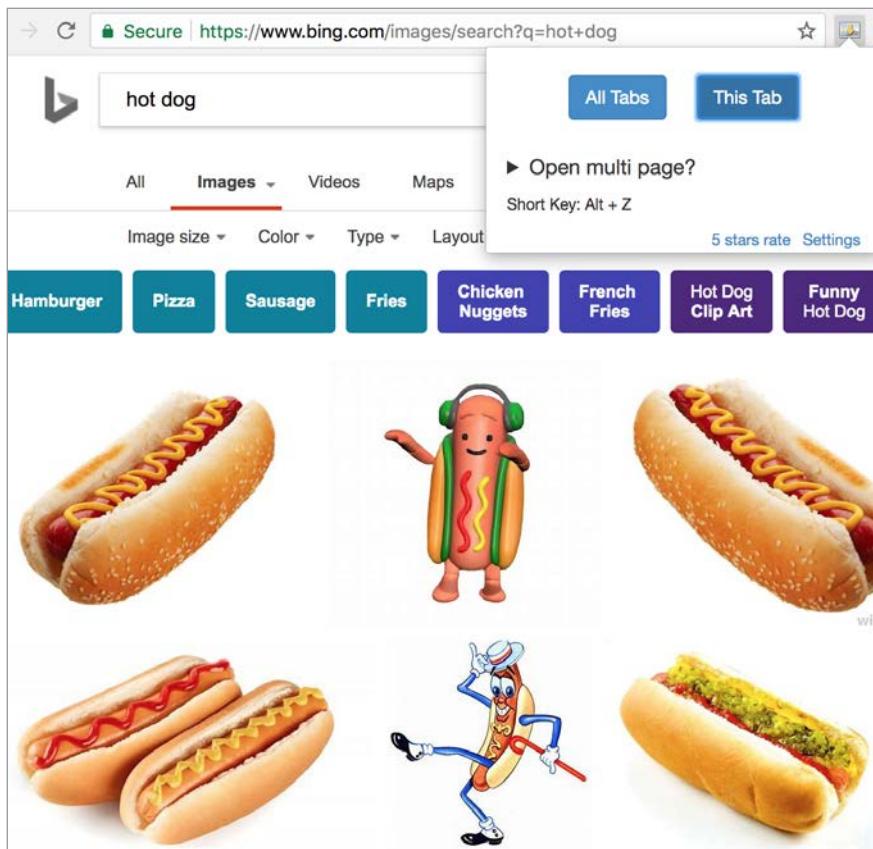
## Подход 2: загрузка изображений с помощью расширения Fatkun для браузера Chrome

Есть несколько расширений, которые позволяют загружать изображения с сайтов в пакетном режиме. Один из примеров — Fatkun Batch Download Image, расширение для Chrome.

Подготовить весь набор с помощью такого подхода можно, выполнив следующие шаги.

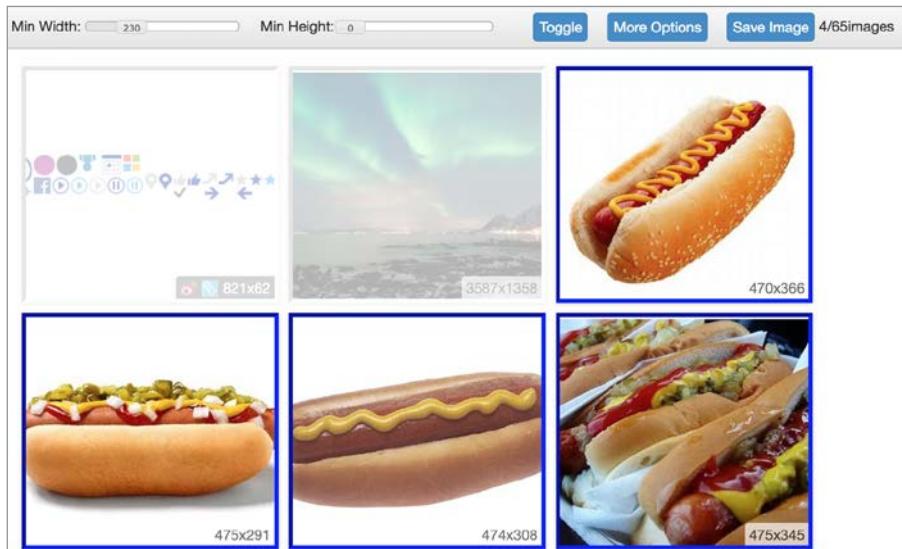
1. Добавьте расширение в браузер.
2. Выполните поиск по ключевому слову в Google или Bing Image.
3. Выберите соответствующий фильтр по лицензиям в настройках поиска.
4. Когда браузер загрузит страницу с результатами поиска, прокрутите ее вниз несколько раз, чтобы получить побольше миниатюр.

5. Откройте расширение и выберите вариант This Tab (Эта вкладка), как показано на рис. 12.3.



**Рис. 12.3.** Результаты поиска по словам «hot dog» в Bing Search

6. Обратите внимание, что по умолчанию будут выбраны все миниатюры. В верхней части окна щелкните на кнопке **Toggle** (Переключить), чтобы снять выделение со всех миниатюр и выбрать только нужные. Попутно можно задать минимальную ширину и высоту, равные 224 (большинство предварительно обученных моделей принимают на входе изображения размером 224 × 224).
7. Щелкните на кнопке **Save Image** (Сохранить изображение) в правом верхнем углу, чтобы загрузить все выбранные миниатюры на компьютер.



**Рис. 12.4.** Выбор изображений в расширении FatKun



Обратите внимание, что на всех изображениях со скриншотов основной объект находится в центре и на чистом фоне. Использование только таких картинок для обучения модели, скорее всего, приведет к тому, что она будет плохо распознавать реальные изображения. Например, по картинке с чистым белым фоном (как на сайте какого-нибудь кафе) нейронная сеть может по ошибке решить, что белый фон равен хот-догу. Поэтому, собирая данные, выбирайте картинки, близкие к реальности.



Для отрицательного класса «не хот-дог» можно собрать случайные изображения. Кроме того, постарайтесь набрать в этот класс изображения предметов, похожих на хот-дог, но не являющихся таковыми; например, сэндвич «субмарина»<sup>1</sup>, хлеб, тарелка, гамбургеры и т. д.

Отсутствие предметов, часто встречающихся вместе с хот-догами, таких как тарелки, салфетки, бутылки или пакеты с кетчупом, может заставить модель думать, что это — настоящие хот-доги. Так что не забудьте добавить их в отрицательный класс.

После установки расширения FatKun оно предложит дать разрешение на чтение и изменение данных со всех посещаемых сайтов, поэтому разумно отключать расширение, если оно не используется для загрузки изображений.

<sup>1</sup> Сэндвич с мясом, сыром, помидорами и т. п. — *Примеч. пер.*

## Подход 3: загрузка с помощью Bing Image Search API

Создание больших датасетов с помощью Fatkun может стать весьма утомительным. Кроме того, изображения, загружаемые Fatkun, являются миниатюрами, то есть имеют уменьшенные размеры. Для создания крупномасштабных коллекций изображений можно использовать специализированные службы поиска, например Bing Image Search API, позволяющие установить определенные ограничения, например ключевое слово, размер изображения и лицензию. Раньше Google поддерживала свой API для поиска изображений, но в 2011 году он был упразднен.

Bing Search API сочетает анализ изображений с помощью ИИ с традиционными методами поиска информации (то есть с использованием меток из полей «alt-text», «metadata» и «caption»). Но из-за вводящих в заблуждение описаний можно получить некоторое количество нерелевантных изображений. Поэтому желательно вручную проанализировать собранные картинки и убедиться, что они действительно соответствуют поставленной задаче.

Если датасет большой, то просмотреть его вручную и отфильтровать плохие образцы будет сложно. Проще использовать итеративный подход, постепенно улучшая качество обучающего датасета с каждой итерацией. Вот общее описание этого процесса:

1. Создайте подмножество обучающих данных, вручную просмотрев небольшое количество изображений. Например, если в исходном датасете 50 000 изображений, то на первой итерации можно вручную отобрать 500 хороших обучающих примеров.
2. Обучите модель на этих 500 изображениях.
3. Протестируйте модель на оставшихся изображениях и получите оценку достоверности для каждого.
4. Среди изображений с наименьшими оценками достоверности (которые обычно неверно классифицируются) просмотрите подмножество (скажем, 500) и отбросьте нерелевантные. Добавьте оставшиеся изображения из этого подмножества в обучающий набор.
5. Повторите шаги с 1-го по 4-й несколько раз, пока качество модели не достигнет удовлетворительного уровня.

Это называется обучением с частичным привлечением учителя.



Если отброшенные изображения использовать в роли представителей отрицательного класса, можно улучшить точность модели.



При формировании большого датасета из картинок из интернета в качестве меток можно использовать сопровождающий текст, например хештеги, смайлики, описание из полей «alt-text» и т. д.

Компания Facebook создала датасет, включающий 3,5 миллиарда изображений, используя хештеги из текста сообщений в качестве слабых меток, обучила на нем свою модель, а затем дообучила ее на датасете ImageNet. Получившаяся модель превзошла лучший результат классификации на 2 % (85 % в топ-1 точности).

Закончив подготовку набора изображений, можно приступать к обучению.

## Обучение модели

Есть три простых способа обучения, и все они обсуждались выше. Здесь мы представим лишь краткий обзор нескольких подходов.

### Подход 1: с помощью инструментов с веб-интерфейсом

Из главы 8 мы знаем, что есть несколько инструментов с веб-интерфейсом для обучения пользовательских моделей на наборах размеченных изображений, в том числе Microsoft CustomVision.ai, Google AutoML, IBM Watson Visual Recognition, Clarifai и Baidu EZDL. Эти инструменты не требуют писать код, и многие из них имеют простой графический интерфейс.

Посмотрим, как с помощью CustomVision.ai создать модель, оптимизированную для мобильных устройств, менее чем за пять минут:

1. Откройте в браузере страницу <http://customvision.ai> и создайте новый проект. Поскольку обученная модель будет работать на мобильных телефонах, выберите компактный тип модели. Наш сценарий связан с едой, поэтому выберите **Food (compact)** (Еда (компактная)), как показано на рис. 12.5.
2. Выгрузите изображения и присвойте им метки, как показано на рис. 12.6. Выгрузите хотя бы по 30 изображений с каждой меткой.
3. Щелкните на кнопке **Train** (Обучить). Откроется диалоговое окно, как показано на рис. 12.7. Если выбрать вариант **Fast Training** (Быстрое обучение), то обучению подвергнется всего несколько слоев, а если выбрать вариант **Advanced Training** (Расширенное обучение), то обучению может подвергнуться вся сеть, что даст более высокую точность классификации (и, очевидно, потребует больше времени и обойдется дороже). В большинстве случаев достаточно быстрого обучения.

Create new project X

Name\*  
Not Hotdog Project

Description  
Next multi-million dollar project idea!

Resource Group create new  
Book [FO] ▼

[Manage Resource Group Permissions](#)

Project Types i  
 Classification  
 Object Detection

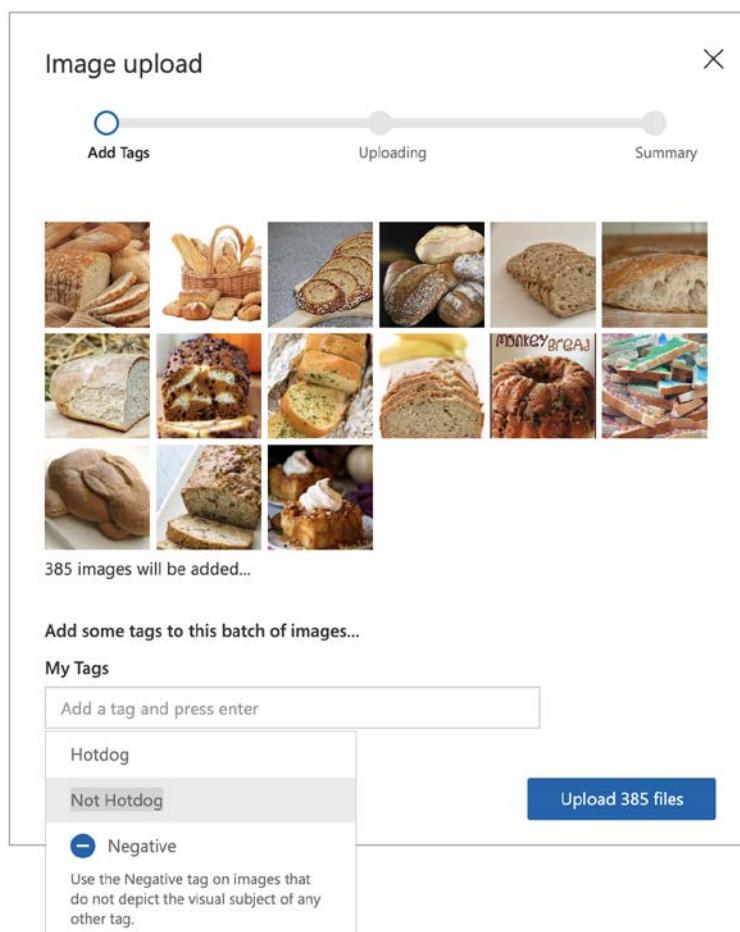
Classification Types i  
 Multilabel (Multiple tags per image)  
 Multiclass (Single tag per image)

Domains: i  
 General  
 Food  
 Landmarks  
 Retail  
 General (compact)  
 Food (compact)  
 Landmarks (compact)  
 Retail (compact)

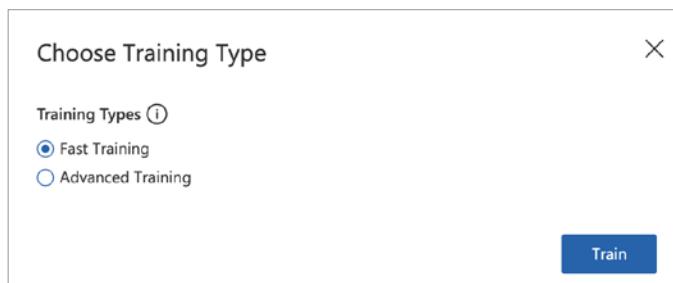
Export Capabilities: i  
 Basic platforms (Tensorflow, CoreML, ONNX, ...)  
 Vision AI Dev Kit

Cancel Create project

Рис. 12.5. Определение нового проекта в CustomVision.ai

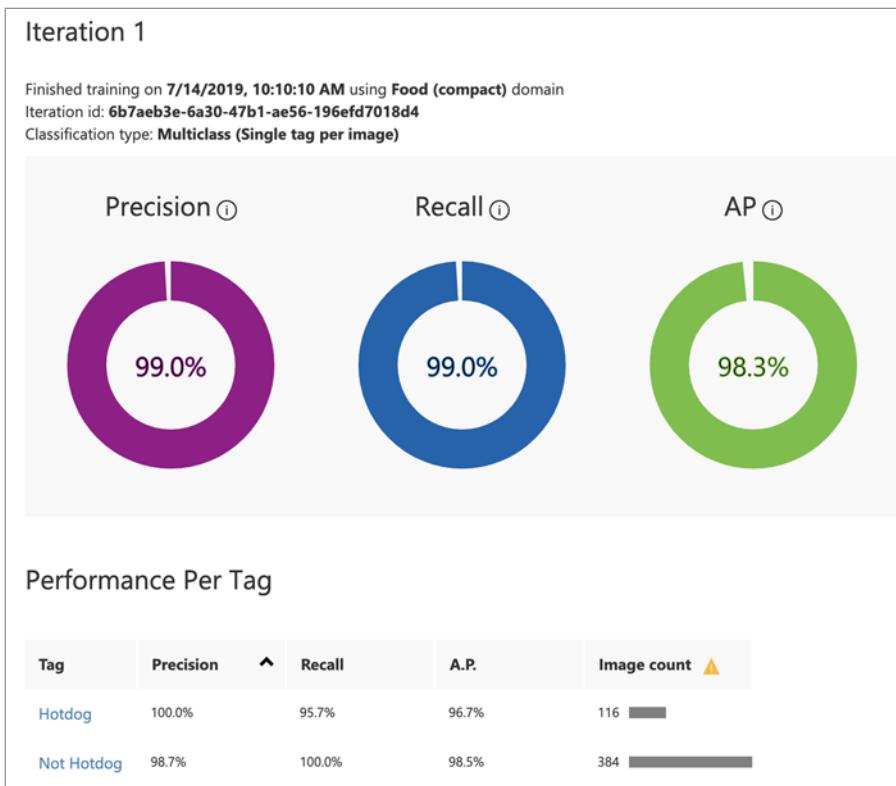


**Рис. 12.6.** Выгрузка изображений в панели управления CustomVision.ai. Обратите внимание, что используются всего две метки, «Hotdog» (хот-дог) и «Not Hotdog» (не хот-дог)



**Рис. 12.7.** Варианты обучения

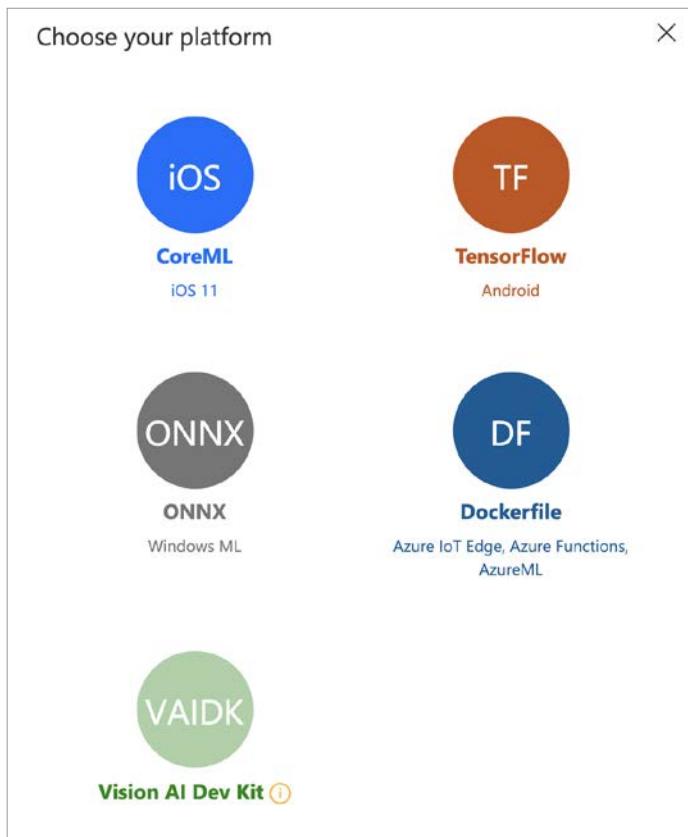
4. Менее чем через минуту должен появиться экран со значениями точности и полноты вновь обученной модели, как показано на рис. 12.8. (Точность и полноту мы уже обсуждали в этой книге.)



**Рис. 12.8.** Точность (Precision), полнота (Recall) и средняя точность (Average Precision) вновь обученной модели

5. Попробуйте поменять порог вероятности и посмотрите, как это повлияет на качество модели. Порог по умолчанию 90% дает довольно хорошие результаты. Чем выше порог, тем точнее модель, но полнота при этом уменьшается.
6. Щелкните на кнопке Export (Экспортировать) и выберите платформу iOS (рис. 12.9). После этого служба CustomVision.ai преобразует модель в формат Core ML (при выборе платформы Android модель будет преобразована в формат TensorFlow Lite).

Мы справились с задачей, не написав ни строчки кода. А теперь рассмотрим еще более удобный способ обучения без программирования.



**Рис. 12.9.** Параметры экспортования моделей, доступные в CustomVision.ai

## Подход 2: с помощью Create ML

В 2018 году Apple выпустила Create ML, чтобы разработчики могли обучать модели компьютерного зрения в экосистеме Apple. Чтобы обучить классификатор изображений, нужно открыть *песочницу* и написать несколько строк кода на Swift. Как вариант можно импортировать `CreateMLUI`, чтобы получить возможность, хотя и ограниченную, обучения в графическом интерфейсе. Это был хороший способ побудить разработчиков на Swift развертывать модели в Core ML без большого опыта в машинном обучении.

Год спустя на Всемирной конференции разработчиков Apple (Worldwide Developers Conference, WWDC) 2019 года Apple снизила порог входа еще больше, объявив о выходе автономного приложения Create ML для macOS Catalina (10.15). Оно имело простой графический интерфейс для обучения нейронных сетей без необходимости писать код. Обучение нейронной сети

сводилось к простому перетаскиванию файлов в окно приложения. Помимо классификации изображений приложение также поддерживало возможность распознавания объектов, обработку текстов на естественном языке, классификацию звуков, активности (с использованием данных с датчиков движения в Apple Watch и iPhone), а также табличных данных (включая системы рекомендаций).

И оно быстро работает! Модель можно обучить менее чем за минуту. Такая высокая скорость достигается переносом обучения, не требующего обучения всех уровней в сети. Приложение поддерживает также различные способы аугментации данных — поворот, размытие, наложение шума и т. д. Пользователю остается установить нужные флагки.

До появления Create ML считалось, что у любого желающего обучить серьезную нейронную сеть за разумное время должен быть NVIDIA GPU. Приложение Create ML способно использовать преимущества интегрированных видеокарт Intel и Radeon, что позволило ускорить обучение на MacBook без покупки дополнительного оборудования. Create ML позволяет одновременно обучать несколько моделей на разных датасетах. Оно сможет извлечь дополнительную выгоду от мощного оборудования, такого как Mac Pro или даже внешнего графического процессора (external GPU, eGPU).

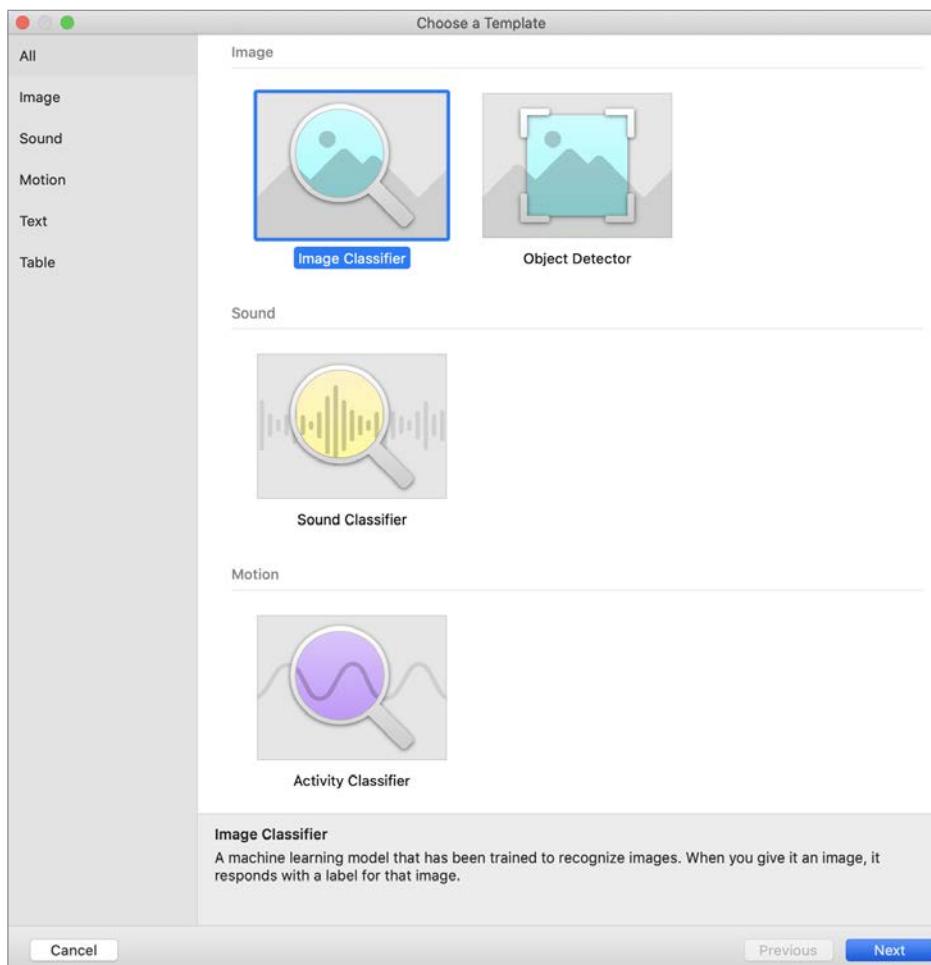
Одна из привлекательных сторон Create ML — размер итоговой модели. Полную модель можно разбить на базовую часть (которая генерирует признаки) и легковесные слои классификации, специализированные для конкретных задач. Apple включает базовые модели во все свои операционные системы. Как результат приложению Create ML остается только сгенерировать классификатор для конкретной задачи. Насколько малы эти модели? Всего несколько килобайт (по сравнению с моделью MobileNet, занимающей более 15 Мбайт, что не так много для сверточных сетей). Это особенно важно в наше время, когда все больше и больше разработчиков начинают включать глубокое обучение в свои приложения, потому что нет нужды хранить копии одних и тех же нейронных сетей в нескольких приложениях, занимающих ценное пространство в хранилище.

Проще говоря, модели в формате Create ML — простые, быстрые и занимают мало места. Звучит слишком хорошо, чтобы быть правдой. Обратной стороной полной интеграции является жесткая привязка разработчиков к экосистеме Apple. Create ML экспортирует файлы только в формате *.mlmodel*, которые можно использовать исключительно в iOS, iPadOS, macOS, tvOS и watchOS. К сожалению, интеграция с Android в Create ML еще не реализована.

В этом разделе создадим классификатор Not Hotdog с помощью Create ML:

1. Откройте приложение Create ML, щелкните на кнопке **New Document** (Новый документ) и выберите шаблон **Image Classifier** (Классификатор изобра-

жений) из числа доступных вариантов (в числе которых классификаторы звуков, активности, текста и таблиц), как показано на рис. 12.10. Обратите внимание, что все это доступно только в версии Xcode 11 (или выше), в macOS 10.15 (или выше).

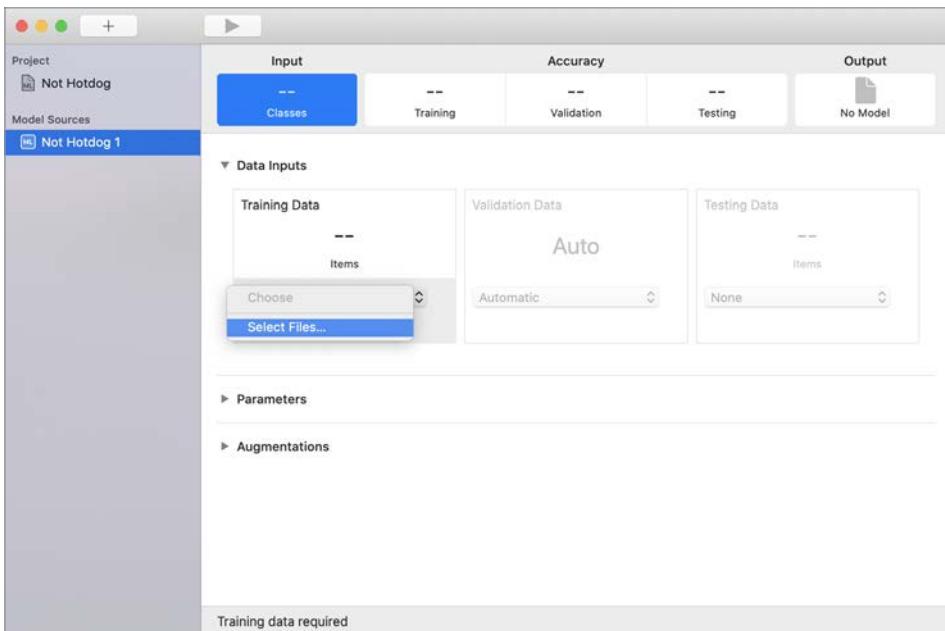


**Рис. 12.10.** Выбор шаблона для нового проекта

2. Затем введите название проекта и щелкните на кнопке **Done** (Готово).
3. Теперь нужно организовать данные в определенную структуру каталогов. Как показано на рис. 12.11, изображения следует поместить в каталоги с именами, соответствующими их меткам. Полезно разделить датасет на обучающую и контрольную выборки с соответствующими каталогами.



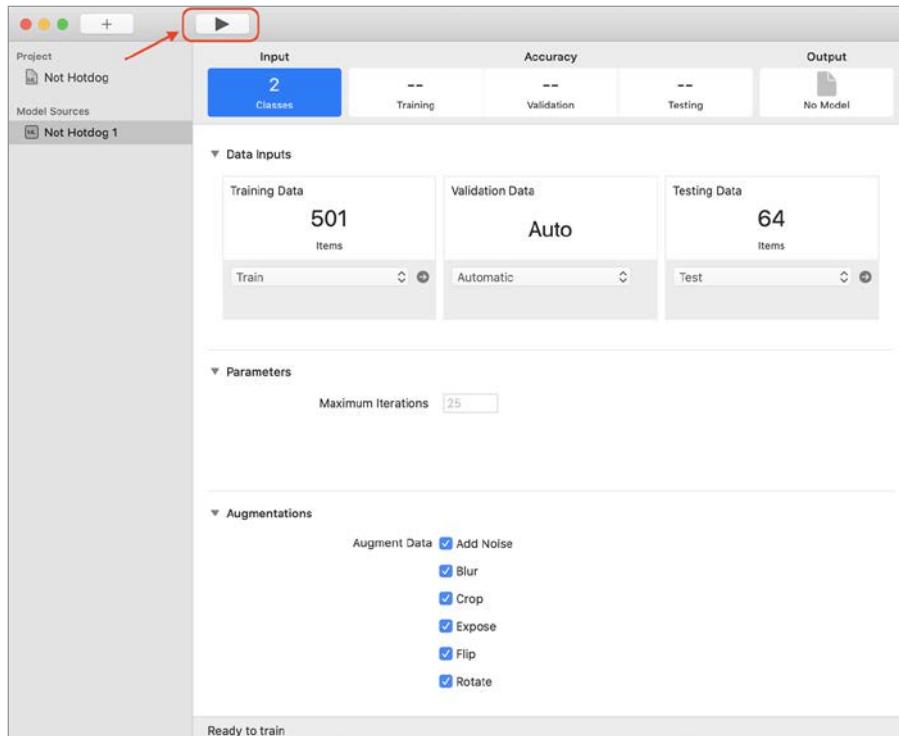
**Рис. 12.11.** Обучающая и контрольная выборки в отдельных каталогах (Train и Test соответственно)



**Рис. 12.12.** Интерфейс обучения в приложении Create ML

4. В пользовательском интерфейсе выберите каталоги с обучающими и контрольными данными, как показано на рис. 12.12.
5. На рис. 12.12 показано, как выглядит UI после выбора каталогов с обучающими и контрольными данными. Обратите внимание, что данные для проверки (Validation Data) приложение Create ML может выбирать автоматически. Также отметьте, какие варианты аугментации данных доступны (раздел Augmentations). Теперь можно щелкнуть на кнопке Play (Пуск) —

это кнопка с изображением треугольника, направленного вправо, см. рис. 12.13, — чтобы запустить процесс обучения.



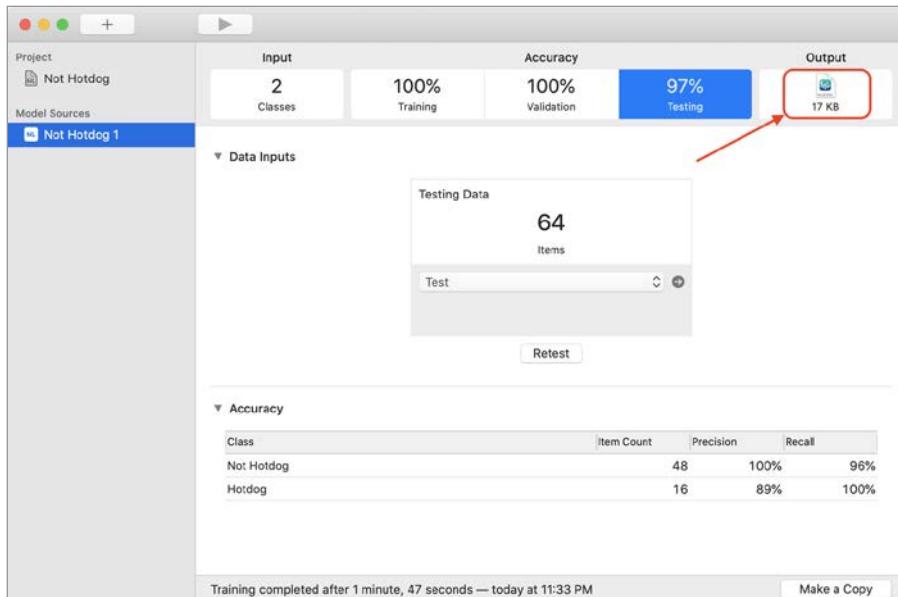
**Рис. 12.13.** Приложение Create ML после загрузки обучающих и контрольных данных



Экспериментируя, вы быстро заметите, что добавление каждого дополнительного варианта аугментации данных замедляет обучение. Чтобы быстро определить базовый уровень качества модели, в первой попытке лучше не использовать аугментацию, а потом можно поэкспериментировать, добавляя все больше и больше вариантов аугментации, чтобы оценить, как они влияют на качество модели.

6. По окончании обучения можно увидеть, насколько хорошо модель справляется с классификацией обучающих, проверочных (выбранных автоматически) и контрольных данных, как показано на рис. 12.14. В нижней части экрана выводится также продолжительность обучения и размер окончательной модели. Точность на контрольных данных составила 97 %, продолжительность обучения — менее двух минут и размер модели — 17 Кбайт. Неплохой результат.

7. Мы почти закончили, осталось лишь экспортить получившуюся модель. Перетащите мышью значок **Output** (Результат) — он обведен рамкой на рис. 12.14 — на рабочий стол, чтобы создать файл *.mlmodel*.



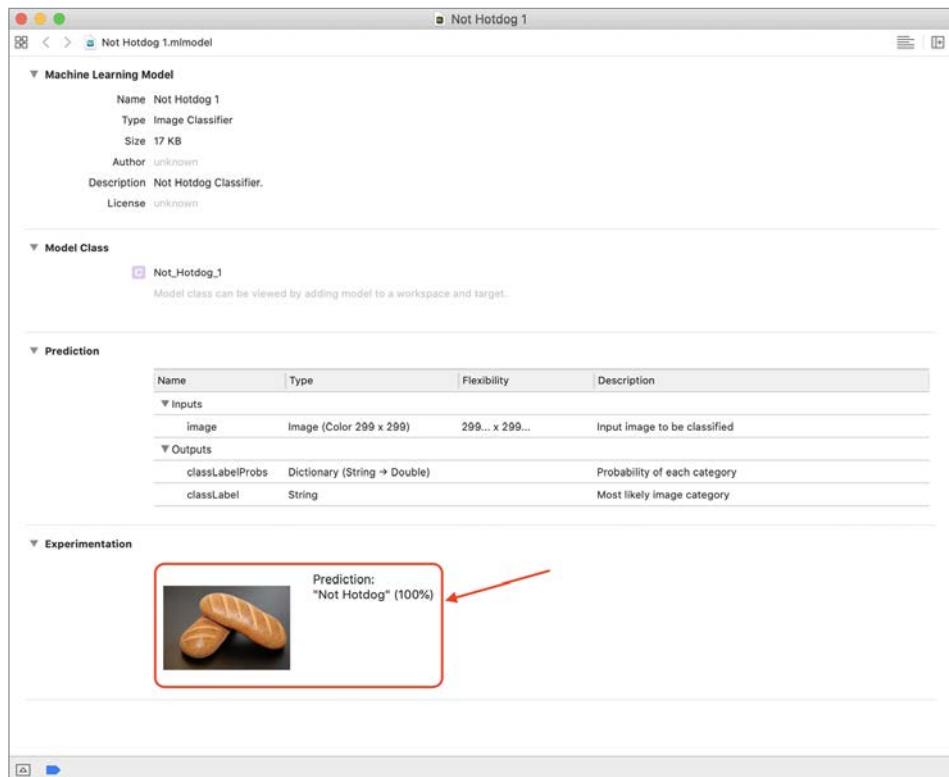
**Рис. 12.14.** Приложение Create ML после завершения обучения

8. Чтобы открыть только что экспортированный файл модели, щелкните по нему дважды. Вы можете изучить входные и выходные слои, а также протестировать модель, перетаскивая на нее изображения, как показано на рис. 12.15.

Теперь модель готова для использования на любом устройстве Apple.



Create ML использует перенос обучения и обучает только несколько последних слоев. В зависимости от варианта использования базовой модели, которую предлагает Apple, этого может быть недостаточно для получения высокоточных прогнозов. Это связано с невозможностью обучения более ранних слоев модели, что ограничивает теоретический потенциал модели. Для большинства повседневных задач это не должно быть проблемой. Но для узкоспециализированных приложений, например анализирующих рентгеновские снимки или определяющих различия между внешне очень похожими объектами, для которых важны даже мельчайшие детали (например, проверяющие банкноты), лучше обучить сверточную сеть с нуля. Рассмотрим этот подход в следующем разделе.



**Рис. 12.15.** Интерфейс инспектора моделей в Xcode

## Подход 3: тонкая настройка с использованием Keras

Вы уже приобрели немалый опыт использования Keras. Подход, о котором мы поговорим дальше, может дать еще более высокую точность, если вы готовы экспериментировать и потратить больше времени на обучение модели. Переиспользуем код из главы 3 и изменим некоторые параметры, такие как каталог и имя файла, размер пакета и количество изображений. Полный код ищите в репозитории на GitHub (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в блокноте `code/chapter-12/1-keras-custom-classifier-with-transfer-learning.ipynb`.

Обучение модели может занять несколько минут, в зависимости от мощности оборудования, и в конце обучения у вас на диске появится файл `NotHotDog.h5`.

## Конвертация модели с использованием Core ML Tools

Как отмечалось в главе 11, есть несколько способов конвертации моделей в формат Core ML.

Модели, созданные в CustomVision.ai, сразу доступны в формате Core ML, поэтому их не придется преобразовывать. Модели, обученные в Keras, можно преобразовать с помощью Core ML Tools, как описывается далее. Обратите внимание, что поскольку мы используем модель MobileNet, в которой есть нестандартный слой `relu6`, нужно импортировать `CustomObjectScope`:

```
from tensorflow.keras.models import load_model
from tensorflow.keras.utils.generic_utils import CustomObjectScope
import tensorflow.keras

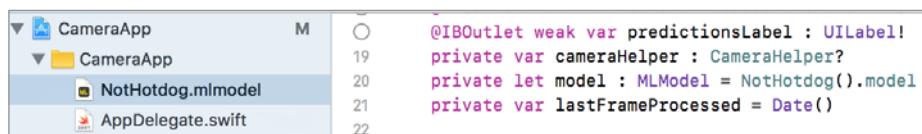
with CustomObjectScope({'relu6':
    tensorflow.keras.applications.mobilenet.relu6,'DepthwiseConv2D':
    tensorflow.keras.applications.mobilenet.DepthwiseConv2D}):
    model = load_model('NotHotDog-model.h5')

import coremltools
coreml_model = coremltools.converters.keras.convert(model)
coreml_model.save('NotHotDog.mlmodel')
```

Теперь, получив готовую к использованию модель, создадим приложение.

## Создание приложения для iOS

Для приложения используем код из главы 11 и просто заменим имя файла `.mlmodel`, как показано на рис. 12.16.



**Рис. 12.16.** Загрузка файла `.mlmodel` в Xcode

Теперь осталось скомпилировать и запустить приложение. Результат показан на рис. 12.17.



**Рис. 12.17.** Приложение правильно определило хот-дог

## Что можно сделать дальше

Можно ли сделать это приложение интереснее? Да. Можно создать настоящий «Шазам для еды», обучив модель распознавать все категории в датасете Food-101, как будет показано в следующей главе. Также можно улучшить UI и выводить не только голые проценты, как это реализовано сейчас. И наконец, чтобы приложение стало вирусным, как «Not Hotdog», реализуйте возможность делиться результатами классификации в соцсетях.

## Итоги

В этой главе мы рассмотрели все этапы пайплайна: сбор данных, обучение и преобразование модели, а также ее использование на устройстве с iOS. Для каждого этапа представили несколько вариантов реализации разной степени сложности. И мы использовали идеи из предыдущих глав для создания реального приложения.

А теперь, как Джин Ян, идите зарабатывать свои миллионы!

## ГЛАВА 13

# Шазам для еды: разработка приложений для Android с помощью TensorFlow Lite и ML Kit

Джин Ян, написавший вирусное приложение «Not Hotdog», вообще-то должен был создать классификатор, распознающий разные блюда. Приложение должно было называться SeeFood, то есть приложение, способное увидеть (see) еду (food) и узнать ее (рис. 13.1). По сути, «Шазам для еды». Тем не менее приложение оказалось слишком успешным, и его купила компания Periscope. Но первоначальная задумка инвестора Эрлиха Бахмана осталась нереализованной. В этой главе попробуем воплотить его мечту.



**Рис. 13.1.** Описание приложения «Not Hotdog» в Apple App Store

Где могла бы пригодиться такая возможность? Например, человек направляет камеру на блюдо и получает информацию о его питательной ценности, включая количество калорий. Или сканирует несколько ингредиентов и получает на их

основе рецепт. Или фотографирует продукт в магазине и узнает, содержит ли продукт какие-либо ингредиенты из черного списка — например, вызывающие аллергию.

Создание такого приложения — интересная задача, так как предполагает решение нескольких проблем:

### *Сбор данных*

В мире насчитывается более сотни национальных кухонь, каждая из которых предлагает сотни, если не тысячи блюд.

### *Точность*

В большинстве случаев приложение должно давать правильные результаты.

### *Производительность*

Результаты должны возвращаться практически мгновенно.

### *Независимость от платформы*

Приложения, работающего только на iPhone, было бы недостаточно. Многие пользователи в развивающихся странах используют менее мощные смартфоны, например устройства на Android. Независимость от платформы — наущная необходимость.

Создать приложение, классифицирующее блюда одной кухни, уже достаточно сложно. А представьте, что оно должно правильно распознавать все блюда и работать на двух платформах! Один человек или небольшая команда быстро столкнется с проблемой масштабирования. Мы используем этот пример, чтобы показать важность изучения различных этапов жизненного цикла разработки мобильного ИИ, которые мы рассмотрели ранее в главе 11.

Материал этой главы относится не только к смартфонам. Полученные знания вы сможете применить также к краевым (edge) устройствам — Google Coral и Raspberry Pi, которые мы обсудим далее в этой книге.

## **Цикл разработки приложения для классификации блюд**

Итак, наша цель — создать всеобъемлющий мультиплатформенный классификатор блюд и продуктов питания. Это кажется непростой задачей, но мы можем разбить ее на управляемые этапы. Так же как в жизни, сначала нужно научиться ползать, потом ходить, а потом бегать. Далее приводится один из возможных подходов:

1. Собрать небольшой начальный набор изображений для одной кухни (например, итальянской).
2. Снабдить эти изображения соответствующими метками — названиями блюд (например, `margherita_pizza`).
3. Обучить модель классификатора.
4. Конвертировать модель в формат, совместимый с мобильной платформой (например, `.tflite`).
5. Создать мобильное приложение с удобным интерфейсом и интегрировать в него модель.
6. Пригласить первых пользователей и поделиться с ними приложением.
7. Собрать информацию о вариантах использования вместе с отзывами активных юзеров, включая кадры с камеры (которые отражают реальное использование) и соответствующие метки (указывающие на правильность или неправильность классификации).
8. Улучшить модель, используя собранные изображения в качестве дополнительных обучающих данных. Этот пункт придется повторить неоднократно.
9. Когда качество модели достигнет минимально приемлемого уровня, передать приложение более широкому кругу пользователей. Продолжать наблюдать за качеством модели для этой кухни.
10. Повторить перечисленные этапы для каждой кухни.



Для этапа 7 можно интегрировать поддержку обратной связи в сам UX. Например, приложение может отображать ранжированный список прогнозов (по величине вероятности) для изображения. Если модель работает надежно, пользователь в большинстве случаев будет выбирать первый вариант. Выбор прогноза с более низким рейтингом должен расцениваться как ошибка модели. В худшем случае, если все предложенные варианты ошибочны, у пользователя должна быть возможность добавить новую метку вручную. Фотографии вместе с метками (во всех трех сценариях) можно использовать в качестве обучающих данных.

Для начала не нужно много данных. Перечисленные этапы могут показаться сложными, но все это можно автоматизировать. Самое замечательное, что чем шире используется приложение, тем лучше оно становится, причем автоматически. Как будто живет своей жизнью. Рассмотрим этот подход саморазвития модели в конце главы.

В этой главе рассмотрим этапы цикла разработки и инструменты, которые помогут на каждом из этапов. Мы затронем весь цикл разработки мобильных приложений не только из этой, но и из предыдущих глав, и покажем, как эффективно использовать эти этапы для создания высококачественных приложений.



Хорошими первыми пользователями могут стать фанаты вашего приложения или компании. В идеале первыми пользователями должны быть люди, заинтересованные в успехе продукта. Для приложения классификации блюд это могут быть любители фитнеса, следящие за каждой съедаемой калорией и ингредиентом. Они понимают, что на раннем этапе приложение не будет отличаться высокой точностью классификации, но они также осознают свою роль в его развитии через обратную связь. Они добровольно согласятся на передачу и использование фотографий своих блюд. Рекомендуем сообщить пользователям, какую информацию от них вы будете собирать, и дать возможность отказаться от участия или удалить свою информацию. Не пугайте их.

Начнем с более близкого знакомства со следующими инструментами экосистемы Google.

### *TensorFlow Lite*

Инструмент для конвертации моделей и мобильного инференса.

### *ML Kit*

Набор инструментов высокого уровня для разработки ПО с несколькими встроенными API, который позволяет запускать пользовательские модели TensorFlow Lite и может интегрироваться с Firebase в Google Cloud.

### *Firebase*

Облачная платформа, обеспечивающая высококачественную инфраструктуру для мобильных приложений с поддержкой аналитики, отчетов о сбоях, А/В-тестирования, пуш-уведомлений и многое другое.

### *Набор инструментов TensorFlow для оптимизации моделей*

Набор инструментов для оптимизации размера и производительности моделей.

## **Обзор TensorFlow Lite**

Как упоминалось в главе 11, Google выпустила TensorFlow Lite — инструмент для инференса на мобильных устройствах, который расширил охват экосистемы TensorFlow за рамки облачных окружений и настольных компьютеров. До этого в экосистеме TensorFlow поддерживался единственный вариант — использование полной библиотеки TensorFlow для iOS (которая была тяжелой и медленной), а позже — ее слегка урезанной версии под названием TensorFlow Mobile (имеющей улучшенные характеристики, но все еще довольно громоздкой).

Библиотека TensorFlow Lite создавалась с нуля специально для мобильных устройств и имеет следующие характеристики.

### *Маленький размер*

В состав TensorFlow Lite входит гораздо более легковесный интерпретатор. Даже с полным набором операторов интерпретатор занимает меньше 300 Кбайт. При типичном использовании с распространенными моделями, например MobileNet, можно ожидать, что занимаемый объем не превысит 200 Кбайт. Для справки: предыдущая версия — библиотека TensorFlow Mobile — занимала 1,5 Мбайт. Кроме того, в TensorFlow Lite используется *выборочная регистрация* — в библиотеку упаковываются только те операции, которые будут использоваться моделью, что позволяет оптимизировать размер библиотеки до необходимого минимума.

### *Высокая скорость работы*

У TensorFlow Lite большая скорость работы, так как она использует преимущества *аппаратного ускорения* на устройстве, например GPU и NPU, если они доступны. В экосистеме Android библиотека использует Android Neural Networks API, а на iPhone — Metal API. По заявлению Google, GPU дает ускорение от двух до семи раз при решении некоторых задач (по сравнению с выполнением на CPU).

Для сериализации и десериализации TensorFlow использует Protocol Buffers (Protobufs). Protobuf — это мощный инструмент представления данных, обладающий большой гибкостью и расширяемостью. Но за эту гибкость приходится платить потерей производительности, которая ощущается на устройствах с низким энергопотреблением, например смартфонах.

Решить эту проблему можно с помощью *FlatBuffers*. Первоначально созданная для разработки видеоигр, где важны низкие издержки и высокая производительность, эта библиотека оказалась хорошим решением для мобильных устройств, позволившем к тому же значительно сократить объем кода, а также использование памяти и процессора, расходуемые на сериализацию и десериализацию моделей. Это сократило и время запуска.

Внутри нейронной сети есть несколько слоев, выполняющих фиксированные вычисления во время инференса, например слои пакетной нормализации. Эти вычисления можно выполнить заранее, так как они основаны на значениях, полученных во время обучения. Это среднее и стандартное отклонение. То есть вычисления в слое пакетной нормализации можно совместить (объединить) с вычислениями в предыдущем слое (и выполнить во время преобразования модели) и тем самым сократить время инференса и ускорить работу модели в целом. Этот прием известен как *предварительная активация* и поддерживается библиотекой TensorFlow Lite.

Интерпретатор использует статическую память и статический план выполнения. Это сокращает время загрузки модели.

## *Меньше зависимостей*

TensorFlow Lite в основном состоит из типичного кода на C/C++ с минимальным количеством зависимостей. Это упрощает упаковку и развертывание, а также уменьшает размер развернутой библиотеки.

## *Поддержка пользовательских операторов*

Библиотека TensorFlow Lite поддерживает операторы квантования и с плавающей запятой, многие из которых настроены с учетом особенностей мобильных платформ и могут применяться для создания и запуска пользовательских моделей. Если нашей модели нужна какая-то операция, которая не поддерживается в TensorFlow Lite, можно реализовать эту операцию и подключить к библиотеке.

## **От создателя**

В 2013 году я написал свой первый инструмент для инференса с нейронной сетью, потому что не нашел другого способа использовать модели распознавания изображений на недорогих серверах AWS – единственное, что мы тогда могли себе позволить. По счастливой случайности один и тот же код можно было собрать для iOS и Android, и я быстро понял, что возможности глубокого обучения на мобильных платформах практически безграничны. Этот код лег в основу библиотеки Jetpac SDK, и мне было очень приятно наблюдать за появлением новых приложений, использующих ее. Когда наш стартап был куплен компанией Google, я встретил Янцина Цзя (YangqingJia), автора Caffe, который работал над внутренним проектом инференса на мобильных устройствах, так что в итоге мы начали сотрудничать, и многое из того, что мы узнали, попало в релиз TensorFlow. Многие люди скептически оценивали важность нейронных сетей для телефонов, но Джекффи Дин (Jeff Dean) всегда был их твердым сторонником и помог реализовать необходимую поддержку сначала в Android, а потом в iOS. Мы сами многому научились, работая над первыми релизами TensorFlow, и команда Mobile Vision в Google тоже добилась потрясающих успехов со своим специализированным внутренним фреймворком, поэтому мы объединили усилия и создали TensorFlow Lite – библиотеку, совместимую с TensorFlow, но гораздо меньшего размера.

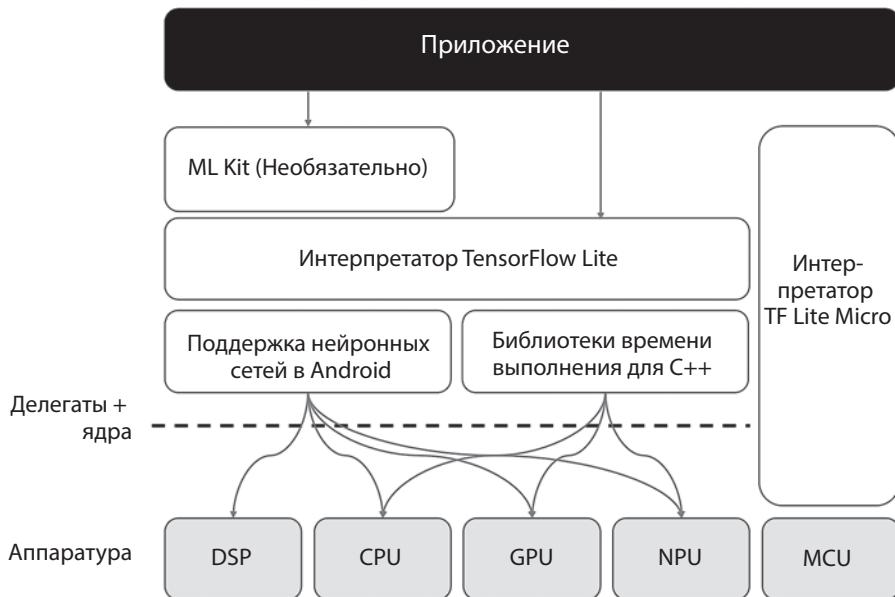
TensorFlow Lite была установлена на более чем двух миллиардах устройств, но ничего из этого не было бы возможно без сообщества, выросшего вокруг нее. Было удивительно приятно получать обратную связь, примеры и поддержку со стороны коллег. Все это помогало людям создавать приложения, невозможные без экосистемы, работающей как хорошо отлаженный механизм.

Пит Уорден (Pete Warden), технид отдела мобильных и встраиваемых приложений TensorFlow, автор книги «TinyML» (O'Reilly, <http://shop.oreilly.com/product/0636920254508.do>)

Прежде чем приступать к созданию приложения для Android, полезно изучить архитектуру TensorFlow Lite.

## Архитектура TensorFlow Lite

На рис. 13.2 изображена общая структура библиотеки TensorFlow Lite.



**Рис. 13.2.** Общая структура экосистемы TensorFlow Lite

Мы как разработчики приложений будем работать на самом верхнем уровне, взаимодействуя с TensorFlow Lite API (или с набором инструментов ML Kit, который, в свою очередь, использует TensorFlow Lite). TensorFlow Lite API скрывает за своим фасадом все сложности, связанные с использованием API нижнего уровня, такими как поддержка нейронных сетей в Android (Android Neural Network API). Эта структура напоминает структуру Core ML в экосистеме Apple.

С другой стороны, вычисления могут выполняться на различных аппаратных процессорах. Самый распространенный — CPU из-за его доступности и гибкости. Современные смартфоны все чаще оснащаются специализированными процессорами, GPU и процессором нейронных сетей NPU (специально созданным для вычислений в нейронных сетях, как на iPhone X). Дополнительно могут использоваться процессоры цифровых сигналов (Digital Signal Processor, DSP), специализирующиеся на аутентификации по лицу или отпечатку пальца и распознавании фразы активизации (например, «Привет, Сири»).

В мире интернета вещей (Internet of Things, IoT) широко используются микроконтроллеры (Microprocessor Control Unit, MCU). Они не имеют операционной системы и отдельного процессора, оснащаются очень небольшим объемом памяти (единицы килобайт), дешевы в массовом производстве и легко встраиваются в различные устройства. С помощью TensorFlow Lite для микроконтроллеров разработчики могут запускать ИИ на таких устройствах без подключения к интернету. Урезанная версия интерпретатора TensorFlow Lite (примерно 20 Кбайт) для микроконтроллеров называется TensorFlow Lite Micro Interpreter.

Но как TensorFlow Lite взаимодействует с оборудованием? Для этого используются делегаты — объекты, учитывающие аппаратные особенности платформы, которые предоставляют единообразный и независимый от платформы API. Иначе говоря, делегаты скрывают от интерпретатора внутренние детали конкретного оборудования, на котором он работает. Они берут на себя всю или почти всю ответственность за выполнение графа операций, которые в противном случае выполнялись бы на CPU вместо более эффективных GPU и NPU. В Android делегат GPU ускоряет производительность с помощью OpenGL, а в iOS использует Metal API.

Учитывая, что TensorFlow Lite сама по себе не зависит от платформы, она должна вызвать библиотеку для конкретной платформы, реализующую известный контракт — контракт делегатов для TensorFlow Lite (TensorFlow Lite Delegate API). В Android этот контракт реализует поддержка нейронных сетей (Android Neural Network API, доступен в Android 8.1 и выше). Поддержка нейронных сетей обеспечивает базовый уровень функциональных возможностей для фреймворков машинного обучения более высокого уровня. Эквивалентом API нейронной сети в мире Apple являются шейдеры Metal Performance Shaders.

Познакомившись с теорией, перейдем к практике.

## Конвертация модели в формат TensorFlow Lite

Сейчас у нас есть модель, предварительно обученная на датасете ImageNet в Keras. Прежде чем использовать ее в приложении для Android, конвертируем модель в формат TensorFlow Lite (файл *.tflite*).

Посмотрим, как это сделать с помощью инструмента TensorFlow Lite Converter, команды `tflite_convert`:

```
# Из формата Keras в формат TensorFlow Lite
$ tflite_convert \
  --output_file=my_model.tflite \
  --keras_model_file=my_model.h5

# Из формата TensorFlow в формат TensorFlow Lite
$ tflite_convert \
  --output_file=my_model.tflite \
  --graph_def_file=my_model/frozen_graph.pb
```

Результатом будет новый файл *my\_model.tflite*, который можно подключить к приложению для Android, как показано в следующем разделе. Позже мы посмотрим, как увеличить производительность этой модели с помощью все того же инструмента *tflite\_convert*. Кроме того, разработчики TensorFlow Lite создали множество предварительно обученных моделей, доступных в формате TensorFlow Lite, что избавляет нас от необходимости останавливаться на этом шаге.

## Создание приложения для распознавания объектов

Самый простой способ поэкспериментировать с TensorFlow Lite — запустить пример приложения из репозитория TensorFlow. Обратите внимание, что для этого понадобится телефон или планшет на Android. Вот как создать и развернуть приложение:

1. Клонируйте репозиторий TensorFlow:

```
git clone https://github.com/tensorflow/tensorflow.git
```

2. Загрузите и установите Android Studio: <https://developer.android.com/studio>.
3. Запустите Android Studio и выберите Open an existing Android Studio project (Открыть существующий проект Android Studio; рис. 13.3).

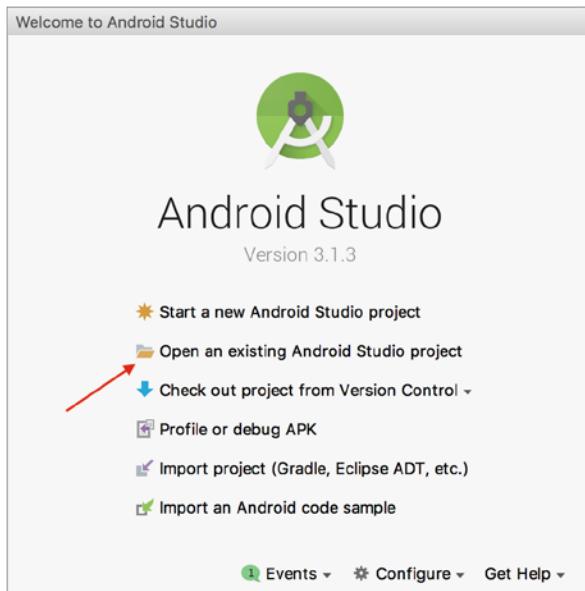
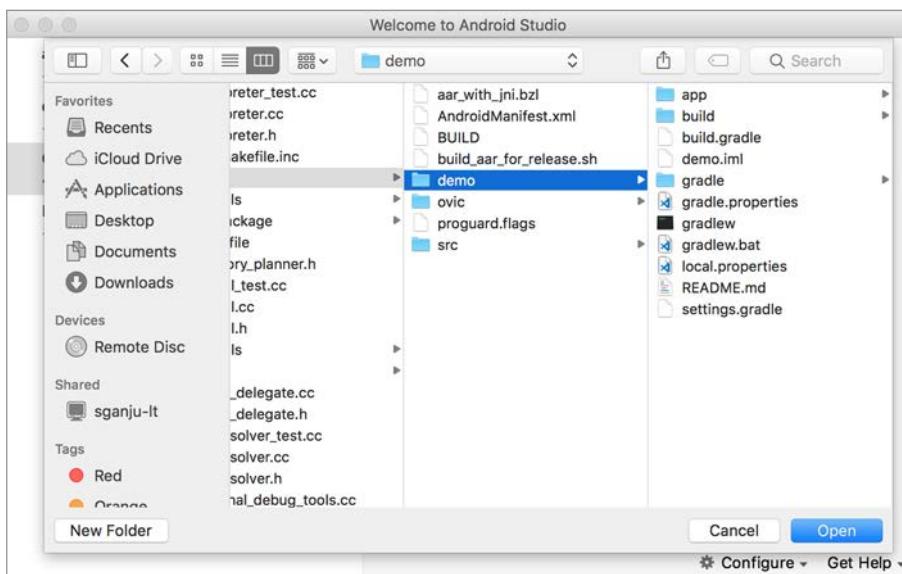


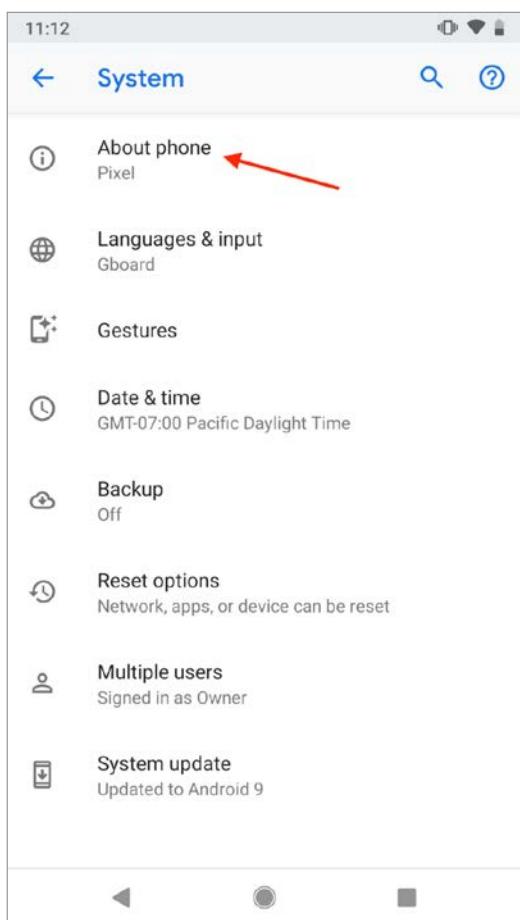
Рис. 13.3. Начальный экран Android Studio

4. Перейдите в каталог, куда вы клонировали репозиторий TensorFlow, а затем зайдите в каталог *tensorflow/tensorflow/contrib/lite/java/demo/* (рис. 13.4). Щелкните на кнопке Open (Открыть).

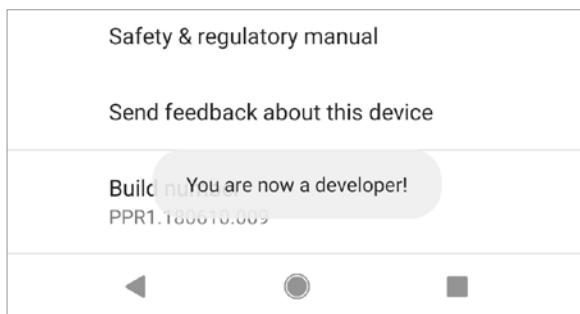


**Рис. 13.4.** Диалог «Open Existing Project» (Открыть существующий проект) в Android Studio

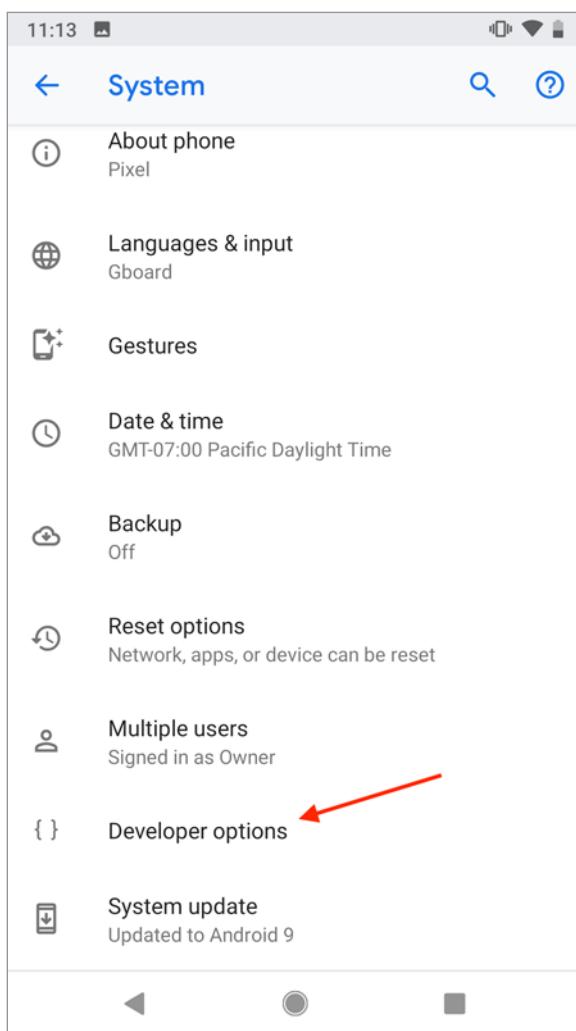
5. На устройстве Android включите режим разработчика. (Обратите внимание, что здесь мы использовали смартфон Pixel от Google со стандартной версией ОС Android. Активация режима разработчика на других устройствах может отличаться.)
- Перейдите в настройки.
  - Прокрутите экран вниз до параметра **About Phone** (О телефоне) или **About Tablet** (О планшете), как показано на рис. 13.5, и коснитесь его.
  - Найдите строку **Build Number** (Номер сборки) и коснитесь ее семь раз. (Да, все правильно, именно семь раз.)
  - Вы должны увидеть сообщение (рис. 13.6), подтверждающее включение режима разработчика.
  - Если вы используете телефон, коснитесь значка **Back** (Назад), чтобы вернуться в предыдущее меню.
  - Вы увидите новый пункт **Developer options** (Для разработчиков) прямо над строкой **About Phone** (О телефоне) или **About Tablet** (О планшете; рис. 13.7). Коснитесь этого пункта, чтобы открыть меню **Developer options** (рис. 13.8).



**Рис. 13.5.** Экран с информацией о системе на телефоне с Android; выберите параметр About Phone (О телефоне)

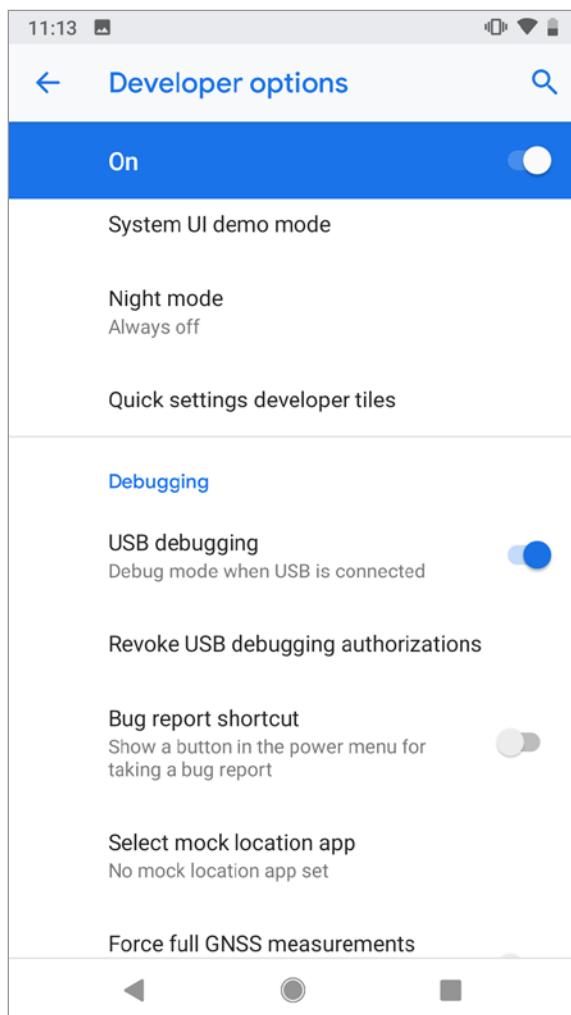


**Рис. 13.6.** Экран «About Phone» (О телефоне) на устройстве с Android



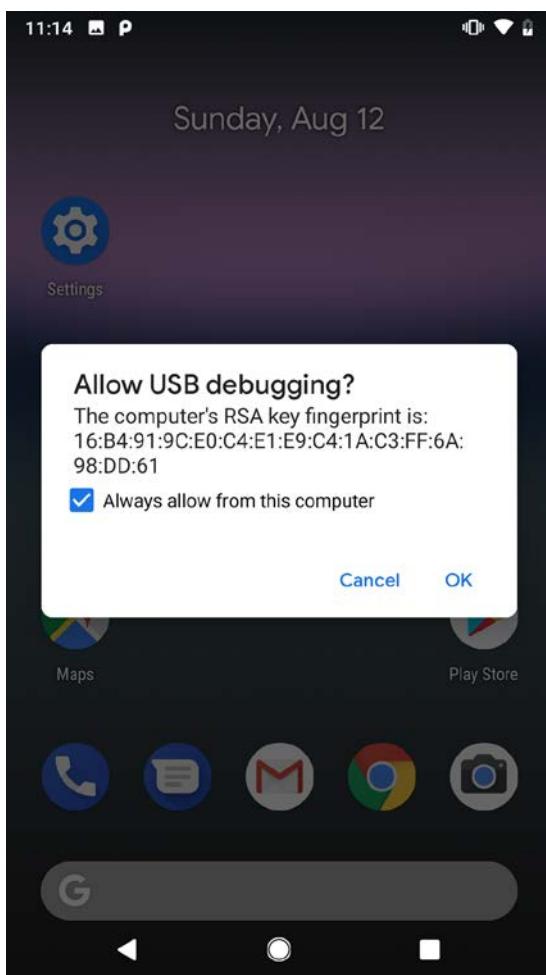
**Рис. 13.7.** Экран с информацией о системе, содержащий пункт Developer options

6. Подключите устройство к компьютеру через кабель USB.
7. На устройстве может появиться запрос на разрешение отладки по USB. Установите флажок Always allow this computer (Всегда разрешать с этого компьютера) и коснитесь кнопки OK (рис. 13.9).
8. В Android Studio в панели инструментов Debug (Отладка) щелкните на кнопке Run App (Запустить приложение) с изображением зеленого треугольника (рис. 13.10).



**Рис. 13.8.** Экран Developer options на устройстве с Android с включенной поддержкой отладки по USB

9. В открывшемся диалоге, где отображаются все доступные устройства и эмуляторы (рис. 13.11), выберите свое устройство и щелкните на кнопке **OK**.
10. После этого приложение должно быть установлено и запущено на телефоне.
11. Приложение запросит разрешение на доступ к камере — дайте такое разрешение.

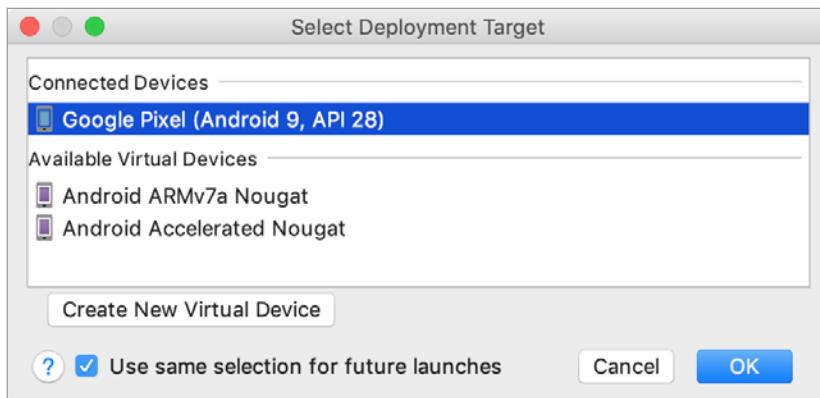


**Рис. 13.9.** Разрешите отладку по USB в ответ на запрос



**Рис. 13.10.** Панель инструментов Debug в Android Studio

12. После этого на экране появится изображение, поступающее с камеры, вместе с результатами классификации объектов, а также информация о времени, которое потребовалось для вычисления прогноза, как показано на рис. 13.12.



**Рис. 13.11.** Выберите телефон в диалоге выбора устройства для развертывания приложения



**Рис. 13.12.** Приложение отображает прогнозы в режиме реального времени

Теперь у вас есть простое приложение, работающее на телефоне, которое получает видеофреймы и классифицирует их. Приложение несложное и показывает хорошие результаты.

В репозитории TensorFlow Lite есть и другие приложения (для iOS и Android), способные решать множество других задач с использованием ИИ, в том числе:

- обнаружение объектов;
- определение позы;
- распознавание жестов;
- распознавание речи.

Следуя простым инструкциям, любой, даже не имеющий опыта разработки мобильных приложений, сможет запустить их на телефоне. А при наличии собственной обученной модели ее можно подключить к приложению и увидеть, как она справляется с вашей конкретной задачей.

Это все хорошо для начала. Но реальный мир намного сложнее. Разработчикам серьезных приложений с тысячами или даже миллионами пользователей нужно думать не только об интерфейсе, но и об обновлении и распространении моделей, тестировании различных версий среди подгрупп пользователей, поддержании паритета между iOS и Android и снижении затрат на разработку. Все это может потребовать много времени и средств, да и, откровенно говоря, совершенно не нужно — ведь уже есть платформы, предоставляющие все это: ML Kit и Firebase.

## ML Kit + Firebase

ML Kit — это комплект инструментов для разработки мобильных приложений, о релизе которого было объявлено на конференции Google I/O 2018. Он включает библиотеки, удобные для начинающих и опытных разработчиков, позволяющие решать широкий круг типичных задач МО. По умолчанию в состав ML Kit входят универсальные средства для компьютерного зрения и анализа текста. В табл. 4.1 перечислены некоторые из типичных задач машинного обучения, которые можно решить, написав всего несколько строк кода.

**Таблица 13.1.** Встроенные возможности ML Kit

Компьютерное зрение	Анализ текста
Классификация объектов	Определение языка
Определение и отслеживание объектов	Перевод с одного языка на другой на устройстве

**Таблица 13.1** (окончание)

Компьютерное зрение	Анализ текста
Определение популярных достопримечательностей	Умные ответы
Распознавание текста	
Определение лиц	
Определение штрих-кодов	

Кроме того, ML Kit позволяет использовать обученные модели TensorFlow Lite. Посмотрим, насколько удобен этот набор инструментов для разработчиков. Представьте, что мы создаем сканер визиток. Можно создать модель обнаружения визитных карточек, которая определяет момент появления визитной карточки в поле зрения камеры и границы визитки (для отображения в пользовательском интерфейсе), запустить другую модель для распознавания текста и отфильтровывать текст, выходящий за эти границы, чтобы исключить из анализа посторонние символы. Или представьте создание игры для изучения иностранного языка, которой можно показывать объекты, а та с помощью классификатора объектов определяет их названия и, используя возможность перевода на устройстве, выводит эти названия на французском языке. Такие приложения можно относительно быстро создать с помощью ML Kit.

Конечно, многие из этих функций доступны и в Core ML, но у ML Kit есть дополнительное преимущество — кроссплатформенность.

Но ML Kit — это лишь часть мозаики. Он интегрируется в Google Firebase, платформу разработки мобильных и веб-приложений, которая является частью Google Cloud и формирует инфраструктуру, которая требуется приложениям промышленного качества. Помимо прочего она поддерживает:

- пуш-уведомления;
- аутентификацию;
- отчеты о сбоях;
- журналирование;
- мониторинг производительности;
- тестирование устройств, используемых для развертывания приложений;
- А/В-тестирование;
- управление моделями.

Последний пункт очень актуален для нас. Одно из самых больших преимуществ Firebase — возможность размещения своих моделей в облаке и их загрузки по

мере необходимости. Просто скопируйте модели в Firebase в Google Cloud, добавьте ссылки на модели в приложение — и все! Поддержка A/B-тестирования позволяет передавать разным пользователям разные версии одной и той же модели и оценивать качество их работы.



Для многих встроенных функций ML Kit предлагает обработку изображений в облаке, где модели будут намного больше, чем модели на устройстве. Очевидно, что более крупным моделям требуется более мощное оборудование для работы и они способны обеспечить более высокую точность классификации с возможно более широким спектром классов (например, они могут быть способны распознавать тысячи классов вместо сотен). Фактически некоторые функции, например функция распознавания достопримечательностей, работают только в облаке.

Обработка в облаке особенно полезна, когда требуется дополнительная точность и/или телефон пользователя имеет низкую вычислительную мощность, что не позволяет приложению использовать модель на устройстве.

## Классификация объектов в ML Kit

Вернемся к нашей задаче классификации объектов в масштабе реального времени. Если вместо обычной библиотеки TensorFlow Lite использовать ML Kit, то код можно сократить до нескольких строк (на языке Kotlin):

```
val image = FirebaseVisionImage.fromBitmap(bitmap)
val detector = FirebaseVision.getInstance().visionLabelDetector
val result = detector.detectInImage(image).addOnSuccessListener { labels ->
    // Вывод меток
}
```

## Использование своих моделей в ML Kit

В дополнение к готовым моделям, входящим в состав ML Kit, можно использовать свои модели. Эти модели должны быть в формате TensorFlow Lite. Вот фрагмент простого кода для загрузки своей модели в приложение:

```
val customModel = FirebaseLocalModelSource.Builder("my_custom_model")
    .setAssetFilePath("my_custom_model.tflite").build()
FirebaseModelManager.getInstance().registerLocalModelSource(customModel)
```

После этого нужно указать конфигурацию входа и выхода модели (в данном случае модель принимает изображение RGB размером  $224 \times 224$  и возвращает вероятности принадлежности к каждому из 1000 классов):

```
val IMAGE_WIDTH = 224
val IMAGE_HEIGHT = 224
val modelConfig = FirebaseModelInputOutputOptions.Builder()
```

```
.setInputFormat(0, FirebaseModelDataType.FLOAT32, intArrayOf(1,
IMAGE_WIDTH, IMAGE_HEIGHT, 3))
.setOutputFormat(0, FirebaseModelDataType.FLOAT32, intArrayOf(1, 1000))
.build()
```

Создать массив с единственным изображением и нормализовать каждый пиксель, приведя его значение в диапазон  $[-1,1]$ :

```
val bitmap = Bitmap.createScaledBitmap(image, IMAGE_WIDTH, IMAGE_HEIGHT, true)
val input = Array(1) {
    Array(IMAGE_WIDTH) { Array(IMAGE_HEIGHT) { FloatArray(3) } }
}

for (x in 0..IMAGE_WIDTH) {
    for (y in 0..IMAGE_HEIGHT) {
        val pixel = bitmap.getPixel(x, y)
        input[0][x][y][0] = (Color.red(pixel) - 127) / 128.0f
        input[0][x][y][1] = (Color.green(pixel) - 127) / 128.0f
        input[0][x][y][2] = (Color.blue(pixel) - 127) / 128.0f
    }
}
```

Настроить интерпретатор модели:

```
val options = FirebaseModelOptions.Builder()
    .setLocalModelName("my_custom_model").build()
val interpreter = FirebaseModelInterpreter.getInstance(options)
```

И наконец, передать пакет с входным изображением в интерпретатор:

```
val modelInputs = FirebaseModelInputs.Builder().add(input).build()
interpreter.run(modelInputs, modelConfig).addOnSuccessListener { result ->
    // Вывод результатов
}
```

И правда просто! Здесь мы показали, как встроить свою модель в приложение. Но иногда нужно, чтобы приложение динамически загружало модель из облака, чтобы, например:

- сократить размер приложения, чтобы пользователи с ограниченными возможностями загрузки по мобильной сети могли загружать его из Play Store;
- поэкспериментировать с различными моделями и выбрать лучшую по полученным результатам;
- пользователь всегда получал доступ к самой свежей и лучшей модели, а нам не надо было каждый раз делать релиз приложения;
- сэкономить место на устройстве пользователя, если функция приложения, которой нужна модель, необязательна.

Посмотрим, как можно использовать модели, размещенные в облаке.

## Модели, размещенные в облаке

ML Kit и Firebase позволяют выгружать и хранить свои модели в Google Cloud и загружать их в приложение по мере необходимости. После загрузки модель работает точно так же, как если бы была частью приложения. Кроме того, такой подход позволяет обновлять модели без необходимости каждый раз делать новый релиз и проводить эксперименты с моделями, чтобы понять, какие из них лучше работают в реальном мире.

Для моделей, которые размещаются в облаке, рассмотрим два момента.

### Доступ к модели в облаке

Эти строки кода говорят Firebase о том, что приложение собирается использовать модель `my_remote_custom_model`:

```
val remoteModel = FirebaseCloudModelSource.Builder("my_remote_custom_model")
    .enableModelUpdates(true).build()
FirebaseModelManager.getInstance().registerCloudModelSource(remoteModel)
```

Обратите внимание, что мы установили признак `enableModelUpdates`, разрешающий отправлять обновления модели из облака в устройство. Так же можно настроить условия, при которых модель будет загружаться в первый раз: когда устройство находится в режиме ожидания, заряжается и/или подключено к WiFi.

Затем настроим интерпретатор, так же как в случае с локальной моделью:

```
val options = FirebaseModelOptions.Builder()
    .setCloudmodelName("my_remote_custom_model").build()
val interpreter = FirebaseModelInterpreter.getInstance(options)
```

Далее следует код, использующий модель для прогнозирования, который выглядит точно так же, как код, использующий локальную модель.

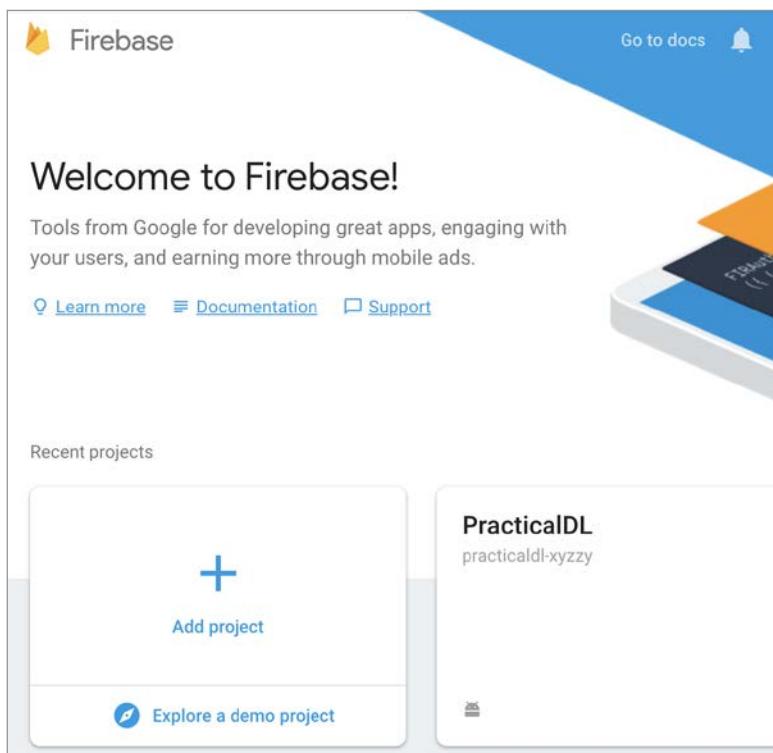
Рассмотрим другой момент, связанный с размещением моделей в облаке, — выгрузку модели.

### Выгрузка модели в облако

На момент написания этих строк Firebase поддерживала размещение моделей только в GCP (Google Cloud Platform). В этом разделе рассмотрим простой процесс создания, выгрузки и сохранения модели в облаке. Будем предполагать, что у вас уже есть своя учетная запись GCP.

Вот список шагов, которые нужно сделать, чтобы сохранить модель в облаке:

1. Откройте в браузере страницу <https://console.firebaseio.google.com>. Выберите существующий проект или создайте новый (рис. 13.13).



**Рис. 13.13.** Домашняя страница Google Cloud Firebase

2. На странице Project Overview (Обзор проекта) создайте приложение для Android (рис. 13.14).
3. Скопируйте идентификатор проекта из Android Studio, как показано на рис. 13.15.
4. Щелкните на Register app (Зарегистрировать приложение) и загрузите файл конфигурации. В этом файле находятся учетные данные, необходимые приложению для доступа к вашей облачной учетной записи. Добавьте файл конфигурации и Firebase SDK в приложение для Android, как показано на странице создания приложения.
5. В разделе ML Kit выберите Get Started (Начать) и щелкните на кнопке Add custom model (Добавить свою модель), как показано на рис. 13.16.
6. В поле имени введите имя модели `my_remote_custom_model`, которое используется в коде.
7. Выгрузите файл модели (рис. 13.17).
8. После завершения выгрузки щелкните на кнопке Publish (Опубликовать).



Рис. 13.14. Страница Project Overview в Google Cloud Firebase

The dialog is titled 'Add Firebase to your Android app'. It has a step indicator '1 Register app'. The first field is 'Android package name' with the value 'com.practicaldl.objectclassifier'. The second field is 'App nickname (optional)' with the value 'Object Classifier'. The third field is 'Debug signing certificate SHA-1 (optional)' containing a long string of zeros. A note below says: 'Required for Dynamic Links, Invites, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.' At the bottom is a 'Register app' button.

Рис. 13.15. Диалог создания приложения в Firebase

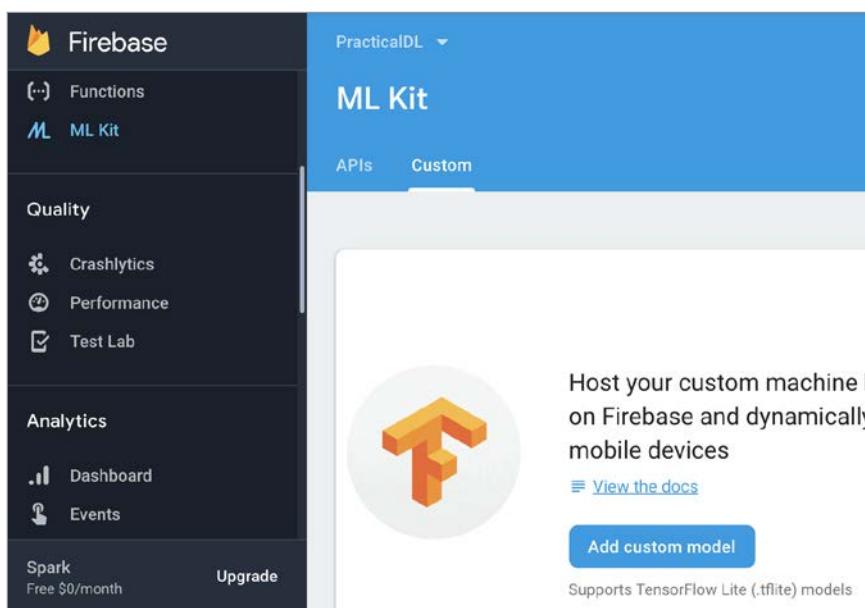


Рис. 13.16. Вкладка добавления своей модели в разделе ML Kit

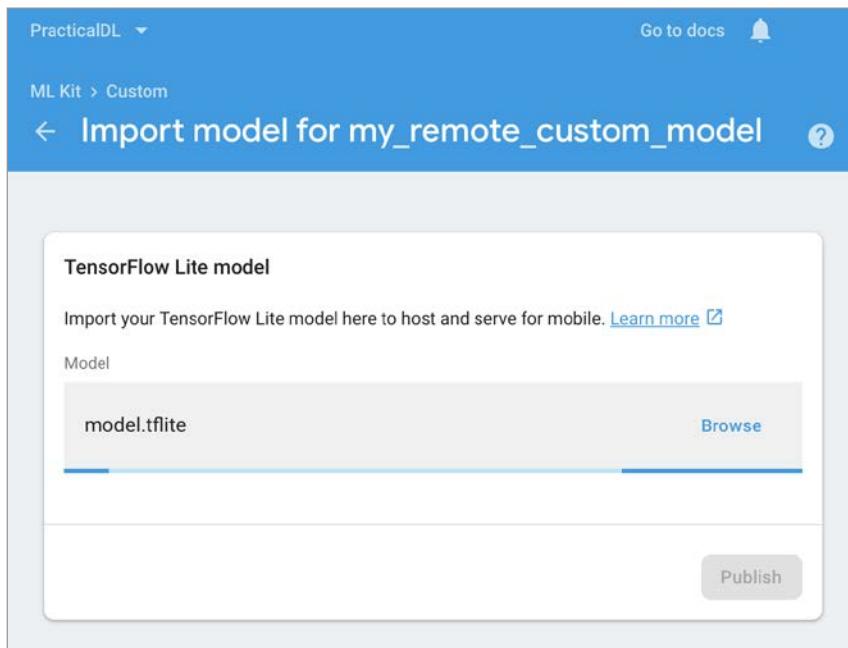
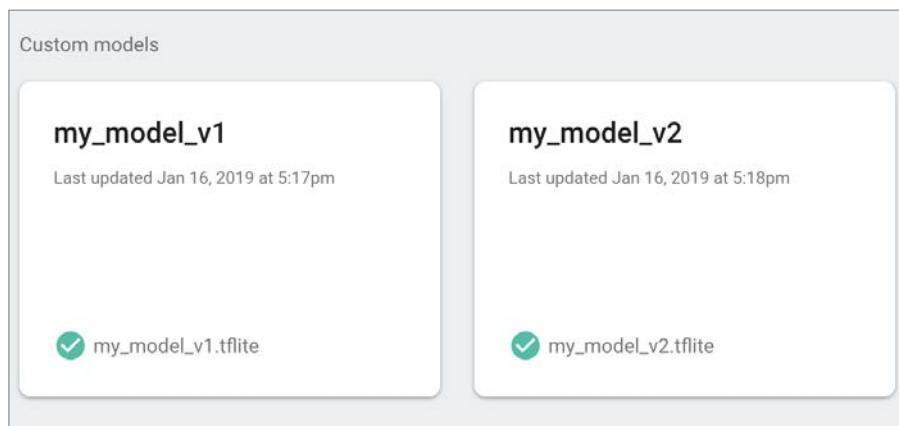


Рис. 13.17. Выгрузка файла модели TensorFlow Lite в Firebase

Вот и все! Теперь модель готова к динамической загрузке из приложения. Далее рассмотрим порядок проведения А/В-тестирования моделей с помощью Firebase.

## А/В-тестирование моделей, размещенных в облаке

Возьмем такой сценарий: есть модель версии 1 под именем `my_model_v1`, которую мы развернули для наших пользователей. Через какое-то время мы получили больше данных для обучения, и в результате появилась модель `my_model_v2` (рис. 13.18). Теперь хотелось бы оценить, повысилось ли качество прогнозирования в этой новой версии. В такой оценке поможет А/В-тестирование.

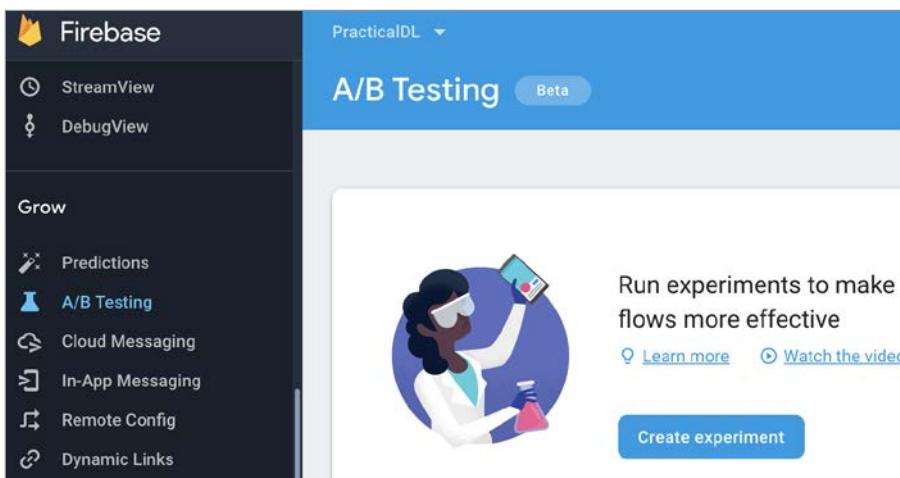


**Рис. 13.18.** В настоящее время в Firebase выгружены две модели

А/В-тестирование — это метод проверки статистических гипотез, помогающий ответить на вопрос: действительно ли В лучше, чем А? Здесь А и В могут быть чем угодно: содержимым сайта, элементами дизайна в приложении для телефона или даже моделями глубокого обучения. А/В-тестирование — действительно полезная возможность, особенно для тех, кто активно развивает модель и кому требуется оценить реакцию пользователей на разные версии.

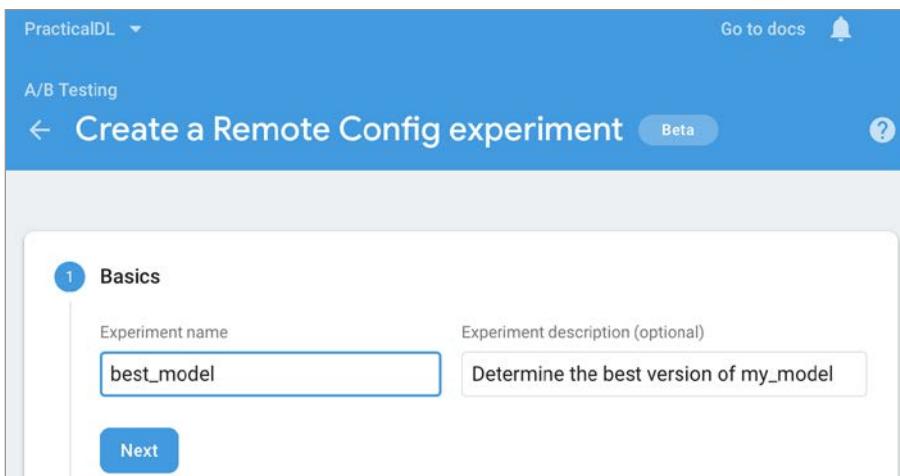
Пользователи уже некоторое время работали с `my_model_v1`, и мы хотим узнать, порадовала ли их наша следующая версия. Не будем откладывать дело в долгий ящик и откроем доступ к модели второй версии 10% наших пользователей. Для этого настроим эксперимент А/В-тестирования:

1. В Firebase выберите раздел **A/B Testing** (А/В-тестирование) и щелкните на кнопке **Create experiment** (Создать эксперимент; рис. 13.19).
2. Выберите параметр **Remote Config** (Удаленная конфигурация).



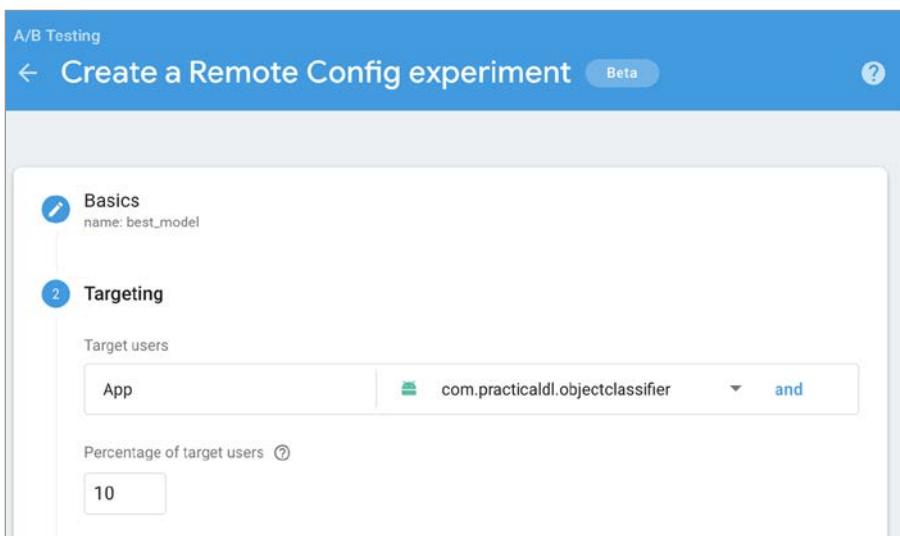
**Рис. 13.19.** Раздел настройки А/В-тестирования в Firebase, где можно создать эксперимент

3. В разделе **Basics** (Основные) введите имя эксперимента в поле **Experiment name** (Имя эксперимента) и, если захотите, краткое описание (рис. 13.20), а затем щелкните на кнопке **Next** (Далее).



**Рис. 13.20.** Раздел Basics, где можно создать удаленную конфигурацию эксперимента

4. В разделе **Targeting** (Таргетинг) в раскрывающемся меню **Target users** (Целевая аудитория) выберите наше приложение и введите процент пользователей, которые будут вовлечены в эксперимент (рис. 13.21).



**Рис. 13.21.** Раздел Targeting в настройках удаленной конфигурации

5. Выберите целевой показатель для оценки (чуть подробнее поговорим об этом в следующем разделе).
6. В разделе **Variants** (Варианты), см. рис. 13.22, создайте новый параметр `model_name` с именем модели, которую будет использовать конкретный пользователь. Контрольная группа будет по умолчанию получать модель `my_model_v1`. Затем создайте дополнительный вариант параметра с именем модели `my_model_v2`, которую получат 10 % пользователей.

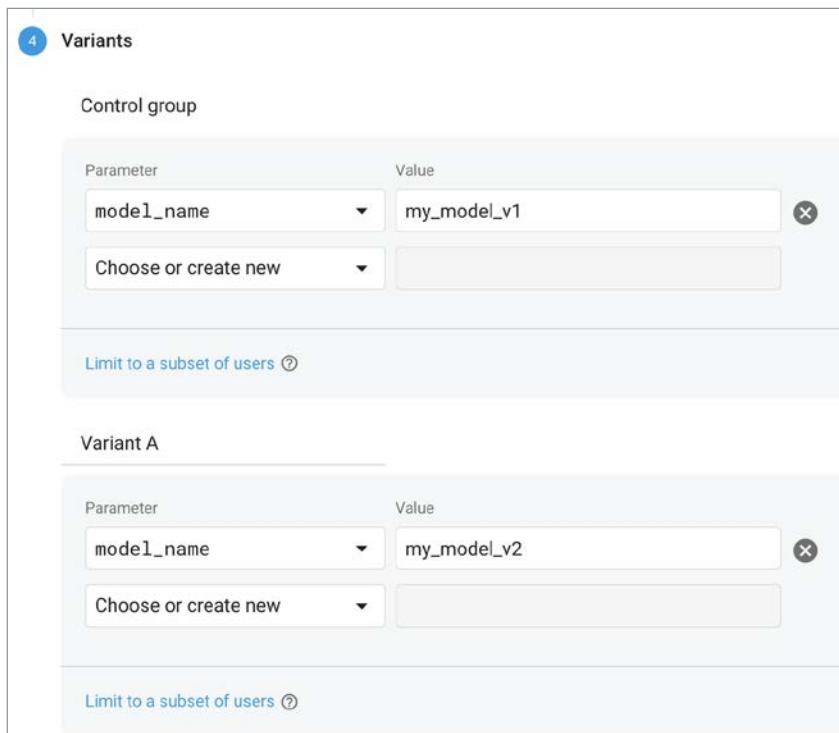
Выберите раздел **Review** (Обзор) и щелкните на кнопке **Start experiment** (Начать эксперимент). С течением времени можно увеличивать процент пользователей, получающих новую модель, используя варианты.

Та-да! Эксперимент создан и запущен.

## Оценка результатов эксперимента

И что дальше? Нужно время, чтобы собрать результаты. В зависимости от типа эксперимента можно выделить от нескольких дней до нескольких недель. Оценить успешность можно по любому количеству критериев. Мы можем использовать уже готовые критерии, предоставленные по умолчанию (рис. 13.23).

Допустим, нас интересует общий доход. В конце концов, все мы хотим разбогатеть, как Джин Ян. Для этого можно измерить доход от пользователей, вовлеченных в эксперимент, и сравнить его с базовым уровнем, то есть с доходом от пользователей, не вовлеченных в эксперимент. Если удельный доход в экспери-



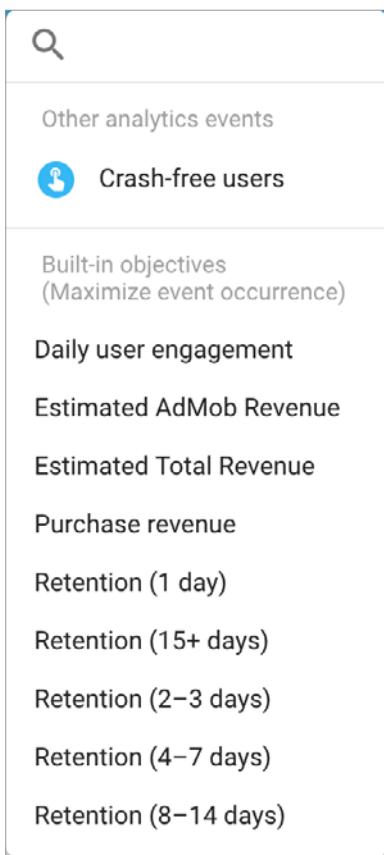
**Рис. 13.22.** Раздел Variants в настройках удаленной конфигурации

ментальной группе выше, то эксперимент считаем успешным. И наоборот, если удельный доход в экспериментальной группе не изменился или уменьшился, то делаем противоположный вывод. В случае успеха эксперимент можно постепенно распространить на всех пользователей, после чего он перестает быть экспериментом и превращается в основное предложение.

## Использование эксперимента в коде

Теперь, настроив эксперимент, посмотрим, как включить его в наше приложение. Чтобы использовать ту или иную модель в приложении, получим объект удаленной конфигурации (`remoteConfig`) и извлечем из него имя модели, которое зависит от того, включен ли пользователь в эксперимент. Код ниже реализует это:

```
val remoteConfig = FirebaseRemoteConfig.getInstance()
remoteConfig.fetch()
val modelName = remoteConfig.getString("current_best_model")
val remoteModel = FirebaseCloudModelSource.Builder(modelName)
    .enableModelUpdates(true).build()
FirebaseModelManager.getInstance().registerCloudModelSource(remoteModel)
```



**Рис. 13.23.** Аналитическая информация, доступная при проведении эксперимента A/B-тестирования

Остальной код, реализующий прогнозирование и показанный в предыдущих разделах, остается неизменным. Теперь приложение готово к использованию правильной модели, как того требует эксперимент.

## TensorFlow Lite в iOS

В предыдущих главах мы показали, насколько просто использовать Apple Core ML на iOS. Можно просто перетащить модель мышью в Xcode и запустить инференс с помощью двух строк кода. Но рассматривая даже простые примеры для iOS, например в репозитории TensorFlow Lite, становится очевидно, что для создания самых простых приложений требуется написать значительный объем шаблонного кода. Однако если использовать TensorFlow Lite вместе с ML Kit,

то эта работа становится намного приятней. В добавок к ясному и лаконичному API мы получаем все возможности, которые подробно описали выше в этой главе: удаленную загрузку и обновление моделей, A/B-тестирование модели и резервное копирование. И все это без лишних усилий. Разработчик, пишущий приложение глубокого обучения для iOS или для Android, может рассматривать работу с ML Kit как способ «написать однажды, использовать везде».

## Оптимизация производительности

В главе 6 мы рассмотрели приемы квантования и прореживания моделей в основном с теоретической точки зрения. Теперь посмотрим на их реализацию с помощью TensorFlow Lite и дополнительных инструментов.

### Квантование с помощью TensorFlow Lite Converter

Для iOS компания Apple предоставляет `quantization_utils` в пакете Core ML Tools. В TensorFlow Lite его эквивалентом является встроенный инструмент `tfLite_convert`, который мы использовали выше в этой главе. Команде `tfLite_convert` можно передать входной файл, граф модели, тип данных для преобразования и имена входов и выходов (узнать которые можно с помощью Netron, как показано в главе 11). Переход от 32-битного к 8-битному целочисленному представлению весов уменьшает размер модели почти в четыре раза и при этом почти не влияет на точность.

```
$ tfLite_convert \
--output_file=quantized-model.tflite \
--graph_def_file=/tmp/some-graph.pb \
--inference_type=QUANTIZED_UINT8 \
--input_arrays=input \
--output_arrays=MobilenetV1/Predictions/Reshape_1 \
--mean_values=128 \
--std_dev_values=127
```

После выполнения этой команды появится модель `quantized-model.tflite`.

### Набор инструментов TensorFlow для оптимизации моделей

TensorFlow Lite Converter – это самый простой способ квантования моделей. Стоит отметить, что это он выполняет квантование модели после ее обучения. Из-за снижения разрешающей способности может наблюдаться небольшая, хотя и заметная потеря точности. Есть ли другое решение? *Обучение с учетом квантования*, как следует из названия, учитывает влияние квантования на этапе обучения и пытается компенсировать и минимизировать потери, которые неизбежны при квантовании после обучения.

Обе формы квантования позволяют уменьшить размер модели на 75 %, но эксперименты показали следующее:

- для модели MobileNetV2 потеря точности при квантовании после обучения составила восемь процентных пунктов, тогда как при обучении с учетом квантования точность снизилась только на один пункт;
- для модели InceptionV3 обучение с учетом квантования дало колоссальное ускорение на 52 %, тогда как квантование после обучения — только на 25 %.



Эти метрики точности есть в наборе тестов для ImageNet с 1000 классов. Большинство моделей имеют меньшую сложность и меньшее количество классов. Потеря точности при квантовании более простых моделей должна быть меньше.

Обучение с учетом квантования можно реализовать с помощью набора инструментов TensorFlow Model Optimization Toolkit. Этот набор предлагает постоянно расширяющийся спектр инструментов для сжатия моделей, включая прореживание. Кроме того, в репозитории TensorFlow Lite уже есть модели, к которым было применено предварительное квантование таким способом. В табл. 13.2 — результаты применения различных стратегий квантования.

**Таблица 13.2.** Результаты применения разных стратегий квантования (до 8 бит) моделей (источник: документация по оптимизации моделей в TensorFlow Lite)

Модель		MobileNet	MobileNetV2	InceptionV3
Топ-1 точности	Исходная	0,709	0,719	0,78
	Квантование после обучения	0,657	0,637	0,772
	Обучение с учетом квантования	0,7	0,709	0,775
Продолжительность работы модели (мс)	Исходная	124	89	1130
	Квантование после обучения	112	98	845
	Обучение с учетом квантования	64	54	543
Размер (Мбайт)	Исходная	16,9	14	95,7
	Квантование после обучения	4,3	3,6	23,9

## Fritz

Как мы уже знаем, основная цель Core ML и TensorFlow Lite — возможность быстрого инференса на мобильных устройствах. Создайте модель, подключите ее к приложению и можете запускать инференс.

Затем появился набор инструментов ML Kit, который кроме поддержки ИИ упростил развертывание моделей и их мониторинг (с помощью Firebase). Отступим на шаг назад, чтобы окинуть одним взглядом весь пайплайн: обучение, конвертирование в мобильный формат, оптимизацию по скорости и развертывание моделей для пользователей, а также мониторинг качества их работы и отслеживание версий. Для выполнения всех этих шагов нужно множество инструментов. И велика вероятность, что у всего пайплайна нет единого владельца. Это связано с тем, что для работы с этими инструментами нужен определенный опыт (например, специалист по обработке данных может не знать, как правильно развернуть модель, чтобы сделать ее доступной пользователям). Стартап из Бостона Fritz предпринял попытку устраниć эти препятствия и упростить полный цикл от разработки модели до ее развертывания как для специалистов по данным, так и для разработчиков мобильных приложений.

Fritz предлагает комплексное решение для разработки мобильного ИИ, обладающее следующими особенностями.

- Возможность развертывать модели непосредственно на пользовательских устройствах из Keras после завершения обучения с использованием обратных вызовов.
- Возможность делать бенчмаркинг модели прямо с компьютера без нужды развертывать ее на телефоне, как показывает следующий код:

```
$ fritz model benchmark <путь к модели Keras в формате .h5>
...
-----
Fritz Model Grade Report
-----

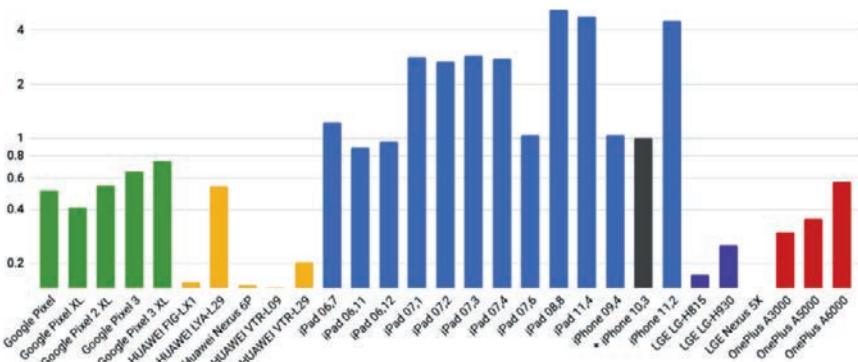
Core ML Compatible:      True
Predicted Runtime (iPhone X): 31.4 ms (31.9 fps)
Total MFLOPS:            686.90
Total Parameters:        1,258,580
Fritz Version ID:       <Идентификатор версии>
```

- Возможность шифровать модели для защиты интеллектуальной собственности от кражи с устройства.
- Поддержка множества передовых алгоритмов компьютерного зрения с использованием всего нескольких строк кода. В комплект инструментов входят готовые мобильные реализации этих алгоритмов, которые умеют обрабатывать кадры, поступающие с высокой частотой. Сюда входит, например, сегментация изображения, передача стиля, обнаружение объектов и опре-

деление позы. На рис. 13.24 показаны бенчмарки обнаружения объектов, выполненные на различных устройствах iOS.

```
let poseModel = FritzVisionPoseModel()
guard let poseResult = try? poseModel.predict(image) else { return }
let imageWithPose = poseResult.drawPose() // Наложить позу на входе.
```

**Действительная производительность**  
Обнаружение объектов на мобильных устройствах —  
относительно iPhone X\*



**Рис. 13.24.** Производительность алгоритма обнаружения объектов из Fritz SDK на различных мобильных устройствах по сравнению с iPhone X

- Возможность дополнительного обучения готовых моделей на пользовательских датасетах с помощью блокнотов Jupyter. Отметим, что эта сложная (даже для профессиональных специалистов по обработке данных) задача упрощается настолько, что разработчику остается только убедиться, что данные для дообучения имеют правильный формат.
- Возможность управлять всеми версиями модели из командной строки.
- Возможность проверить пригодность моделей для использования на мобильных устройствах с помощью приложения с открытым исходным кодом Heartbeat (доступно в магазинах приложений для iOS и Android), созданным компанией Fritz. Достаточно просто клонировать приложение, взять модель для проверки, заменить ею существующую модель и посмотреть, как она работает на телефоне.
- Активное сообщество блогеров, пишущих о достижениях в мобильном ИИ на heartbeat.fritz.ai.

## От создателя

Мотивом к созданию Fritz послужили два противоречивых чувства: вдохновение и разочарование. Вдохновляло решение множества проблем, которые раньше считались неразрешимыми, с помощью машинного обучения. Заработало сочетание данных, вычислений и моделей. В то же время мы были разочарованы нехваткой ресурсов для развертывания и обслуживания МО где угодно, кроме облака. По соображениям производительности, конфиденциальности и стоимости предпочтительнее перенести обученные модели на устройства, где собираются данные, а не посыпать данные в облако, обрабатывать их там и возвращать прогнозы обратно. Это подтолкнуло нас к созданию платформы Fritz для разработчиков, позволяющей собирать, обучать и развертывать модели МО и управлять ими на мобильных устройствах.

Во многих отношениях процесс создания решений с помощью МО похож на бег с препятствиями. Мы хотим уменьшить или устраниć эти препятствия, чтобы инженеры, даже не имеющие опыта использования глубокого обучения или работы с мобильными устройствами, могли с помощью наших разработок создавать лучшие мобильные приложения. У них должна быть возможность быстро начать использовать эффективные и предварительно обученные модели или обучать собственные. Наши инструменты призваны упростить развертывание моделей, позволить оценить качество их работы, поддерживать их в актуальном состоянии и защищать. Частичным результатом наших усилий стали модели, которые прекрасно работают на краевых устройствах.

Упрощение внедрения функций МО в мобильную среду приведет к появлению более мощных и персонализированных приложений, которые помогут не потерять текущих пользователей, привлечь новых и изменить наши представления о том, что возможно и что невозможно. Мы считаем, что машинное обучение — это самое фундаментальное изменение в информатике, случившееся за последние десятилетия, и хотим помочь в создании, обучении и развитии сообщества.

Дэн Абдинур (Dan Abdinoor),  
генеральный директор и соучредитель Fritz

## Целостный взгляд на цикл разработки мобильных приложений ИИ

Мы рассмотрели множество методов и технологий, позволяющих решать отдельные задачи в цикле разработки мобильного ИИ. Здесь мы свяжем их воедино и ответим на вопросы, возникающие на протяжении всего цикла. На рис. 13.25 представлен общий вид цикла разработки.



**Рис. 13.25.** Цикл разработки мобильных приложений ИИ

## Как собирать данные?

Есть нескольких разных стратегий сбора данных:

- найти интересующий объект и вручную сделать несколько фотографий с разных ракурсов, при разном освещении, в разных окружениях, с разного расстояния и т. д.;
- загрузить данные из интернета с помощью Fatkun (глава 12);
- найти готовые датасеты (например, с помощью Google Dataset Search (<https://oreil.ly/NawZ6>)), например Food-101 с фотографиями блюд;
- синтезировать свой датасет.

Для этого:

1. Поместите объект (на переднем плане) перед зеленым экраном (фоном) и сфотографируйте его. Замените фон случайными изображениями, чтобы синтезировать большой датасет. Затем, меняя масштаб, обрезая и поворачивая исходные фотографии, можно создать сотни дополнительных изображений.
2. Если это невозможно, то вырежьте объект из имеющихся фотографий и повторите предыдущий шаг, чтобы получить достаточно большой

и разнообразный датасет. Изображения самого объекта на переднем плане тоже должны отличаться, иначе сеть может переобучиться и просто запомнить один пример вместо обобщения объекта.

3. Найдите реалистичную трехмерную модель интересующего объекта, поместите ее в реалистичное окружение, используя фреймворк трехмерной графики, например Unity. Отрегулируйте освещение и положение камеры, масштабирование и поворот и сфотографируйте объект под разными углами. В главе 7 мы говорили о компаниях AI.Reverie и CVEDIA, которые работают в этой сфере. Мы сами тоже использовали фотопортфолио симуляторы, чтобы обучить модели беспилотных автомобилей (см. главы 16 и 17).

## Как размечать данные?

Большинство шагов, описанных в предыдущем разделе, уже должны были внести метки для данных. Для разметки коллекций без меток используйте специализированные инструменты: Supervisely, Labelbox и Diffgram. Для обработки действительно больших датасетов, разметить которые самостоятельно невозможно, воспользуйтесь предложениями специализированных компаний Digital Data Divide, iMerit и SamaSource.

## Как обучить модель?

Вот два основных подхода к обучению моделей:

- с написанием своего кода: с помощью Keras и TensorFlow (глава 3);
- без написания своего кода: с помощью сервисов создания собственных классификаторов — Google Auto ML, Microsoft CustomVision.ai и Clarifai (эти сервисы мы тестировали в главе 8) или Apple Create ML (глава 12).

## Как конвертировать модель в формат для мобильных устройств?

Вот несколько способов:

- с помощью Core ML Tools (только для устройств Apple);
- с помощью TensorFlow Lite Converter — для iOS и Android;
- в качестве альтернативы можно создать сквозной пайплайн на основе Fritz.

## Как оптимизировать производительность модели?

Вот несколько приемов, которые помогут повысить производительность модели:

- начните с эффективной модели, например из семейства MobileNet или, что еще лучше, из семейства EfficientNet;
- уменьшите размер модели путем квантования и прореживания, чтобы сократить время загрузки и вычисления прогнозов, сохранив при этом точность (как рассказывалось в главах 6, 11 и 13). На этом этапе размер модели можно уменьшить на 75 % с небольшой потерей точности;
- используйте Core ML Tools для экосистемы Apple или TensorFlow Model Optimization Kit для iOS и Android.

## Как повысить привлекательность для пользователей?

Очевидно, что ответ на этот вопрос во многом зависит от типа решаемой задачи. Но можем дать общую рекомендацию — найти баланс между интерактивностью, производительностью и использованием ресурсов (память, процессор, аккумулятор и т. д.). И конечно же, подумайте о внедрении интеллектуального механизма обратной связи, который поможет собирать данные и обеспечит обратную связь. Геймификация сможет поднять этот механизм на совершенно новый уровень.

Возьмем для примера приложение, классифицирующее продукты питания по фотографии. После того как пользователь сфотографирует блюдо, ему покажут список из пяти наиболее вероятных прогнозов (в порядке убывания достоверности). Например, если пользователь сфотографировал пиццу, на экране могут появиться такие прогнозы: «пирог — 75 %», «пицца — 15 %», «наан — 6 %», «хлеб — 3 %», «запеканка — 1 %». В этом случае пользователь выберет второй прогноз. В случае идеальной модели пользователь всегда будет выбирать первый прогноз, а поскольку наша модель несовершенна, рейтинг прогноза, выбранного пользователем, становится сигналом, который поможет улучшить модель в будущем. В худшем случае, когда все предложенные прогнозы ошибочны, пользователь может вручную ввести название блюда. Добавленная функция автозаполнения при ручном вводе сохранит единообразие меток в датасете.

Еще лучше, если пользователю не придется фотографировать и достаточно будет просто навести камеру на блюдо, а приложение сформирует прогноз в режиме реального времени.

## Как развернуть модель?

Вот несколько способов развертывания моделей:

- объедините модель с приложением в один бинарный файл и опубликуйте его в магазинах приложений;
- разместите в облаке и реализуйте ее загрузку в приложении;
- используйте службу управления моделями, например Fritz или Firebase (интегрированную с ML Kit).

## Как оценить успешность модели?

Первым делом определите критерии успешности. Рассмотрим следующие примеры:

- в 95 % случаев модель должна вычислять прогноз менее чем за 200 мс;
- пользователи, использующие эту модель, открывают приложение каждый день;
- для приложения классификации продуктов питания критерием успеха может быть что-то вроде: 80 % пользователей выбрали первый прогноз в списке.

Это не фиксированные критерии, и со временем они должны корректироваться. Очень важно, чтобы оценка успешности опиралась на конкретные данные. Если у вас есть новая версия модели, запустите А/В-тестирование и оцените успешность этой версии по тем же критериям, чтобы определить, действительно ли она лучше предыдущей.

## Как совершенствовать модель?

Вот некоторые рекомендации по улучшению качества модели:

- собираите отзывы пользователей о прогнозах: какие прогнозы были правильными, а какие, что особенно важно, — неправильными. Передайте эти изображения вместе с соответствующими метками на вход модели в следующем цикле ее обучения. Этот подход иллюстрирует рис. 13.26;
- автоматически собираите фотографии, опознанные неправильно, если пользователи дали на это явное согласие. Выполните разметку этих фотографий вручную и используйте их в следующем цикле обучения.



**Рис. 13.26.** Цикл обратной связи для ошибочных прогнозов, помогающий собрать больше обучающих данных и улучшить модель

## Как обновить модель на телефонах пользователей?

Вот несколько способов:

- для каждой новой версии модели создавать новый релиз. Это медленно и негибко;
- разместите новую модель в облаке и реализуйте в приложении загрузку новой версии модели при ее появлении. Загрузку лучше выполнять, когда устройство пользователя подключено к Wi-Fi;
- используйте систему управления моделями Firebase (в сочетании с ML Kit) или Fritz, чтобы автоматизировать большую часть рутинной работы.

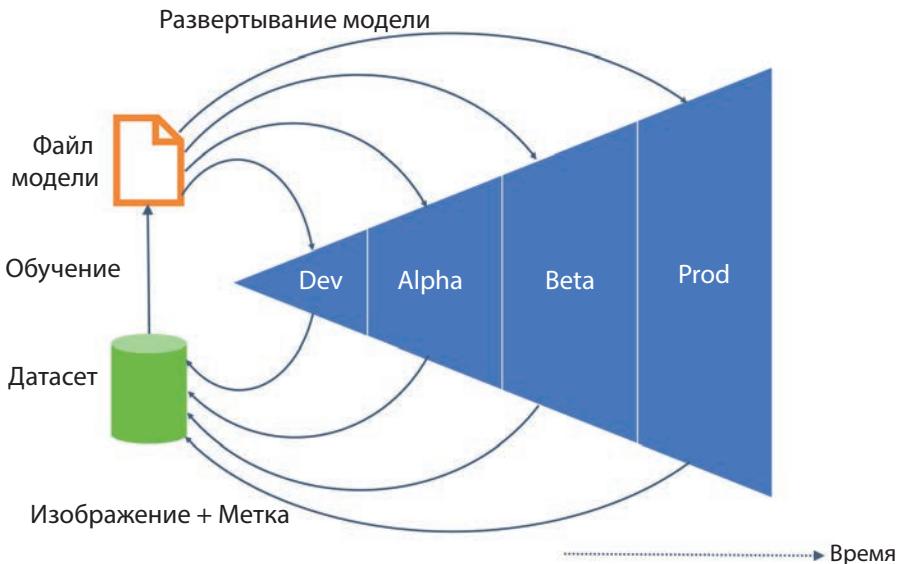
А теперь посмотрим, как организовать автоматическое улучшение модели.

## Самосовершенствующаяся модель

Выше мы рассмотрели все, что нужно, для сценариев с развитыми и предварительно обученными моделями. Просто подключите модель к приложению, и все

готово к работе. В сценарии, где требуются узкоспециализированные модели, часто обучаемые на скучных датасетах, можно привлечь пользователей и создать самосовершенствующуюся, постоянно развивающуюся модель.

Суть в следующем: каждый раз, когда пользователь использует приложение, он предоставляет обратную связь (изображение + метка) для дальнейшего улучшения модели, как показано на рис. 13.27.



**Рис. 13.27.** Цикл развития самосовершенствующейся модели

Модель должна пройти несколько этапов разработки, прежде чем попадет к конечным пользователям. Ниже перечислены этапы подготовки ПО к релизу, которые применимы к моделям ИИ.

1. *Dev* (версия в разработке).

Начальный этап разработки, когда единственные пользователи приложения — это разработчики. Накопление данных происходит медленно, приложение работает неустойчиво, а прогнозы могут быть ненадежными.

2. *Alpha* (альфа-версия).

Когда приложение будет готово к тестированию уже другими пользователями, оно вступает в фазу альфа-версии. На этом этапе обратная связь чрезвычайно важна, потому что помогает улучшить работу приложения, а также предоставляет большой объем дополнительных данных для пайплайна.

Приложение работает устойчивее, и модель дает более надежные прогнозы. В некоторых компаниях этот этап также известен как этап *внутреннего тестирования* (то есть тестирование приложения производится сотрудниками).

### 3. *Beta* (бета-версия).

На этапе бета-версии круг пользователей значительно расширяется в сравнении с этапом альфа-версии. Обратная связь также очень важна. Скорость сбора новых данных сильно возрастает, так как в реальном мире намного больше различных устройств, собирающих данные, которые помогают быстро улучшить модель. Благодаря отзывам пользователей альфа-версии интерфейс приложения стал намного стабильнее, а прогнозы — надежнее. Бета-версии приложений для iOS обычно размещаются на Apple TestFlight, а для Android — на Google Play Console. Для размещения бета-версий также используются HockeyApp и TestFairy.

### 4. *Prod* (готовая, то есть продакшен-версия).

Продакшен-версия приложения стабильна и доступна многим пользователям. Поступление новых данных увеличивается, но модель на этом этапе уже довольно стабильна и дополнительное обучение не дает значительного эффекта. Но с ростом количества пользователей модель может улучшать надежность прогнозов в некоторых пограничных случаях, которые могли не возникнуть на первых трех этапах. Когда модель станет достаточно развитой, последующие улучшения можно протестировать, повторив этапы альфа- и бета-версий или проведя A/B-тестирование с привлечением подмножества конечных пользователей.

Специалисты по обработке данных обычно предполагают, что данные будут доступны до начала разработки. Но в мире мобильного ИИ мы можем начать с небольшого датасета и постепенно улучшать систему.

Для саморазвивающегося приложения «Шазам для еды» самым сложным решением разработчиков станет выбор ресторанов, в которые они должны отправиться для сбора исходных данных.

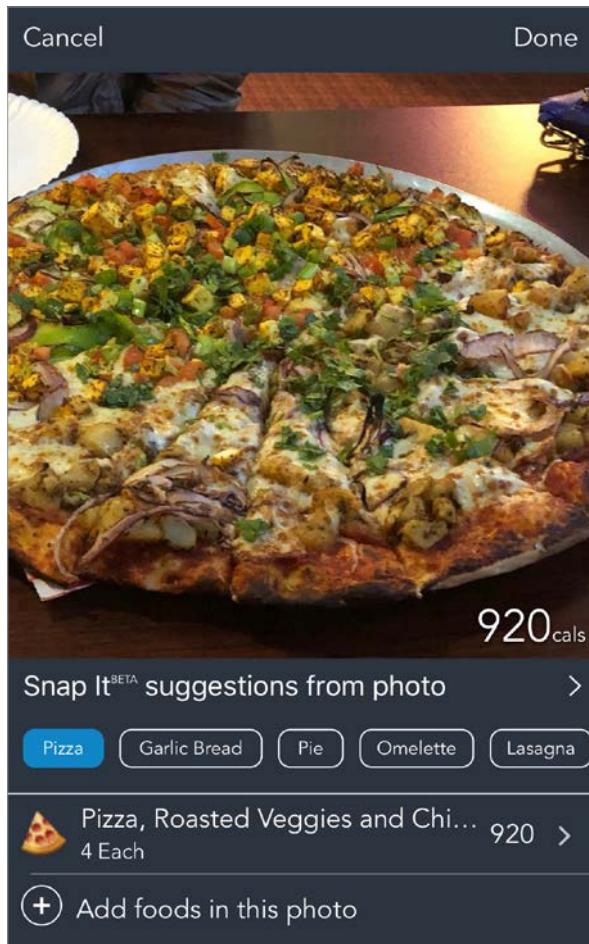
## Примеры из практики

Рассмотрим несколько интересных практических примеров.

### Lose It!

Признаемся: приложение, которое мы создавали в этой главе, уже существует. Мечта Эрлиха Бахмана была реализована компанией Fit Now из Бостона. Раз-

работчики приложения Lose It! (рис. 13.28) утверждают, что помогли 30 миллионам пользователей похудеть на более чем 38 миллионов килограммов, дав им возможность следить за калориями. Пользователь направляет камеру телефона на штрихкод, на этикетку с названием продукта и на само блюдо, которое он собирается съесть, и узнает, сколько там калорий.



**Рис. 13.28.** Функция Snap It из приложения Lose It! показывает предложения для названия сканируемого продукта

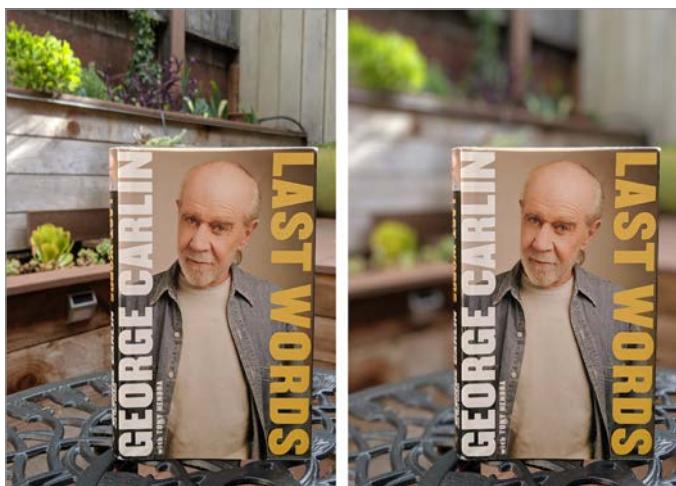
На первом этапе эта система сканирования продуктов питания использовала облачные алгоритмы. Но из-за задержек в сети прогнозирование не происходило в режиме реального времени. Тогда команда Fit Now перенесла свои модели на TensorFlow Lite, чтобы оптимизировать их для мобильных устройств,

и использовала ML Kit для развертывания на устройствах миллионов пользователей. Благодаря постоянной обратной связи, обновлениям моделей и А/В-тестированию приложение продолжает постепенно самосовершенствоваться.

## Режим портретной съемки на телефонах Pixel 3

Один из важных визуальных эффектов, отличающих работы профессиональных фотографов от любительских снимков, — это боке, или размытие фона позади главного объекта. (Не путайте с Bokeh — инструментом визуализации в Python.) Чем ближе камера к объекту, тем сильнее размывается фон. С помощью профессиональных фотообъективов с низкой диафрагмой (стоимость которых нередко достигает нескольких тысяч долларов) можно создавать впечатляющие размытия.

Но если у вас нет таких денег, достичь того же эффекта поможет ИИ. Смартфон Google Pixel 3 предлагает режим портретной съемки, который автоматически создает эффект боке. Для этого используется сверточная сеть, которая оценивает глубину каждого пикселя в кадре и определяет, какие пиксели принадлежат переднему плану, а какие — фону. Затем пиксели фона размываются и получается эффект боке, как показано на рис. 13.29.



**Рис. 13.29.** Режим портретной съемки в Pixel 3 — он визуально отделяет передний план от заднего фона с помощью эффекта размытия

Оценка глубины — это задача, требующая большого объема вычислений, поэтому очень важно выполнять ее быстро. На помощь приходит Tensorflow Lite с поддержкой вычислений на GPU, который работает примерно в 3,5 раза быстрее CPU.

Для этого в Google обучили нейронную сеть, специализирующуюся на оценке глубины. Они использовали установку с несколькими телефонными камерами (которую назвали «Франкенфон»), с помощью которой снимали одну и ту же сцену всеми этими камерами и вычисляли *параллакс* каждого пикселя, чтобы точно определить его глубину. Затем полученная карта глубины загружалась в сверточную сеть вместе с изображениями.

## Распознавание голоса от Alibaba

Представьте, что вы взяли телефон друга и говорите: «Привет, Сири» или «Окей Google», а затем произносите волшебные слова «прочти последний текст». С точки зрения конфиденциальности было бы ужасно, если бы этот трюк сработал. Именно поэтому большинство современных мобильных ОС предлагают сначала разблокировать телефон, и только потом продолжить. Но все это может раздражать реального владельца телефона.

Лаборатория машинного интеллекта Alibaba Machine Intelligence Labs решила эту проблему, используя следующий подход. Сначала речь говорящего преобразуется в изображения спектрограмм (путем извлечения аудиопризнаков с применением алгоритма мел-частотного кепстрапа), затем на этих изображениях обучается сверточная сеть (переносом обучения) и, наконец, полученная модель развертывается на устройствах с Tensorflow Lite. Как видите, сверточные сети используются не только в компьютерном зрении!

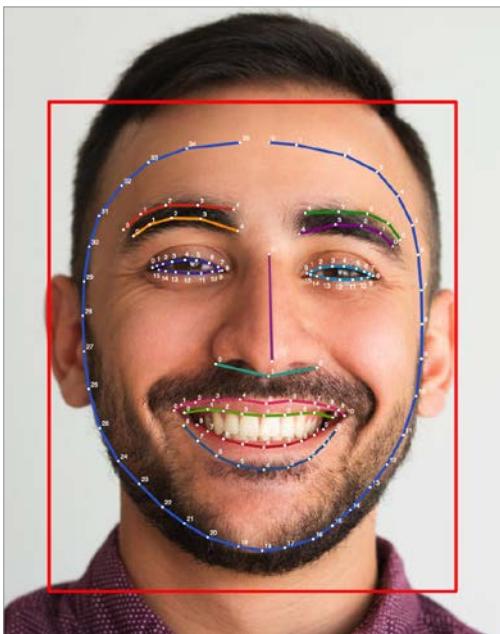
Благодаря поддержке распознавания голоса они смогли персонализировать контент на устройствах с большим количеством пользователей в семье (например, в функции голосового управления телевизором в приложении Netflix TV). Кроме того, умея распознавать звук человеческого голоса, можно очистить произносимые команды от фонового шума. Чтобы ускорить обработку с помощью Tensorflow Lite, инженеры оставили флаг `USE_NEON` для использования специализированных наборов инструкций процессоров на базе ARM. Команда сообщила, что эта оптимизация увеличила скорость вычислений в четыре раза.

## Определение контуров лица с помощью ML Kit

Хотите быстро реализовать возможности, как в фильтрах Snapchat? В ML Kit для этого есть все, что нужно, — API для определения контуров лица, то есть набора точек (с координатами X и Y), повторяющих черты лица на входном изображении. В общей сложности API генерирует 133 точки, отражающие разные черты лица, включая 16 точек, представляющих каждый глаз, и 36 точек, представляющих овал лица, как показано на рис. 13.30.

Внутри ML Kit — модель глубокого обучения, для обслуживания которой используется TensorFlow Lite. В экспериментах с новой поддержкой GPU

в TensorFlow Lite компания Google отметила четырехкратное увеличение скорости на смартфонах Pixel 3 и Samsung Galaxy S9 и шестикратное на iPhone 7 по сравнению с предыдущей реализацией, использующей CPU. В результате мы можем с высокой точностью добавить на изображение шляпу или солнцезащитные очки в режиме реального времени.



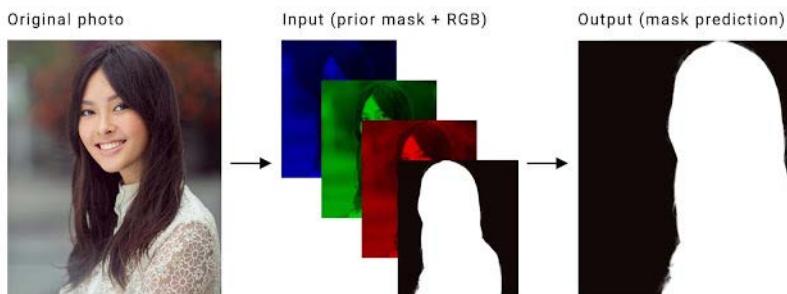
**Рис. 13.30.** 133 точки, обозначающие черты лица, которые определяются ML Kit (изображение взято с сайта <https://oreil.ly/8PMGa>)

## Сегментация видео в реальном времени в YouTube Stories

Зеленые экраны — основное оборудование для работающих в видеиндустрии. Правильный выбор цвета фона дает возможность изменить его «на лету» с использованием техники *цветовой рип-проекции* (chroma keying). Для этого нужно дорогое ПО и мощные компьютеры. Многие пользователи YouTube тоже с удовольствием использовали бы этот прием, но у них нет бюджета. Но теперь есть решение — поддержка сегментации видео в масштабе реального времени в приложении YouTube Stories, реализованная с помощью TensorFlow Lite.

Ключевые требования: высокая скорость семантической сегментации (более 30 кадров в секунду) и согласованность во времени, например плавность изменения фона по краям от кадра к кадру. Попытавшись выполнить семантическую сегментацию по нескольким кадрам, можно быстро заметить, что края многих

сегментированных масок как бы выбирают. Трюк, помогающий избавиться от этого неприятного эффекта, состоит в том, чтобы передавать сегментированные маски лица человека в следующий кадр как предыдущий. Поскольку мы традиционно работаем с тремя каналами (RGB), хитрость в том, чтобы добавить четвертый канал, который по сути является результатом преобразования предыдущего кадра, как показано на рис. 13.31.



**Рис. 13.31.** Входное изображение (слева) разбито на три составляющих слоя (R, G, B). Затем к ним добавляется выходная маска из предыдущего кадра (изображение взято с веб-сайта <https://oreil.ly/rHNH5>).

Применив еще ряд оптимизаций к базовой сверточной сети, YouTube удалось достичь скорости обработки более чем 100 кадров в секунду на iPhone 7 и более чем 40 кадров в секунду на Pixel 2. Кроме того, сравнивая серверную часть CPU и GPU в TensorFlow Lite, наблюдалось увеличение скорости в 18 раз при выборе серверной части GPU вместо CPU (гораздо большее ускорение по сравнению с обычным ускорением в 2–7 раз, достигаемым в других задачах семантической сегментации только для изображений).

## Итоги

В этой главе мы обсудили архитектуру TensorFlow Lite, касающуюся Android Neural Network API. Затем создали простое приложение для распознавания объектов на Android. Далее познакомились с набором инструментов Google ML Kit и узнали, почему им стоит пользоваться. Мы также узнали, как конвертировать модели из формата TensorFlow в формат TensorFlow Lite, чтобы использовать их на устройствах Android. Наконец, мы обсудили несколько примеров использования TensorFlow Lite на практике.

В следующей главе посмотрим, как использовать глубокое обучение в реальном времени для разработки интерактивного приложения.

## ГЛАВА 14

---

# Создание приложения **Purrfect Cat Locator с помощью TensorFlow Object Detection API**

На участок Боба часто приходит соседский кот, и эти посещения имеют весьма неприятные последствия. Дело в том, что Боб разбил на участке красивый большой сад, за которым трепетно ухаживает. А пушистый гость приходит каждую ночь и жует растения. Месяцы тяжелой работы уничтожаются в мгновение ока. Недовольный такой ситуацией, Боб хочет что-нибудь предпринять.

Ведомый своим внутренним Эйсом Вентурой<sup>1</sup>, Боб пытался охранять сад по ночам, чтобы отгонять кота, но в итоге это оказалось неэффективно. Когда терпение Боба лопнуло, он решил прибегнуть к мощному оружию: использовать ИИ в сочетании с системой полива, чтобы направлять струю воды на кота.

Для этого Боб установил в саду камеру, которая должна отслеживать перемещения кота и включать ближайший разбрзыватель. У Боба уже был старый телефон, и ему оставалось только выбрать способ определения местоположения кота в реальном времени, чтобы знать, какой разбрзыватель включить и отпугнуть пушистого гостя. Будем надеяться, что после нескольких водных процедур, как показано на рис. 14.1, кот не захочет повторять набеги на сад.

В этой главе поможем Бобу создать детектор котов. (Примечание для зоозащитников: при написании этой главы ни один кот не пострадал.) В процессе ответим на следующие вопросы:

- Какими бывают задачи компьютерного зрения?

---

<sup>1</sup> Эйс Вентура — вымышленный комедийный персонаж, эксцентричный частный детектив, специализирующийся на розыске домашних животных. — Примеч. ред.

- Как переиспользовать модель, предварительно обученную для распознавания определенного класса объектов?
- Можно ли обучить модель детектора объектов, не написав ни строки кода?
- Как обучить детектор нестандартных объектов, если возникнет потребность в более детальном контроле для большей точности и скорости?



**Рис. 14.1.** Система полива с искусственным интеллектом для отпугивания кошек должна действовать как устройство Havahart Spray Away с датчиком движения



Внимательные читатели могли заметить, что мы пытаемся найти положение кота в двумерном пространстве, тогда как сам кот существует в трехмерном пространстве. Чтобы упростить задачу определения реальных координат местоположения кота, предположим, что камера всегда будет находиться в фиксированном положении. Тогда для определения расстояния до кота можно взять размер среднего животного и сравнивать его с видимым размером на кадрах, полученных с камеры через определенные промежутки времени. Например, средний кот умещается в рамку высотой 300 пикселей, находясь на расстоянии около полуметра от камеры, и в рамку высотой 250 пикселей, находясь на расстоянии около метра.

## Виды задач компьютерного зрения

В большинстве глав мы фактически рассматривали один вид задач: классификацию объектов. При этом мы выясняли, содержит ли изображение объекты

определенного класса. Но приложение Боба должно не только распознавать кота в кадре, но и определять его точное местоположение, чтобы включить нужный разбрызгиватель. Определение местоположения объекта на изображении — это задача другого вида: *обнаружение объекта*. Но прежде чем перейти к обнаружению объектов, давайте посмотрим, какие еще виды задач компьютерного зрения бывают и на какие вопросы они отвечают.

## Классификация

В этой книге мы видели несколько примеров классификации. Задача классификации заключается в присваивании изображению класса(ов) объекта(ов), присутствующих на изображении. Она отвечает на вопрос, есть ли на изображении объект класса X? Классификация — одна из фундаментальных задач компьютерного зрения. На картинке могут быть объекты более чем одного класса — это называется многоклассовой классификацией. Меткой для изображения в этом случае будет список всех классов объектов, которые есть на изображении.

## Локализация

Недостаток классификации состоит в том, что такие задачи не сообщают нам ни где именно находится конкретный объект, ни насколько он велик или мал, ни сколько на изображении объектов этого класса. Нам лишь сообщают, что где-то на изображении есть объект определенного класса. *Локализация*, также известная как *классификация с локализацией*, может не только перечислить, какие классы есть на изображении, но и указать их место. Ключевое слово здесь — *классы*, а не отдельные объекты, потому что локализация дает только одну ограничивающую рамку для каждого класса. Как и классификация, локализация не может ответить на вопрос, сколько объектов класса X есть на этом изображении?

Как мы увидим, локализация работает правильно, если гарантируется наличие одного экземпляра каждого класса. Если на изображении более одного экземпляра класса, одна ограничивающая рамка может охватывать несколько или все объекты этого класса (в зависимости от вероятностей). Но единственной ограничивающей рамки на класс кажется маловато, не так ли?

## Обнаружение

Когда на одном изображении несколько объектов, принадлежащих нескольким классам, локализации будет недостаточно. Обнаружение объектов даст ограничивающую рамку для каждого экземпляра каждого класса. Оно также

сообщит класс объекта в каждой из рамок. Например, обнаружение объектов в беспилотном автомобиле вернет по одной рамке для каждой машины, человека, фонарного столба и т. д., находящихся в поле зрения камеры автомобиля. Благодаря этому обнаружение можно использовать в приложениях, которые должны подсчитывать количество объектов, например людей в толпе.

В отличие от локализации, у обнаружения нет ограничений на количество экземпляров класса, которые есть на изображении. Вот почему оно используется в реальных приложениях.



Часто люди используют термин «локализация объекта», подразумевая «обнаружение объекта». Важно помнить о разнице между ними.

## Сегментация

Цель сегментации — присвоить метку класса отдельным пикселям на изображении. Аналогия из реального мира — детская раскраска, где цифрами указаны цвета для раскрашивания тех или иных фрагментов. Так как класс присваивается каждому пикселю на изображении, такая задача требует значительных вычислительных ресурсов. В отличие от задач локализации и обнаружения объектов, которые формируют ограничивающие рамки, задача сегментации создает группы пикселей, также известные как *маски*. Очевидно, что сегментация намного точнее определяет границы объектов, чем обнаружение. Например, косметическое приложение, изменяющее на изображении цвет волос пользователя в режиме реального времени, неизбежно будет использовать сегментацию. Сегментация бывает разной, в зависимости от типов масок.

### Семантическая сегментация

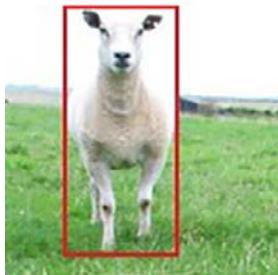
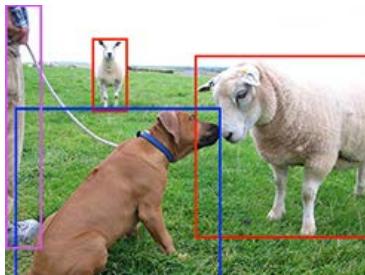
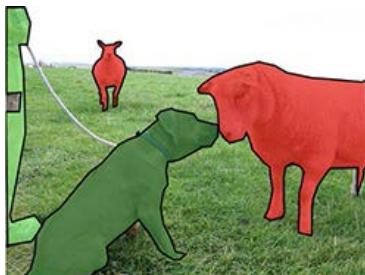
Для изображения семантическая сегментация назначает одну маску каждому классу. Если имеется несколько объектов одного класса, всем им назначается одна маска. Разные экземпляры одного класса не различаются. Семантическая сегментация, как и локализация, не может подсчитывать объекты.

### Сегментация экземпляров

Для изображения сегментация на уровне экземпляров определяет область, занимаемую каждым экземпляром каждого класса. Разные экземпляры одного класса будут сегментироваться отдельно.

Задачи компьютерного зрения перечислены в табл. 14.1.

**Таблица 14.1.** Задачи компьютерного зрения на примере изображения 120524 (<https://oreil.ly/ieJ2d>) из датасета MS COCO

Задача	Изображение	Простыми словами	Результат
Классификация объектов		Есть ли овца на изображении?	Вероятности классов
Локализация объектов		Есть ли овца на изображении и где?	Ограничивающая рамка и вероятность класса
Обнаружение объектов		Где и какие объекты есть на изображении?	Ограничивающие рамки, вероятности классов, прогнозируемые классы
Семантическая сегментация		Какие пиксели на этом изображении принадлежат разным классам, например классам «овца», «собака», «человек»?	Одна маска для каждого класса

**Таблица 14.1 (окончание)**

Задача	Изображение	Простыми словами	Результат
Сегментация экземпляров		Какие пиксели на этом изображении принадлежат разным экземплярам каждого класса, например экземплярам классов «овца», «собака», «человек»?	Одна маска для каждого экземпляра каждого класса

## Способы обнаружения объектов

Есть несколько способов обнаружения объектов в приложениях в зависимости от сценария, потребностей и технических возможностей. Некоторые из них перечислены в табл. 14.2.

**Таблица 14.2.** Разные способы обнаружения объектов, их плюсы и минусы

Способ	Нестандартные классы	Время	Облачные или локальные решения	Плюсы и минусы
Определение объектов с использованием облачных API	Нет	< 5 минут	Только облачные	+ Тысячи классов + Современные решения + Быстрая настройка + Масштабируемость – Невозможность определения нестандартных классов – Задержки в сети
Предварительно обученные модели	Нет	< 15 минут	Локальные	+ ~100 классов + Выбор модели по скорости и точности + Возможность использования на краевых устройствах – Невозможность определения нестандартных классов

Способ	Нестандартные классы	Время	Облачные или локальные решения	Плюсы и минусы
Обучение модели в облаке	Да	< 20 минут	Облачные и локальные	+ Обучение с помощью графического интерфейса без программирования + Возможность определения нестандартных классов + Возможность выбора между мощными (в облаке) и быстрыми (на устройстве) моделями + Масштабируемость – Использование переноса обучения, что может не позволить обучить модель с нуля
Обучение собственной модели	Да	от 4 часов до 2 дней	Локальные	+ Точная настройка + Большой выбор моделей под разные требования к скорости и точности – Время подготовки и существенная вовлеченность в процесс

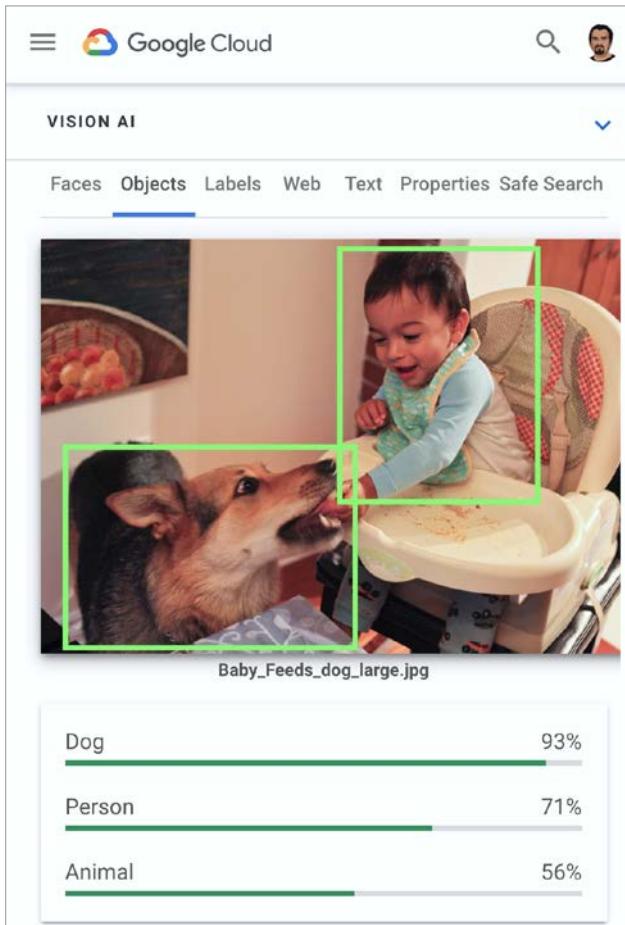
Стоит отметить, что варианты с пометкой «облачные» изначально предусматривают возможность масштабирования. В то же время варианты с пометкой «локальные» также можно развернуть в облаке для достижения высокой масштабируемости, как мы делали это в главе 9.

## Использование готовых облачных API обнаружения объектов

Из главы 8 мы узнали, что облачные API относительно просты в использовании. Достаточно получить учетную запись и ключ API, прочитать документацию и написать клиент REST API. Чтобы упростить этот процесс, многие облачные службы предоставляют SDK для Python (и других языков). К основным облачным API обнаружения объектов относятся Google Vision AI (рис. 14.2) и Microsoft Cognitive Services.

Преимущества облачных API заключаются в их масштабируемости и возможности распознавания тысяч классов объектов. Эти облачные гиганты построили

свои модели, используя большие, постоянно увеличивающиеся проприетарные датасеты, что позволило определить очень богатую таксономию. В наиболее крупных общедоступных размеченных датасетах содержится всего несколько сотен классов, поэтому в мире нет непроприетарных решений, способных обнаруживать тысячи классов.



**Рис. 14.2.** Результаты обнаружения объектов с помощью Google Vision AI API на уже знакомой нам фотографии

Основным недостатком облачных API, конечно же, являются задержки в сети. Из-за этого невозможно выполнять определение в масштабе реального времени. В следующем разделе посмотрим, как обеспечить возможность определения в реальном времени без использования каких-либо данных или обучения.

# Использование предварительно обученных моделей

В этом разделе мы посмотрим, как легко запустить детектор объектов на смартфоне с помощью предварительно обученной модели. Надеемся, вы внимательно прочитали предыдущие главы и освоили работу с нейронными сетями на смартфоне. Код, запускающий модели обнаружения объектов в iOS и Android, вы найдете в репозитории (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке *code/chapter-14*. Для замены модели достаточно заменить существующий файл *.tflite* новым.

В предыдущих главах мы часто использовали модель MobileNet для решения задач классификации. Семейство MobileNet, в том числе MobileNetV2 (2018) и MobileNetV3 (2019), включает только классификационные сети, тем не менее они могут служить основой для архитектуры обнаружения объектов SSD MobileNetV2, которую мы используем в этой главе.

## Получение модели

Лучший способ разобраться в магии — внимательно рассмотреть модель. Обратимся к репозиторию TensorFlow Models (<https://oreil.ly/9CsHM>), где есть модели для решения более чем 50 задач глубокого обучения, включая классификацию аудио, суммаризацию текста и обнаружение объектов. Кроме моделей, там есть также служебные скрипты, которые мы используем в этой главе. Но хватит разговоров! Клонируем репозиторий на свою машину:

```
$ git clone https://github.com/tensorflow/models.git && cd models/research
```

Добавьте дополнительные каталоги в переменную окружения PYTHONPATH, чтобы сделать скрипты доступными для Python:

```
$ export PYTHONPATH="${PYTHONPATH}:`pwd`:`pwd`/slim"
```

Скопируйте командный скрипт компилятора Protocol Buffer (protobuf) из репозитория книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в текущий каталог, чтобы сделать файлы *.proto* доступными для Python:

```
$ cp {path_to_book_github_repo}/code/chapter-14/add_protoc.sh . && \
chmod +x add_protoc.sh && \
./add_protoc.sh
```

Наконец, запустите скрипт *setup.py*, как показано ниже:

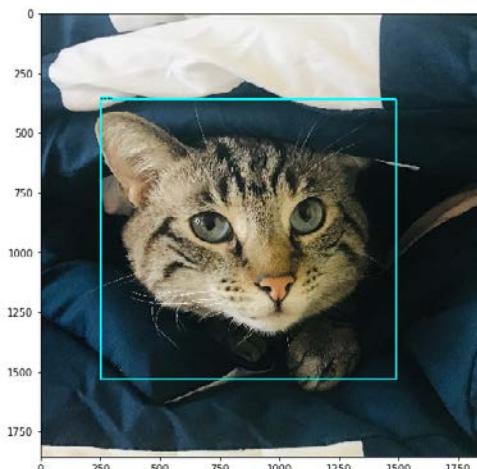
```
$ python setup.py build
$ python setup.py install
```

Теперь загрузите готовую модель. Здесь мы используем модель SSD MobileNetV2. В репозитории вы найдете зоопарк моделей TensorFlow Object Detection ([https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/tf1\\_detection\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md)), обученных обнаружению объектов, в котором для каждой модели указываются датасет, использовавшийся для обучения, скорость инференса и средний показатель точности (Mean Average Precision, mAP). Отыщите в этом списке модель `ssd_mobilenet_v2_coco`, обученную на датасете MS COCO, и загрузите ее. В ходе испытаний эта модель выполняла прогон за 31 мс со значением mAP 22 на графическом процессоре NVIDIA GeForce GTX TITAN X, получая на входе изображения разрешением  $600 \times 600$  пикселей. После загрузки модели распакуйте ее в каталог `models/research/object_detection` внутри репозитория TensorFlow. Вот как можно проверить содержимое каталога после распаковки модели:

```
$ cd object_detection/  
$ ls ssd_mobilenet_v2_coco_2018_03_29  
checkpoint           model.ckpt.data-00000-of-00001  model.ckpt.meta  
saved_model          model.ckpt.index            pipeline.config  
frozen_inference_graph.pb
```

## Тест-драйв модели

Прежде чем подключить модель к мобильному приложению, проверим, как она делает прогнозы. В репозитории TensorFlow Models есть блокнот Jupyter, куда можно вставить фотографию и получить прогноз. Он доступен как `models/research/object_detection/object_detection_tutorial.ipynb`. На рис. 14.3 показан прогноз для уже знакомой нам фотографии, полученный в этом блокноте.



**Рис. 14.3.** Прогноз, полученный от модели обнаружения объектов в блокноте Jupyter из репозитория TensorFlow Models

## Развертывание на устройстве

Убедившись, что модель работает, преобразуем ее в формат для мобильных устройств. Для этого используем уже знакомый инструмент `tflite_convert` из главы 13. Следует отметить, что этот инструмент работает с файлами `.pb`, тогда как зоопарк моделей TensorFlow Object Detection предоставляет только контрольные точки модели. Поэтому сначала из контрольной точки и графа сгенерируем модель `.pb`. Сделаем это с помощью скрипта `export_tflite_ssd_graph.py` из репозитория TensorFlow Models, который можно найти в папке `models/research/object_detection`:

```
$ python export_tflite_ssd_graph.py \
--pipeline_config_path=ssd_mobilenet_v2_coco_2018_03_29/pipeline.config \
--trained_checkpoint_prefix=ssd_mobilenet_v2_coco_2018_03_29/
model.ckpt.data-00000-of-00001 \
--output_directory=tflite_model \
--add_postprocessing_op=true
```

Если скрипт выполнится успешно, то в каталоге `tflite_model` появятся следующие файлы:

```
$ ls tflite_model
tflite_graph.pb
tflite_graph.pbtxt
```

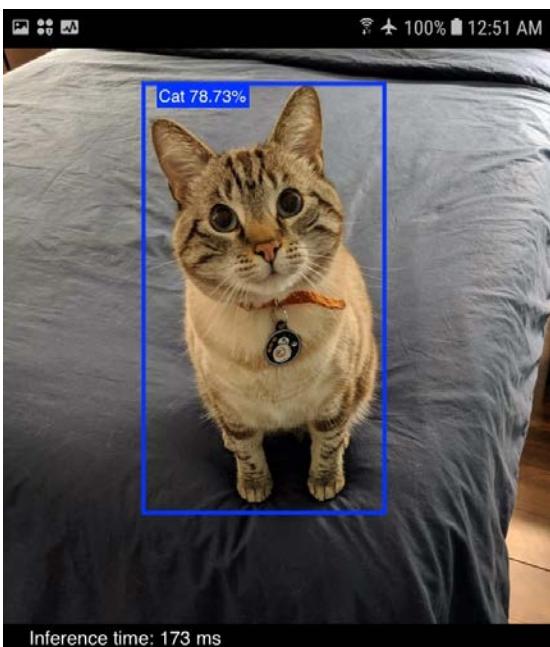
Преобразуйте их в формат TensorFlow Lite с помощью `tflite_convert`.

```
$ tflite_convert --graph_def_file=tflite_model/tflite_graph.pb \
--output_file=tflite_model/model.tflite
```

Теперь остается только подключить модель к приложению. Приложение, которое мы будем использовать, вы найдете в репозитории книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке `code/chapter-14`. Мы уже показывали, как менять модели в приложении в главах 11, 12 и 13, поэтому не будем на этом останавливаться. На рис. 14.4 показан результат, полученный от модели обнаружения объектов после ее подключения к приложению Android.

Причина, почему мы сразу же увидели прогноз «Cat» (кот), объясняется тем, что «Cat» — это одна из 80 категорий, которая есть в датасете MS COCO, использовавшемся для обучения модели.

Бобу будет достаточно развернуть эту модель на своих устройствах (Боб может не использовать мобильные телефоны для наблюдения за садом, но процесс развертывания моделей TensorFlow Lite на краевых и мобильных устройствах почти ничем не отличается). Но правильнее было бы повысить точность модели, дополнительного обучив ее на фотографиях, снятых в саду Боба. В следующем разделе мы посмотрим, как использовать перенос обучения для создания своей модели обнаружения объектов, используя только веб-инструмент. Если вы читали главу 8, то процесс покажется вам знакомым.



**Рис. 14.4.** Обнаружение объектов в режиме реального времени на устройстве с Android

## Создание своей модели обнаружения объектов без программирования

*От моей кошки пахнет кошачьей едой!*

*Ральф Виггам*

Скажем честно. Для этого эксперимента мы — авторы книги — могли бы пойти и нафотографировать кучу котов в своих садах. Но это довольно утомительно и дало бы прибавку к точности и полноте всего в несколько процентных пунктов. С другой стороны, можно было бы рассмотреть по-настоящему интересный пример и заодно проверить пределы возможностей CNN. Речь, конечно, про «Симпсонов». Учитывая, что для обучения большинства моделей не используют персонажей мультфильмов, было бы интересно посмотреть, как модель справится с ними.

Для начала нужно сформировать датасет. К счастью, нужный нам готовый датасет уже есть на Kaggle (<https://oreil.ly/-3wvj>). Загрузим его и воспользуемся сервисом CustomVision.ai (как было показано в главе 8) для обучения своего классификатора Simpsons Classifier, выполнив следующие шаги.



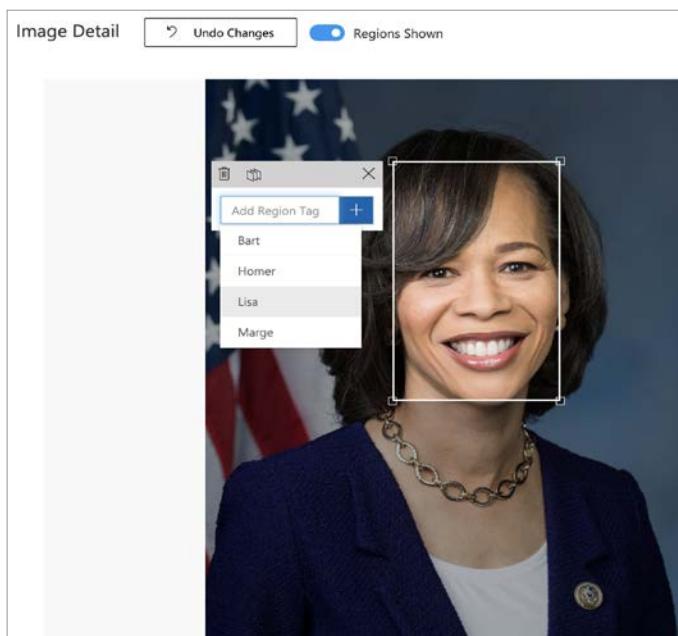
К сожалению, у авторов этой книги нет миллионов долларов. Поэтому мы не можем позволить себе выкупить права на «Симпсонов» у кинокомпании Fox. Законы об авторском праве не позволяют нам напечатать в книге изображения героев. Поэтому мы используем фотографии женщин и мужчин — конгрессменов США, имена которых совпадают с именами героев мультсериала: *Гомера Хоха* (Homer Hoch), *Мардж Рукемы* (Marge Roukema), *Барта Гордона* (Bart Gordon) и *Лизы Блант Рочестер* (Lisa Blunt Rochester). Но имейте в виду — эти фотографии только для печати в книге, в реальности мы обучали свой классификатор на исходном датасете с героями мультфильма.

1. Откройте сайт *CustomVision.ai* и создайте новый проект типа Object Detection (Обнаружение объектов; рис. 14.5).

The screenshot shows the 'Create new project' dialog box. It includes fields for 'Name\*' (Simpsons Detector), 'Description' (¡Ay Caramba!), a 'Resource' dropdown (Simpsons [FO]), 'Project Types' (Object Detection selected), 'Domains' (General selected), and 'Export Capabilities' (Basic platforms selected). At the bottom are 'Cancel' and 'Create project' buttons.

**Рис. 14.5.** Создание нового проекта типа Object Detection (Обнаружение объектов) в CustomVision.ai

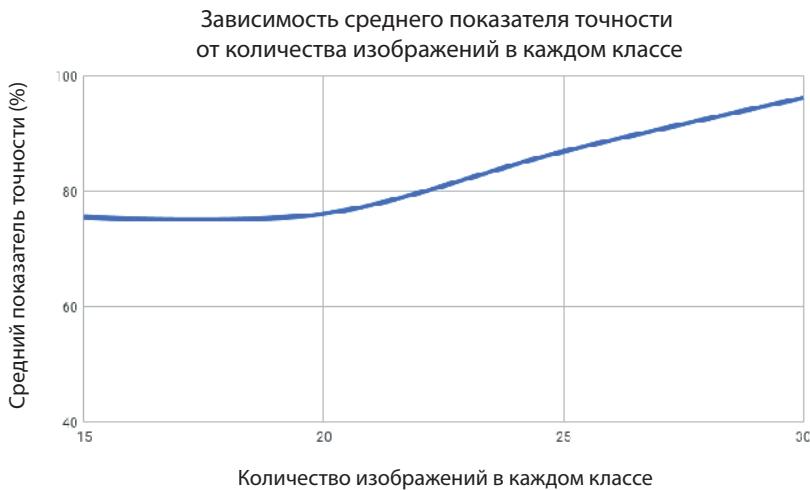
2. Добавьте теги для разметки фотографий Гомера, Мардж, Барта и Лизы. Загрузите по 15 изображений каждого персонажа и на каждом изображении нарисуйте ограничивающую рамку. Панель управления при этом должна выглядеть, как показано на рис. 14.6.



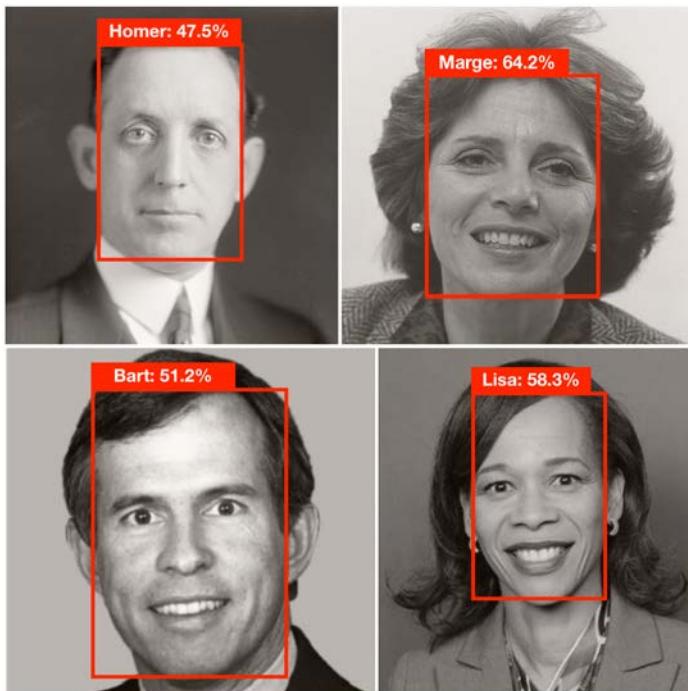
**Рис. 14.6.** Панель управления с ограничивающей рамкой и названием класса

3. Щелкните на кнопке **Train** (Обучить), чтобы запустить процесс обучения. На дашборде есть ползунки для управления порогами вероятности (probability) и частичного совпадения предсказанной ограничивающей рамки с истинной (overlap). При желании вы можете поэкспериментировать с ними и подобрать настройки, обеспечивающие максимальную точность и полноту.
4. Щелкните на кнопке **Quick Test** (Быстрая проверка) и используйте любое случайное изображение Симпсонов (ранее не использовавшееся для обучения), чтобы проверить качество работы модели. Результаты могут получиться не очень хорошими, но ничего страшного: эту проблему можно исправить, повторив обучение на большем количестве данных.
5. Давайте поэкспериментируем и узнаем, сколько изображений нужно, чтобы повысить точность, полноту и метрику mAP. Для начала добавим еще пять изображений каждого класса и повторно обучим модель.
6. Повторяйте, пока не получите приемлемые результаты. На рис. 14.7 показаны результаты нашего эксперимента.

Окончательная модель достаточно хорошо обнаруживает объекты на случайных изображениях из интернета, как показано на рис. 14.8. Самое удивительное, хотя мы и не ожидали этого, что модель, обученную на естественных изображениях, можно точно настроить на изображениях из мультфильмов, используя относительно небольшое количество данных.



**Рис. 14.7.** Зависимость среднего показателя точности от количества изображений в каждом классе



**Рис. 14.8.** Окончательная модель обнаруживает персонажей Симпсонов, которых представляют члены Конгресса США с теми же именами (см. примечание в начале этого раздела)

Как мы уже знаем, *CustomVision.ai* позволяет экспортить модель в различные форматы, включая Core ML, TensorFlow Lite, ONNX и др. То есть можно просто загрузить модель в желаемом формате (в нашем случае *.tflite*) и подключить этот файл к приложению, как мы сделали это с предварительно обученной моделью в предыдущем разделе. Модель, обнаруживающую Симпсонов в режиме реального времени, можно показать друзьям и семье, когда в следующий раз директор Скиннер заведет рассказ о вареной ветчине по телевизору.



*CustomVision.ai* — не единственная платформа, позволяющая размечать, обучать и развертывать модели онлайн без программирования. Похожие возможности предлагают Google Cloud AutoML (бета-версия по состоянию на октябрь 2019 года) и Apple Create ML (только для экосистемы Apple). Кроме того, служба Matroid позволяет создавать пользовательские модели обнаружения объектов в видеопотоках, что может быть очень полезно для обучения сети без затрат значительных усилий на создание датасета.

Мы рассмотрели три быстрых способа сделать детектор объектов с минимальным объемом кода. По нашим оценкам, любого из них будет вполне достаточно в 90 % случаев. Если какие-то из этих вариантов или все подходят для вашего сценария, то с этого места сразу переходите к разделу «Примеры из практики».

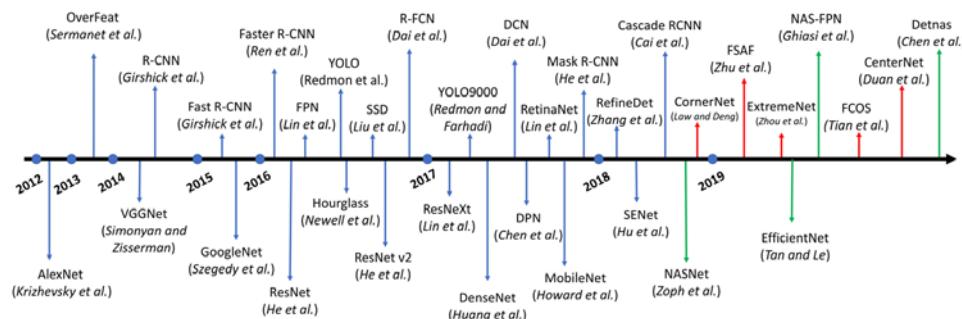
В очень редких ситуациях потребуется более полный контроль над точностью, размером модели и использованием ресурсов. В следующих разделах подробнее изучим мир обнаружения объектов, начиная от разметки данных и заканчивая развертыванием модели.

## Развитие технологий обнаружения объектов

С революцией в области глубокого обучения наступил ренессанс не только в сфере классификации, но и для других задач компьютерного зрения, включая обнаружение объектов. За последние годы было предложено несколько архитектур обнаружения объектов, часто основанных друг на друге. Некоторые из них показаны на рис. 14.9.

В случае с классификацией объектов сверточная сеть извлекает признаки из изображения и вычисляет вероятности для определенного количества классов. Эти сверточные архитектуры (ResNet, MobileNet) — основа для моделей обнаружения объектов. В моделях обнаружения объектов конечным результатом являются ограничивающие рамки (определяются точкой центра, высотой и шириной). Обсуждение внутреннего устройства этих моделей займет много времени (но эту информацию можно найти в репозитории книги на GitHub <https://github.com/>

*PracticalDL/Practical-Deep-Learning-Book*), поэтому ограничимся более общими представлениями и разделим их на две категории (табл. 14.3).



**Рис. 14.9.** Хронология различных архитектур обнаружения объектов (источник: статья «Recent Advances in Deep Learning for Object Detection», написанная Сюнвэй Би (Xiongwei Wu) с коллегами)

**Таблица 14.3.** Категории моделей обнаружения объектов

	Описание	Достоинства и недостатки
Двухступенчатые детекторы	С помощью сети выделения предполагаемых регионов (Region Proposal Network, RPN) генерируются ограничивающие рамки кандидатов (интересующих регионов) независимо от категории. Далее множество кандидатов подается на вход сверточной сети, чтобы получить категорию для каждого. Примеры: Faster R-CNN, Mask R-CNN	+ Высокая точность – Низкая скорость
Одноступенчатые детекторы	Напрямую предсказывают категорию для каждого местоположения в карте признаков; обучаются сквозным обучением. Примеры: YOLO, Single-Shot Detector (SSD)	+ Высокая скорость, лучше подходит для приложений, действующих в режиме реального времени. – Низкая точность



В мире обнаружения объектов хорошо известен Росс Гиршик (Ross Girshik), чье имя часто можно встретить в статьях. До и после начала эпохи глубокого обучения он работал над моделями Deformable Part Models (DPM), R-CNN, Fast R-CNN, Faster R-CNN, You Only Look Once (YOLO), Mask R-CNN, Feature Pyramid Network (FPN), RetinaNet и ResNeXt. Это лишь немногие из тех моделей, что из года в год ставят новые рекорды в публичных состязаниях.

Я участвовал в нескольких конкурсах PASCAL VOC по обнаружению объектов, где брал первые места и был награжден «за достижения в профессиональной деятельности» за свою работу над деформируемыми моделями частей (Deformable Part Models, DPM). Думаю, что основная заслуга принадлежит конкурсу PASCAL, а не мне.

*Росс Гиришик*

## Вопросы производительности

Для практиков наиболее важны точность и скорость. Эти два фактора часто обратно пропорциональны. Нужна такая модель обнаружения объектов, которая имела бы идеальный баланс. В табл. 14.4 перечислены некоторые предварительно обученные модели из репозитория TensorFlow. Скорость прогнозирования измерялась для входных изображений с разрешением  $600 \times 600$  пикселей с использованием видеокарты NVIDIA GeForce GTX TITAN X.

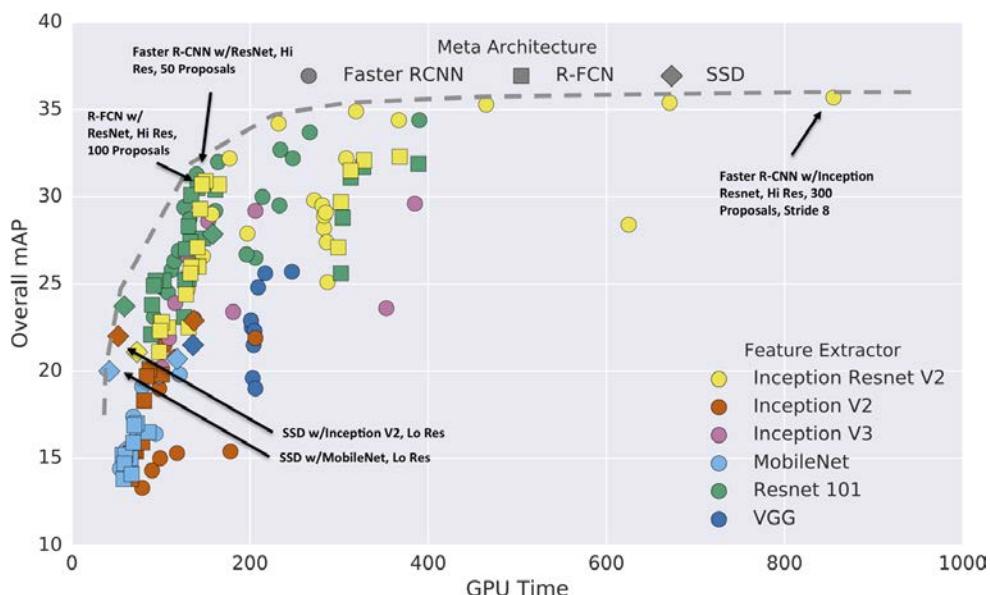
**Таблица 14.4.** Скорость и средний показатель точности для некоторых предварительно обученных моделей из репозитория TensorFlow

	Время инференса (мс)	mAP на MS COCO (%)
ssd_mobilenet_v1_coco	30	21
ssd_mobilenet_v2_coco	31	22
ssdlite_mobilenet_v2_coco	27	22
ssd_resnet_50_fpn_coco	76	35
faster_rcnn_nas	1.833	43

По результатам исследований, проведенных Google в 2017 году (рис. 14.10), были сделаны следующие выводы.

- Одноступенчатые детекторы быстрее двухступенчатых, но менее точны.
- Чем выше точность базовой архитектуры для задач классификации, тем выше точность обнаружения объектов.
- Объекты небольшого размера менее надежно определяются детекторами (mAP обычно ниже 10 %). Этот эффект сильнее проявляется в одноступенчатых детекторах.
- При анализе объектов большого размера большинство детекторов имеют схожие характеристики.

- Изображения с более высоким разрешением дают лучшие результаты, потому что для сети небольшие объекты выглядят крупнее.
- Двухступенчатые детекторы внутренне генерируют большое количество кандидатов, которые нужно рассмотреть в качестве потенциальных местоположений объектов. Предельное количество кандидатов настраивается. Уменьшение порога может привести к ускорению с небольшой потерей точности (порог может зависеть от конкретной ситуации).



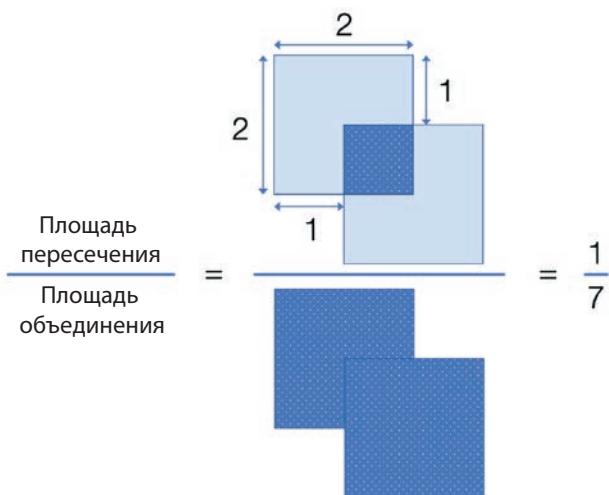
**Рис. 14.10.** Влияние архитектуры модели обнаружения объектов, а также базовой архитектуры (используемой для извлечения признаков), на усредненный показатель точности и время вычисления прогноза (обратите внимание, что в черно-белом варианте цвета могут быть нечеткими). Источник: статья «Speed/accuracy trade-offs for modern convolutional object detectors» Джонатана Хуанга (Jonathan Huang) и др.

## Ключевые термины в обнаружении объектов

В обнаружении особенно часто используются следующие термины: Intersection over Union (отношение площади пересечения к площади объединения), Mean Average Precision (средний показатель точности), Non-Maximum Suppression (подавление немаксимумов) и Anchor (якорь). Давайте посмотрим, что означает каждый из них.

## Intersection over Union

В пайплайне обучения моделей-детекторов показатель IoU обычно используется в качестве порогового значения для фильтрации обнаружений определенного качества. Для вычисления метрики IoU используются две ограничивающие рамки — предсказанная и истинная — и на их основе вычисляются два значения. Первое — площадь области пересечения предсказанной и истинной рамок — пересечение. Второе — площадь фигуры, образованной наложением предсказанной и истинной рамок, — объединение. Как следует из названия<sup>1</sup>, метрика IoU вычисляется делением площади пересечения на площадь объединения. Рисунок 14.11 показывает идею вычисления IoU для двух квадратов  $2 \times 2$ , которые имеют область пересечения  $1 \times 1$  посередине.



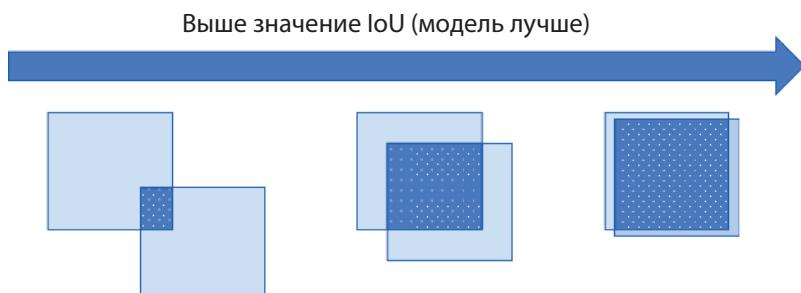
**Рис. 14.11.** Визуальное представление отношения IoU

В идеальном случае, когда предсказанная рамка совпадает с истинной, метрика IoU будет иметь значение 1. В худшем случае предсказанная рамка не будет перекрываться с истинной, и метрика IoU будет равна 0. Как видите, значение IoU изменяется в диапазоне от 0 до 1, причем чем выше значение, тем качественнее прогноз, как показано на рис. 14.12.

Для фильтрации некачественных результатов можно установить минимальное пороговое значение IoU. Установка излишне агрессивного порога (например, 0,9) приведет к потере многих прогнозов, которые могут оказаться важными в даль-

<sup>1</sup> Термин «Intersection over Union» можно дословно перевести как «пересечение на объединение». — Примеч. пер.

нейшем. И наоборот, установка слишком низкого порога приведет к появлению слишком большого количества ложных ограничивающих рамок. Обычно в качестве порога IoU для моделей-детекторов используется значение не ниже 0,5.



**Рис. 14.12.** Обычно чем лучше модель, тем больше перекрываются предсказанные ограничивающие рамки с истинными, что дает более высокое значение IoU

Еще раз отметим, что IoU рассчитывается для каждого экземпляра в каждой категории, а не для изображения. Но для оценки качества детектора по большему набору изображений часто в сочетании с IoU используется еще одна метрика — средний показатель точности, mAP.

## Mean Average Precision

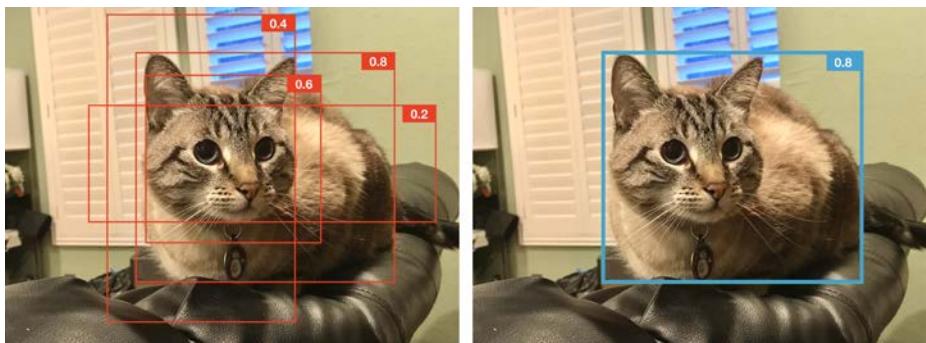
Читая исследовательские работы по обнаружению объектов, мы часто сталкиваемся с показателем AP@0,5. Так обозначается средняя точность при  $\text{IoU} = 0,5$ . Другое более сложное представление: AP@[0,6:0,05:0,75] — это средняя точность при IoU от 0,6 до 0,75 с шагом 0,05. Средний показатель точности, или mAP, — это просто средняя точность по всем категориям. В состязаниях COCO используется метрика mAP: AP@[0,5:0,05:0,95].

## Non-Maximum Suppression

Алгоритмы обнаружения объектов делают предложения о потенциальных местоположениях объектов, которые есть на изображении. Ожидается, что для каждого объекта будет обнаружено несколько таких предполагаемых ограничивающих рамок с различными значениями достоверности. Наша задача — выбрать те из них, которые наиболее точно отражают реальные местоположения объектов. Наивно было бы рассматривать только предсказанные рамки с максимальной оценкой достоверности. Этот подход был бы оправдан, если бы на изображении был один объект. Но он не годится, если на одном изображении несколько категорий с несколькими экземплярами в каждой.

В этой ситуации на помощь приходит подавление не-максимумов (Non-Maximum Suppression, NMS; рис. 14.13). Ключевая идея NMS состоит в том, что два экземпляра, принадлежащие одной категории, не будут иметь сильно перекрывающиеся ограничивающие рамки. Например, значение IoU ограничивающих рамок таких экземпляров будет меньше определенного порога (скажем, 0,5). Избыточный способ заключается в том, чтобы повторить следующие шаги для каждой категории:

1. Отфильтровать все предсказания с оценкой достоверности ниже минимального порога.
2. Выбрать предсказание с максимальным значением оценки достоверности.
3. Для всех оставшихся предсказаний, отсортированных в порядке убывания их оценок достоверности, проверить, не превышает ли метрика IoU текущей рамки по отношению к одному из ранее принятых предсказаний величины 0,5. Если превышает, то отбросить это предсказание, иначе принять как допустимое для дальнейшего анализа.



**Рис. 14.13.** Использование метрики NMS для поиска ограничивающих рамок, точнее представляющих местоположение объекта на изображении

## Создание своих моделей с помощью TensorFlow Object Detection API

В этом разделе рассмотрим пример создания детектора объектов, включая сбор, разметку и предварительную обработку данных, обучение модели и ее экспорт в формат TensorFlow Lite.

Ниже приведены некоторые стратегии сбора данных.

## Сбор данных

Вы уже знаете, что в мире глубокого обучения данные правят всем. Есть несколько способов получить данные об объекте, который нужно обнаружить.

### *Использование готовых датасетов*

Есть несколько общедоступных датасетов, пригодных для обучения моделей-детекторов, в том числе MS COCO (80 категорий), ImageNet (200 категорий), Pascal VOC (20 категорий) и более новый датасет Open Images (600 категорий). MS COCO и Pascal VOC используются в большинстве состязаний по обнаружению объектов, причем состязания на основе COCO более подходят для реальных сценариев из-за более сложных изображений.

### *Загрузка изображений из интернета*

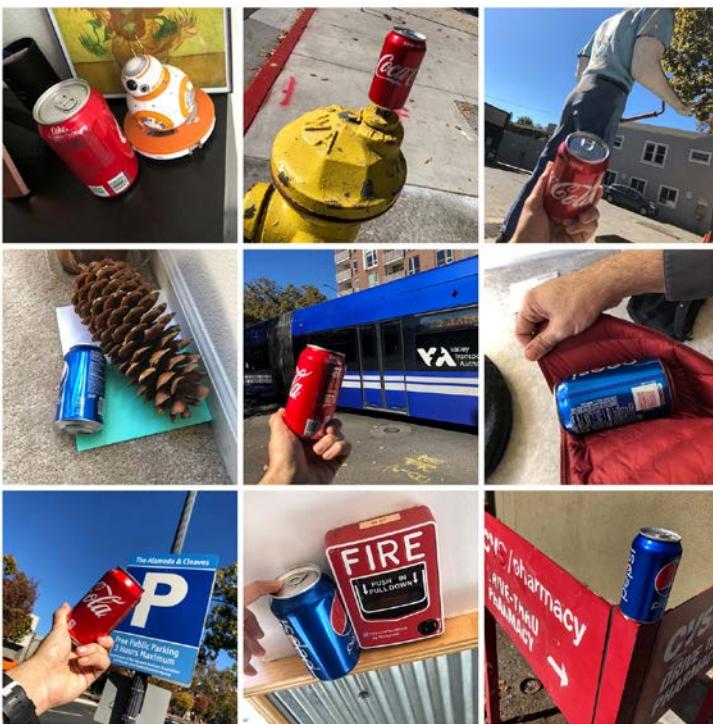
Мы использовали этот подход в главе 12, где собирали изображения для классов «хот-дог» и «не хот-дог». Для быстрой загрузки большого количества изображений можно использовать расширение Fatkun.

### *Фотографирование вручную*

Это наиболее трудоемкий (но и наиболее интересный) вариант. Для создания надежной модели важно обучать ее на изображениях, сделанных в самых разных условиях. Имея в руках объект для обнаружения, нужно сделать не менее 100–150 его фотографий на разном фоне, под разными ракурсами и в разных условиях освещения. На рис. 14.14 показаны примеры изображений, созданных для обучения модели обнаружения объектов «Coke and Pepsi». Учитывая, что модель может запомнить ложную корреляцию, посчитав, что красный цвет означает кока-колу, а синий — пепси, желательно сфотографировать объекты на разном фоне, который потенциально может сбить с толку. Так в итоге получится более надежная модель.

Объект желательно снимать там, где его присутствие маловероятно, — это разнообразит датасет и повышает надежность модели. Такой подход, помимо прочего, превращает скучный и утомительный процесс в интересное занятие. Чтобы было еще интересней, можно задаться целью создать уникальные и новаторские фотографии. На рис. 14.15 — примеры таких необычных фотографий, сделанных для датасета детектора банкнот.

Поскольку наша модель должна обнаруживать котов, переиспользуем картинки из датасета Kaggle «Cats and Dogs», который был в главе 3. Для этого примера мы случайным образом выбрали изображения и разделили их на обучающую и контрольную выборки.



**Рис. 14.14.** Фотографии объектов, сделанные в различных условиях, для обучения модели. По QR-коду вы можете посмотреть рисунок в цвете

Для единообразия размеров входных данных для нашей сети приведем все изображения к фиксированным размерам. Эти же размеры будем использовать при определении сети и при конвертации в модель *.tflite*. С помощью ImageMagick можно изменить размеры всех изображений сразу:

```
$ apt-get install imagemagick
$ mogrify -resize 800x600 *.jpg
```



Если в текущем каталоге у вас собрано много изображений (например, несколько десятков тысяч), то предыдущая команда может обработать не все. Чтобы исправить проблему, можно использовать команду *find*, чтобы вывести список всех изображений, и передать полученный список команде *mogrify*:

```
$ find . -type f | awk -F. '!a[$NF]++{print $NF}' |
xargs -I{} mogrify -resize 800x600 *.jpg
```



**Рис. 14.15.** Необычные фотографии, сделанные при создании датасета для обучения модели обнаружения банкнот

## Разметка данных

Следующий шаг после сбора данных — их разметка. В отличие от задачи классификации, когда достаточно поместить изображения в соответствующие каталоги, здесь надо вручную нарисовать ограничивающие рамки вокруг интересующих объектов. Для этого прекрасно подойдет инструмент LabelImg (доступен для Windows, Linux и Mac), потому что он:

- позволяет сохранять аннотации в файлах XML в формате PASCAL VOC (также принятом в ImageNet) или в формате YOLO;
- поддерживает одно- и многоклассовые метки.

Загрузите приложение с GitHub (<https://oreil.ly/jH31E>) в каталог на своем компьютере. В каталоге вы найдете выполняемый файл и каталог *data* с образцами данных. Поскольку мы собираемся использовать свои данные, данные из каталога *data* не потребуются. Чтобы запустить приложение, дважды щелкните на выполняемом файле.



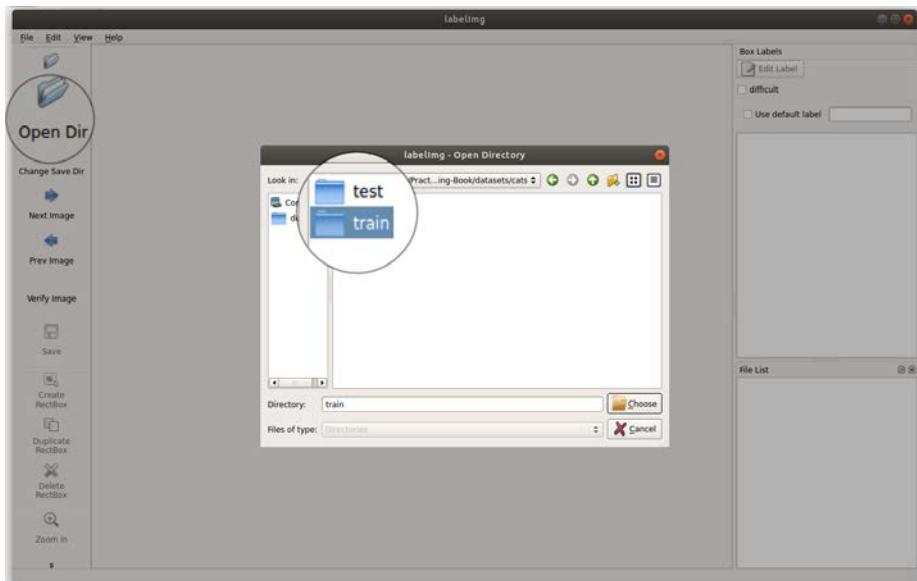
Если вам интересно, чем отличаются форматы YOLO и PASCAL VOC, рассказываем: в формате YOLO используются простые файлы `.txt`, по одному на каждое изображение и с теми же именами, в которых хранится информация о классах и ограничивающих рамках. Вот как выглядит типичный файл `.txt` в формате YOLO:

```
class_for_box1 box1_x box1_y box1_w box1_h  
class_for_box2 box2_x box2_y box2_w box2_h
```

Координаты  $x$ ,  $y$ , а также ширина и высота рамок в этом примере нормализованы по полной ширине и высоте изображений.

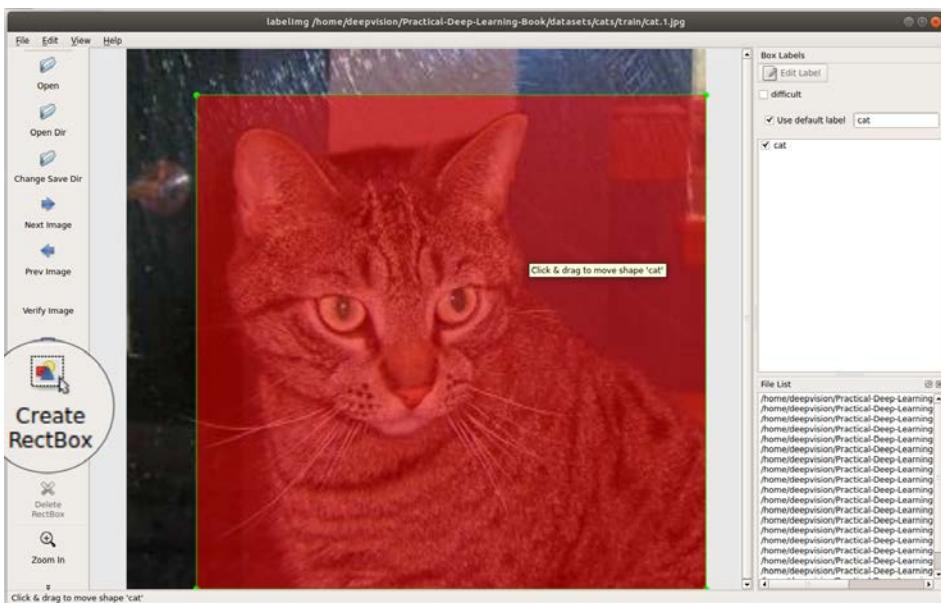
Формат PASCAL VOC, напротив, основан на XML. Так же как в формате YOLO, каждому изображению соответствует один XML-файл (в идеале с тем же именем). Типичным представителем файлов в этом формате является файл `code/chapter-14/pascal-voc-sample.xml` в репозитории книги.

На данный момент у вас уже должна быть коллекция изображений для использования в обучении. Сначала их нужно случайным образом разделить на обучающую и контрольную выборки и поместить в отдельные каталоги. Для этого достаточно перетащить случайные изображения в любой каталог. После создания каталогов `train` и `test` загрузите каталог `train`, щелкнув на кнопке Open Dir (Открыть каталог), как показано на рис. 14.16.



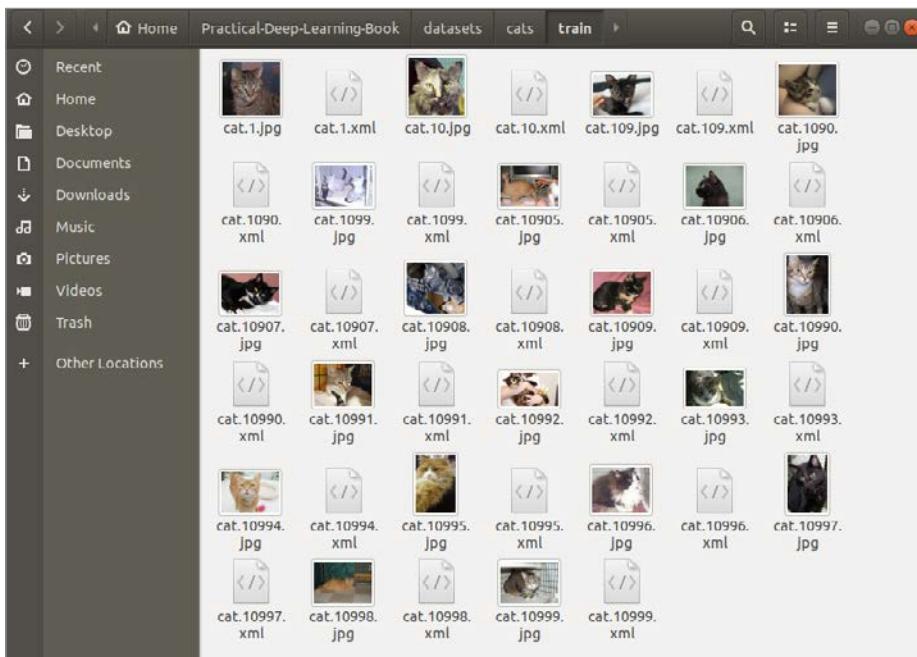
**Рис. 14.16.** Щелкните на кнопке Open Dir и выберите каталог с обучающей выборкой

После того как LabelImg загрузит каталог *train*, начинаем разметку. Для этого нужно просмотреть каждое изображение и вручную нарисовать ограничивающие рамки вокруг каждого объекта (в нашем случае только вокруг котов), как показано на рис. 14.17. После добавления ограничивающей рамки вам будет предложено указать для нее метку. Для этого введите название объекта: «*cat*». После ввода метки установите флажок, чтобы переиспользовать эту же метку при разметке последующих изображений. На картинках с несколькими объектами нарисуйте несколько рамок и добавьте соответствующие метки. Для объектов разных классов добавьте соответствующие им метки.



**Рис. 14.17.** Щелкните на кнопке Create RectBox (Создать рамку) на панели слева, чтобы создать рамку, охватывающую кота

Повторите этот шаг для всех картинок в обучающей и контрольной выборках. Нарисовать рамки вокруг каждого объекта надо так, чтобы ни одна часть объекта не выходила за границы рамки и при этом внутрь рамки не попало ничего лишнего. После этого в каталогах *train* и *test* появятся файлы *.xml*, по одному для каждого файла, как показано на рис. 14.18. Откройте файл *.xml* в текстовом редакторе и проверьте метаданные — имя файла изображения, координаты ограничивающей рамки и имя метки.



**Рис. 14.18.** У каждой картинки есть XML-файл с информацией о метке и ограничивающей рамке

В главе 1 мы говорили, что при создании датасета MS COCO требовалось примерно по три секунды, чтобы добавить метку с именем каждого объекта на изображении, примерно по 30 секунд, чтобы добавить ограничивающую рамку вокруг каждого объекта, и по 79 секунд, чтобы нарисовать очертания каждого объекта.

Разметка для задач обнаружения объектов может занять в 10 раз больше времени, чем для задач классификации. В больших масштабах при сохранении высокого качества такая работа может стоить очень дорого. Что удорожает работу? Рисование рамок вручную. Есть ли другие операции, менее дорогостоящие? Да, проверка правильности нарисованных рамок. Было бы здорово, если можно было бы сократить затраты на первые за счет вторых.

Вместо разметки большого объема данных вручную мы использовали следующий итеративный полуавтоматический процесс, который вы можете повторить. Для этого:

1. Выберите небольшую часть изображений из общего объема данных.
2. Выполните их разметку вручную, указав класс объекта и информацию об ограничивающей рамке.
3. Обучите модель на размеченных данных.

4. С помощью этой модели получите прогнозы для оставшихся неразмеченных данных.
5. Если достоверность прогноза выше установленного порога, присвойте метку.
6. Выберите изображения с достоверностью ниже установленного порога.
7. Проверьте правильность прогнозов. Если прогноз неверен, исправьте его вручную. На практике для этого обычно нужно немного сместить рамку, что намного быстрее, чем рисование рамки с нуля.
8. Добавьте изображения из выборки, проверенной вручную, в обучающий датасет.
9. Повторяйте, пока модель не достигнет приемлемого качества на выделенном проверочном наборе.

На первых итерациях этого процесса модель может часто ошибаться. Это нормально. Используя подход с участием в процессе человека, вы постепенно научите модель не допускать ошибок. Довольно скоро она сможет достаточно уверенно автоматически размечать большую часть данных, сократив затраты на ручную разметку.

Это один из примеров *активного обучения*. Многие компании, предлагающие услуги разметки данных, например Figure Eight, активно используют этот прием для снижения трудозатрат на разметку и повышения эффективности. Поэтому значительную часть времени можно уделить проверке, а не самой разметке.

## Предварительная обработка данных

Теперь у нас есть XML-данные, которые описывают рамки вокруг всех наших объектов. Но чтобы использовать эти данные для обучения, их следует обработать и преобразовать в формат, который понимает TensorFlow, а именно в формат TFRecords. Прежде чем преобразовать данные в этот формат, нужно объединить данные из всех файлов XML в один файл CSV (Comma-Separated Values — значения, разделенные запятыми). Для этого используем скрипты, которые предлагает TensorFlow.

Теперь, когда все готово, приступим к работе:

1. Объедините обучающие данные из разрозненных XML-файлов в один CSV-файл с помощью инструмента `xml_to_csv` из репозитория `gacoon_dataset` (<https://oreilly/k8QGI>) Дата Трэна (Dat Tran). В нашем репозитории вы найдете отредактированную копию этого инструмента в файле `code/chapter-14/xml_to_csv.py`:

```
$ python xml_to_csv.py -i {путь к обучающей выборке cats training dataset} \
-o {путь к выходному файлу train_labels.csv}
```

2. Проделайте ту же операцию с контрольными данными:

```
$ python xml_to_csv.py -i { путь к контрольной выборке cats test dataset} \
-o {путь к выходному файлу test_labels.csv}
```

3. Создайте файл *label\_map.pbtxt*, определяющий соответствия между метками и идентификаторами всех классов. Мы используем этот файл для преобразования текстовых меток в целочисленные идентификаторы, как того требует формат TFRecord. Поскольку у нас всего один класс, файл получился коротким и имеет следующее содержимое:

```
item {
    id: 1
    name: 'cat'
}
```

4. Сгенерируйте файлы в формате TFRecord с данными, которые в дальнейшем будут использоваться для обучения и тестирования модели. Этот инструмент также можно найти в репозитории racoon\_dataset. В нашем репозитории вы найдете отредактированную копию этого инструмента в файле *code/chapter-14/generate\_tfrecord.py*. (Путь в аргументе *image\_dir* должен совпадать с путями в XML-файлах; LabelImg использует абсолютные пути.)

```
$ python generate_tfrecord.py \
--csv_input={путь к train_labels.csv} \
--output_path={путь к train.tfrecord} \
--image_dir={путь к cat training dataset}

$ python generate_tfrecord.py \
--csv_input={путь к test_labels.csv} \
--output_path={путь к test.tfrecord} \
--image_dir={путь к cat test dataset}
```

После создания файлов *train.tfrecord* и *test.tfrecord* можно начинать обучение.

## Исследование модели

(Этот раздел носит исключительно информационный характер и не содержит сведений, важных для процесса обучения. Если хотите, переходите прямо к разделу «Обучение» ниже.)

Исследовать модель можно с помощью инструмента *saved\_model\_cli*:

```
$ saved_model_cli show --dir ssd_mobilenet_v2_coco_2018_03_29/saved_model \
--tag_set serve \
--signature_def serving_default
```

Вот как выглядят результаты анализа нашей модели:

```
The given SavedModel SignatureDef contains the following input(s):
  inputs['inputs'] tensor_info:
    dtype: DT_UINT8
    shape: (-1, -1, -1, 3)
    name: image_tensor:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['detection_boxes'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, 100, 4)
    name: detection_boxes:0
  outputs['detection_classes'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, 100)
    name: detection_classes:0
  outputs['detection_scores'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, 100)
    name: detection_scores:0
  outputs['num_detections'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1)
    name: num_detections:0
  outputs['raw_detection_boxes'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, -1, 4)
    name: raw_detection_boxes:0
  outputs['raw_detection_scores'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, -1, 2)
    name: raw_detection_scores:0
Method name is: tensorflow/serving/predict
```

Ниже приводится подробная интерпретация этих результатов:

1. Форма входного слоя `inputs` определена как  $(-1, -1, -1, 3)$ . Это означает, что обученная модель может принимать на входе изображения произвольного размера, в которых значения пикселей определяются тремя каналами. Из-за гибкости в плане размеров входных данных преобразованная модель получается больше, чем аналогичная модель, принимающая данные фиксированного размера. Когда дальше в этой главе мы начнем обучать свой детектор, то зафиксируем размер входных данных так, чтобы модель принимала только изображения  $800 \times 600$  пикселей. Это позволит сделать модель более компактной.
2. Теперь перейдем к выходным слоям. Самый первый выходной слой — это `detection_boxes`. Его форма  $(-1, 100, 4)$  сообщает о возможном количестве ограничивающих рамок, а также о том, как они выглядят. В частности, первое число (то есть  $-1$ ) указывает, что в прогнозе может быть произволь-

ное количество рамок для всех 100 классов (второе число) и каждая рамка представлена четырьмя координатами (третье число) — x, y, ширина и высота. Иначе говоря, модель не ограничивает количество обнаруженных экземпляров для каждого из 100 классов.

3. Слой `detection_classes` — это список из двух элементов: первый элемент определяет количество обнаруженных объектов, а второй — вектор в формате унитарного (one-hot) кодирования, представляющий обнаруженный класс.
4. `num_detections` — это количество объектов, обнаруженных на изображении. Это единственное число с плавающей запятой.
5. `raw_detection_boxes` определяет координаты каждой ограничивающей рамки для каждого обнаруженного объекта до применения NMS.
6. `raw_detection_scores` — список из двух чисел с плавающей запятой. Первое описывает общее количество обнаружений, а второе — общее количество категорий, включая фон, если он считается отдельной категорией.

## Обучение

Поскольку мы используем модель SSD MobileNetV2, то нужно создать копию файла `pipeline.config` (из репозитория TensorFlow) и привести его в соответствие с параметрами конфигурации. Создайте копию файла конфигурации и найдите в нем строку `PATH_TO_BE_CONFIGURED`. Этот параметр содержит список путей в файловой системе, которые требуется изменить (желательно использовать абсолютные пути). Также нужно отредактировать некоторые другие параметры конфигурации: количество классов (`num_classes`), количество шагов (`num_steps`), количество образцов, выделяемых для проверки (`num_examples`) и путь к файлам с картами меток для обучения и тестирования. (Нашу версию файла `pipeline.config` можно найти в репозитории книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>).)

```
$ cp object_detection/samples/configs/ssd_mobilenet_v2_coco.config  
./pipeline.config  
$ vim pipeline.config
```



В нашем примере мы используем модель SSD MobileNetV2 для обучения детектора объектов. Вы же можете выбрать в качестве основы какую-то другую модель. Поскольку каждая модель распространяется со своим файлом конфигурации пайплайна, следует изменить файл конфигурации для выбранной модели. Процесс здесь аналогичный: определить, какие пути нужно изменить, и исправить их с помощью текстового редактора.

Помимо изменения путей может понадобиться изменить другие параметры в файле конфигурации: размер изображения, количество классов, выбор оптимизатора, скорость обучения и количество эпох.

До сих пор мы находились в каталоге *models/research* в репозитории TensorFlow. Теперь перейдем в каталог *object\_detection* и запустим скрипт *model\_main.py*, входящий в состав TensorFlow. Он обучит нашу модель, опираясь на конфигурацию в только что отредактированном файле *pipeline.config*:

```
$ cd object_detection/  
$ python model_main.py \  
--pipeline_config_path=../pipeline.config \  
--logtostderr \  
--model_dir=training/
```

Понять, что все в порядке, можно по появлению в выводе таких строк:

```
Average Precision (AP) @[ IoU=0.50:0.95 | area=all | maxDets=100 ] = 0.760
```

В зависимости от заданного количества итераций обучение займет от нескольких минут до пары часов, поэтому пока перекусите, выпейте кофе, поменяйте наполнитель в кошачьем лотке или дочитайте главу до конца. Когда обучение завершится, в каталоге *training/* появится файл последней контрольной точки. В процессе обучения также будут созданы следующие файлы:

```
$ ls training/  
  
checkpoint                         model.ckpt-13960.meta  
eval_0                             model.ckpt-16747.data-00000-of-00001  
events.out.tfevents.1554684330.computername model.ckpt-16747.index  
events.out.tfevents.1554684359.computername model.ckpt-16747.meta  
export                            model.ckpt-19526.data-00000-of-00001  
graph.pbtxt                         model.ckpt-19526.index  
label_map.pbtxt                     model.ckpt-19526.meta  
model.ckpt-11180.data-00000-of-00001    model.ckpt-20000.data-00000-of-00001  
model.ckpt-11180.index              model.ckpt-20000.index  
model.ckpt-11180.meta               model.ckpt-20000.meta  
model.ckpt-13960.data-00000-of-00001    pipeline.config  
model.ckpt-13960.index
```

В следующем разделе мы преобразуем файл последней контрольной точки в формат *tflite*.



Иногда для отладки оптимизации или просто ради интереса нужно более глубоко исследовать модель. Это как сцены после титров в кино. Запустите следующую команду, передав в аргументах файл контрольной точки, файл конфигурации и тип ввода, чтобы получить различные параметры, отчет об анализе модели и другие бесценные крупицы информации:

```
$ python export_inference_graph.py \  
--input_type=image_tensor \  
--pipeline_config_path=training/pipeline.config \  
--output_directory=inference_graph \  
--trained_checkpoint_prefix=training/model.ckpt-20000
```

Вот как примерно выглядит вывод этой команды:

```
=====Options=====
-max_depth 10000
-step -1
...
=====Model Analysis Report=====
...
Doc:
scope: The nodes in the model graph are organized by
their names, which is hierarchical like filesystem.
param: Number of parameters (in the Variable).

Profile:
node name | # parameters
_TFProfRoot (--/3.72m params)
    BoxPredictor_0 (--/93.33k params)
        BoxPredictor_0/BoxEncodingPredictor
            (--/62.22k params)
                BoxPredictor_0/BoxEncodingPredictor/biases
                    (12, 12/12 params)
...
FeatureExtractor (--/2.84m params)
...
```

Этот отчет поможет узнать количество параметров, что, в свою очередь, даст возможности для оптимизации модели.

## Конвертация модели

Теперь, имея файл последней контрольной точки (имя файла должно отражать количество эпох, заданное в файле *pipeline.config*), преобразуем его с помощью скрипта `export_tflite_ssd_graph`, как мы делали выше в этой главе с предварительно обученной моделью:

```
$ python export_tflite_ssd_graph.py \
--pipeline_config_path=training/pipeline.config \
--trained_checkpoint_prefix=training/model.ckpt-20000 \
--output_directory=tflite_model \
--add_postprocessing_op=true
```

Если скрипт выполнился успешно, то в каталоге *tflite\_model* появятся следующие файлы:

```
$ ls tflite_model
tflite_graph.pb
tflite_graph.pbtxt
```

Остался последний шаг: конвертировать файлы зафиксированного графа в формат *.tflite*. Сделать это можно с помощью инструмента `tflite_convert`:

```
$ tflite_convert --graph_def_file=tflite_model/tflite_graph.pb \
--output_file=tflite_model/cats.tflite \
--input_arrays=normalized_input_image_tensor \
--output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_\
PostProcess:1', \
'TFLite_Detection_PostProcess:2','TFLite_Detection_PostProcess:3' \
--input_shape=1,800,600,3 \
--allow_custom_ops
```

Некоторые аргументы в этой команде требуют дополнительных пояснений.

- Аргумент в параметре `--input_arrays` просто указывает, что изображения для прогнозирования будут представлены нормализованными тензорами значений типа `float32`.
- Аргументы в параметре `--output_arrays` сообщают, что каждый прогноз будет содержать четыре типа информации: количество ограничивающих рамок, оценки обнаружения, обнаруженные классы и координаты ограничивающих рамок. Это возможно благодаря тому, что мы использовали аргумент `--add_postprocessing_op = true` в скрипте экспорта графа на предыдущем шаге.
- Аргументы в параметре `--input_shape` определяют те же значения размеров, что указаны в файле конфигурации пайплайна `pipeline.config`.

Назначение остальных аргументов ясно без дополнительных пояснений. У вас в каталоге `tflite_model/` должен появиться новый файл модели `cats.tflite`. Его можно подключить к приложению на устройстве с Android или iOS или к приложению на краевом устройстве и приступить к обнаружению котов в режиме реального времени. Модель спасет сад Боба от посягательств хвостатого гостя!



На вход модели MobileNet данные должны подаваться в виде тензора `normalized_image_tensor` нормализованных значений в диапазоне  $[-1,1]$ . Это означает, что каждый пиксель нужно преобразовать (линейно) в диапазон  $[-1,1]$ . Для этого достаточно каждое значение во входном изображении, находящееся в диапазоне от 0 до 255, разделить на 128 (чтобы привести к диапазону  $[0, 2]$ ), а затем прибавить  $-1$ , чтобы привести к диапазону  $[-1,1]$ .

В квантованной модели MobileNet '`normalized_image_tensor`' содержит значения в диапазоне  $[0, 255]$ .

Чтобы узнать больше, посмотрите исходный код функции предварительной обработки в классе экстрактора признаков в репозитории TensorFlow Models, который можно найти в папке `models/research/object_detection/models`.

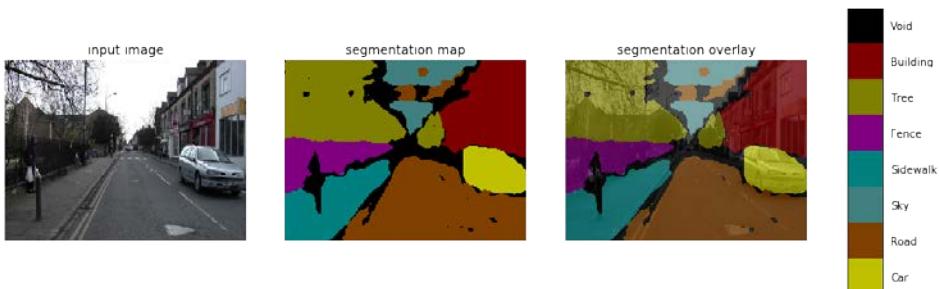
## Сегментация изображений

Но не будем ограничиваться обнаружением объектов и сделаем еще кое-что, чтобы точнее определить местоположение объектов — сегментацию объектов.

Как было показано выше в этой главе, этот шаг включает в себя прогнозирование категории для каждого пикселя в кадре изображения. Для решения задач сегментации обычно используются архитектуры U-Net, Mask R-CNN и DeepLabV3+. По аналогии с обнаружением объектов сейчас наблюдается тенденция к использованию сетей сегментации в режиме реального времени, в том числе на устройствах с ограниченными ресурсами, например в смартфонах. Использование в реальном времени открывает массу новых возможностей для приложений в потребительском сегменте, например наложение фильтров на область лица (рис. 14.19), и в промышленности, например для определения проезжей части беспилотными автомобилями (рис. 14.20).



**Рис. 14.19.** Окрашивание волос с помощью ModiFace путем точного картирования пикселей, соответствующих волосам



**Рис. 14.20.** Сегментация изображения на кадрах с видеорегистратора (датасет CamVid)

Как мы не раз говорили, многое из всего этого можно реализовать, написав совсем немного кода. Инструменты разметки Supervisely, LabelBox и Diffgram помогут не только выполнить разметку данных, но также загрузить ранее размеченные данные и дообучить предварительно обученную модель сегментации объектов. Кроме того, эти инструменты поддерживают разметку с привлечением ИИ, что значительно ускорит (~ в 10 раз) и удешевит трудоемкий и дорогостоящий процесс разметки. Если вас это заинтересовало, то вам повезло! Мы разместили в репозитории книги

(см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) дополнительное руководство о том, как создавать и обучать модели сегментации.

## Примеры из практики

Посмотрим, как обнаружение объектов используется в реальных приложениях.

### Умный холодильник

Умные холодильники становятся все более доступными, а поэтому популярными. Люди обожают заглядывать в холодильники, даже когда они не дома и не могут физически сделать это. Microsoft и швейцарский производственный гигант Liebherr объединили свои усилия и использовали глубокое обучение в холодильнике SmartDeviceBox нового поколения (рис. 14.21). Liebherr использовала модель обнаружения объектов Fast R-CNN, чтобы определять наличие различных продуктов на полках холодильника, вести их учет и держать пользователя в курсе текущего статус-кво.



**Рис. 14.21.** Обнаруженные объекты и их классы в холодильнике SmartDeviceBox (изображение с сайта <https://oreil.ly/xg0wT>)

## Подсчет толпы

Подсчетом людей занимаются не только кошки, глядящие в окно. Это полезно при управлении безопасностью и логистикой на крупных спортивных мероприятиях, политических митингах и других местах с интенсивным движением. Подсчет толпы, как нетрудно догадаться, можно использовать для подсчета любых объектов, включая людей и животных. Охрана дикой природы является исключительно сложной проблемой из-за отсутствия размеченных данных и подходящих стратегий сбора данных.

## Охрана дикой природы

Несколько организаций, в том числе университет Глазго, университет Кейптауна, музей естественной истории имени Филда и научно-исследовательский институт дикой природы в Танзании, собрались вместе, чтобы подсчитать животных, участвующих в крупнейшей на земле миграции: 1,3 миллиона голубых антилоп гну и 250 000 зебр между Серенгети и национальным заповедником Масай Мара в Кении (рис. 14.22). Они выполнили разметку данных с помощью 2200 ученых-добровольцев, а также платформы под названием Zooniverse и использовали алгоритмы автоматического обнаружения объектов, такие как YOLO, для подсчета популяции антилоп гну и зебры. При этом было отмечено, что результаты подсчета добровольцами и алгоритмом обнаружения объектов отличались друг от друга в пределах 1 %.



**Рис. 14.22.** Аэроснимок антилоп гну, сделанный с небольшого самолета  
(изображение с сайта <https://oreil.ly/hTodA>)

## Кумбха Мела

Другой пример — город Праяградж в Индии. Один раз в 12 лет в этом городе проводится фестиваль Кумбха Мела (рис. 14.23), на который съезжается примерно 250 миллионов человек. Управление такой толпой — очень трудная задача. Например, в 2013 году из-за неэффективного управления толпой возникла давка, в результате которой погибло 42 человека.

В 2019 году правительство штата заключило контракт с Larsen & Toubro на использование ИИ для решения различных логистических задач, включая мониторинг трафика, уборку мусора, оценку безопасности, а также мониторинг толпы. Используя более тысячи камер видеонаблюдения, власти проанализировали плотность скопления людей и разработали систему оповещения, основанную на плотности людей в фиксированной зоне. В местах с повышенной плотностью наблюдение было более интенсивным. Эта система вошла в Книгу рекордов Гиннеса как самая большая система управления толпой, когда-либо созданная в истории человечества.



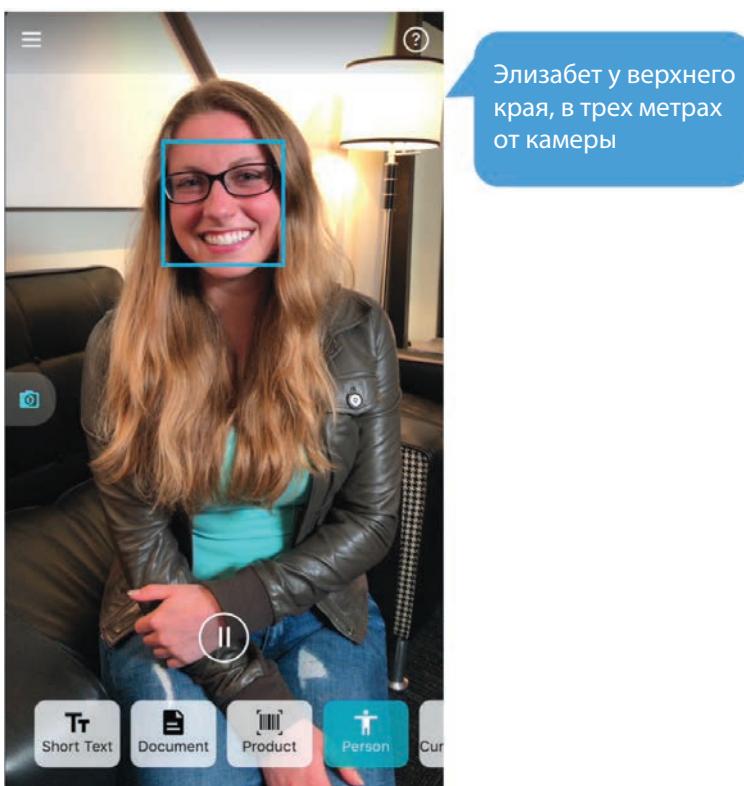
**Рис. 14.23.** Фестиваль Кумбха Мела 2013 года, снято участником  
(изображение с сайта <https://oreil.ly/0ly7N>)

## Распознавание лиц в приложении Seeing AI

Приложение Microsoft Seeing AI (для слепых и слабовидящих) реализует функцию распознавания лиц в реальном времени и информирует пользователя о людях, находящихся перед камерой телефона, их относительном местонахождении и расстоянии от камеры. Типичное уведомление может звучать так: «Одно лицо в верхнем левом углу на расстоянии полутора метров». Кроме того, приложение может выявлять лица по списку известных лиц. Если лицо совпадает с одним из списка, то программа объявит имя человека, например: «Элизабет у верхнего края, в трех метрах от камеры» (рис. 14.24).

Для этого система использует быстрый детектор объектов, оптимизированный для мобильных устройств. Затем часть изображения с лицом передается для дальнейшего анализа алгоритмам распознавания возраста, эмоций, прически и др., действующим в облаке Microsoft Cognitive Services. Для безопасной,

с точки зрения конфиденциальности, идентификации людей приложение просит друзей и семью пользователя сделать три селфи с лицами и генерирует характеристическое (векторное) представление лица, которое сохраняется на устройстве (то есть сами изображения нигде не хранятся). Когда в будущем в потоке кадров с камеры обнаруживается лицо, для него вычисляется векторное представление, которое сравнивается с представлениями в базе данных. Эта функция основана на идеи обучения с одного раза (one-shot), подобно сиамским сетям, которые мы видели в главе 4. Еще одно преимущество отказа от сохранения изображений — размер приложения растет не так сильно, даже при сохранении информации о большом количестве лиц.



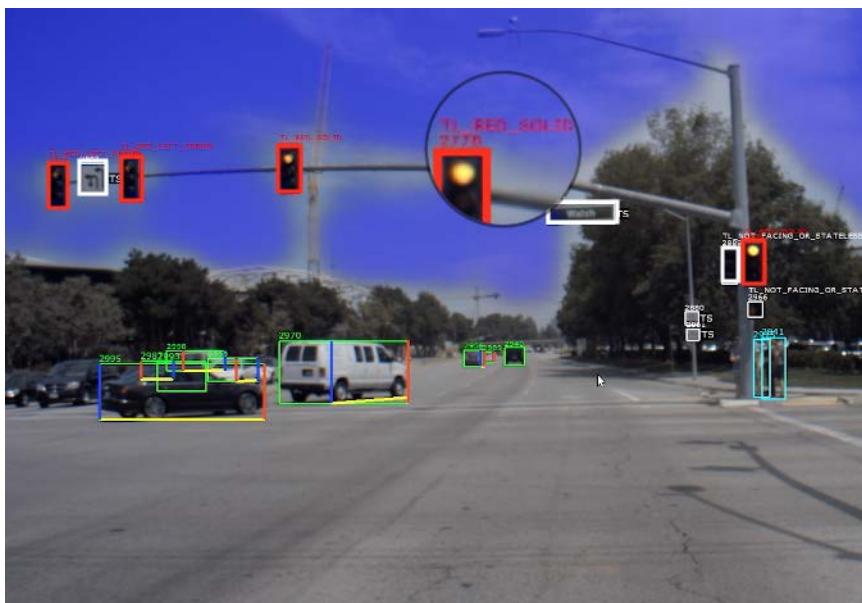
**Рис. 14.24.** Функция обнаружения лиц в Seeing AI

## Беспилотные автомобили

Несколько компаний по производству беспилотных автомобилей, включая Waymo, Uber, Tesla, Cruise, NVIDIA и др., используют обнаружение объектов,

сегментацию и другие методы глубокого обучения для создания беспилотных автомобилей. Усовершенствованные системы помощи водителю, обнаруживающие пешеходов, идентифицирующие тип транспортного средства и распознавающие дорожные знаки, — это важные компоненты беспилотного автомобиля.

Для принятия решений в системе управления беспилотными автомобилями NVIDIA использует специализированные сети, решающие самые разные задачи. Например, WaitNet — нейронная сеть, используемая для быстрого обнаружения светофоров, перекрестков и дорожных знаков. Эта сеть должна работать не только быстро, но и надежно, даже в очень сложных погодных условиях. Ограничивающие рамки вокруг объектов, обнаруженных сетью WaitNet, передаются в специализированные сети для более детальной классификации. Например, если в кадре обнаружился светофор, его изображение передается в LightNet — сеть, определяющую форму светофора (круг или стрелка) и его состояние (красный, желтый, зеленый). Если в кадре обнаружился дорожный знак, он передается в SignNet — сеть классификации, распознающую несколько сотен типов дорожных знаков, утвержденных правилами дорожного движения в США и Европе. Такая передача результатов из более быстрых сетей в специализированные сети помогает повысить производительность и модульность различных сетей.



**Рис. 14.25.** Обнаружение светофоров и дорожных знаков беспилотным автомобилем, использующим платформу NVIDIA Drive (изображение с сайта <https://oreil.ly/vcJY6>)

## Итоги

В этой главе мы рассмотрели задачи компьютерного зрения, взаимосвязи и различия между ними. Мы изучили задачу обнаружения объектов, принцип работы пайплайна и развитие пайплайна. Затем собрали данные, разметили их, обучили модель (с кодом и без кода) и развернули ее для использования на мобильных устройствах. Потом познакомились с реальными примерами обнаружения объектов разными компаниями и организациями. А под конец заглянули в мир сегментации объектов. Обнаружение объектов — очень мощный инструмент, огромный потенциал которого раскрывается каждый день. А какое инновационное применение обнаружения объектов придумаете вы? Поделитесь своими идеями в Twitter @PracticalDLBook (<https://www.twitter.com/PracticalDLBook>).

## ГЛАВА 15

---

# Как стать творцом: ИИ в краевых устройствах

Написана Сэмом Стерквалом

Вы знаете, как создать приложение ИИ, но хотите большего? Хочется перенести его в реальный мир, а не ограничиваться запуском ИИ на каком-то компьютере? Хотите создавать интерактивные устройства, облегчающие жизнь людям или просто приносящие удовольствие? Может, вы хотите создать интерактивную картину, которая будет улыбаться при взгляде на нее? Или чтобы камера на входной двери подавала громкий сигнал, когда кто-то попытается украсть вашу почту? Может, вы хотите создать роботизированный манипулятор, сортирующий вторсырье и мусор? Или устройство, останавливающее браконьеров в лесу? Или дрон, который автономно обследует большие территории и находит людей во время катастроф? Или инвалидное кресло, которое двигается самостоятельно? В любом случае понадобится создать интеллектуальное электронное устройство. Но как его сконструировать? Сколько оно будет стоить? Насколько мощным может быть? В этой главе постараемся ответить на эти вопросы.

Мы посмотрим, как реализовать ИИ на краевом (edge) устройстве — устройстве, которое можно использовать в проекте «творца». Творцы в нашем контексте — это люди с DIY-философией (DIY — Do It Yourself, сделай сам), воплощающие творческие замыслы во что-то новое. Часто они начинают с простого хобби и затем постепенно становятся изобретателями, робототехниками, новаторами или предпринимателями.

Цель этой главы — показать вам, как подбирать подходящие устройства для решения конкретных задач (это значит не пытаться запустить тяжелую GAN на слабеньком процессоре и не использовать GPU с квадриллионом ядер для запуска классификатора «не хот-дог») и настраивать их максимально быстро и легко. Мы изучим несколько наиболее известных устройств и посмотрим, как можно использовать их для инференса с помощью нашей модели. Наконец,

посмотрим, как производители по всему миру используют ИИ для создания роботизированных проектов.

А теперь сделаем первый шаг и посмотрим, какие сегодня существуют краевые устройства ИИ.

## Обзор краевых устройств ИИ

В этом разделе рассмотрим несколько хорошо известных краевых устройств с ИИ, перечисленных в табл. 15.1. Поговорим об их внутреннем устройстве и различиях, а потом протестируем.

**Таблица 15.1.** Список устройств

Устройство	Описание
Raspberry Pi 4	Самый известный на момент написания книги одноплатный компьютер
Intel Movidius NCS2	USB-ускоритель с 16-ядерным процессором машинного зрения (Visual Processing Unit, VPU)
Google Coral USB	USB-ускоритель, использующий специализированную интегральную схему (Application-Specific Integrated Circuit, ASIC)
NVIDIA Jetson Nano	Одноплатный компьютер, использующий комбинацию CPU и 128-ядерного CUDA GPU
PYNQ-Z2	Одноплатный компьютер, использующий комбинацию CPU и программируемой логической интегральной схемы (Field-Programmable Gate Array, FPGA) с 50K программируемых блоков (CLB)

Не будем рассуждать, что лучше, а что хуже. Наша цель — научиться правильно выбирать устройство для конкретного проекта. Обычно мы не видим «Феррари» на улицах, чаще нам встречаются более экономичные автомобили, которые выполняют свою работу ничуть не хуже, если не лучше. Так и использование мощного GPU NVIDIA 2080 Ti стоимостью более 1000 долларов будет излишним для управления дроном с питанием от батарей. Чтобы понять, какое из этих краевых устройств лучше всего соответствует потребностям, ответим на несколько вопросов:

1. Насколько устройство большое (например, в сравнении с монетой)?
2. Сколько стоит устройство? Оцените размер своего бюджета.
3. Насколько быстро должно работать устройство? Достаточно ли частоты 1 кадр в секунду? Или оно должно обрабатывать 100 кадров в секунду?

4. Какую мощность (в ваттах) будет потреблять устройство? Для проектов с батарейным питанием это особенно важный показатель.

Рассмотрим некоторые устройства, показанные на рис. 15.1.



**Рис. 15.1.** Краевые устройства, где можно реализовать ИИ. Сверху и по часовой стрелке: PYNQ-Z2, Arduino UNO R3, Intel Movidius NCS2, Raspberry Pi 4, Google Coral USB Accelerator, NVIDIA Jetson Nano и в середине монета в 1 евро для масштаба

## Raspberry Pi

Начнем с одного из самых популярных электронных проектов, Raspberry Pi (рис. 15.2). Это недорогое и легко расширяемое устройство, собравшее вокруг себя огромное сообщество любителей.

Размер	85,6 × 56,5 мм
Цена	От 35 долларов США
Процессор	ARM Cortex-A72 CPU
Потребляемая мощность	15 Вт



**Рис. 15.2.** Raspberry Pi 4

Что такое Raspberry Pi? Это «одноплатный компьютер», то есть компьютер, все компоненты которого, необходимые для выполнения вычислений, размещены на одной печатной плате. Четвертая версия одноплатного компьютера содержит SoC Broadcom (system-on-a-chip – однокристальную систему), содержащую (что особенно важно) четырехъядерный процессор ARMv8-A72, модуль VideoCore VI 3D, а также некоторые видеокодеры и видеодекодеры. Снабжается оперативной памятью объемом до 4 Гбайт.

Эта версия – большой шаг вперед по сравнению с Raspberry Pi 3 с точки зрения производительности, но она обладает несколько худшей энергоэффективностью. Это связано с тем, что в Raspberry Pi 3 используется более энергоэффективное ядро ARMv8-A53, тогда как ядро ARMv8-A72 (используемое в версии 4) является высокопроизводительной версией. Это легко заметить по требованиям к источнику питания. Raspberry Pi 3 потребляет ток 2,5 ампера, а Raspberry Pi 4 – 3 ампера. Также важно отметить, что Raspberry Pi 4 имеет порты USB 3, тогда как Raspberry Pi 3 – только порты USB 2. Это окажется важным в будущем.

А теперь поговорим о машинном обучении. Каким бы мощным ни было устройство Raspberry Pi 4, оно все еще имеет всего четыре последовательных ядра CPU (которые теперь поддерживают выполнение не по порядку [Out of Order, OoO]). В нем есть модуль VideoCore VI, но на момент написания этих строк TensorFlow не поддерживала эту архитектуру. Коити Накамура (Koichi Nakamura) из Idein Inc. создал библиотеку py-videoocore (<https://oreil.ly/AuEzr>) для Python для работы с четырехъядерными процессорами (Quad Processing Units, QPU; GPU-подобные ядра SoC в Raspberry Pi). Он и раньше использовал эту библиотеку для ускорения нейронных сетей, но ускорить TensorFlow с ее помощью пока невозможно. Есть библиотеки на C++ для доступа к этим ядрам. Но, как вы

могли догадаться, эти ядра не такие мощные, поэтому даже при использовании ускорения трудно достичь желаемых результатов. Не будем вдаваться в тонкости — изучение алгоритмов выходит за рамки этой книги. Как мы увидим далее, в этом может не быть никакой необходимости.

Raspberry Pi оказался чрезвычайно полезным устройством, пригодным для решения множества задач. Оно широко используется в образовательных целях, а также любителями. Вы могли бы удивиться, узнав, насколько широко используется Raspberry Pi в промышленности (есть даже устройство под названием netPi, которое является версией Raspberry Pi в прочном корпусе).

## Intel Movidius Neural Compute Stick

А теперь рассмотрим одну из причин, почему Raspberry Pi используется как отправная точка во множестве проектов. Это Intel Movidius Neural Compute Stick 2 (рис. 15.3), второе поколение USB-ускорителя, созданного Intel.



**Рис. 15.3.** Intel Neural Compute Stick 2

Размер	72,5 × 27 мм
Цена	87,99 доллара США
Процессор	Myriad X VPU
Потребляемая мощность	1 Вт

Это устройство выполняет довольно простую функцию: вы передаете ему нейронную сеть и некоторые данные, а оно выполняет все вычисления, не-

обходимые для инференса. Устройство подключается только через USB, поэтому его легко подключить к Raspberry Pi, запустить вычисления на USB-ускорителе и освободить процессор Raspberry Pi для решения других, не менее интересных задач.

В основе устройства лежит видеопроцессор Myriad VPU. В первом поколении использовался видеопроцессор Myriad 2 VPU, в текущем — Myriad X VPU. Видеопроцессор имеет 16 ядер SHAVE (Streaming Hybrid Architecture Vector Engine — потоковая гибридная архитектура для векторных вычислений), которые похожи на ядра GPU, но менее приспособлены для операций с графикой. Устройство имеет также собственную оперативную память, что особенно ценно при выполнении расчетов с использованием нейронных сетей, потому что эти сети создают большие объемы данных во время вычислений, которые могут храниться рядом с ядром, что значительно сокращает время доступа к ним.

## Google Coral USB Accelerator

Устройство Google Coral (рис. 15.4) с тензорным процессором Google Edge TPU — это второй USB-ускоритель, который мы рассмотрим. Для начала проясню, почему мы рассматриваем два разных USB-ускорителя. Устройство от Intel, как уже отмечалось, имеет несколько ядер SHAVE, которые поддерживают множество инструкций, как ядра GPU, и используются для ускорения вычислений. Устройство Google Edge TPU, напротив, — это специализированная интегральная схема (Application-Specific Integrated Circuit, ASIC), которая также выполняет некоторую обработку, но служит одной цели (отсюда и ключевое слово «*Specific*»). ASIC имеет несколько свойств, присущих аппаратному обеспечению, некоторые из них действительно хороши, а другие — не очень:

### *Скорость*

Поскольку все электронные схемы внутри TPU служат одной цели, отсутствуют издержки на операции декодирования. Вы вводите входные данные и веса, а процессор почти мгновенно возвращает результат.

### *Энергоэффективность*

Все устройства ASIC имеют одну цель и поэтому не требуют дополнительной энергии. Соотношение производительность/Ватт для ASIC обычно является самым высоким.

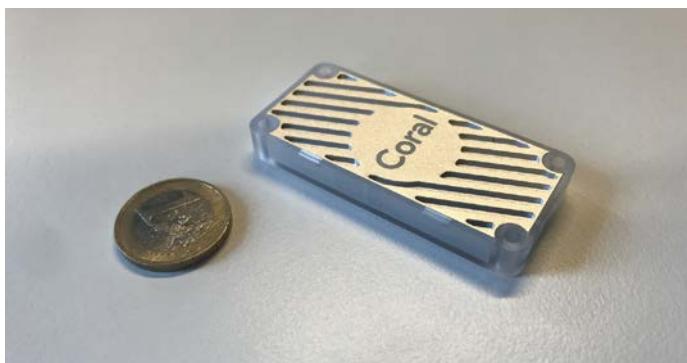
### *Гибкость*

Устройство ASIC может делать только то, для чего оно было создано, в данном случае оно ускоряет работу нейронных сетей TensorFlow Lite. Но чтобы использовать его, придется ограничить себя компилятором Google Edge TPU и 8-битными моделями *.tflite*.

### Простота

Google – это компания-разработчик ПО, и ее сотрудники знают, как упростить работу. Именно это они и воплотили в устройстве. Начать работу с Google Coral невероятно просто.

Как же работает тензорный процессор Google Edge TPU? Информация о самом Edge TPU не разглашается, но есть информация о Cloud TPU, поэтому можно предположить, что в целом Edge TPU работает так же, как Cloud TPU. Он имеет специальное оборудование для операций умножения-сложения, активации и объединения. Все транзисторы на микросхеме соединены так, что она может принимать веса и входные данные и вычислять результат, распараллеливая операции. Самая большая часть микросхемы (не считая встроенной памяти) выполняет именно то, что обозначает ее название Matrix Multiply Unit – модуль умножения матриц. В нем используется довольно хитрый, хотя и не новый принцип под названием *систолическое выполнение* (*systolic execution*, <https://oreil.ly/R5VpP>). Этот принцип снижает требования к пропускной способности памяти за счет сохранения промежуточных результатов в элементах обработки, а не в памяти.



**Рис. 15.4.** Google Coral USB Accelerator

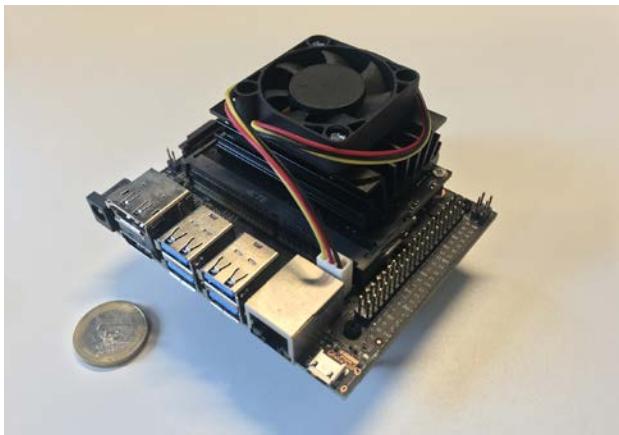
Размер	65 × 30 мм
Цена	74,99 доллара США
Процессор	Google Edge TPU
Потребляемая мощность	2,5 Вт



Какое главное отличие между этими двумя USB-ускорителями? Google Coral более мощный, но (немного) менее гибкий, чем Intel Movidius NCS2. При этом Coral намного проще в настройке и использовании, особенно если есть обученная и сконвертированная модель.

## NVIDIA Jetson Nano

Перейдем к оборудованию другого типа: NVIDIA Jetson Nano (рис. 15.5), одноплатному компьютеру с искусственным интеллектом. Это устройство похоже на Raspberry Pi, но имеет 128-ядерный графический процессор Maxwell GPU с поддержкой CUDA. Наличие дополнительного графического процессора делает это устройство похожим на комбинацию Raspberry Pi + Intel NCS2, но в отличие от NCS2 с 16 ядрами, устройство Jetson имеет 128 ядер. Что еще в нем имеется? Четырехъядерный процессор A57 ARMv8, который является предшественником ARMv8 A72 в Raspberry Pi 4 (кроме того, само устройство на несколько месяцев старше Raspberry Pi 4), 4 Гбайт оперативной памяти Low-Power Double Data Rate 4 (LPDDR4), которая довольно удобно распределяется между CPU и GPU, что позволяет обрабатывать данные на GPU, не копируя их.



**Рис. 15.5.** NVIDIA Jetson Nano

Размер	100 × 79 мм
Цена	99 долларов США
Процессор	ARM A57 + 128-ядерный Maxwell GPU
Потребляемая мощность	10 Вт (при высокой нагрузке потребляемая мощность может увеличиться)

Особенность этого одноплатного компьютера в том, что, как уже говорилось выше, ядра графического процессора поддерживают программно-аппаратную архитектуру параллельных вычислений CUDA (Compute Unified Device Architecture), поэтому могут использоваться библиотекой TensorFlow-GPU или любым другим фреймворком, что придает устройству дополнительную привлекательность по сравнению с Raspberry Pi, особенно если предполагается обучать

на нем сети. Кроме того, это очень недорого для графического процессора. Сейчас за Jetson Nano придется выложить 99 долларов, а взамен вы получите довольно высокопроизводительный процессор ARM и 128-ядерный графический процессор. Для сравнения: Raspberry Pi 4 с 4 Гбайт памяти стоит около 55 долларов, ускоритель Coral USB — около 75 долларов, а Movidius NCS2 — около 90 долларов. Последние два устройства не являются самостоятельными и требуют дополнительно использовать как минимум Raspberry Pi, а у Pi нет графического процессора, способного ускорить приложения глубокого обучения.

Еще одно замечание о Jetson Nano: это устройство способно ускорять операции с 32-битными значениями с плавающей запятой, которые TensorFlow использует по умолчанию, но эффективность возрастает еще больше с 16-битными значениями с плавающей запятой и еще больше, если использовать фреймворк TensorRT. К счастью, компания разработала отличную небольшую библиотеку с открытым исходным кодом — TensorFlow-TensorRT (TF-TRT), которая автоматически ускоряет доступные операции с TensorRT, позволяя TensorFlow делать все остальное. Библиотека обеспечивает значительное ускорение, примерно в четыре раза по сравнению с TensorFlow-GPU. Учитывая все это, Jetson Nano выглядит самым гибко настраиваемым устройством.

### От создателя

Работа с платформой NVIDIA Jetson доставляет большое удовольствие, потому что дает программные и аппаратные возможности для управления роботами с ИИ, дронами, приложениями интеллектуальной видеоаналитики (Intelligent Video Analytics, IVA) и другими автономными устройствами, способными самостоятельно принимать решения. Jetson Nano — это мощь современного ИИ в форме небольшой и простой в использовании платформы.

Несколько лет назад у нас была группа старшеклассников, которые в летние каникулы разрабатывали проекты с глубоким обучением на краевых устройствах. Они придумывали творческие проекты, впечатляющие своим техническим воплощением и остроумием. Мы были рады передать в их распоряжение Jetson Nano, потому что чувствовали, что именно в таких проектах Jetson Nano может использоваться для широкой аудитории.

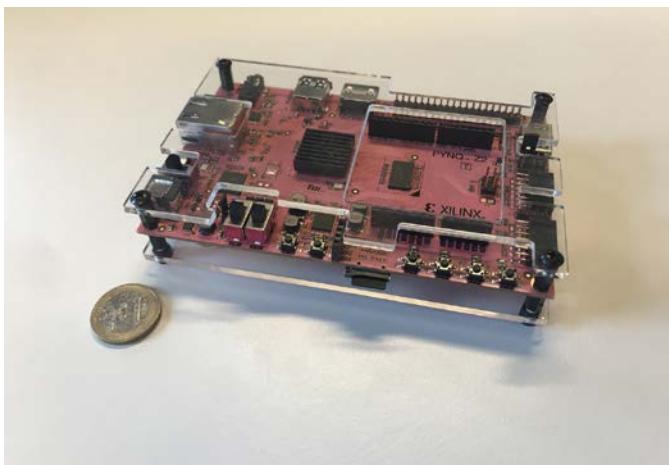
Приятно быть частью талантливой команды в NVIDIA, которая предлагает инструменты, вдохновляющие сообщество на быстрое освоение ИИ. Не устаем восхищаться некоторыми интересными проектами, которые творцы, разработчики и студенты создают на основе Jetson Nano — от самодельных роботов до автоматизации домашнего быта, и это только начало. Наша цель — сделать ИИ доступным для всех.

Джон Уэлш (John Welsh) и Читоку Ято (Chitoku Yato),  
разработчики NVIDIA Jetson Nano

## FPGA + PYNQ

А теперь пристегните ремни: нам предстоит глубокое погружение в темный мир электроники. Платформа PYNQ (рис. 15.6), основанная на семействе микросхем Xilinx Zynq, представляет совершенно иную сторону мира электроники, чем остальные устройства из этого раздела. PYNQ снабжена двухъядерным процессором ARM-A9 с колоссальной частотой 667 МГц.

Вы могли подумать: «Да ладно? Как это можно сравнивать с четырехъядерным процессором A72 в Raspberry Pi с частотой 1,5 ГГц?!» И будете правы, CPU в этой штуке почти бесполезен. Но... На борту устройства есть кое-что еще — программируемая логическая интегральная схема (Field-Programmable Gate Array, FPGA).



**Рис. 15.6.** Xilinx PYNQ-Z2

Размер	140 × 87 мм
Цена	119 долларов США
Процессор	Xilinx Zynq-7020
Потребляемая мощность	13,8 Вт

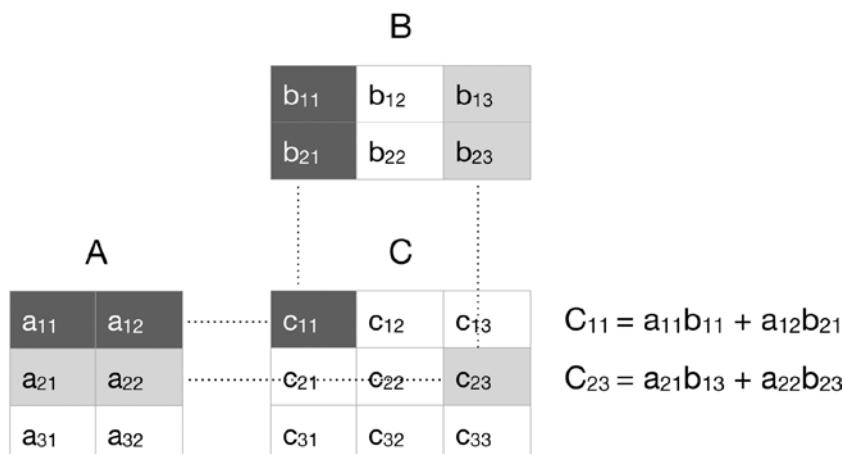
## FPGA

Чтобы понять, что такое FPGA, сначала вспомним некоторые базовые понятия. Начнем с CPU: устройства с известным набором операций называют архитектурой набора команд (Instruction Set Architecture, ISA). Этот набор определяет

круг возможностей CPU и обычно включает такие операции, как **Load Word** (Загрузка слова) (размер слова определяется разрядностью шины данных, как правило, это 32 или 64 разряда), которая загружает некоторое значение во внутренний регистр CPU, и **Add** (Сложение), которая может сложить содержимое двух внутренних регистров и сохранить результат, например, в третьем регистре.

CPU может выполнять эти операции, так как содержит большое количество транзисторов (в принципе их можно рассматривать как электрические переключатели), которые жестко связаны так, что CPU автоматически транслирует операции и выполняет соответствующие им действия. Программа — это длинный список таких операций, следующих друг за другом в определенном порядке. Одноядерный процессор будет по очереди извлекать эти операции из списка и выполнять их с довольно внушительной скоростью.

Теперь рассмотрим идею параллельного выполнения. Как вы знаете, нейронные сети состоят в основном из сверток или линейных узлов, которые можно преобразовать в матричные операции умножения. Исследовав математику, лежащую в основе этих операций, можно заметить, что каждый выход слоя можно вычислить независимо от других выходов, как показано на рис. 15.7.



**Рис. 15.7.** Матричные операции

Проще говоря, все операции могут выполняться параллельно. Одноядерный процессор не дает такой возможности, потому что способен выполнять операции только последовательно. Вот тут и пригодятся многоядерные процессоры. Четырехъядерный процессор может выполнять четыре операции одновременно, то есть теоретически может ускорить вычисления в четыре раза. Ядра SHAVE в Intel Movidius NCS2 и ядра CUDA в Jetson Nano не такие сложные, как ядро полноценного процессора, но их возможностей достаточно для выполнения

операций умножения и сложения, а кроме того, у NCS2 16 таких ядер, а у Jetson Nano — 128. Более мощный графический процессор RTX 2080 Ti вообще имеет 4352 ядра CUDA. Теперь легко понять, почему GPU лучше справляются с задачами глубокого обучения, чем обычные процессоры.

Вернемся к FPGA. В отличие от CPU и GPU, которые представляют собой огромные наборы транзисторов, жестко запрограммированные для выполнения определенного набора инструкций, FPGA — это точно такой же огромный набор транзисторов, но не связанных между собой. Можно реализовать свою уникальную схему соединений и изменить эту схему в будущем. Можно подключить FPGA к процессору. Вы можете также найти схемы и проекты, в которых другие люди подключили FPGA к графическому процессору. Но самое интересное, что схема соединений в FPGA может повторять архитектуру вашей нейронной сети, что фактически превращает FPGA в физическую реализацию такой сети. Слово «соединения» здесь использовано намеренно. Часто можно встретить людей, которые называют эти схемы соединений словом «программа», но это не совсем правильно. В действительности, программируя FPGA, вы перенастраиваете реальное оборудование, в отличие от CPU или GPU, для которых загружаете программу, выполняемую оборудованием.

## Платформа PYNQ

Обычно файл схемы соединений для FPGA называют *битовым потоком* или *битовой картой*, потому что содержимое этого файла, которое записывается в микросхему, по сути представляет карту соединений, которые должны быть установлены. Как вы понимаете, создавать такие битовые потоки намного сложнее, чем написать скрипт на Python. Вот где на помощь приходит платформа PYNQ. Ее девиз — «Продуктивность Python для Zynq». Компания устанавливает образ PYNQ, который автоматически запускает Jupyter Notebook на ARM внутри чипа, и имеет в своем составе несколько основных битовых потоков. В будущем, вероятно, число битовых потоков увеличится. В мире PYNQ эти битовые потоки называются *оверлеями* или *наложениями*.

Если вы погуглите, то быстро наткнетесь на пример BNN-PYNQ, реализующий простую шестислойную сверточную сеть в стиле VGG, способную обрабатывать изображения с частотой 3000 кадров в секунду на PYNQ-Z1 и Z2 и около 10 000 кадров в секунду на устройстве ZCU104, которое снабжено версией Ultrascale+ чипа Zynq. Эти цифры могут вызывать недоверие, поэтому уточню, что обработке подвергаются изображения  $32 \times 32$  пикселей, а не обычные изображения  $224 \times 224$ , а кроме того, сама сеть «бинаризована», то есть роль весов и активаций в ней играют однобитные данные, а не 32-битные, как в TensorFlow. Для более полноценного сравнения производительности я попытался воссоздать аналогичную сеть в Keras.

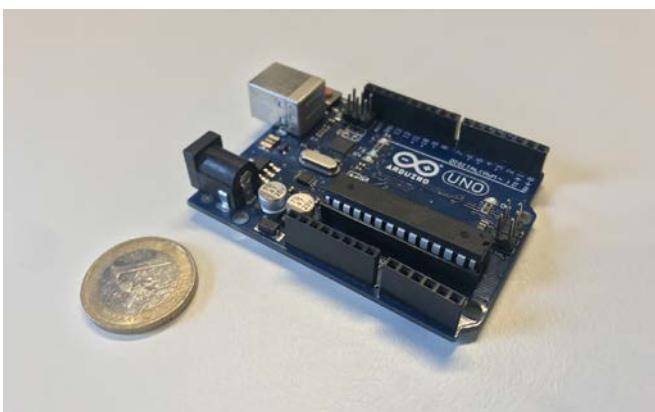
Я построил модель FP32 и обучил ее на датасете CIFAR-10, который входит в Keras и доступен как `keras.datasets.cifar10`. Модель FP32 достигла уровня ошибок примерно 17 % и, что удивительно, оказалась всего на 2 % лучше бинарной модели. Скорость вычисления прогноза на восьмиядерном процессоре Intel i9 составила около 132 кадров в секунду. Насколько мне известно, пока нет простого и эффективного способа, позволяющего использовать бинарную сеть на CPU, — вам придется покопаться в некоторых специализированных пакетах Python или в коде на C, чтобы получить максимальную отдачу от оборудования. Теоретически можно добиться ускорения в три-пять раз. Но эта скорость все равно будет намного ниже скорости не самых производительных FPGA, а кроме того, CPU обычно потребляет намного больше энергии. Конечно, у всего есть обратная сторона, и для FPGA — это сложность конструкции. Есть фреймворки с открытым исходным кодом, например FINN (<https://oreil.ly/CijXF>), основанный на Xilinx, которые смягчают этот недостаток. Также некоторые производители предлагают свои фреймворки, но ни один из них не может сравниться с простотой использования программных пакетов и библиотек, таких как TensorFlow. Проектирование нейронной сети на FPGA также требует более глубоких знаний в области электроники, поэтому дальнейшее обсуждение выходит за рамки этой книги.

## Arduino

Arduino (рис. 15.8) может значительно упростить вашу жизнь, предлагая возможность взаимодействий с реальным миром через датчики и исполнительные механизмы.

Микроконтроллер (MicroController Unit, MCU) Arduino — это плата для разработки микроконтроллеров, построенная на базе 8-битного микроконтроллера Advanced Virtual RISC (AVR) ATMega328p, работающего на частоте 16 МГц (поэтому не стоит пытаться запустить на нем хоть сколько-нибудь серьезную нейронную сеть). Микроконтроллер — это то, с чем сталкивался каждый из нас, их можно найти почти в любых электронных устройствах, от монитора и телевизора до клавиатуры персонального компьютера, и, черт возьми, даже в велосипедных фарах сейчас могут быть микроконтроллеры, управляющие режимом мигания.

Эти платы бывают разных форм, размеров и характеристик, и хотя MCU в Arduino UNO имеет довольно низкую производительность, доступны более мощные микроконтроллеры. Самой известной, пожалуй, является серия ARM Cortex-M. Недавно разработчики TensorFlow и uTensor (легковесный фреймворк инференса, предназначенный для использования на микроконтроллерах) объединили свои усилия, чтобы модели TensorFlow можно было запускать на этих микроконтроллерах.



**Рис. 15.8.** Arduino Uno R3

Размер	68,6 × 53,3 мм
Цена	10 долларов США
Процессор	AVR ATMega328p
Потребляемая мощность	0,3 Вт

### От создателя

Когда в 2014 году я пришел в Google, то был поражен, обнаружив, что команда «Ok Google» использовала модели размером всего 13 Кбайт, работающие на крошечных цифровых сигнальных процессорах (Digital Signal Processor, DSP) и распознающие эту фразу. Это были очень опытные и прагматичные инженеры, и я знал, что они делают это не ради того, чтобы покрасоваться. Они много экспериментировали и нашли лучшее решение, какое только возможно. К тому времени для меня обыденностью были модели размером в мегабайты, поэтому открытие, что даже такие крошечные модели могут быть полезны, стало для меня откровением. Я не мог не задаться вопросом, какие еще задачи решают подобные сети. С тех пор я стремлюсь донести технологию, которую использовала эта команда, до более широкой аудитории, чтобы узнать, какие еще практические применения этих сетей возможны. Вот почему в начале 2019 года мы выпустили нашу первую версию TensorFlow Lite для микроконтроллеров, и было удивительно наблюдать, как много творческих инженеров создают инновационные решения и даже совершенно новые продукты с использованием этих технологий.

Пит Уорден (Pete Warden), техлид Mobile and Embedded TensorFlow, автор TinyML (<http://shop.oreilly.com/product/0636920254508.do>)

Arduino UNO малопригоден для объемных и сложных вычислений, зато у него есть много других преимуществ, которые делают его хорошим орудием для любого творца. Это недорогое, простое и надежное устройство, у которого есть огромное сообщество. Для большинства датчиков и исполнительных механизмов написаны библиотеки и созданы платы расширения для Arduino, что делает платформу действительно очень простой в применении. Архитектура AVR – это довольно устаревшая 8-битная модифицированная гарвардская архитектура, которую легко освоить, особенно с фреймворком Arduino. В огромном сообществе Arduino было написано великое множество мануалов, поэтому просто просмотрите любое руководство по Arduino для своего датчика, и вы обязательно найдете то, что нужно для реализации проекта.



Используя библиотеку на Python, можно с легкостью организовать взаимодействие с Arduino через универсальный асинхронный приемопередатчик (Universal Asynchronous Receiver/Transmitter, UART) по USB-кабелю для отправки команд или данных туда и обратно.

## Сравнение краевых устройств для ИИ

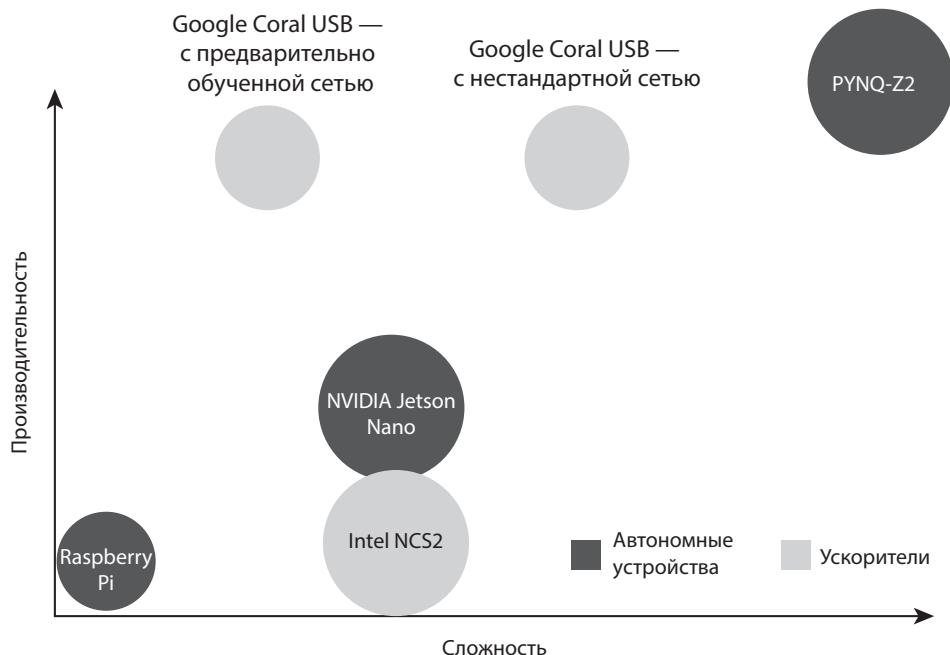
В табл. 15.2 перечислены платформы, о которых рассказывалось выше, а на рис. 15.9 показано место каждого устройства на схеме «производительность–сложность».

**Таблица 15.2.** Достоинства и недостатки различных платформ

Устройство	Достоинства	Недостатки
Raspberry Pi 4	<ul style="list-style-type: none"> <li>Простота</li> <li>Большое сообщество</li> <li>Доступность</li> <li>Дешевизна</li> </ul>	<ul style="list-style-type: none"> <li>Недостаточная вычислительная мощность</li> </ul>
Intel Movidius NCS2	<ul style="list-style-type: none"> <li>Простота оптимизации</li> <li>Поддержка большинства платформ</li> </ul>	<ul style="list-style-type: none"> <li>Дороговизна</li> <li>Недостаточная вычислительная мощность</li> </ul>
Google Coral USB	<ul style="list-style-type: none"> <li>Простота освоения</li> <li>Высокая скорость</li> <li>Хорошее соотношение цена/скорость</li> </ul>	<ul style="list-style-type: none"> <li>Поддерживает только модели <i>.tflite</i></li> <li>Требуется 8-битное квантование</li> </ul>

Таблица 15.2 (окончание)

Устройство	Достоинства	Недостатки
NVIDIA Jetson Nano	<ul style="list-style-type: none"> <li>«Все в одном»</li> <li>Возможность обучения моделей</li> <li>Дешевизна</li> <li>Позволяет добиться достойной производительности</li> <li>Наличие CUDA GPU</li> </ul>	<ul style="list-style-type: none"> <li>Производительности может не хватать для некоторых задач</li> <li>Сложность выполнения более продвинутых оптимизаций</li> </ul>
PYNQ-Z2	<ul style="list-style-type: none"> <li>Потенциально огромная энергоэффективность</li> <li>Возможность аппаратной реализации сети</li> <li>Потенциально огромная производительность</li> <li>Еще большая универсальность, чем у Jetson Nano</li> </ul>	<ul style="list-style-type: none"> <li>Огромная сложность</li> <li>Долгий процесс проектирования</li> </ul>



**Рис. 15.9.** Производительность и сложность некоторых устройств (чем больше круг, тем выше цена)

В этом разделе протестируем некоторые платформы, выполнив 250 классификаций одного и того же изображения с использованием модели MobileNetV2, обученной на датасете ImageNet. Каждый раз мы будем использовать одно и то же изображение — это позволит обойти узкие места, которые могут возникнуть в системах с недостаточным объемом оперативной памяти для хранения всех весов и множества разных изображений.

Для начала выберем изображение. Кто из нас не любит кошек? Поэтому возьмем изображение красивого кота (рис. 15.10) размером  $224 \times 224$  пикселя, чтобы его не пришлось масштабировать.



**Рис. 15.10.** Изображение кота, которое мы возьмем для экспериментов

Для сравнения полезно иметь некоторую точку отсчета, поэтому сначала запустим модель на персональном компьютере. Как это сделать, объясняется в главе 2, поэтому напишем скрипт, который 250 раз вычислит прогноз и измерит затраченное время. Полный скрипт ищите в репозитории книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>) в папке `code/chapter-15`:

```
$ python3 benchmark.py
Using TensorFlow backend.
tf version : 1.14.0
keras version : 2.2.4
input tensor shape : (1, 224, 224, 3)
warmup prediction
[('n02124075', 'Egyptian_cat', 0.6629321)]
starting now...
Time[s] : 16.704
FPS      : 14.968
Model saved.
```

Когда у нас есть с чем сравнивать, посмотрим, как запустить эту модель на краевых устройствах, сможем ли мы поднять производительность до приемлемого уровня и как.

## Raspberry Pi

Начнем с самого известного и простого устройства: Raspberry Pi (или просто RPi для краткости).

Начнем с установки Raspbian версии Linux, созданной специально для Raspberry Pi. Для простоты предположим, что у нас уже есть Raspberry Pi со всем установленным, обновленным и готовым к работе системным ПО. Теперь можно перейти к установке библиотеки TensorFlow, для чего достаточно выполнить простую команду pip:

```
$ pip3 install tensorflow
```



Из-за недавнего перехода на Raspbian Buster мы столкнулись с некоторыми проблемами, которые были решены с помощью следующего пакета piwheel:

```
$ pip3 install  
https://www.piwheels.org/simple/tensorflow/tensorflow  
-1.13.1-cp37-none-linux_armv7l.whl
```

Установим библиотеку Keras с помощью pip. Не забудьте установить пакет libhdf5-dev, который необходим для загрузки весов в нейронную сеть:

```
$ pip3 install keras  
$ apt install libhdf5-dev
```

Поскольку установка библиотеки OpenCV, которая в коде используется под именем cv2, на RPi может вызывать проблемы (особенно из-за недавнего перехода на новую версию ОС), для загрузки изображения вместо OpenCV можно использовать PIL. Для этого вместо cv2 импортируем PIL и изменим код загрузки изображения:

```
#input_image = cv2.imread(input_image_path)  
#input_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB)  
input_image = PIL.Image.open(input_image_path)  
input_image = np.asarray(input_image)
```



В отличие от OpenCV, библиотека PIL загружает изображения в формате RGB, поэтому преобразование из BGR в RGB больше не требуется.

Бот и все! Тот же самый скрипт теперь должен работать на Raspberry Pi:

```
$ python3 benchmark.py  
Using TensorFlow backend.  
tf version : 1.14.0
```

```
keras version : 2.2.4
input tensor shape : (1, 224, 224, 3)
warmup prediction
[('n02124075', 'Egyptian_cat', 0.6629321)]
starting now...
Time[s] : 91.041
FPS      : 2.747
```

Как видите, на этой платформе скрипт работает намного медленнее — скорость обработки не дотягивает даже до трех кадров в секунду. Можно ли как-то поднять скорость?

Первое, что можно сделать, — посмотреть в сторону TensorFlow Lite. Библиотека TensorFlow, как обсуждалось в главе 13, имеет встроенный инструмент для преобразования любых моделей в формат TensorFlow Lite (*.tflite*).

Мы написали небольшой скрипт, очень похожий на исходный *benchmark.py*, который использует модель в формате TensorFlow Lite и вычисляет те же 250 прогнозов. Скрипт доступен в репозитории книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>).

Давайте продолжим и запустим этот скрипт, чтобы оценить успех наших усилий.

```
$ python3 benchmark_tflite.py
Using TensorFlow backend.
input tensor shape : (1, 224, 224, 3)
conversion to tflite is done
INFO: Initialized TensorFlow Lite runtime.
[['n02124075', 'Egyptian_cat', 0.68769807]]
starting now (tflite)...
Time[ms] : 32.152
FPS : 7.775
```



На сайте Google Coral доступны 8-битные квантованные модели процессоров, которые можно запускать с помощью TensorFlow Lite.

Скорость увеличилась в три раза (табл. 15.3). Неплохо для первой попытки, но мы пока достигли только 7,8 FPS, что вроде как неплохо и вполне подходит для многих приложений. Но что, если изначально планировалось обрабатывать видеопоток в режиме реального времени? Тогда этой скорости недостаточно. Квантование модели может увеличить ее производительность, но это увеличение будет незначительным, учитывая, что CPU в Raspberry Pi не оптимизирован для 8-битных целочисленных операций.

Можно ли добиться еще большего ускорения, чтобы Raspberry Pi можно было использовать, например, для управления беспилотными гоночными автомобилями или автоматическими дронами? Да, можно, с помощью USB-ускорителей.

**Таблица 15.3.** Бенчмарки разных конфигураций Raspberry Pi

Конфигурация	Кадров/с
Raspberry Pi 4 (tflite, 8-битная)	8,6
Raspberry Pi 4 (tflite)	7,8
Raspberry Pi 3B+ (tflite, 8-битная)	4,2
Raspberry Pi 4	2,7
Raspberry Pi 3B+	2,1

## Ускорение с помощью Google Coral USB Accelerator

Эти ускорители будут весьма кстати. Чтобы использовать их, нам не придется ничего менять в аппаратной конфигурации, нужно лишь кое-что добавить. Как заставить Google Coral USB Accelerator работать в комплексе с Raspberry Pi и ускорит ли это процесс?

Прежде всего, намереваясь использовать USB-ускоритель, найдите руководство от производителя «Getting Started» (Приступая к работе). Откройте страницу <https://coral.withgoogle.com/>, перейдите в раздел *Docs & Tools > Documentation > USB Accelerator > Get Started* (Документация и инструменты > Документация > USB-ускоритель > Начать работу). Ознакомившись с описанием, вы сможете приступить к работе за считанные минуты.

На момент написания этой книги Coral USB Accelerator не был полностью совместим с Raspberry Pi 4, но, немного изменив скрипт *install.sh*, эту проблему довольно легко решить. Достаточно добавить в скрипт установки фрагмент, распознающий устройство Pi 4 и показанный на рис. 15.11.

Теперь скрипт *install.sh* будет работать правильно. Переименуем файл *.so* библиотеки, чтобы ее можно было использовать в Python 3.7. Для этого используем команду копирования:

```
$ sudo cp /usr/local/lib/python3.7/dist-packages/
edgetpu/swig/_edgetpu_cpp_wrapper.cpython-
35m-arm-linux-gnueabihf.so
/usr/local/lib/python3.7/dist-packages/
edgetpu/swig/_edgetpu_cpp_wrapper.cpython-
37m-arm-linux-gnueabihf.so
```

После этого все должно работать как надо, в том числе и примеры от Google.

Теперь, подключив и настроив Coral, снова попробуем сделать бенчмаркинг модели MobileNetV2. Но на этот раз ее необходимо квантовать, потому что Google

```

LIBEDGETPU_SUFFIX=arm32
HOST_GNU_TYPE=arm-linux-gnueabihf
elif [[ "${MODEL}" == "Raspberry Pi 3 Model B Plus Rev"* ]]; then
    info "Recognized as Raspberry Pi 3 B+."
    LIBEDGETPU_SUFFIX=arm32
    HOST_GNU_TYPE=arm-linux-gnueabihf
elif [[ "${MODEL}" == "Raspberry Pi 4 Model B Rev"* ]]; then
    info "Recognized as Raspberry Pi 4."
    LIBEDGETPU_SUFFIX=arm32
    HOST_GNU_TYPE=arm-linux-gnueabihf
fi
elif [[ "${CPU_ARCH}" == "aarch64" ]]; then
    info "Recognized as generic ARM64 platform."
    LIBEDGETPU_SUFFIX=arm64
    HOST_GNU_TYPE=aarch64-linux-gnu

```

**Рис. 15.11.** Изменения, которые необходимо внести в сценарий установки Google Coral

Edge TPU поддерживает только 8-битные целочисленные операции. Google уже подготовила квантованную и преобразованную модель MobileNetV2, готовую к использованию с Edge TPU. Получить ее можно на веб-сайте Coral в разделе *Examples > Code examples > Models page > MobileNet V2 (ImageNet) > Edge TPU Model* (Примеры > Примеры кода > Страница с моделями > MobileNet V2 (ImageNet) > Модель для Edge TPU).



Если вы решите создать, обучить, квантовать и конвертировать свою модель для Edge TPU, то придется приложить немного больше усилий и проделать работу, которую нельзя выполнить только с помощью Keras. Как это делается, можно узнать на сайте Google Coral (<https://oreil.ly/ydoSy>).

Итак, у нас есть действующий USB-ускоритель и модель для него. Теперь создадим новый скрипт для проверки производительности этой конфигурации. Этот скрипт тоже очень похож на предыдущий и во многом основан на примерах, которые Google предоставила вместе с Coral. Как всегда, файл скрипта доступен для загрузки в репозитории книги (см. <http://PracticalDeepLearning.ai>):

```
$ python3 benchmark_edgetpu.py
INFO: Initialized TensorFlow Lite runtime.
warmup prediction
Egyptian cat
0.59375
starting now (Edge TPU)...
Time[s] : 1.042
FPS      : 240.380
```

Здесь все верно. Скрипт действительно выполнил классификацию 250 раз! На Raspberry Pi 4 с USB3 для этого потребовалось всего 1,04 секунды, что соответствует 240,38 кадра в секунду! Это настоящее ускорение. Как вы понимаете, обработка видеопотока в реальном времени не будет проблемой.

Сейчас доступно множество предварительно скомпилированных моделей для самых разных целей, поэтому, если хотите, можете проверить их. Может быть, вам удастся найти подходящую модель и не придется преодолевать все этапы по созданию, обучению, квантованию и конвертации собственной модели.

В Raspberry Pi 3 есть только порт USB2. Как показано в табл. 15.4, это узкое место при использовании Coral.

**Таблица 15.4.** Бенчмарки Google Coral в разных конфигурациях

Конфигурация	Кадров/с
i7-7700K + Coral (tflite, 8-битная)	352,1
Raspberry Pi 4 + Coral (tflite, 8-битная)	240,4
Jetson Nano + Coral (tflite, 8-битная)	223,2
RPi3 + Coral (tflite, 8-битная)	75,5

## NVIDIA Jetson Nano

Coral Edge TPU — прекрасный USB-ускоритель, но что, если для вашего проекта нужна узкоспециализированная модель, обученная на каком-то сумасшедшем датасете, собранном вами? Как уже говорилось выше, создание новой модели для Coral Edge TPU требует больше усилий, чем просто создание модели Keras. Есть ли простой способ получить нестандартную модель, работающую на краевом устройстве с приличной производительностью? NVIDIA спешит на помощь! Компания создала замену Raspberry Pi — ускоритель Jetson Nano с поддержкой CUDA и довольно эффективным графическим процессором, который позволяет ускорять не только TensorFlow, но и все, что потребуется.

Начнем с установки необходимых пакетов. Желательно установить NVIDIA JetPack — версию Debian-подобной ОС для платформы Jetson. Инструкцию по установке TensorFlow и других необходимых библиотек ищите на сайте NVIDIA (<https://oreil.ly/d11bQ>).



Обратите внимание на известную проблему с pip3: `cannot import name 'main'` (невозможно импортировать имя 'main').

Бот ее решение:

```
$ sudo apt install nano
$ sudo nano /usr/bin/pip3
```

```
Replace : main => __main__
Replace : main() => __main___.__main__()
```



Обновление всех библиотек с помощью `pip` может занять много времени, поэтому для проверки — не зависло ли ваше устройство Jetson — установите `htop`:

```
$ sudo apt install htop
$ htop
```

Эта утилита позволяет следить за использованием CPU. Если утилита работает, значит, устройство Jetson не зависло.



Устройство Jetson Nano может сильно нагреваться во время компиляции, поэтому стоит организовать дополнительное охлаждение с помощью вентилятора. На монтажной плате есть разъем, но имейте в виду, что он подает на вентилятор напряжение 5 В, тогда как большинство вентиляторов рассчитаны на 12 В.

При питании 5 В некоторые вентиляторы, рассчитанные на 12 В, запускаются без проблем, а некоторые требуется немного «подтолкнуть», чтобы они запустились. Выбирайте вентилятор размером 40 × 40 мм. Также есть версии, рассчитанные на 5 В.

К сожалению, пошаговое руководство от NVIDIA сложнее, чем руководство от Google. Скорее всего, вы столкнетесь с трудностями и в конце концов перевернете стол<sup>1</sup>.

Но держитесь! Рано или поздно вы все преодолеете.



Чтобы установить Keras на Jetson, предварительно нужно установить пакет `scipy`, который в свою очередь требует установки `libatlas-base-dev` и `gfortran`, поэтому начните с установки последнего и двигайтесь от конца списка к началу:

```
$ sudo apt install libatlas-base-dev gfortran
$ sudo pip3 install scipy
$ sudo pip3 install keras
```

Когда все будет готово, можно запустить файл `benchmark.py` напрямую. Благодаря GPU он будет работать намного быстрее, чем на Raspberry Pi:

```
$ python3 benchmark.py
Using TensorFlow backend.
tf version : 1.14.0
keras version : 2.2.4
input tensor shape : (1, 224, 224, 3)
warmup prediction
```

---

<sup>1</sup> Отсылка к мему «переворачивание стола». — Примеч. ред.

```
[('n02124075', 'Egyptian_cat', 0.6629321)]  
starting now...  
Time[s] : 20.520  
FPS      : 12.177
```

Полученные результаты показывают мощь Jetson Nano: это устройство работает почти так же, как любой другой персональный компьютер с графическим процессором. Однако 12 FPS — это не так много, поэтому давайте посмотрим, можно ли улучшить результат.



Подключите к Jetson Nano USB-ускоритель Google Coral, что позволит одновременно запустить две модели — одну на Coral, а другую на GPU.

Графический процессор в Jetson сконструирован специально для операций с 16-битными числами с плавающей запятой, что, по сути, является лучшим компромиссом между точностью и производительностью. Как упоминалось выше, NVIDIA предлагает пакет TF-TRT, который упрощает оптимизацию и преобразование. Но он добавляет немного сложности по сравнению с примером Coral (но не забывайте, что Google предоставляет предварительно скомпилированный файл модели для Coral). Нужно зафиксировать модель Keras, а затем создать график вычисления прогноза с помощью TF-TRT. Поиски всего в GitHub могут занять некоторое время, поэтому мы разместили все это в репозитории книги (см. <https://github.com/PracticalDL/Practical-Deep-Learning-Book>).

Оптимизировать свои модели для инференса на Jetson Nano можно с помощью скрипта `tftrt_helper.py`. Он фиксирует модель Keras, удаляет обучающие ноды, а затем использует пакет NVIDIA TF-TRT для Python (входящий в состав TensorFlow Contrib) для оптимизации.

```
$ python3 benchmark_jetson.py  
FrozenGraph build.  
TF-TRT model ready to rumble!  
input tensor shape : (1, 224, 224, 3)  
[['n02124075', 'Egyptian_cat', 0.66293204]]  
starting now (Jetson Nano)...  
Time[s] : 5.124  
FPS      : 48.834
```

С помощью всего нескольких строк кода мы достигли скорости 48 FPS, то есть ускорили вычисления в четыре раза. Интересно, правда? Аналогичного ускорения можно добиться на собственных моделях, адаптированных к конкретным потребностям вашего проекта. Результаты бенчмаркинга можно найти в табл. 15.5.

**Таблица 15.5.** Бенчмаркинг NVIDIA Jetson Nano в разных конфигурациях

Конфигурация	Кадров/с
Jetson Nano + Coral (tflite, 8-битная)	223,2
Jetson Nano (TF-TRT, 16-битная)	48,8
Jetson Nano 128CUDA	12,2
Jetson Nano (tflite, 8-битная)	11,3

## Сравнение производительности краевых устройств

В табл. 15.6 приводится количественное сравнение производительности нескольких краевых устройств при использовании модели MobileNetV2.

**Таблица 15.6.** Полные результаты бенчмаркинга

Конфигурация	Кадров/с
i7-7700K + Coral (tflite, 8-битная)	352,1
i7-7700K + GTX1080 2560CUDA	304,9
Raspberry Pi 4 + Coral	240,4
Jetson Nano + Coral (tflite, 8-битная)	223,2
RPi3 + Coral (tflite, 8-битная)	75,5
Jetson Nano (TF-TRT, 16-битная)	48,8
i7-7700K (tflite, 8-битная)	32,4
i9-9880HQ (2019 MacBook Pro)	15,0
Jetson Nano 128CUDA	12,2
Jetson Nano (tflite, 8-битная)	11,3
i7-4870HQ (2014 MacBook Pro)	11,1
Jetson Nano (tflite, 8-битная)	10,9
Raspberry Pi 4 (tflite, 8-битная)	8,6
Raspberry Pi 4 (tflite)	7,8
RPi3B+ (tflite, 8-битная)	4,2
Raspberry Pi 4	2,7
RPi3B+	2,1

Теперь обобщим результаты экспериментов:

1. Хорошая оптимизация очень важна. Мир оптимизации вычислений продолжает быстро развиваться, и в ближайшие несколько лет мы еще не раз увидим впечатляющие успехи.
2. Когда дело доходит до искусственного интеллекта, начинает действовать правило: чем больше вычислительных устройств (процессоров или ядер), тем лучше.
3. ASIC > FPGA > GPU > CPU — в таком порядке убывает отношение производительность/Батт.

TensorFlow Lite — великолепная библиотека, особенно для маломощных процессоров. Она специально создавалась для маломощных процессоров, и использовать ее на машинах с архитектурой x64 будет не так интересно.

## Примеры из практики

Эта глава посвящена творцам и поэтому была бы неполной без примеров их творчества. Посмотрим на несколько гаджетов и автоматов, созданных благодаря использованию ИИ в краевых устройствах.

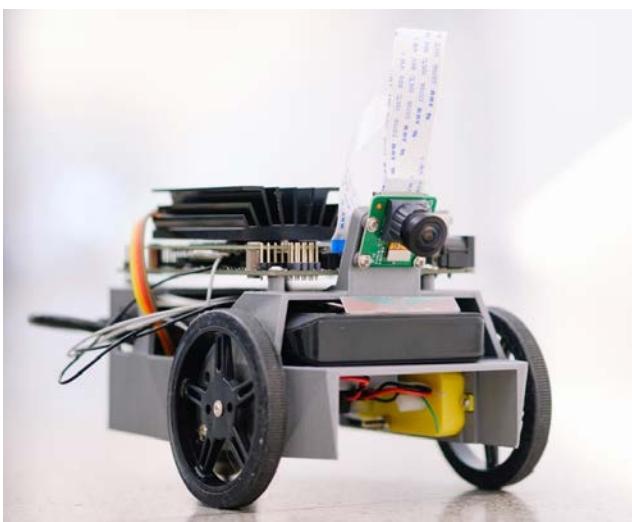
### JetBot

NVIDIA находится в авангарде революции глубокого обучения, предлагая исследователям и разработчикам мощное оборудование и софт. В 2019 году они сделали еще один шаг и выпустили Jetson Nano. Мы уже познакомились с возможностями этого устройства. Вы могли бы использовать его для воплощения какой-нибудь DIY-идеи, например для создания миниатюрного самодельного беспилотного автомобиля. К счастью, NVIDIA выпустила бесплатную книгу рецептов JetBot для сборки робота (рис. 15.12), которым можно управлять с помощью Jetson Nano.

На вики-странице JetBot (<https://oreil.ly/3nExK>) приводятся простые и понятные пошаговые инструкции по сборке. Вот как они выглядят в кратком изложении:

1. Купите детали из списка: электромотор, колесики, камеру, антенну Wi-Fi. Кроме Jetson Nano за 99 долларов в списке еще около 30 деталей общей стоимостью около 150 долларов.
2. Возьмите набор плоскогубцев и отверток и соберите детали. Конечный результат должен быть похож на рис. 15.12.

3. Затем настройте программное обеспечение. Для этого запишите образ JetBot на SD-карту, загрузите Jetson Nano, подключите его к Wi-Fi и, наконец, подключитесь к роботу JetBot с помощью веб-браузера.
4. После этого изучите примеры блокнотов Jupyter, позволяющие с помощью небольших фрагментов кода не только управлять роботом из браузера, но и использовать его камеру для сбора обучающих данных, обучать модели прямо на устройстве (в конце концов, на борту Jetson Nano имеется свой небольшой GPU) и использовать их для обезьяньих препятствий и следования за объектами.



**Рис. 15.12.** NVIDIA JetBot

Многие собрали эту простую конструкцию и пытаются расширить ее возможности, например добавляют лидары для получения большего объема информации об окружающей обстановке, конструируют на основе JetBot различные игрушки и бытовые приборы — танки, роботы-пылесосы (JetRoomba), игрушечных пауков (JetSpider), следящих за объектами, и многое другое. Поскольку все пути в итоге приводят к гоночным автомобилям, команда Jetson Nano выпустила аналогичную книгу рецептов для конструирования гоночного автомобиля JetRacer. В конструкции используются более быстрое шасси, видеокамера с более высокой частотой кадров и фреймворк TensorRT для оптимизации вычисления прогнозов. Модели гоночных автомобилей JetRacer уже участвуют в состязаниях, например на встречах DIY Robocars.

Самый долгий шаг на этом пути — ожидание посылки с деталями.

### От создателя

Мы стремились создать эталонную платформу, чтобы показать другим разработчикам, что можно сделать с этим крошечным устройством — Jetson Nano. Работая над проектом, который превратился в JetBot, мы поняли, что его требования к платформе соответствуют требованиям многих проектов с ИИ следующего поколения.

Для воплощения JetBot требовалась платформа, которая могла обрабатывать изображения в реальном времени, использовать недорогую камеру и легко взаимодействовать с оборудованием. По мере развития Jetson Nano все эти требования были выполнены. Учитывая, что Jetson Nano — это мозг, который управляет роботом JetBot, мы посчитали, что это устройство станет отличной платформой для всех, кто хочет создавать свои проекты с ИИ.

Мы старались сделать JetBot простым и поучительным, чтобы вдохновить разработчиков на творчество с использованием Jetson Nano. Хотя JetBot показывают в презентациях по всему миру, его предназначение намного шире. Этот проект призван стать примером, когда творцы объединяют наши технологии со своим творческим потенциалом и изобретательностью. Появление JetBot уже дало много выгод, и мы надеемся, что это отличное руководство способствует изучению и исследованию ИИ и, что самое главное, получению удовольствия от работы с ним.

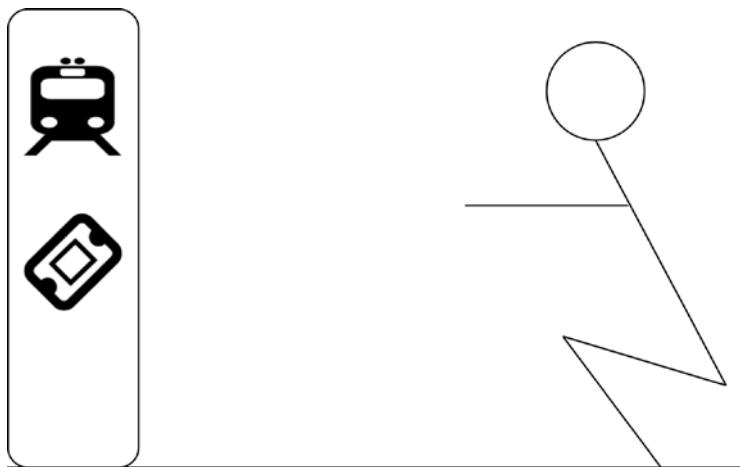
Джон Уэлш (John Welsh) и Читоку Ято (Chitoku Yato),  
разработчики NVIDIA Jetson Nano

## Билеты на проезд в метро за приседания

Когда крепкое здоровье не является достаточным мотивом для занятий физкультурой, что еще могло бы быть мотивом? Как оказывается... бесплатный проезд в метро! Чтобы привлечь внимание общественности к нарастающей проблеме ожирения, на станциях метро в Мехико установили билетные автоматы с камерами, которые выдают бесплатные билеты на проезд, но только тем, кто занимается физкультурой. Бесплатный билет можно получить, сделав десять приседаний (рис. 15.13). В Москве тоже внедрили аналогичную систему на станциях метро, но, судя по всему, там был принят гораздо более высокий стандарт физической подготовки — 30 приседаний на человека.

Вдохновленные этим примером, мы могли бы разработать свое устройство распознавания приседаний. Реализовать такое устройство можно разными способами, и самый простой из них — обучить модель классификатора на классах изображений «есть приседание» и «нет приседания». Однако для этого потребуются тысячи и тысячи изображений людей, которые стоят и сидят на корточках.

Более удачное решение — использовать модель PoseNet (которая определяет местоположение суставов, как было показано в главе 10), скажем, на ускорителе Google Coral USB Accelerator. С его более чем 10 кадрами в секунду мы смогли бы намного точнее подсчитать, сколько раз точки бедер опускались достаточно низко. Для создания такого устройства достаточно одноплатного компьютера Raspberry Pi, камеры Pi Camera, ускорителя Coral USB Accelerator и согласия оператора общественного транспорта предоставить принтер для печати бесконечного количества бесплатных билетов. Воплотив этот проект, мы могли бы сделать жителей наших городов более подтянутыми.



**Рис. 15.13.** Бесплатные билеты за приседания

## Сортировщик огурцов

Макото Койке (Makoto Koike), сын фермеров, выращивающих огурцы в Японии, заметил, что его родители тратят много времени на сортировку огурцов после сбора урожая. Они делили огурцы на девять категорий на основе мельчайших различий в текстуре, наличия крошечных царапин и колючек, а также размера, толщины и кривизны. Система сортировки была сложной, и нанимать работников, занятых неполный рабочий день, было практически невозможно из-за того, что их требовалось долго обучать особенностям сортировки. Макото не стал сидеть сложа руки, наблюдая за своими родителями, занимающимися сортировкой с утра до вечера.

Мы не упомянули одну деталь: г-н Койке занимался созданием краевых систем в Toyota. Он знал, что визуальный контроль можно автоматизировать с помощью глубокого обучения. Он сделал более 7000 снимков разных огурцов, которые его мать вручную отсортировала за три месяца. Затем обучил классификатор,

который мог по изображению огурца с высокой степенью точности определить его категорию. Но ведь сам классификатор почти ничего не делает?

Используя свой опыт работы со краевыми системами, он развернул классификатор на одноплатном компьютере Raspberry Pi, к которому была подключена камера, конвейерная лента и сортировочный манипулятор, помещающий каждый огурец в одну из нескольких корзин, опираясь на прогноз, сделанный Raspberry Pi (рис. 15.14). На первом этапе в Raspberry Pi использовался небольшой классификатор огурец/не огурец. Для изображений, классифицированных как огурец, Raspberry Pi посыпал изображение на сервер, где работал более сложный классификатор, определяющий категорию огурцов. Имейте в виду, что это было в 2015 году, вскоре после появления TensorFlow (задолго до того, как появились TensorFlow Lite, MobileNet и были увеличены вычислительные возможности Raspberry Pi 4 или даже 3). Благодаря этому устройству родители Макото больше времени стали уделять уходу за растениями, а не перебирать целыми днями собранные огурцы.



**Рис. 15.14.** Устройство для сортировки огурцов Макото Койке (с сайта <https://oreil.ly/0cS0p>)

Несмотря на неутихающие дебаты о том, что ИИ отнимает рабочие места, история г-на Койке подчеркивает реальную ценность искусственного интеллекта: повышение производительности людей, расширение их возможностей и, конечно же, гордость родителей.

## Что изучать дальше

Встав на стезю творчества и изобретательства, постарайтесь сосредоточиться на двух одинаково важных аспектах: аппаратном и программном обеспечении. Одно без другого невозможно. Число DIY-проектов намного больше, чем можно собрать за всю свою жизнь. Поэтому выбирайте проекты, которые нравятся вам, попробуйте собрать их от начала до конца и почувствуйте волнение, когда они заработают. Постепенно вы накопите опыт и в следующий раз, читая описание очередного проекта, уже сами догадаетесь, как его реализовать.

Так что засучите рукава и начните воплощать в жизнь проект за проектом. Источниками для вдохновения станут сайты *Hackster.io*, *Hackaday.io* и *Instructables.com*, содержащие широкий спектр тем, платформ и уровней навыков (от начального до продвинутого), нередко с пошаговыми инструкциями.

Расходы на сборку оборудования для начинающих можно уменьшить, купив готовые комплекты. Например, AI Kits для Raspberry Pi, включая AIY Vision Kit (Google), AIY Speech Kit (Google), GoPiGo (Dexter Industries) и платформу Duckie-Town (MIT), и это лишь малая часть. Они не требуют большого опыта для начала работы и делают область электроники более доступной — от старшеклассников до любителей-конструкторов.

Кроме того, более высокая скорость работы ИИ на устройствах с низким энергопотреблением означает более высокую производительность моделей. Добиться более высокой скорости помогут методы сжатия моделей: квантование и проеживание (о них шла речь в главе 6). Чтобы еще больше увеличить скорость работы (за счет некоторой потери точности), используйте бинарные нейронные сети, веса которых хранятся в виде однобитных значений вместо 32-битных. Создавать подобные модели позволяют разработки XNOR.ai и Microsoft Embedded Learning Library. Книга Пита Уордена «*TinyML*» станет отличным источником знаний для запуска ИИ даже на устройствах с низким энергопотреблением — микроконтроллерах с объемом памяти менее 100 Кбайт.

## Итоги

В этой главе мы познакомились с известными краевыми устройствами, пригодными для ИИ, и посмотрели, как запустить модель Keras на некоторых из них. Мы составили общее представление об их внутреннем устройстве и узнали, как добиться достаточной вычислительной мощности для работы нейронных сетей. В заключение мы представили сравнительный анализ этих устройств, чтобы вы могли осознанно выбирать их для своих проектов в зависимости от размера, стоимости и требований к быстродействию и питанию. Затем познакомились с некоторыми проектами роботизированных гаджетов,

рассмотрели способы создания таких проектов, которые используют творцы во всем мире. Гаджеты и приспособления, обсуждаемые в этой главе, — это лишь очень узкая часть спектра. Со временем будут появляться новые устройства, потому что краевые устройства становятся все более желанным местом применения ИИ.

Благодаря широчайшим возможностям ИИ на краевых устройствах, творцы могут придумать и воплотить в жизнь проекты своей мечты, которые еще несколько лет назад были научной фантастикой. Помните, что в своем путешествии по миру DIY-приспособлений и гаджетов с ИИ вы находитесь в сплоченном сообществе энтузиастов-изобретателей, которые открыты и готовы поделиться своим опытом и знаниями. В интернете есть много форумов, и, столкнувшись с какой-нибудь проблемой, вы всегда сможете найти людей, которые помогут. Надеюсь, реализуя свои проекты, вы сможете вдохновить других стать творцами.

## ГЛАВА 16

# Моделирование беспилотного автомобиля методом сквозного глубокого обучения с использованием Keras

Написана Адитьей Шарма и Митчеллом Сприном

От бэтмобиля до КИТТ<sup>1</sup>, от роботакси до автоматического доставщика пиццы беспилотные автомобили стали неотъемлемой частью поп-культуры. А почему бы и нет? Не редкость, когда идея из научно-фантастического романа воплощается в жизнь. Что еще важно, беспилотные автомобили обещают решить некоторые ключевые проблемы, с которыми сейчас сталкиваются жители городов: ДТП, высокий уровень загрязнения, перегруженные дороги и траты времени в пробках. Этот список можно продолжать долго.

Создание полноценной системы автономного вождения — довольно сложная задача, и ее невозможно в полной мере охватить в одной главе книги. Любая сложная проблема — как луковица: ее можно разделить на слои и решать по частям. И здесь мы намерены решить одну фундаментальную проблему: управление направлением движения. Самое интересное, что для этого не нужна настоящая машина. Мы будем обучать нейронную сеть управлять автомобилем в виртуальной среде, используя реалистичную физику и визуальные эффекты.

Но сначала немного истории.

<sup>1</sup> Автомобиль с искусственным интеллектом из американского телесериала «Рыцарь дорог». — Примеч. пер.

## Уровни автоматизации вождения

«Автономное вождение» — зонтичный термин, который часто используется для описания множества вещей, от технологии адаптивного круиз-контроля в Chrysler 200 до управления автомобилями будущего, не имеющими руля. Чтобы несколько упорядочить использование этого термина, международное сообщество инженеров автомобильной промышленности (Society of Automotive Engineers, SAE) предложило разделить вождение на шесть уровней, начиная с уровня 0 («без автоматизации») до 5 («полная автоматизация»), как показано на рис. 16.1. Конечно, эти уровни требуют некоторой дополнительной интерпретации, но даже в таком виде дают четкие ориентиры, когда речь заходит об автоматизации управления. Адаптивный круиз-контроль — это пример автоматизации уровня 1, тогда как автопилоты на автомобилях Tesla находятся где-то между уровнями 2 и 3. Многие технологические компании и автопроизводители работают над автомобилями уровня 4, но пока не было создано ни одного промышленного образца. Пройдет некоторое время, прежде чем воплотится в жизнь наша мечта о полностью автоматическом автомобиле пятого уровня.

Уровни автоматизации, предложенные SAE

Полная  
автоматизация



Рис. 16.1. Уровни автоматизации вождения, предложенные SAE (с сайта <https://oreil.ly/RHKUt>)

## Краткая история автоматизации вождения

Может показаться, что автономное вождение — недавняя технологическая разработка, но исследования в этой области начались более 30 лет назад. Ученые из университета Карнеги — Меллона впервые попробовали свои силы в этой, казалось бы, фантастической области еще в 1984 году, когда начали работу над проектом Navlab 1. «Navlab» — это сокращение от Carnegie Mellon Navigation Laboratory (навигационная лаборатория университета Карнеги — Меллона). Этот проект стал первым шагом в удивительное будущее человечества. Navlab 1 представлял собой фургон Chevrolet (рис. 16.2) с установленным оборудованием на борту, в том числе — рабочие станции для исследователей и суперкомпьютер Sun Microsystems. Этот проект считался самой передовой вычислительной системой на колесах. В то время практически не было беспроводной связи на большие расстояния, а облачные вычисления даже не были оформлены в виде идеи. Размещение всей группы в кузове экспериментального фургона могло показаться опасной идеей, но максимальная скорость NavLab составляла 32 километра в час, и двигался он только по пустым дорогам. Так что работающие на борту ученые были в безопасности. Технология, использованная в этом автомобиле, заложила основу для технологий, используемых в современных беспилотных автомобилях. Например, для наблюдения за окружающей обстановкой в Navlab 1 использовались видеокамеры и дальномер (ранняя версия лидара). Навигацию осуществляла однослойная нейронная сеть, определяющая углы поворота по датчикам. Довольно красиво, не правда ли?



**Рис. 16.2.** NavLab 1 во всей своей красе (изображение взято с веб-сайта <https://oreil.ly/b2Bnn>)

### **Расскажите о своих достижениях в робототехнике**

Было создано несколько версий Navlab 1. Например, в 1995 году Navlab 5 смог проехать весь путь от Питтсбурга до Сан-Диего, преодолев почти 4586 километров, за исключением 80 километров на некоторых участках. За это достижение Navlab 5 был занесен в Зал славы роботов. Попадание в Зал славы в мире роботов делает вас элитой. Если вы когда-нибудь будете в Питтсбурге и захотите увидеть некоторых представителей, удостоенных этой престижной награды, обязательно посетите выставку Roboworld в научном центре Карнеги.

## **Глубокое обучение, автономное вождение и проблема данных**

35 лет тому назад ученые уже знали, что нейронные сети будут играть важную роль в беспилотных автомобилях. Тогда, конечно, не было технологий (GPU, облачных вычислений, FPGA и т. д.), необходимых для обучения и развертывания достаточно масштабных глубоких нейронных сетей для воплощения автономного вождения в реальность. Современные беспилотные автомобили используют глубокое обучение для выполнения самых разных задач. В табл. 16.1 приведены некоторые примеры.

**Таблица 16.1.** Примеры применения глубокого обучения в беспилотных автомобилях

Задача	Примеры
Восприятие	Обнаружение проезжей части, дорожной разметки, перекрестков, пешеходных переходов и т. д.
	Обнаружение других транспортных средств, пешеходов, животных, предметов на дороге и т. д.
	Обнаружение дорожных знаков и светофоров
Навигация	Определение угла поворота рулевого колеса, положения педали газа и т. д. на основе данных, поступающих с датчиков
	Выполнение обгонов, смена полосы движения, развороты и т. д.
	Выезд на дорогу и съезд с нее, пересечение перекрестков с круговым движением и т. д.
	Проезд по туннелям, мостам и т. д.
	Реакция на нарушения правил дорожного движения другими водителями, неожиданные изменения в окружающей обстановке и т. д.
	Навигация по пробкам

## Симуляция и автономное вождение

Симуляция играет очень важную роль в разработке автопилотов. Прежде чем систему автономного вождения можно будет безопасно развернуть на реальных транспортных средствах, она должна преодолеть часы и часы испытаний и проверки на симуляторах, которые точно имитируют среду, в которой эти транспортные средства будут перемещаться. Симуляторы обеспечивают возможность проверки алгоритмов с обратной связью. Беспилотное транспортное средство, действующее в симуляторе, получает данные от датчиков и на их основе производит управляющие воздействия, оказывая прямое влияние на окружающую среду и на следующий набор показаний с датчиков, тем самым замыкая цикл.

Для проверки разных частей системы автономного вождения используются разные типы симуляторов (рис. 16.3). Для задач, требующих реалистичного визуального представления окружающей среды, таких как задачи распознавания объектов, используются фотогенеративные симуляторы. Эти симуляторы также пытаются точно имитировать физику объектов, динамику материалов и т. д. Примеры фотогенеративных симуляторов — решения с открытым исходным кодом AirSim и Carla. С другой стороны, для задач, где реалистичные визуальные эффекты не требуются, можно сэкономить ресурсы GPU и использовать симуляторы построения приятия. Большинство этих задач используют результаты, полученные системой восприятия для планирования маршрута и навигации. Примером симуляторов этой категории может служить симулятор с открытым исходным кодом Baidu Apollo.



**Рис. 16.3.** Фотореалистичный симулятор AirSim и симулятор построения приятия Apollo (с сайта <https://oreil.ly/rag7v>)

Стартап Cognata из Израиля, занимающийся проблемой автономного вождения, — лидер в этой области благодаря своей фотогенеративной облачной платформе моделирования. Cognata решает некоторые из ключевых задач, используя инновационный подход с применением глубокой нейронной сети. «Я пришел в эту сферу после почти десяти лет работы над усовершенствованными системами помощи водителю, основанными на компьютерном зрении», — сказал CEO и основатель Cognata Дэнни Атсмон (Danny Atsmon).

«Я обнаружил, что использование методов визуализации на основе OpenGL не дает того качества изображения, которое нужно для обучения или проверки глубокой нейронной сети, и пришел к выводу, что для этого необходим новый метод визуализации. Наш механизм визуализации использует глубокую нейронную сеть для изучения распределений на основе реальных данных и визуализирует смоделированные данные, стараясь как можно точнее сымитировать реальный датчик».

Мы действительно живем в фантастическое время, когда инновации в аппаратном и программном обеспечении объединяются для решения одной из самых сложных технических задач, которые мы когда-либо видели: создание беспилотного автомобиля.

Мы знаем, что для глубокого обучения нужны данные. Чем больше данных, тем выше качество модели. В предыдущих главах мы видели, что для обучения классификаторов могут потребоваться миллионы изображений. Можно представить, сколько данных нужно, чтобы обучить модель управлять автомобилем. Но в случае с беспилотными автомобилями ситуация меняется, когда дело доходит до данных, необходимых для проверки и тестирования. Автомобиль не только должен уметь ездить, он также должен делать это безопасно. Поэтому тестирование и проверка модели принципиально важны и требуют гораздо большего количества данных, чем для обучения, в отличие от традиционных задач глубокого обучения. Но точно предсказать объем нужных данных довольно трудно. Как показали исследования, проведенные в 2016 году, нейронная сеть должна проехать почти 20 миллиардов километров, чтобы достичь уровня хорошего водителя-человека. Для парка из 100 беспилотных автомобилей, собирающих данные 24 часа в сутки 365 дней в году и движущихся со средней скоростью 40 километров в час, на это уйдет более 500 лет!

Конечно, было бы крайне непрактично и дорого собирать данные на протяжении 11 миллиардов миль с использованием парка автомобилей, разъезжающих по реальному миру. Вот почему почти каждый, кто работает в этой сфере, будь то гигант-автопроизводитель или стартап, для сбора данных и проверки обученных моделей использует симуляторы. Например, Waymo (<https://waymo.com/tech>, команда Google по созданию беспилотных автомобилей) к концу 2018 года проехала по дорогам почти 18 миллионов километров. Это намного больше, чем у любой другой компании на планете, но менее одного процента от требуемых 20 миллиардов километров. А в симуляторах Waymo проехала более 11 миллиардов километров. Практически все используют симуляторы для проверки своих моделей, но одни компании создают собственные симуляторы, а другие лицензируют их у Cognata, Applied Intuition и AVSimulation. Есть отличные симуляторы с открытым исходным кодом: Microsoft AirSim (<https://github.com/Microsoft/AirSim>), Intel Carla (<http://carla.org/>) и Baidu Apollo Simulation (<https://oreil.ly/rag7v>). Благодаря этим инструментам не

нужно подключаться к CAN-шине нашего автомобиля, чтобы изучать основы автономногоождения.

В этой главе используем версию AirSim, специально разработанную для *Autonomous Driving Cookbook* (<https://oreil.ly/uzOGL>).

## «Hello, World!» в автономномождении: управление в моделируемой среде

В этом разделе реализуем одну из простых задач в сфере автономногоождения. Беспилотные автомобили сложны, имеют десятки датчиков, передающих гигабайты данных, и принимают множество решений во время движения. Как и в программировании, решая задачу «Hello, World!» в области автономногоождения, мы ограничимся следующими базовыми требованиями:

- Автомобиль всегда остается на дороге.
- В качестве источника входной информации будет использоваться один кадр изображения, полученный с единственной камеры, установленной на передней части капота. Никакие другие датчики (лидары, радары и др.) не используются.
- На основе этого входного одиночного изображения движущийся с постоянной скоростью автомобиль должен выбрать угол поворота рулевого колеса. Никаких других управляющих воздействий (торможение, нажатие/отпускание педали газа, переключение передач и т. д.) не производится.
- Ни на дороге, ни вокруг нее нет других транспортных средств, пешеходов, животных или чего-либо еще.
- Дорога является однополосной, не имеет разметки, дорожных знаков и светофоров, а также не предъявляется требование придерживаться левой или правой стороны.
- Дорога меняет только направление (что требует изменения угла поворота рулевого колеса), но не меняет уклон (что может потребовать изменения положения педали газа, применения тормозов и т. д.).

### Инструменты и требования

Реализуем задачу «Hello, World!» на ландшафтной карте в AirSim (рис. 16.4). AirSim — это платформа с открытым исходным кодом для фотопрералистичного моделирования и популярный исследовательский инструмент для сбора обучающих данных и проверки моделей глубокого обучения для автономногоождения.



**Рис. 16.4.** Ландшафтная карта в AirSim

Код для этой главы можно найти в руководстве по глубокому обучению ([https://oreil.ly/\\_Jl9](https://oreil.ly/_Jl9)) из *Autonomous Driving Cookbook* (<https://oreil.ly/uF5Bz>) на GitHub в виде блокнотов Jupyter. Перейдите в папку *AirSimE2EDeepLearning/* (<https://oreil.ly/YVKPp>). Получить руководство можно также, клонировав репозиторий к себе на компьютер:

```
$ git clone https://github.com/Microsoft/AutonomousDrivingCookbook.git  
$ cd AutonomousDrivingCookbook/AirSimE2EDeepLearning/
```

Для реализации нейронной сети используем уже знакомую библиотеку Keras. Для решения этой задачи не требуется знакомства с какими-либо новыми идеями глубокого обучения, кроме тех, которые мы рассмотрели в предыдущих главах.

Для этой главы мы создали датасет, моделируя поездки в AirSim. Размер несжатого датасета — 3,25 Гбайт. Он ненамного больше датасетов, которые мы использовали раньше, но не забывайте, что мы реализуем упрощенную версию сложной задачи. Для сравнения: автопроизводители ежедневно собирают по несколько петабайт данных. Датасет доступен по адресу <https://aka.ms/AirSimTutorialDataset>.

Можно запустить предлагаемый код и обучить свою модель на компьютере на Windows или Linux. Для этой главы мы создали автономную сборку AirSim, которую можно получить по адресу <http://aka.ms/ADCookbookAirSimPackage>. После загрузки распакуйте пакет с симулятором в каталог по вашему выбору. Запомните или запишите путь к нему — он понадобится позже. Обратите внимание: эта сборка работает только в Windows, но она нужна, только чтобы

протестировать готовую обученную модель. Если вы используете Linux, то возьмите файлы модели и запустите их на компьютере с Windows, использовав предлагаемый пакет симулятора.

AirSim – это фотореалистичный симулятор среды, он способен создавать реалистичные изображения окружающей среды, на которых можно обучить модель. Похожую графику можно увидеть в видеоиграх высокого разрешения. Учитывая объем данных и качество графики в симуляторе, желательно, чтобы на компьютере, где будет выполняться код, был GPU.

Для запуска кода установите некоторые дополнительные инструменты и зависимости. Подробнее о них написано разделе «Environment Setup» (настройка среды) в файле README (<https://oreil.ly/RzZe7>) в репозитории. В общем случае вам понадобятся дистрибутив Anaconda с Python 3.5 (или выше), библиотека TensorFlow (которая используется Keras) и инструмент h5py (для сохранения и чтения файлов данных и моделей). После установки и настройки среды Anaconda можно установить все другие необходимые зависимости Python, запустив файл *InstallPackages.py* от имени суперпользователя root или администратора.

В табл. 16.2 – список всех требований, которые мы только что определили.

**Таблица 16.2.** Список компонентов и требований

Компонент	Требования или ссылка	Примечания
Программный код	<a href="https://oreil.ly/_IJl9">https://oreil.ly/_IJl9</a>	Можно запускать в Windows или Linux
Jupyter Notebook	<i>DataExplorationAndPreparation.ipynb</i> ( <a href="https://oreil.ly/6yvCq">https://oreil.ly/6yvCq</a> ) <i>TrainModel.ipynb</i> ( <a href="https://oreil.ly/rCr47">https://oreil.ly/rCr47</a> ) <i>TestModel.ipynb</i> ( <a href="https://oreil.ly/FE-EP">https://oreil.ly/FE-EP</a> )	
Датасет	<a href="http://aka.ms/AirSimTutorialDataset">http://aka.ms/AirSimTutorialDataset</a>	3,25 Гбайт, необходим для обучения модели
Симулятор	<a href="http://aka.ms/ADCookbookAirSimPackage">http://aka.ms/ADCookbookAirSimPackage</a>	Не требуется для обучения модели — необходим только для ее тестирования; работает только в Windows
GPU	Рекомендуется для обучения, необходим для симулятора	
Другие инструменты + зависимости для Python	Раздел «Environment Setup» в файле README ( <a href="https://oreil.ly/RzZe7">https://oreil.ly/RzZe7</a> )	

А теперь, разобравшись со всеми инструментами, приступим к делу.

## Исследование и подготовка данных

По мере погружения в мир нейронных сетей и глубокого обучения вы неизбежно заметите, что магия машинного обучения заключается не в структуре нейронных сетей и особенностях их обучения, а в понимании сути проблемы, данных и предметной области. Автономное вождение не исключение. Как вы вскоре увидите, глубокое понимание данных и решаемой задачи — ключевое условие успеха в обучении модели беспилотного автомобиля.

Все шаги, представленные здесь, описаны в блокноте *DataExplorationAndPreparation.ipynb* (<https://oreil.ly/6aE7z>). Начнем с импорта всех модулей для этой части упражнения:

```
import os
import random

import numpy as np
import pandas as pd
import h5py

import matplotlib.pyplot as plt
from PIL import Image, ImageDraw

import Cooking #Этот модуль содержит вспомогательный код. Не правьте этот файл!
```

Убедимся, что программа сможет найти распакованный датасет. Также определим место для хранения обработанных (то есть «подготовленных») данных:

```
# << Укажите здесь путь к каталогу с исходными данными >>
RAW_DATA_DIR = 'data_raw/'

# << Укажите здесь путь к каталогу для сохранения подготовленных (.h5) данных >>
COOKED_DATA_DIR = 'data_cooked/'

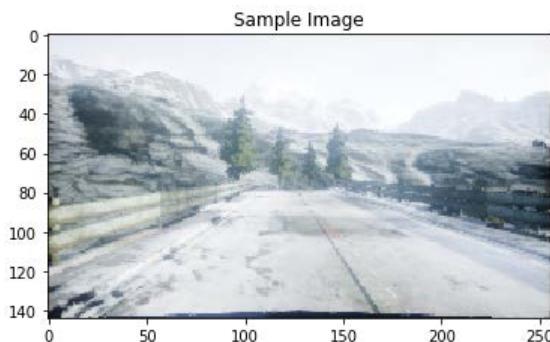
# Подпапки в каталоге RAW_DATA_DIR для поиска данных
# Например, сначала поиск будет выполнен в папке RAW_DATA_DIR/normal_1
DATA_FOLDERS = ['normal_1', 'normal_2', 'normal_3', 'normal_4', 'normal_5',
                'normal_6', 'swerve_1', 'swerve_2', 'swerve_3']

# Размеры используемых изображений и иллюстраций
FIGURE_SIZE = (10,10)
```

Заглянув в каталог с датасетом, можно заметить, что в нем девять папок — от *normal\_1* до *normal\_6* и от *swerve\_1* до *swerve\_3*. Вернемся к этим названиям позже. В каждой из этих папок есть изображения, а также файлы *.tsv* (или *.txt*). Посмотрим, как выглядит одно из этих изображений:

```
sample_image_path = os.path.join(RAW_DATA_DIR, 'normal_1/images/img_0.png')
sample_image = Image.open(sample_image_path)
plt.title('Sample Image')
plt.imshow(sample_image)
plt.show()
```

Вы должны увидеть результат, как на рис. 16.5. Это изображение было снято камерой, расположенной на капоте автомобиля (край которого виден внизу на рис. 16.5). На картинке мы видим окружающий ландшафт, генерированный симулятором. Обратите внимание, что, согласно нашим требованиям, на дороге нет других машин или объектов. Дорога кажется заснеженной, но для нашего эксперимента снег является только визуальным эффектом и не повлияет на физику поведения машины. Конечно, в реальном мире очень важно точно воспроизвести физику снега, дождя и т. д., чтобы симуляторы могли создавать высокоточные версии реальности. Это очень сложная задача, и многие компании выделяют на нее много ресурсов. К счастью, в нашей задаче мы можем спокойно игнорировать все это и рассматривать снег просто как белые пиксели на изображении.



**Рис. 16.5.** Содержимое файла normal\_1/images/img\_0.png

Теперь посмотрим, как выглядят остальные данные, отобразив содержимое одного из файлов `.tsv/.txt`:

```
sample_tsv_path = os.path.join(RAW_DATA_DIR, 'normal_1/airsim_rec.txt')
sample_tsv = pd.read_csv(sample_tsv_path, sep='\t')
sample_tsv.head()
```

Вы увидите следующее:

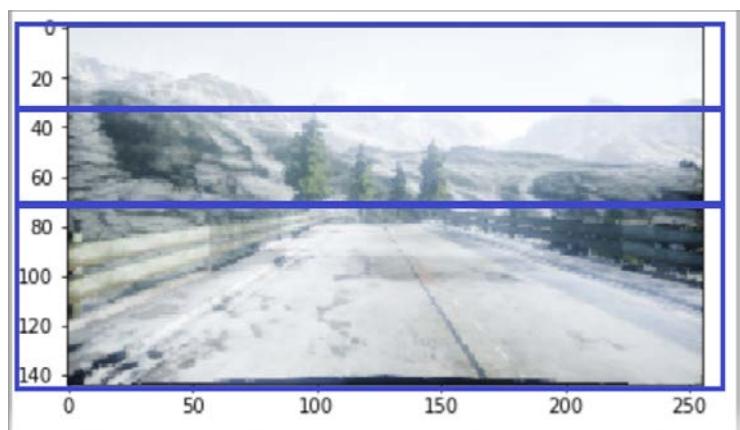
	Timestamp	Speed (kmph)	Throttle	Steering	Brake	Gear	ImageName
0	93683464	0	0.0	0.000000	0.0	N	img_0.png
1	93689595	0	0.0	0.000000	0.0	N	img_1.png
2	93689624	0	0.0	-0.035522	0.0	N	img_2.png
3	93689624	0	0.0	-0.035522	0.0	N	img_3.png
4	93689624	0	0.0	-0.035522	0.0	N	img_4.png

Здесь много интересной информации, поэтому ненадолго приостановимся и изучим ее. Каждая строка в таблице соответствует кадру изображения (здесь показаны только первые пять кадров). Поскольку это начало датасета, мы видим, что автомобиль еще не начал двигаться: скорость (столбец *Speed (kmph)*) и положение педали газа (столбец *Throttle*) равны 0, и выбрана нейтральная передача (столбец *Gear*). Поскольку в нашей задаче модель должна предсказывать только углы поворота, не будем прогнозировать значение скорости, положение педали газа, использование тормоза или выбор передачи. Но эти значения будут использоваться как часть входных данных (подробнее об этом позже).

## Определение области интереса

А теперь вернемся к образцу изображения. Все наблюдения были сделаны с точки зрения человека. Давайте еще раз взглянем на изображение, и на этот раз рассмотрим его с точки зрения автопилота, который только учится управлять автомобилем и пытается понять, что видит.

Я как автопилот, глядя на это изображение, полученное с камеры, могу разделить его на три части (рис. 16.6). Нижняя треть выглядит более или менее однородно. Она состоит в основном из прямых линий (дорога, разделитель полос, дорожные ограждения и т. д.) и имеет однородную окраску с участками белого, черного, серого и коричневого цвета. Еще есть странная черная дуга в самом низу (край капота машины). Верхняя треть изображения, как и нижняя, тоже относительно однородна. У нее преимущественно серо-белая окраска (небо и облака). Но средняя третья выбивается из общего ряда — на ней множество деталей.



**Рис. 16.6.** Три части изображения, как их видит автопилот

Есть большие коричневые и серые образования (горы), которые совершенно не однородны. Также есть четыре высокие зеленые фигуры (деревья), их форма и цвет отличаются от всего, что я вижу, поэтому они, вероятно, очень важны. Последовательно просматривая другие изображения, я замечаю, что верхняя и нижняя трети не сильно меняются, но средняя треть претерпевает большие изменения. Следовательно, я делаю вывод, что это та часть, на которой я должен сосредоточиться больше всего.

Видите, какие сложности испытывает автопилот? Он не может с ходу определить, какая часть наиболее важна. Он может попытаться связать изменяющиеся углы поворота с изменяющимся пейзажем вместо того, чтобы сосредоточиться на поворотах дороги, которая обозначена очень тонкими линиями, теряющимися на фоне остального пейзажа. Изменения в пейзаже не только сбивают с толку, но и не имеют никакого отношения к задаче, которую мы пытаемся решить. Небо в основном неизменное, не дает никакой информации, касающейся вождения. Часть капота, запечатленная на каждом снимке, тоже не важна.

Давайте поможем автопилоту сконцентрироваться только на соответствующей части изображения. Для этого определим область интереса (Region Of Interest, ROI) на изображении. Как нетрудно догадаться, жестких правил, определяющих, как должна выглядеть область интереса, нет. Выбор такой области во многом зависит от решаемой задачи. В нашем случае, поскольку камера находится в фиксированном положении и уклон дороги не изменяется, ROI представляет собой простой прямоугольник, охватывающий дорогу и ее границы. Можно увидеть ROI, запустив следующий фрагмент кода:

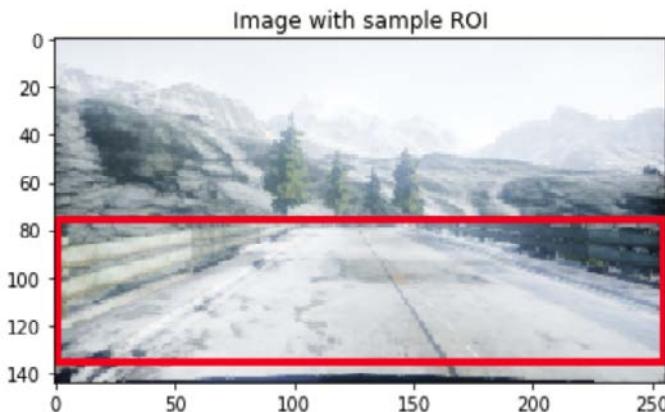
```
sample_image_roi = sample_image.copy()

fillcolor=(255,0,0)
draw = ImageDraw.Draw(sample_image_roi)
points = [(1,76), (1,135), (255,135), (255,76)]
for i in range(0, len(points), 1):
    draw.line([points[i], points[(i+1)%len(points)]], fill=fillcolor, width=3)
del draw

plt.title('Image with sample ROI')
plt.imshow(sample_image_roi)
plt.show()
```

В результате вы должны увидеть изображение, как на рис. 16.7.

Теперь все внимание наша модель будет уделять только дороге, и изменения в окружающем ландшафте мешать ей не будут. Такое ограничение также позволит уменьшить размер изображения вдвое, что упростит обучение нейронной сети.



**Рис. 16.7.** Область, на которой автопилот должен сконцентрироваться при обучении

## Аугментация данных

Как упоминалось выше, для обучения глубоких нейронных сетей лучше иметь как можно больше данных. В предыдущих главах мы видели, насколько важна аугментация данных и как это помогает не только увеличить объем обучающих данных, но и избежать переобучения. Но большинство методов аугментации данных, описанных ранее, применяется для обучения классификаторов изображений, и они бесполезны для задачи обучения автопилота. Возьмем, например, поворот изображения. Случайный поворот изображений на 20 градусов очень полезен при обучении классификатора, работающего с камерой смартфона, но камера на капоте нашей машины зафиксирована и никогда не увидит повернутое изображение (если только машина не перевернется, но это значит, что у нас очень большая проблема). То же верно и в отношении случайного сдвига — сдвиг изображения никогда не встретится во время поездки. Но все же есть кое-какие методы, подходящие для аугментации данных в текущей задаче?

Присмотревшись к изображению, можно заметить, что зеркальное отражение относительно оси Y дает изображение, которое могло быть получено в другом тестовом прогоне (рис. 16.8). То есть мы с легкостью можем удвоить объем датасета, если используем зеркальное отражение. Но просто перевернуть изображение недостаточно, нужно изменить связанный с ним угол поворота. Поскольку новое изображение — это зеркальное отображение оригинала, можно просто изменить знак угла поворота (например, с 0,212 на -0,212).

Есть еще один прием, который применяют для аугментации данных. Автопилоты должны быть готовы не только к изменениям на дорогах, но также к изменениям внешних условий — погоды, времени суток и света. Большинство современных симуляторов позволяют создавать эти условия синтетическим путем. Все наши данные были собраны в условиях яркого дневного освещения. Но чтобы не

возвращаться к симулятору для сбора дополнительных данных при различных условиях освещения, просто внесем случайные изменения в освещение, регулируя яркость изображения во время обучения на имеющихся данных. Пример такого изменения показан на рис. 16.9.



**Рис. 16.8.** Зеркальное отражение изображения по оси Y



**Рис. 16.9.** Уменьшение яркости изображения на 40 %

## Дисбаланс датасета и стратегии вождения

Модели глубокого обучения хороши ровно настолько, насколько хороши обучающие данные. В отличие от людей, современный ИИ не способен думать и рассуждать. Поэтому, столкнувшись с совершенно новой ситуацией, он будет опираться на виденное им ранее и делать прогнозы исходя из данных, на которых был обучен. Кроме того, статистический характер глубокого обучения заставляет его игнорировать редкие случаи и интерпретировать их как отклонения и выбросы, даже если они важны. Это называется *дисбалансом датасета*, и это головная боль для специалистов по данным.

Представьте, что мы обучаем классификатор, предназначенный для анализа дерматоскопических снимков и определения доброкачественности или злокачественности поражений кожи. Предположим, что в нашем датасете миллион изображений, на 10 000 из которых есть злокачественные поражения. Вероятно, что классификатор, обученный на этих данных, всегда будет предсказывать доброкачественность поражений и никогда — злокачественность. Это

объясняется тем, что алгоритмы глубокого обучения предназначены для минимизации ошибок (повышения точности) во время обучения. Классифицируя все изображения как доброкачественные, модель достигает 99 % точности и решает все проблемы, но при этом не справляется с задачей, которой ее обучили: обнаружению раковых образований. Эта проблема обычно решается путем обучения предикторов на более сбалансированном датасете.

Автопилот не застрахован от проблемы дисбаланса датасета. Возьмем нашу модель прогнозирования угла поворота. Что мы знаем об углах поворота из повседневного опыта вождения? В основном мы ездим по дорогам прямо. Если модель обучить только на типичных примерах дорожной ситуации, она никогда не научится ориентироваться в поворотах из-за их относительной редкости в наборе обучающих данных. Чтобы решить эту проблему, важно, чтобы датасет содержал информацию не только для обычной стратегии вождения, но и для стратегии вождения с поворотами, причем в статистически значимых количествах. Вернемся к нашим файлам *.tsv/.txt*. Мы обращали ваше внимание на названия папок с данными. Теперь становится ясно, что в датасете есть данные, полученные в шести прогонах сбора данных с использованием обычной стратегии вождения и в трех прогонах — с использованием стратегии вождения с поворотами.

Объединим данные из всех файлов *.tsv/.txt* в единую коллекцию `DataFrame`, чтобы упростить анализ:

```
full_path_raw_folders = [os.path.join(RAW_DATA_DIR, f) for f in DATA_FOLDERS]

dataframes = []
for folder in full_path_raw_folders:
    current_dataframe = pd.read_csv(os.path.join(folder, 'airsim_rec.txt'),
                                    sep='\t')
    current_dataframe['Folder'] = folder
    dataframes.append(current_dataframe)
dataset = pd.concat(dataframes, axis=0)
```

Нанесем некоторые углы поворота для обеих стратегий вождения на диаграмму рассеяния:

```
min_index = 100
max_index = 1500
steering_angles_normal_1 = dataset[dataset['Folder'].apply(lambda v: 'normal_1' in v)][['Steering']][min_index:max_index]
steering_angles_swerve_1 = dataset[dataset['Folder'].apply(lambda v: 'swerve_1' in v)][['Steering']][min_index:max_index]

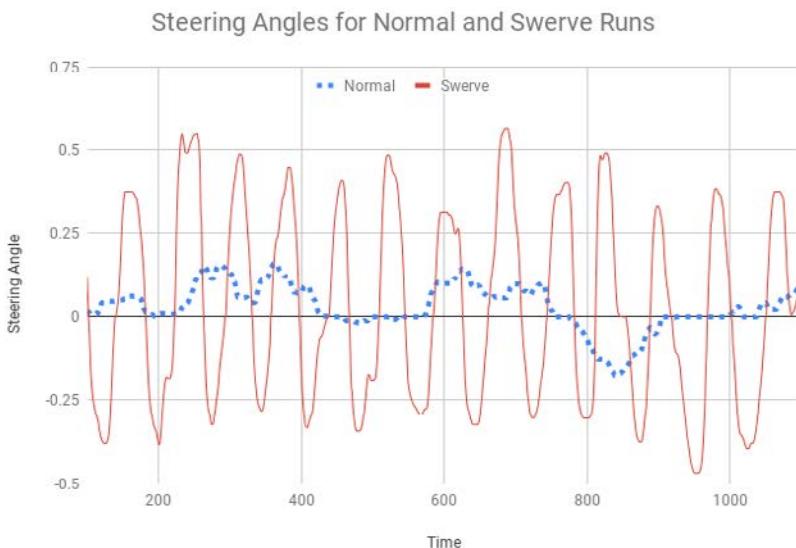
plot_index = [i for i in range(min_index, max_index, 1)]
fig = plt.figure(figsize=FIGURE_SIZE)
ax1 = fig.add_subplot(111)
ax1.scatter(plot_index, steering_angles_normal_1, c='b', marker='o',
            label='normal_1')
ax1.scatter(plot_index, steering_angles_swerve_1, c='r', marker='_',
            label='swerve_1')
```

```

plt.legend(loc='upper left');
plt.title('Steering Angles for normal_1 and swerve_1 runs')
plt.xlabel('Time')
plt.ylabel('Steering Angle')
plt.show()

```

Результат показан на рис. 16.10.



**Рис. 16.10.** Диаграмма, показывающая углы поворота для двух стратегий вождения

Изобразим также количество точек данных, собранных с помощью каждой стратегии:

```

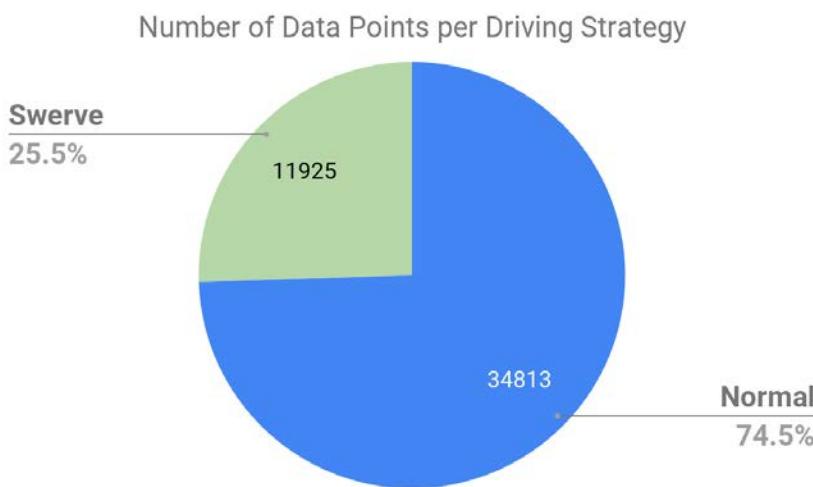
dataset['Is Swerve'] = dataset.apply(lambda r: 'swerve' in r['Folder'], axis=1)
grouped = dataset.groupby(by=['Is Swerve']).size().reset_index()
grouped.columns = ['Is Swerve', 'Count']

def make_autopct(values):
    def my_autopct(percent):
        total = sum(values)
        val = int(round(percent*total/100.0))
        return '{0:.2f}% ({1:d})'.format(percent, val)
    return my_autopct

pie_labels = ['Normal', 'Swerve']
fig, ax = plt.subplots(figsize=FIGURE_SIZE)
ax.pie(grouped['Count'], labels=pie_labels,
       autopct=make_autopct(grouped['Count']), explode=[0.1, 1],
       textprops={'weight': 'bold'}, colors=['lightblue', 'salmon'])
plt.title('Number of data points per driving strategy')
plt.show()

```

Результат показан на рис. 16.11.



**Рис. 16.11.** Деление датасета на две стратегии вождения

Глядя на рис. 16.10, мы видим, что нормальная стратегия вождения дает углы поворота, которые наблюдаются в повседневной езде: движение в основном происходит по прямой с редкими поворотами. Напротив, стратегия вождения с поворотами в основном ориентирована на крутые повороты и, соответственно, имеет более высокие значения углов поворота. Как показано на рис. 16.11, комбинация этих двух стратегий дает хорошее, хотя и далекое от реальной жизни распределение обучающих данных с соотношением 75/25. Это еще больше подтверждает важность использования симуляторов для обучения автопилота, потому что маловероятно, что удастся собрать равноценный объем данных с использованием стратегии с поворотами на реальном автомобиле.

Прежде чем завершить обсуждение приемов предварительной обработки и дисбаланса датасета, рассмотрим распределение углов поворота в двух стратегиях вождения, построив их гистограмму (рис. 16.12):

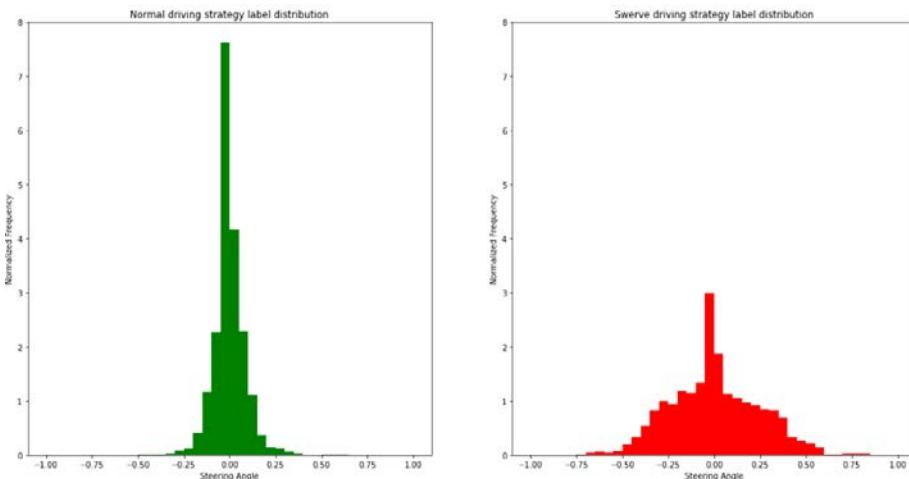
```

bins = np.arange(-1, 1.05, 0.05)
normal_labels = dataset[dataset['Is Swerve'] == False]['Steering']
swerve_labels = dataset[dataset['Is Swerve'] == True]['Steering']

def steering_histogram(hist_labels, title, color):
    plt.figure(figsize=FIGURE_SIZE)
    n, b, p = plt.hist(hist_labels.as_matrix(), bins, normed=1, facecolor=color)
    plt.xlabel('Steering Angle')
    plt.ylabel('Normalized Frequency')
    plt.title(title)
    plt.show()

```

```
steering_histogram(normal_labels, 'Normal driving strategy label
distribution', 'g')
steering_histogram(swerve_labels, 'Swerve driving strategy label
distribution', 'r')
```



**Рис. 16.12.** Распределение углов поворота в двух стратегиях вождения

Как мы говорили, стратегия вождения с поворотами дает гораздо более широкий диапазон углов, чем обычная стратегия вождения (рис. 16.12). Это разнообразие поможет нейронной сети правильно реагировать, если она когда-нибудь обнаружит, что автомобиль съезжает с дороги. Проблема дисбаланса в датасете решена, но лишь отчасти. У нас по-прежнему много нулей в обеих стратегиях вождения. Чтобы еще больше сбалансировать датасет, можно просто игнорировать часть нулевых значений во время обучения. Этот прием улучшит сбалансированность датасета, но значительно сократит общее количество точек данных. Важно помнить об этом при создании и обучении нейронной сети.

Прежде чем идти дальше, проверим, насколько хорошо наши данные подходят для обучения. Возьмем исходные данные из всех папок, разделим их на обучающий, проверочный и контрольный датасеты и сожмем их в файлы HDF5. Формат HDF5 позволяет получать доступ к данным по частям, без необходимости читать весь датасет в память. Благодаря этой его особенности он идеально подходит для задач глубокого обучения. Он также поддерживается в Keras. Следующий код потребует некоторого времени для выполнения. В итоге у нас будет три файла с наборами данных: *train.h5*, *eval.h5* и *test.h5*.

```
train_eval_test_split = [0.7, 0.2, 0.1]
full_path_raw_folders = [os.path.join(RAW_DATA_DIR, f) for f in DATA_FOLDERS]
Cooking.cook(full_path_raw_folders, COOKED_DATA_DIR, train_eval_test_split)
```

Каждый файл состоит из четырех частей:

#### *Image*

Массив NumPy, содержащий изображение.

#### *Previous\_state*

Массив NumPy с последним известным состоянием автомобиля. Это кортеж (рулевое управление, педаль газа, тормоз, скорость).

#### *Label*

Массив NumPy с углами поворота, которые требуется спрогнозировать (нормализованные в диапазон  $-1..1$ ).

#### *Metadata*

Массив NumPy с метаданными о файлах (из какой папки они получены и т. д.).

Теперь извлечем наблюдения из датасета и начнем обучать модель.

## Обучение модели автопилота

Все шаги, перечисленные в этом разделе, подробно описаны в блокноте *TrainModel.ipynb* (<https://oreil.ly/ESHVS>). Как обычно, начнем с импорта некоторых библиотек и определения путей:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Lambda,
Input, concatenate
from tensorflow.keras.optimizers import Adam, SGD, Adamax, Nadam
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,
CSVLogger
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow.keras.backend as K
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,
Dense,
from tensorflow.keras.layers import Lambda, Input, concatenate,
BatchNormalization

from keras_tqdm import TQDMNotebookCallback
import json
import os
import numpy as np
import pandas as pd
from Generator import DriveDataGenerator
```

```
from Cooking import checkAndCreateDir
import h5py
from PIL import Image, ImageDraw
import math
import matplotlib.pyplot as plt

# << Каталог с данными, подготовленными на предыдущем шаге >>
COOKED_DATA_DIR = 'data_cooked/'

# << Каталог для размещения модели >>
MODEL_OUTPUT_DIR = 'model'
```

Далее настроим датасеты:

```
train_dataset = h5py.File(os.path.join(COOKED_DATA_DIR, 'train.h5'), 'r')
eval_dataset = h5py.File(os.path.join(COOKED_DATA_DIR, 'eval.h5'), 'r')
test_dataset = h5py.File(os.path.join(COOKED_DATA_DIR, 'test.h5'), 'r')

num_train_examples = train_dataset['image'].shape[0]
num_eval_examples = eval_dataset['image'].shape[0]
num_test_examples = test_dataset['image'].shape[0]

batch_size=32
```

## Генератор данных

В предыдущих главах мы познакомились с идеей генераторов данных в Keras. Генератор данных выполняет итерации по датасету и читает данные с диска порциями. Это позволяет равномерно загрузить работой и CPU, и GPU и тем самым увеличить пропускную способность пайплайна обучения. Для реализации идей из предыдущего раздела мы создали свой генератор для Keras под названием `DriveDataGenerator`.

Вспомним некоторые из наблюдений, сделанных в предыдущем разделе.

- Модель должна фокусироваться только на ROI в пределах изображений.
- Можно провести аугментацию датасета, зеркально отражая изображения по горизонтали и меняя знак угла поворота.
- Дополнительно в аугментацию данных можно добавить случайное изменение яркости изображений. Так мы сымитируем различные условия освещения, что поможет сделать модель более надежной.
- Можно выборочно отбросить некоторую часть точек данных с углом поворота, равным нулю, чтобы сделать обучающий датасет более сбалансированным.
- После балансировки количество точек данных значительно сократится.

Посмотрим, как в `DriveDataGenerator` реализованы первые четыре пункта из этого списка. Последний пункт мы рассмотрим, когда начнем проектировать нейронную сеть.

ROI легко выделить простой обрезкой изображения. Список [76,135,0,255] в следующем блоке кода — это координаты  $[x1, x2, y1, y2]$  углов прямоугольника, представляющего область интереса. Генератор вырезает этот прямоугольник из каждого изображения. Изменить границы ROI можно с помощью параметра `roi`.

Отражение по горизонтали реализуется довольно просто. При создании пакетов, если параметру `horizontal_flip` присвоено значение `True`, случайные изображения переворачиваются по оси Y, а соответствующие им значения угла поворота меняются на противоположные.

Случайным изменением яркости управляет параметр `brighten_range`. При значении `0.4` в этом параметре яркость изображений в любом заданном пакете будет случайным образом изменяться в пределах 40%. Не рекомендуем использовать значения выше `0.4`. Для вычисления нового значения яркости изображение преобразуется из формата RGB в формат HSV, значения «V» масштабируются, и затем изображение снова преобразуется в формат RGB.

Балансировкой датасета путем отбрасывания наблюдений с нулевыми углами поворота управляет параметр `zero_drop_percentage`. При значении `0.9` в этом параметре будет удалено 90 % точек данных с нулевой меткой (нулевым углом поворота).

Инициализируем наш генератор следующими параметрами:

```
data_generator = DriveDataGenerator(rescale=1./255., horizontal_flip=True,
                                     brighten_range=0.4)

train_generator = data_generator.flow\
    (train_dataset['image'], train_dataset['previous_state'],
     train_dataset['label'], batch_size=batch_size, zero_drop_percentage=0.95,
     roi=[76,135,0,255])
eval_generator = data_generator.flow\
    (eval_dataset['image'], eval_dataset['previous_state'],
     eval_dataset['label'], batch_size=batch_size, zero_drop_percentage=0.95,
     roi=[76,135,0,255])
```

Визуализируем некоторые образцы данных, нарисовав угол поворота на соответствующих изображениях:

```
def draw_image_with_label(img, label, prediction=None):
    theta = label * 0.69 # Диапазон углов поворота для автомобиля
    # составляет +- 40 градусов -> 0.69 радиан
    line_length = 50
    line_thickness = 3
    label_line_color = (255, 0, 0)
    prediction_line_color = (0, 255, 255)
    pil_image = image.array_to_img(img, K.image_data_format(), scale=True)
    print('Actual Steering Angle = {}'.format(label))
    draw_image = pil_image.copy()
    image_draw = ImageDraw.Draw(draw_image)
    first_point = (int(img.shape[1]/2), img.shape[0])
```

```

second_point = (int((img.shape[1]/2) + (line_length * math.sin(theta))),  

                int(img.shape[0] - (line_length * math.cos(theta))))  

image_draw.line([first_point, second_point], fill=label_line_color,  

               width=line_thickness)  

if (prediction is not None):  

    print('Predicted Steering Angle = {0}'.format(prediction))  

    print('L1 Error: {0}'.format(abs(prediction-label)))  

    theta = prediction * 0.69  

    second_point = (int((img.shape[1]/2) + ((line_length/2) *  

                           math.sin(theta))), int(img.shape[0] - ((line_length/2) *  

                           math.cos(theta))))  

    image_draw.line([first_point, second_point], fill=prediction_line_color,  

                   width=line_thickness * 3)  

del image_draw  

plt.imshow(draw_image)  

plt.show()  

[sample_batch_train_data, sample_batch_test_data] = next(train_generator)  

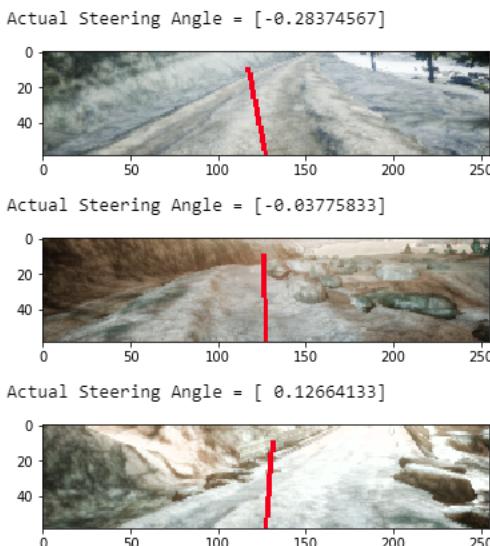
for i in range(0, 3, 1):  

    draw_image_with_label(sample_batch_train_data[0][i],  

                          sample_batch_test_data[i])

```

В результате получатся изображения, как на рис. 16.13. Обратите внимание, что теперь при обучении модель увидит только области интереса, и ей не будет мешать другая нерелевантная информация на исходных изображениях. Красная линия показывает угол поворота на местности. Этот угол был выбран водителем, когда изображение было снято камерой во время сбора данных.



**Рис. 16.13.** Углы поворота на изображениях

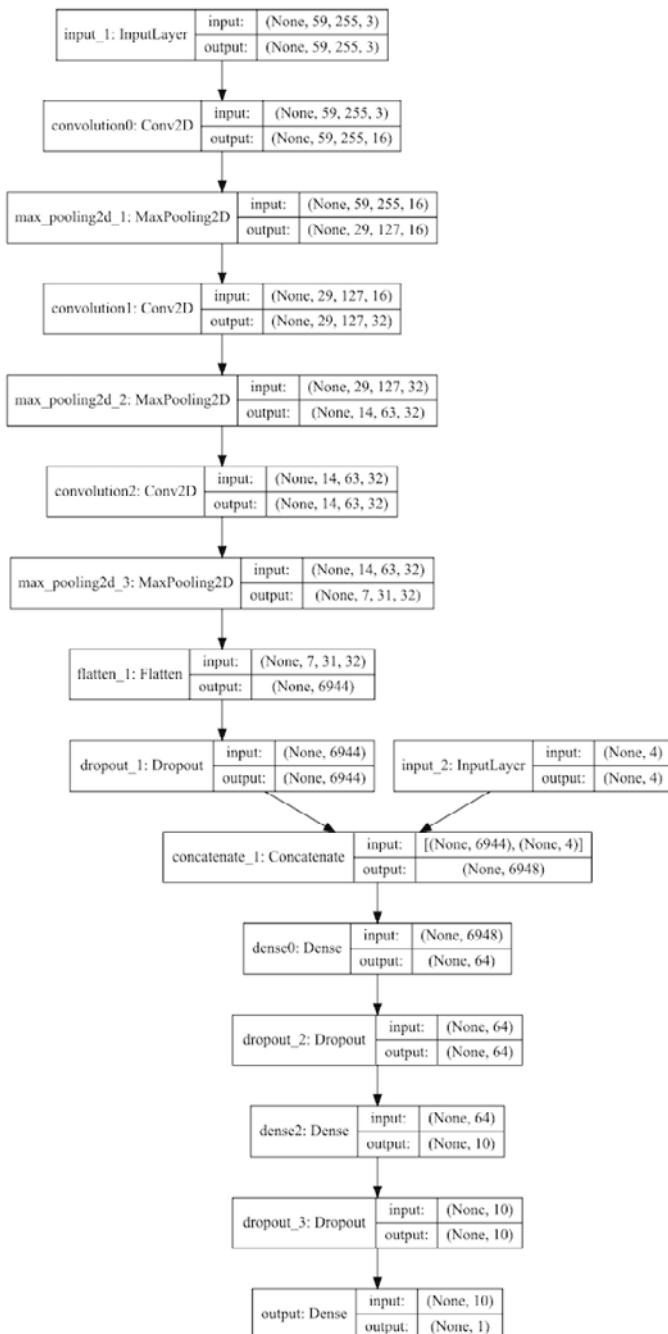
## Определение модели

Теперь приступим к определению архитектуры нейронной сети. На этом этапе мы должны принять во внимание проблему уменьшения объема данных после удаления нулей. Из-за этого мы не можем создать слишком глубокую сеть. Поскольку сеть будет обучаться на изображениях, для извлечения признаков понадобится несколько пар операций свертки/объединения по максимальному значению (convolutional/maxpooling).

Но одних только изображений может оказаться недостаточно, чтобы обеспечить сходимость модели. Использование для обучения одних только изображений также не согласуется с тем, как принимаются решения в реальном мире. Когда мы едем по дороге, то учитываем не только окружающую обстановку, но и скорость движения, поворот руля и состояние педалей газа и тормоза. Входные данные от камер, лидаров, радаров и т. д., поступающие в нейронную сеть, — это лишь часть той информации, которую использует водитель при принятии решений в реальном мире. Изображение, которое видит наша нейронная сеть, могло быть получено из неподвижного автомобиля или автомобиля, движущегося со скоростью 100 километров в час; сеть не имеет информации о скорости. Поворот руля вправо на два градуса при движении со скоростью 10 и 100 километров в час приведет к очень разным результатам. Проще говоря, модель, пытающаяся предсказать углы поворота, не должна полагаться только на сенсорную информацию. Ей нужна информация о текущем состоянии автомобиля. К счастью, у нас есть эта информация.

Мы отметили, что наши датасеты состоят из четырех частей. Для каждого изображения, кроме метки с величиной угла поворота и метаданных, имеется также последнее известное состояние автомобиля. Эти сведения хранятся в виде кортежа (угол поворота руля, положение педали газа, тормоз, скорость), и мы будем подавать их на вход нейронной сети вместе с изображениями. Обратите внимание, что это не противоречит нашему требованию к простоте, потому что мы все еще используем единственную камеру в качестве единственного внешнего датчика.

Теперь взгляните на нейронную сеть, которую мы будем использовать для решения этой задачи. Она показана на рис. 16.14. Мы используем три сверточных слоя с 16, 32 и 32 фильтрами соответственно и сверточным окном (3, 3). Мы объединяем признаки, выделенные из изображения (вывод сверточных слоев), с входным слоем, передающим предыдущее состояние автомобиля. Затем комбинированный набор признаков проходит через два полно связанных слоя с 64 и 10 скрытыми нейронами соответственно. В сети используется функция активации ReLU. Обратите внимание, что в отличие от задач классификации из предыдущих глав, последний слой нашей сети — это отдельный нейрон без активации. Причина в том, что решаемая нами задача относится к классу задач регрессии. Результатом работы будет угол поворота, число с плавающей запятой, а не дискретный набор классов.



**Рис. 16.14.** Архитектура сети

Теперь реализуем нашу сеть. Проверить ее архитектуру можно с помощью метода `model.summary()`:

```
image_input_shape = sample_batch_train_data[0].shape[1:]
state_input_shape = sample_batch_train_data[1].shape[1:]
activation = 'relu'

# Создать стопку сверточных слоев
pic_input = Input(shape=image_input_shape)

img_stack = Conv2D(16, (3, 3), name="convolution0", padding='same',
activation=activation)(pic_input)
img_stack = MaxPooling2D(pool_size=(2,2))(img_stack)
img_stack = Conv2D(32, (3, 3), activation=activation, padding='same',
name='convolution1')(img_stack)
img_stack = MaxPooling2D(pool_size=(2, 2))(img_stack)
img_stack = Conv2D(32, (3, 3), activation=activation, padding='same',
name='convolution2')(img_stack)
img_stack = MaxPooling2D(pool_size=(2, 2))(img_stack)
img_stack = Flatten()(img_stack)
img_stack = Dropout(0.2)(img_stack)

# Добавить ввод последнего состояния
state_input = Input(shape=state_input_shape)
merged = concatenate([img_stack, state_input])

# Добавить несколько полносвязных слоев в конец модели
merged = Dense(64, activation=activation, name='dense0')(merged)
merged = Dropout(0.2)(merged)
merged = Dense(10, activation=activation, name='dense2')(merged)
merged = Dropout(0.2)(merged)
merged = Dense(1, name='output')(merged)

adam = Nadam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model = Model(inputs=[pic_input, state_input], outputs=merged)
model.compile(optimizer=adam, loss='mse')
```

## Обратные вызовы

Одна из приятных особенностей Keras — возможность объявлять *обратные вызовы*. Обратные вызовы — это функции, которые вызываются после каждой эпохи обучения и помогают получить представление о том, как протекает обучение и до определенной степени контролировать гиперпараметры. С их помощью можно также проверять условия и выполнять определенные действия во время обучения: например, преждевременную остановку обучения, если потери перестают уменьшаться. В нашем эксперименте используем несколько обратных вызовов.

### ReduceLROnPlateau

Если модель близка к минимуму, а скорость обучения слишком высока, то модель будет кружить вокруг этого минимума, но так и не достигнет его. Этот

обратный вызов позволяет снизить скорость обучения, когда потери на этапе проверки выйдут на плато и перестанут уменьшаться, чтобы попытаться достичь оптимальной точки.

### **CSVLogger**

Этот обратный вызов дает возможность записать выходные данные модели после каждой эпохи обучения в файл CSV, чтобы изучить прогресс обучения без использования консоли.

### **ModelCheckpoint**

Желательно, чтобы модель имела как можно меньшие потери на проверочном наборе. Этот обратный вызов будет сохранять модель всякий раз, когда потери на этапе проверки будут уменьшаться.

### **EarlyStopping**

Когда потери на этапе проверки перестанут уменьшаться, лучше прервать обучение, иначе есть риск переобучить модель. Этот обратный вызов определит момент, когда потери на этапе проверки перестанут уменьшаться, и остановит обучение.

Продолжим и реализуем эти обратные вызовы:

```
plateau_callback = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
                                      min_lr=0.0001, verbose=1)

checkpoint_filepath = os.path.join(MODEL_OUTPUT_DIR, 'models', '{0}_model.{1}'
                                   -{2}.h5'.format('model', '{epoch:02d}', '{val_loss:.7f}'))
checkAndCreateDir(checkpoint_filepath)
checkpoint_callback = ModelCheckpoint(checkpoint_filepath, save_best_only=True,
                                       verbose=1)

csv_callback = CSVLogger(os.path.join(MODEL_OUTPUT_DIR, 'training_log.csv'))

early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10,
                                         verbose=1)

nbcallback = TQDMNotebookCallback()
setattr(nbcallback, 'on_train_batch_begin', lambda x,y: None)
setattr(nbcallback, 'on_train_batch_end', lambda x,y: None)
setattr(nbcallback, 'on_test_begin', lambda x: None)
setattr(nbcallback, 'on_test_end', lambda x: None)
setattr(nbcallback, 'on_test_batch_begin', lambda x,y: None)
setattr(nbcallback, 'on_test_batch_end', lambda x,y: None)

callbacks=[plateau_callback, csv_callback, checkpoint_callback,
          early_stopping_callback, nbcallback]
```

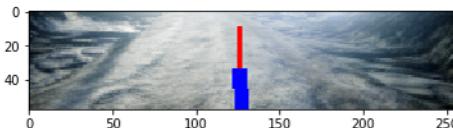
Теперь приступим к обучению. Это займет некоторое время, так что пока сделайте перерыв и посмотрите кино. Обучение должно завершиться с уровнем потери на этапе проверки, составляющим примерно 0,0003:

```

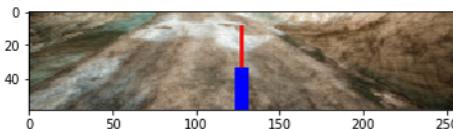
history = model.fit_generator(train_generator,
    steps_per_epoch=num_train_examples // batch_size, epochs=500,
    callbacks=callbacks, validation_data=eval_generator,
    validation_steps=num_eval_examples // batch_size, verbose=2)

Epoch 1/500
Epoch 00001: val_loss improved from inf to 0.02338, saving model to
model\models\model_model.01-0.0233783.h5
- 442s - loss: 0.0225 - val_loss: 0.0234
Epoch 2/500
Epoch 00002: val_loss improved from 0.02338 to 0.00859, saving model to
model\models\model_model.02-0.0085879.h5
- 37s - loss: 0.0184 - val_loss: 0.0086
Epoch 3/500
Epoch 00003: val_loss improved from 0.00859 to 0.00188, saving model to
model\models\model_model.03-0.0018831.h5
- 38s - loss: 0.0064 - val_loss: 0.0019
.....
```

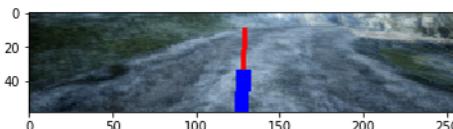
Actual Steering Angle = [-0.043457]  
Predicted Steering Angle = [-0.04056546]  
L1 Error: [0.00289154]



Actual Steering Angle = [-0.01323133]  
Predicted Steering Angle = [-0.02355941]  
L1 Error: [0.01032808]



Actual Steering Angle = [0.06888733]  
Predicted Steering Angle = [0.06637219]  
L1 Error: [0.00251514]



**Рис. 16.15.** Отображение фактических и прогнозируемых углов поворота на изображениях

Теперь модель обучена и готова к работе. Но прежде чем посмотреть ее в деле, давайте быстро проверим ее работоспособность и получим прогнозы для некоторых изображений:

```
[sample_batch_train_data, sample_batch_test_data] = next(train_generator)
predictions = model.predict([sample_batch_train_data[0],
                             sample_batch_train_data[1]])
for i in range(0, 3, 1):
    draw_image_with_label(sample_batch_train_data[0][i],
                          sample_batch_test_data[i], predictions[i])
```

Вы увидите результат, как на рис. 16.15. Толстая линия на этом рисунке — это прогнозируемый результат, а тонкая линия — фактический угол поворота, указанный в метке. Похоже, наши прогнозы довольно точны (фактические и прогнозируемые значения можно увидеть над изображениями). Пришло время развернуть модель и посмотреть на ее работу.

## Развертывание модели автопилота

Все шаги, перечисленные в этом разделе, подробно описаны в блокноте *TestModel.ipynb* (<https://oreil.ly/5saDl>). Теперь, когда у нас есть обученная модель, запустим симулятор и попробуем использовать наш автопилот.

Начнем с импорта некоторых библиотек и определения путей:

```
from tensorflow.keras.models import load_model
import sys
import numpy as np
import glob
import os

if ('../../PythonClient/' not in sys.path):
    sys.path.insert(0, '../../PythonClient/')
from AirSimClient import *

# << Указать путь к модели >>
# Если оставить значение None, то будет использоваться модель с наименьшими
# потерями на этапе проверки, выявленными в процессе обучения
MODEL_PATH = None

if (MODEL_PATH == None):
    models = glob.glob('model/models/*.h5')
    best_model = max(models, key=os.path.getctime)
    MODEL_PATH = best_model

print('Using model {} for testing.'.format(MODEL_PATH))
```

Загрузите модель и подключитесь к ландшафтной карте в AirSim. Чтобы запустить симулятор, на компьютере с Windows откройте PowerShell, перейдите в каталог, куда распаковали архив с пакетом симулятора, и запустите следующую команду:

```
. \AD_Cookbook_Start_AirSim.ps1 landscape
```

Вернитесь в блокнот Jupyter и выполните следующий код, чтобы подключить модель к клиенту AirSim. Перед началом убедитесь, что симулятор запущен:

```
model = load_model(MODEL_PATH)

client = CarClient()
client.confirmConnection()
client.enableApiControl(True)
car_controls = CarControls()
print('Connection established!')
```

После установки соединения определите начальное состояние автомобиля, а также инициализируйте некоторые буфера, используемые для хранения выходных данных модели:

```
car_controls.steering = 0
car_controls.throttle = 0
car_controls.brake = 0

image_buf = np.zeros((1, 59, 255, 3))
state_buf = np.zeros((1,4))
```

Следующий шаг — настройка модели на получение входных изображений RGB из симулятора. Для этого определим вспомогательную функцию:

```
def get_image():
    image_response = client.simGetImages([ImageRequest(0, AirSimImageType.Scene,
                                                       False, False)][0]
    image1d = np.fromstring(image_response.image_data_uint8, dtype=np.uint8)
    image_rgba = image1d.reshape(image_response.height, image_response.width, 4)

    return image_rgba[76:135,0:255,0:3].astype(float)
```

И в заключение организуем бесконечный цикл, в котором будем получать изображения из симулятора вместе с текущим состоянием автомобиля, прогнозировать угол поворота и возвращать его обратно в симулятор. Поскольку модель предсказывает только углы поворота, организуем подачу управляющего сигнала для поддержания скорости. Настроим его так, чтобы машина двигалась с постоянной скоростью 5 м/с:

```
while (True):
    car_state = client.getCarState()

    if (car_state.speed < 5):
        car_controls.throttle = 1.0
    else:
        car_controls.throttle = 0.0

    image_buf[0] = get_image()
```

```
state_buf[0] = np.array([car_controls.steering, car_controls.throttle,
                      car_controls.brake, car_state.speed])
model_output = model.predict([image_buf, state_buf])
car_controls.steering = round(0.5 * float(model_output[0][0]), 2)

print('Sending steering = {0}, throttle = {1}'.format(car_controls.steering,
                                                      car_controls.throttle))

client.setCarControls(car_controls)
```

Вы увидите примерно такой вывод:

```
Sending steering = 0.03, throttle = 1.0
Sending steering = -0.1, throttle = 1.0
Sending steering = -0.12, throttle = 1.0
Sending steering = -0.13, throttle = 1.0
Sending steering = -0.13, throttle = 1.0
Sending steering = -0.13, throttle = 1.0
Sending steering = -0.14, throttle = 1.0
Sending steering = -0.15, throttle = 1.0
```

Мы сделали это! Автомобиль уверенно едет по дороге, стараясь придерживаться правой стороны, аккуратно преодолевая все крутые повороты и моменты, когда он может съехать с дороги. Мы успешно справились с задачей обучения нашей первой модели автопилота!



**Рис. 16.16.** Обученная модель управляет автомобилем

Прежде чем закончить, отметим несколько важных моментов. Во-первых, обратите внимание, что автомобиль движется не идеально плавно. Это обусловлено тем, что мы решаем задачу регрессии и определяем угол поворота для каждого кадра, видимого автомобилем. Исправить эту проблему можно усреднением

прогнозов методом скользящего окна по буферу последовательных изображений. Еще одна идея — преобразовать задачу регрессии в задачу классификации. Для этого можно определить диапазоны углов поворота (... , -0,1, -0,05, 0, 0,05, 0,1, ...), распределить метки по сегментам и прогнозировать сегмент для каждого изображения.

Если понаблюдать за работой модели какое-то время (чуть больше пяти минут), то мы увидим, что в итоге машина случайно съезжает с дороги и разбивается. Это происходит на участке с крутым уклоном. Помните наше последнее требование при подготовке задачи? Изменение уклона требует манипулирования педалью газа и тормозами. Поскольку наша модель управляет только углом поворота, она плохо справляется с преодолением крутых уклонов.

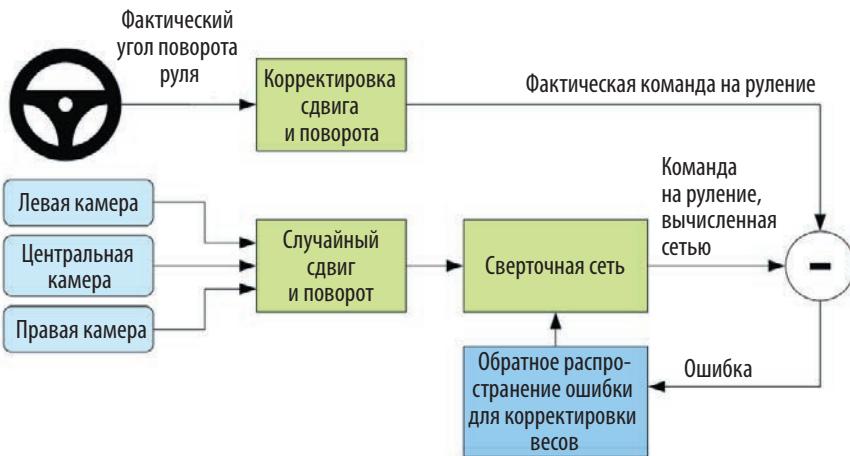
## Что изучать дальше

Мы обучили упрощенную модель и выяснили, что для автопилота она не идеальна. Но не отчаивайтесь. Не забывайте, что мы лишь слегка коснулись возможностей на стыке сквозного обучения и беспилотных автомобилей. То, что мы смогли на маленьком датасете обучить автопилот ездить почти идеально, дает право гордиться собой!

### Сквозное глубокое обучение для беспилотных автомобилей в реальном мире

В начале 2016 года сквозное глубокое обучение стало особенно популярно в робототехнике. Гипотеза заключалась в том, что при наличии достаточного количества визуальных данных глубокая нейронная сеть сможет изучить задачу от начала до конца, отображая входные данные в непосредственные действия. Следовательно, должна быть возможность обучать роботов выполнять сложные задачи, используя только изображения с камер.

Одно из самых популярных решений в контексте автономного вождения реализовано в NVIDIA в виде системы DAVE-2. В отличие от традиционных систем, в DAVE-2 использовалась одна глубокая нейронная сеть для прогнозирования управляющих сигналов по пикселям изображения (рис. 16.17). В NVIDIA показали, что небольшого количества обучающих данных, охватывающих менее чем сто часов вождения, достаточно, чтобы научить автопилот водить автомобиль в различных условиях, на шоссе и проселочных дорогах, в солнечную и облачную погоду, в дождь и снег. Более подробную информацию об этом решении можно найти по адресу <https://devblogs.nvidia.com/deep-learning-self-driving-cars>.



**Рис. 16.17.** Архитектура системы NVIDIA DAVE-2 (с сайта <https://oreil.ly/jVwEv>)

Вот несколько идей, которые вы можете развить, опираясь на опыт, полученный выше. Реализовать их можно, используя среду, подготовленную в этой главе.

## Расширение датасета

Как правило, большой объем данных помогает повысить качество прогнозов модели. Теперь, когда у вас есть действующий симулятор, полезно поупражняться в расширении датасета и выполнить еще несколько прогонов сбора данных. Можно даже попытаться объединить данные из различных сред, доступных в AirSim, и посмотреть, как модель, обученная на этих данных, работает в разных средах.

В этой главе мы использовали только изображения RGB, полученные единственной камерой. Но AirSim обладает гораздо более широкими возможностями. Например, можно собрать изображения с картами глубин, с сегментированными объектами, с видом сверху и т. д. для каждой из доступных камер. Можно получить до 20 разных изображений (пять камер, работающих во всех четырех режимах) для каждого экземпляра. Помогут ли все эти дополнительные данные улучшить только что обученную модель?

## Обучение на последовательных данных

Сейчас наша модель вычисляет каждый прогноз, используя одно изображение и одно состояние автомобиля. Но мы в реальной жизни видим совсем не так. Наши действия всегда учитывают серию прошедших событий, которые при-

вели к текущему моменту. В нашем датасете есть информация о времени, когда было получено то или иное изображение, и можно использовать ее для создания последовательностей. Можно изменить модель, чтобы она делала прогнозы, используя предыдущие  $N$  изображений и состояний. Например, прогнозировала угол следующего поворота по последним 10 изображениям и 10 состояниям. (Подсказка: для этого может потребоваться рекуррентная нейронная сеть.)

## Обучение с подкреплением

В следующей главе мы познакомимся с обучением с подкреплением. После этого можно вернуться к руководству «Distributed Deep Reinforcement Learning for Autonomous Driving» (<https://oreil.ly/u1hoC>) из *Autonomous Driving Cookbook* (<https://oreil.ly/bTphH>) и использовать среду Neighborhood в AirSim, которая также входит в пакет, загруженный для этой главы. Мы увидим, как масштабировать глубокое обучение с подкреплением и сократить время обучения с недели до часа, применив перенос обучения и облачные технологии.

## Итоги

Эта глава показала применение глубокого обучения в сфере беспилотных автомобилей. Воспользовавшись навыками из предыдущих глав, с помощью Keras мы реализовали простой автопилот. Исследовав имеющиеся исходные данные, выяснили, как их подготовить для обучения высококачественной модели. И смогли добиться высоких результатов с очень небольшим датасетом. Мы развернули обученную модель, и автопилот показал себя в симуляторе. Согласитесь, в этом есть что-то волшебное.

## ГЛАВА 17

---

# Создание беспилотного автомобиля менее чем за час: обучение с подкреплением с помощью AWS DeepRacer

Написана Сунилом Маллья

Если вы следите за новостями в сфере технологий, то наверняка заметили споры о том, когда же компьютеры захватят мир. Такие разговоры вызывают улыбку, но чем же они обусловлены? По большей части появлением новостей о победах компьютеров над людьми в задачах, где нужно принимать решения. Они научились выигрывать у людей в шахматы и го (2016), достигли значительных успехов в Atari (2013) и, наконец, в 2017 году выиграли у команд из людей в Dota 2. Самое поразительное во всем этом, что «боты» научились выигрывать, играя друг против друга и закрепляя найденные успешные стратегии.

Такое взаимообучение, по сути, ничем не отличается от того, как люди учат своих питомцев. Дрессировщик подкрепляет правильное поведение собаки, поощряя ее угощением, похвалой и лаской, а любое нежелательное поведение пресекает окриком «фу» или «нельзя». Идея подкрепления хорошего поведения и пресечения плохого и составляет основу *обучения с подкреплением* (*reinforcement learning*).

Компьютерные игры, как и игры в целом, требуют принятия последовательности решений, поэтому традиционные методы обучения с учителем здесь не подходят — они часто фокусируются на принятии одного решения (например, на картинке кошка или собака?). Специалисты, занимающиеся обучением с подкреплением, часто шутят, что целыми днями просто играют в видеоигры (и это правда!). Обучение с подкреплением используется во всех сферах: для торгов-

ли акциями, управления отоплением и кондиционирования больших зданий и вычислительных центров, проведения торгов в режиме реального времени, оптимизации качества потокового видео и даже оптимизации химических реакций в лабораториях. Настоятельно рекомендуем использовать обучение с подкреплением для принятия последовательностей решений и задач оптимизации. В этой главе мы сосредоточимся на исследовании этой парадигмы машинного обучения и ее применении в реальной задаче: создании модели беспилотного автомобиля в масштабе 1:18 менее чем за час.

## Краткое введение в обучение с подкреплением

Последние несколько лет, после того как были побиты ранее установленные рекорды в компьютерных играх, обучение с подкреплением переживает ренессанс. Расцвет теории обучения с подкреплением пришелся на 1990-е годы, но она не распространилась на крупные промышленные системы из-за высоких требований к вычислительным ресурсам и трудностей в обучении таких систем. Традиционно обучение с подкреплением считалось сложной вычислительной задачей, тогда как нейронные сети относятся к задачам, работающим с большими объемами данных. Но развитие глубоких нейронных сетей пошло на пользу обучению с подкреплением. Сейчас нейронные сети начали широко использоваться в моделях обучения с подкреплением, что дало начало *глубокому обучению с подкреплением*. В этой главе мы используем термины «обучение с подкреплением» и «глубокое обучение с подкреплением» как синонимы, но почти во всех случаях, если явно не указано иное, мы подразумеваем глубокое обучение.

Несмотря на недавние достижения, ландшафт обучения с подкреплением остается не очень удобным для разработчиков. Интерфейсы для обучения моделей постепенно становятся проще, но эта тенденция пока не проникла в сферу обучения с подкреплением. Еще одна сложность — значительные требования к вычислительным ресурсам и ко времени сходимости модели (до момента, когда обучение можно считать завершенным). Обучение модели может занимать дни, а то и недели. Кроме того, даже если предположить, что мы обладаем ангельским терпением, прекрасно знаем нейронные сети и имеем большой бюджет, остается еще одна проблема: нехватка образовательных ресурсов, посвященных обучению с подкреплением. Большинство из них ориентировано на опытных специалистов по обработке данных, и разработчики не всегда понимают суть. А как же тот беспилотный автомобиль в масштабе 1:18, о котором мы говорили? Это AWS DeepRacer (рис. 17.1). Одним из главных мотивов создания AWS DeepRacer было стремление сделать обучение с подкреплением доступным для разработчиков. Этот проект основан на Amazon SageMaker — универсальной

платформе обучения с подкреплением. И давайте честно: кто из нас не любит беспилотные автомобили?



**Рис. 17.1.** Модель беспилотного автомобиля AWS DeepRacer в масштабе 1:18

## Почему для изучения обучения с подкреплением выбран беспилотный автомобиль?

В последние годы в технологию беспилотного управления было вложено много денег, и в ней достигнут значительный успех. Стали очень популярны гонки самодельных беспилотных и радиоуправляемых моделей автомобилей. Такой беспрецедентный энтузиазм разработчиков, создающих масштабные модели с помощью проверенного в реальных сценариях оборудования, буквально вынудил нас использовать автомобиль, как пример для изучения обучения с подкреплением. Несмотря на возможность создания автопилотов с использованием традиционного компьютерного зрения или обучения с учителем (копирование поведения), мы считаем, что у обучения с подкреплением есть преимущества.

В табл. 17.1 перечислены некоторые популярные комплекты для сборки моделей беспилотных автомобилей, доступные разработчикам, и технологии, лежащие в их основе. Одно из ключевых преимуществ обучения с подкреплением — возможность обучать модели исключительно на тренажерах. Но системы обучения с подкреплением имеют и свои проблемы, самая главная — проблема переноса симуляции в реальность (*sim2real*). Разворачивание в реальных условиях моделей, полностью обученных на симуляторах, всегда было сложно. В DeepRacer для этого есть несколько простых, но эффективных решений, которые мы обсудим далее в этой главе. За первые шесть месяцев после выпуска DeepRacer в ноябре 2018 года около 9000 разработчиков обучили свои модели в симуляторах и успешно протестировали их на реальных трассах.

**Таблица 17.1.** Ландшафт технологий автономных автомобилей

Платформа	Оборудование	Сборка	Технология	Стоимость	Пример
AWS DeepRacer	Intel Atom с GPU мощностью 100 Гфлопс	Предварительная сборка	Обучение с подкреплением	339 долларов США	
OpenMV	OpenMV H7	Требует сборки (собирается за пару часов)	Традиционное компьютерное зрение	90 долларов США	
Duckietown	Raspberry Pi	Предварительная сборка	Обучение с подкреплением, копирование поведения	279–350 долларов США	
DonkeyCar	Raspberry Pi	Требует сборки (собирается за два-три часа)	Копирование поведения	250 долларов США	
NVIDIA JetRacer	Jetson Nano	Требует сборки (собирается за три–пять часов)	Обучение с учителем	~400 долларов США	

## От создателя

Будущие роботизированные автомобили, создаваемые DIY-энтузиастами, будут:

### *Быстрее*

Мы стремимся к тому, чтобы наши машины неизменно побеждали людей не только в скорости, но и в выборе наступательной и оборонительной тактики в гонках. Чтобы наши машины были не персонажами в игре Mario Kart, а участвовали в невероятно сложных гонках в реальности. ;-)

### *Проще*

По аналогии с отличной игрой, в которую «легко играть, но трудно освоить на уровне мастера», мы хотим, чтобы наши машины хорошо ездили изначально, но для победы требовали использования программных ухищрений.

### *Дешевле*

Стоимость и уровень владения машинным обучением для входа в гоночные лиги DIY-моделей значительно снижается.

Крис Андерсон (Chris Anderson),  
CEO 3DR и основатель DIY Robocars

# Практика глубокого обучения с подкреплением с DeepRacer

Теперь перейдем к самой захватывающей части этой главы: созданию первой модели для участия в гонках беспилотников. Но перед этим составим шпаргалку, которая поможет не заблудиться в терминологии обучения с подкреплением.

### *Цель*

Пройти круг по треку без отклонения.

### *Вход*

Человек, управляющий машиной, наблюдает за окружающей обстановкой и использует навыки вождения для принятия решений и выбора направления. Система DeepRacer тоже основана на компьютерном зрении, поэтому в качестве входных данных мы будем использовать изображение, поступающее с единственной камеры. В частности, будем использовать черно-белые изображения размером  $120 \times 160$ .

### *Выход (действия)*

В реальном мире мы ведем машину, управляя педалью газа, тормозами и рулем. Система DeepRacer, установленная на крыше радиоуправляемого автомобиля, выдает два управляющих сигнала: положение педали газа и поворот руля, каждый из которых является традиционным сигналом с широтно-импульсной модуляцией (ШИМ). Отображение управления в сигналы ШИМ кажется неинтуитивными, поэтому мы дискретизируем управляющие воздействия, которые может генерировать автопилот. Вспомните старые добрые гонки, в которые мы играли на компьютере. Для управления автомобилем там использовались клавиши со стрелками — влево, вправо и вверх. Точно так же можно определить фиксированный набор действий, которые должен генерировать автопилот, но с более точным управлением педалью газа и рулем. После обучения модель будет принимать решения о том, какое действие предпринять, чтобы успешно пройти трассу. У нас будет возможность определить эти действия при создании модели.



Сервопривод в радиоуправляемых машинках обычно управляется ШИМ-сигналом, который представляет собой серию импульсов различной ширины. Чтобы сервопривод оказался в нужном положении, ему отправляется импульсный сигнал заданной ширины. Импульсы определяются минимальной шириной импульса, максимальной шириной импульса и частотой повторения.

### *Агент*

Система, которая учится и принимает решения. В нашем случае это автопилот, который учится ориентироваться в окружающей обстановке (на трассе).

### *Среда*

Среда, с которой агент учится взаимодействовать, генерируя управляющие воздействия. В DeepRacer среда включает трек, по которому агент может двигаться и который не должен покидать. Агент исследует среду и собирает данные для обучения базовой нейронной сети глубокого обучения с подкреплением.

### *Состояние ( $s$ , state)*

Представление местоположения агента в среде. Это моментальный снимок агента. В DeepRacer для представления состояния используется изображение.

### *Действия ( $a$ , action)*

Набор решений, которые может принимать агент.

### *Шаг*

Дискретный переход из одного состояния в другое.

### Эпизод

Определяет попытку автопилота достичь своей цели, то есть пройти круг по трассе. Иначе говоря, эпизод — это последовательность шагов, или опыта. Разные эпизоды могут иметь разную продолжительность.

### Вознаграждение ( $r$ , reward)

Оценка действия, предпринятого агентом при заданном состоянии.

### Политика ( $\pi$ , policy)

Стратегия или функция принятия решений; отображение состояния в действие.

### Функция оценки ( $V$ , Value function)

Отображение состояния в оценку, представляющую собой ожидаемое вознаграждение за действие, предпринятое в данном состоянии.

### Буфер повторения опыта

Временный буфер, в котором хранится опыт в форме кортежа ( $s, a, r, s'$ ), где « $s$ » обозначает изображение (или состояние) с камеры, « $a$ » — действие, предпринимаемое автопилотом, « $r$ » — ожидаемое вознаграждение за выбранное действие и « $s'$  — новое наблюдение (или новое состояние) после выполнения действия.

### Функция вознаграждения

Любая система обучения с подкреплением нуждается в руководстве, то есть в чем-то, что подсказывает, какое действие хорошее или плохое в данной ситуации. Функция вознаграждения играет роль такого руководства, оценивая действия, предпринятые автопилотом, и присуждая вознаграждение (скалярное значение), пропорциональное целесообразности этого действия в данной ситуации. Например, при повороте налево действие «поворот налево» будет считаться оптимальным (вознаграждение = 1 по шкале от 0 до 1), но действие «поворот направо» будет считаться плохим (вознаграждение = 0). Система обучения с подкреплением собирает эти оценки, возвращаемые функцией вознаграждения, и обучает модель. Это самая важная часть в обучении автопилота, и мы уделим ей большое внимание.

Наконец, объединив все перечисленное в систему, получаем следующий схематический поток:

Вход (черно-белое изображение 120x160) → (модель обучения с подкреплением) →  
Выход (влево, вправо, прямо)

В AWS DeepRacer функция вознаграждения — важная часть процесса построения модели. Мы должны определить ее, чтобы обучить модель AWS DeepRacer.

В эпизоде агент взаимодействует с треком, чтобы определить оптимальный набор действий, которые нужно предпринять, и получить максимальное совокупное вознаграждение. Но ни один эпизод не дает достаточно большого объема данных для обучения агента. Поэтому сбор данных производится по множеству эпизодов. Периодически в конце каждого  $n$ -го эпизода мы запускаем очередную итерацию обучения модели с подкреплением. Чтобы получить наилучшую из возможных моделей, нужно выполнить множество итераций. Этот процесс подробно описывается в следующем разделе. После обучения агент выполняет пробную поездку по трассе, чтобы научиться выполнять оптимальные действия с учетом входного изображения. Оценка модели выполняется либо в симуляторе с виртуальным агентом, либо в реальном мире с физическим автомобилем AWS DeepRacer.

А теперь приступим к созданию нашей первой модели. Поскольку вход имеет фиксированную форму — одно изображение с камеры, сосредоточимся только на выходе (действиях) и функции вознаграждения. Чтобы начать обучение модели, выполним следующие шаги.

### Проблема переноса симуляции в реальность (Sim2Real)

Сбор данных для роботизированных устройств может быть чрезвычайно сложной и затратной по времени задачей. Поэтому обучение таких моделей в симуляторе дает большие преимущества. Но создание определенных сценариев из реального мира может быть непрактично, как, например, имитация столкновения с другим автомобилем или с человеком. Следовательно, в симуляторе могут быть проблемы — различия в восприятии и точность имитации (качество изображения, частота дискретизации и т. д.), физические различия (трение, масса, плотность и т. д.) и неправильное физическое моделирование (взаимодействия и столкновения между физическими объектами). Все это может привести к тому, что среда в симуляторе будет сильно отличаться от реального мира.

## Создание первой модели обучения с подкреплением

Для выполнения этого упражнения нужен аккаунт AWS. Войдите в консоль AWS, используя свои учетные данные, как показано на рис. 17.2.

Убедимся, что мы находимся в регионе Северная Вирджиния, так как сервис доступен только в этом регионе, а затем перейдем на страницу консоли DeepRacer: <https://console.aws.amazon.com/deepracer/home?region=us-east-1#getStarted>.

После выбора Reinforcement learning (Обучение с подкреплением) откроется страница модели. На этой странице отображается список всех созданных моделей и состояние каждой из них. Чтобы создать новую модель, начните с выбора Create model (Создать модель).

Account: [redacted]

User Name: jeff-console

Password: [redacted]

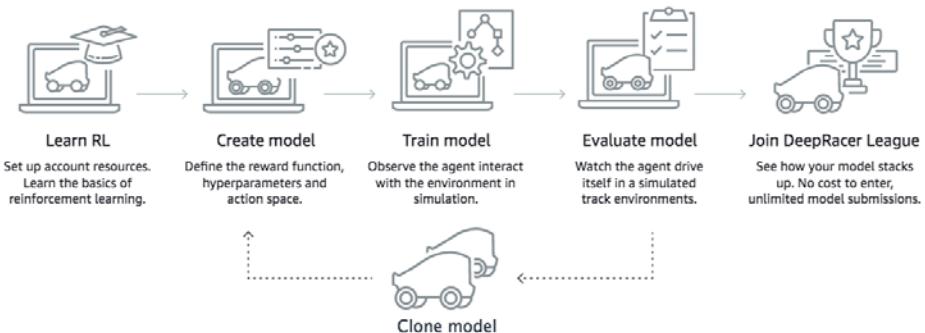
I have an MFA Token (more info)

MFA Code: [redacted]

**Sign In**

[Sign-in using root account credentials](#)

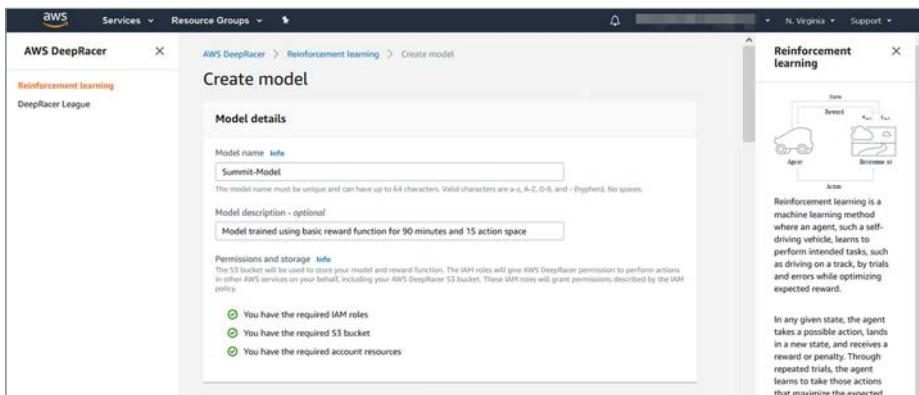
**Рис. 17.2.** Вход в консоль AWS



**Рис. 17.3.** Рабочий процесс для обучения модели AWS DeepRacer (Learn RL — познакомиться с основами обучения с подкреплением; Create model — создать модель; Train model — обучить модель; Evaluate model — оценить модель; Join DeepRacer League — присоединиться к лиге DeepRacer; Clone model — клонировать модель)

## Шаг 1: создание модели

Теперь создадим модель, которую можно использовать на беспилотном автомобиле AWS DeepRacer для движения по гоночной трассе. Но сначала нужно выбрать конкретную гоночную трассу, определить, какие действия будет выполнять модель, и функцию вознаграждения для стимулирования желаемого поведения, а также настроить гиперпараметры, используемые во время обучения.



**Рис. 17.4.** Создание модели в консоли AWS DeepRacer

## Шаг 2: настройка процесса обучения

Здесь мы выбираем среду обучения, настраиваем пространство действий, пишем функцию вознаграждения и настраиваем другие параметры обучения. И только после этого сможем приступить непосредственно к обучению.

### Выбор среды обучения

Обучение с подкреплением будет проводиться в симуляторе гоночных трасс, и мы должны выбрать конкретную трассу. Для развертывания среды обучения будем использовать облачную службу AWS RoboMaker, упрощающую создание роботизированных приложений.

Для обучения модели желательно выбирать трассу, наиболее похожую на ту, где будут проводиться состязания. По состоянию на июль 2019 года в AWS DeepRacer предлагалось семь трасс для обучения. Такая дополнительная настройка, как выбор трассы, не обязательна и не гарантирует получение хорошей модели, но увеличивает шансы получить модель, которая сможет показать высокий результат на гоночной трассе. Кроме того, если обучить модель на имитации прямой дороги, то маловероятно, что она научится поворачивать. Как и в случае обучения с учителем, когда маловероятно, что модель усвоит что-то, что не является частью обучающих данных, при обучении с подкреплением агент едва ли научится чему-то, чего нет в обучающей среде. Для первого упражнения выберем трек re:Invent 2018, как показано на рис. 17.5.

Теперь выберем алгоритм обучения. В настоящее время консоль AWS DeepRacer поддерживает только алгоритм проксимальной оптимизации политики (Proximal Policy Optimization, PPO). Со временем предполагается увеличить количество поддерживаемых алгоритмов, но на первом этапе был выбран алгоритм PPO

как обеспечивающий высокую скорость обучения и обладающий превосходным свойством сходимости. Обучение с подкреплением — итеративный процесс, потому что, во-первых, нужно определить функцию вознаграждения, которая охватывала бы сразу все важные действия агента в среде, и во-вторых, часто требуется настраивать гиперпараметры, чтобы получить более или менее удовлетворительные результаты обучения. И то и другое определяется экспериментально. Поэтому наиболее разумный подход — начать с простой функции вознаграждения. Так и поступим: выберем простую функцию вознаграждения, а затем будем постепенно совершенствовать ее. AWS DeepRacer упрощает этот процесс, позволяя клонировать обученную модель и улучшать функцию вознаграждения для учета ранее игнорировавшихся переменных или систематически корректировать гиперпараметры, пока результат не сойдется. Самый простой способ обнаружить это схождение — заглянуть в логи и посмотреть, доходит ли машина до финишной черты, то есть пройдена ли трасса целиком. Как вариант, можно наблюдать за поведением машины и воочию убедиться, что она достигает финиша.

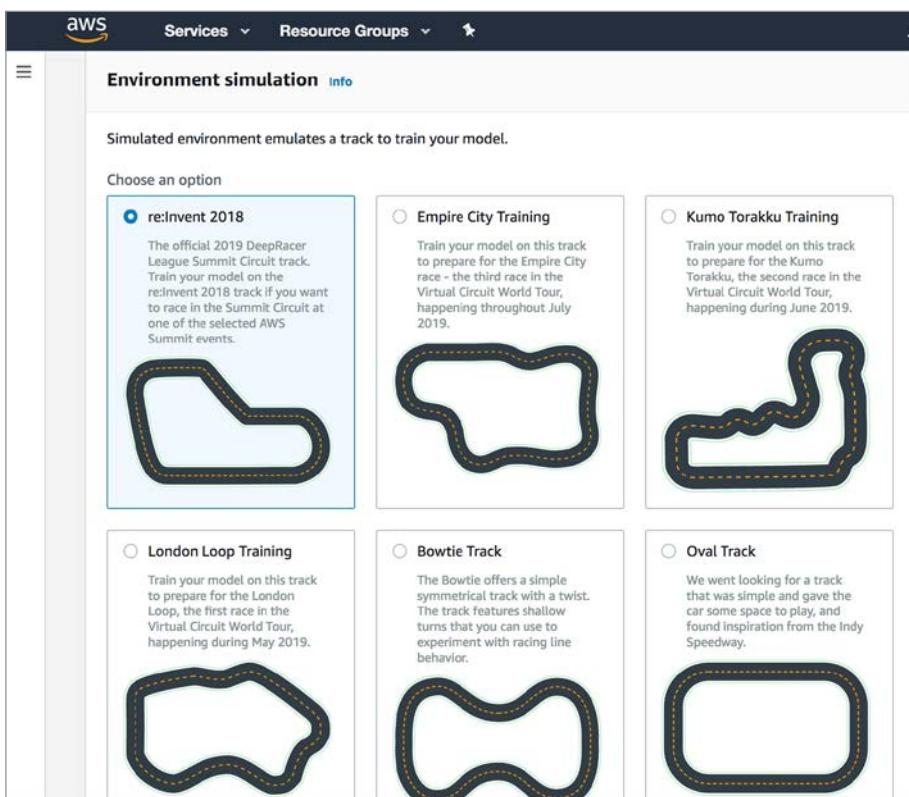


Рис. 17.5. Выбор трассы в консоли AWS DeepRacer

## Настройка пространства действий

Теперь настроим пространство действий, доступных модели для выбора во время и после обучения. Действие (выход) – это комбинация скорости и угла поворота. Сейчас в AWS DeepRacer используется дискретное пространство действий (фиксированный набор действий), отличное от непрерывного пространства действий (в котором угол поворота и скорость могут задаваться любыми действительными значениями). Это связано с тем, что так проще будет отобразить действия в значения на физическом автомобиле, но об этом мы поговорим позже, в разделе «Гонки на автомобиле AWS DeepRacer». Чтобы создать дискретное пространство действий, определим максимальную скорость, уровни скорости, максимальный угол поворота и уровни угла поворота, как показано на рис. 17.6.

Вот как настраиваются параметры пространства действий:

### *Максимальный угол поворота*

Это максимальный угол в градусах, на который могут поворачиваться передние колеса автомобиля влево и вправо. Передние колеса поворачиваются не более чем на определенный угол, поэтому выберем максимальный угол поворота равным 30 градусам.

### *Количество вариантов угла поворота*

Определяет количество фиксированных углов поворота между крайними положениями колес, повернутыми вправо и влево на максимальный угол. То есть для максимального угла поворота 30 градусов (+30 градусов влево и -30 градусов вправо) и при количестве вариантов, равном 5, в пространстве действий на выбор будут доступны следующие углы поворота, слева направо: 30 градусов, 15 градусов, 0 градусов, -15 градусов и -30 градусов, как показано на рис. 17.6. Углы поворота всегда симметричны относительно отметки 0 градусов.

### *Максимальная скорость*

Определяет максимальную скорость, которую автомобиль может развивать в симуляторе, измеряется в метрах в секунду (м/с).

### *Уровни скорости*

Определяет количество уровней скорости от максимальной (включительно) до нуля. То есть если максимальная скорость составляет 3 м/с, а количество уровней скорости равно 3, то в пространстве действий будут доступны для выбора скорости 1 м/с, 2 м/с и 3 м/с. Проще говоря, если 3 м/с разделить на 3, получится 1 м/с – шаг изменения скорости от 0 м/с до 3 м/с. Скорость 0 м/с не входит в пространство действий.

Исходя из сказанного выше, окончательное пространство будет включать 15 дискретных действий (три скорости на пять углов поворота), которые нужно

Servicess Resource Groups N. Virginia

### Action space [Info](#)

Action space defines the specific actions an agent can take in both the simulator and physical world. While a real vehicle can choose from a continuum of actions, AWS DeepRacer simplifies the agent's decision-making process by reducing that space to a set of discrete actions.

Configure this discrete action space by setting the range and granularity for speed and steering angle. The system automatically generates an action space according to that specification. Note that your model will take longer to train under a larger action space.

Maximum steering angle  
30 degrees  
Max values are between 1 and 30.

Steering angle granularity  
5

Maximum speed  
3 m/s  
Max values are between 0.8 and 8.

Speed granularity  
3

### Action list

Action number	Steering	Speed
0	-30 degrees	1 m/s
1	-30 degrees	2 m/s
2	-30 degrees	3 m/s
3	-15 degrees	1 m/s
4	-15 degrees	2 m/s
5	-15 degrees	3 m/s
6	0 degrees	1 m/s
7	0 degrees	2 m/s
8	0 degrees	3 m/s
9	15 degrees	1 m/s
10	15 degrees	2 m/s
11	15 degrees	3 m/s
12	30 degrees	1 m/s
13	30 degrees	2 m/s
14	30 degrees	3 m/s

Рис. 17.6. Определение пространства действий в консоли AWS DeepRacer

перечислить в AWS DeepRacer. При желании можно попробовать другие варианты, просто помните, что для обучения с большими пространствами действий потребуется немного больше времени.



Вот несколько советов из нашего опыта по настройке пространства действий:

- Как показали эксперименты, модели с более высокой максимальной скоростью обучаются дольше. В некоторых случаях (в зависимости от функции вознаграждения и трассы) для обучения модели с максимальной скоростью 5 м/с может потребоваться более 12 часов.
- Модель не будет выполнять действий, которых нет в пространстве действий. Аналогично, если модель обучается на трассе, не требующей использовать некоторое действие, как, например, прямолинейная трасса, где нет поворотов, то модель не будет знать, как использовать это действие, потому что не была мотивирована к его применению. Приступая к созданию надежной модели, помните о пространстве действий и тренировочной трассе.
- Выбор высокой максимальной скорости или широкого угла поворота — это здорово, но не нужно забывать о функции вознаграждения. Также подумайте о том, стоит ли входить в повороты на максимальной скорости или ехать зигзагообразно на прямых участках.
- Важно помнить о физике. Если обучить модель с максимальной скоростью больше 5 м/с, то можно столкнуться с неприятностью, когда машина будет терять управление в поворотах, что, вероятно, увеличит время обучения модели.

## Настройка функции вознаграждения

Как отмечалось выше, функция вознаграждения оценивает качество выбранного действия с учетом текущей ситуации и вознаграждает за выбор этого действия. На практике вознаграждение вычисляется во время обучения после каждого действия и составляет ключевую часть опыта, используемого для обучения модели. После оценки кортеж (состояние, действие, следующее состояние, вознаграждение) сохраняется в буфере памяти. Логику функции вознаграждения можно построить с использованием переменных, которые предоставляет симулятор. Эти переменные хранят угол поворота и скорость, а также положение автомобиля на треке, например координаты (X, Y) и путевые точки (маркеры этапов на треке). Эти показатели можно использовать в логике функции вознаграждения на Python 3.

Все параметры функции вознаграждения доступны в виде словаря. Ключи, типы данных и описание словаря даны на рис. 17.7, а некоторые из наиболее важных — на рис. 17.8.

```
{  
    "all_wheels_on_track": Boolean, # признак, что автомобиль находится  
                                    # в пределах трека  
    "x": float,                  # координата X автомобиля в метрах
```

```

    "y": float,                                # координата Y автомобиля в метрах
    "distance_from_center": float,              # расстояние от осевой линии трека в метрах
    "is_left_of_center": Boolean,                # признак, что автомобиль находится слева
                                                # от осевой линии трека
    "heading": float,                           # отклонение автомобиля от основного
                                                # направления в градусах
    "progress": float,                          # доля трека в процентах, которую преодолел
                                                # автомобиль
    "steps": int,                               # количество пройденных шагов
    "speed": float,                            # скорость автомобиля в м/с
    "steering_angle": float,                   # угол поворота колес автомобиля в градусах
    "track_width": float,                      # ширина трека
    "waypoints": [(float, float), ...],          # список координат (x,y) путевых точек на
                                                # осевой линии трека
    "closest_waypoints": [int, int],             # индексы двух ближайших путевых точек
}

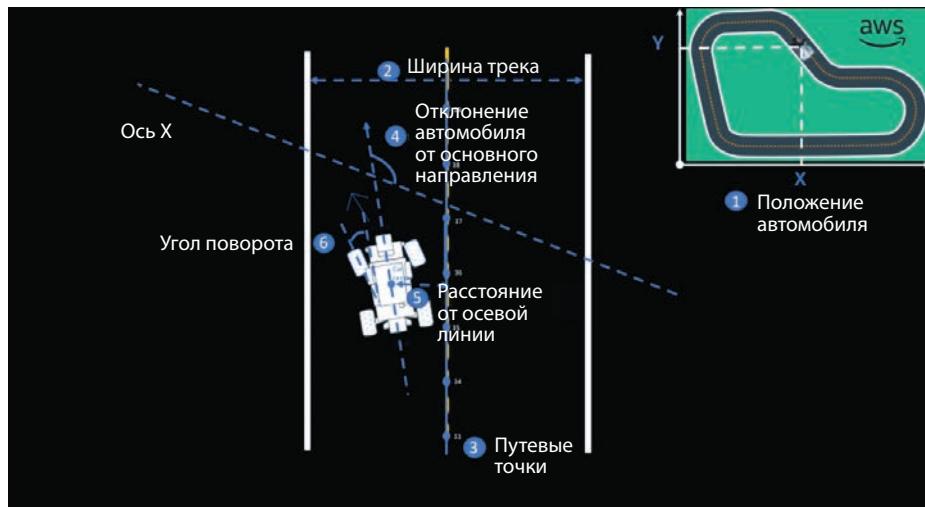
```

```

{
    "all_wheels_on_track": Boolean,           # flag to indicate if the vehicle is on the track
    "x": float,                             # vehicle's x-coordinate in meters
    "y": float,                             # vehicle's y-coordinate in meters
    "distance_from_center": float,           # distance in meters from the track center
    "is_left_of_center": Boolean,             # Flag to indicate if the vehicle is on the left side to the track center or not.
    "heading": float,                        # vehicle's yaw in degrees
    "progress": float,                       # percentage of track completed
    "steps": int,                            # number steps completed
    "speed": float,                          # vehicle's speed in meters per second (m/s)
    "steering_angle": float,                 # vehicle's steering angle in degrees
    "track_width": float,                    # width of the track
    "waypoints": [{float, float}, ...],        # list of [x,y] as milestones along the track center
    "closest_waypoints": [int, int]           # indices of the two nearest waypoints.
}

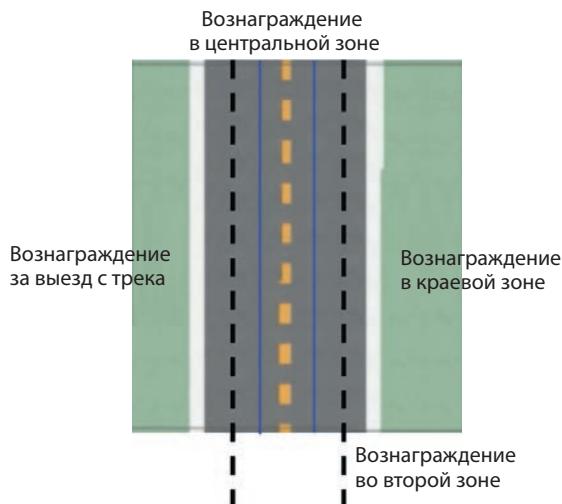
```

**Рис. 17.7.** Параметры функции вознаграждения (подробности ищите в документации)



**Рис. 17.8.** Визуальное объяснение некоторых параметров функции вознаграждения

Теперь определим простую функцию вознаграждения и обучим нашу первую модель. Воспользуемся шаблоном по умолчанию (рис. 17.9), следуя которому автопилот будет пытаться ехать вдоль осевой пунктирной линии. Логика, лежащая в основе этой функции вознаграждения, в том, чтобы выбрать самый безопасный путь по трассе, потому что, находясь на осевой линии, автомобиль имеет меньше шансов выкатиться за пределы трека. Для этого функция вознаграждения создает три зоны вдоль трека, используя три маркера, и дает наибольшее вознаграждение, если машина находится в центральной зоне. Обратите внимание на разницу в размере вознаграждения. За нахождение в узкой центральной зоне дается вознаграждение в размере 1, за нахождение во второй (нецентральной) зоне — вознаграждение 0,5, и за пребывание в краевой зоне — вознаграждение 0,1. Если уменьшить вознаграждение за нахождение в центральной зоне или увеличить вознаграждение за нахождение во второй зоне, мы фактически побудим машину использовать большую часть поверхности трека. Помните свой экзамен по вождению? Экзаменатор наверняка сделал бы то же самое и снизил оценку при приближении к бордюру или линиям разметки. Это может пригодиться, особенно когда на трассе есть острые углы.



**Рис. 17.9.** Пример функции вознаграждения

Вот код, реализующий эту логику:

```
def reward_function(params):
    ...
    Пример функции вознаграждения агента, стимулирующей его придерживаться
    осевой линии
    ...
```

```

# Прочитать входные параметры
track_width = params['track_width']
distance_from_center = params['distance_from_center']

# Вычислить три маркера, определяющие зоны на треке
marker_1 = 0.1 * track_width
marker_2 = 0.25 * track_width
marker_3 = 0.5 * track_width

# Дать более высокое вознаграждение за нахождение в центральной зоне
if distance_from_center <= marker_1:
    reward = 1.0
elif distance_from_center <= marker_2:
    reward = 0.5
elif distance_from_center <= marker_3:
    reward = 0.1
else:
    reward = 1e-3 # вероятно потерпел аварию или выехал за пределы трека

return float(reward)

```

Поскольку это первая попытка обучения, сосредоточимся на создании и оценке базовой модели, а затем перейдем к ее оптимизации. В этом случае пропустим разделы с описанием настроек алгоритма и гиперпараметров и используем значения по умолчанию.



**FAQ:** должны ли вознаграждения быть в определенном диапазоне и можно ли давать отрицательные вознаграждения?

Нет никаких ограничений в отношении того, что можно вознаграждать или не вознаграждать, но обычно принято использовать вознаграждения из диапазона от 0 до 1 или от 0 до 100. Что еще более важно, выбранная шкала должна давать относительное вознаграждение за соответствующие действия. Например, поворот руля вправо при входе в правый поворот должен высоко вознаграждаться, поворот руля влево — вознаграждение, близкое к нулю, а отказ от поворота руля можно вознаградить промежуточным значением — более высоким, чем за поворот руля влево, потому что это может быть не самое плохое действие.

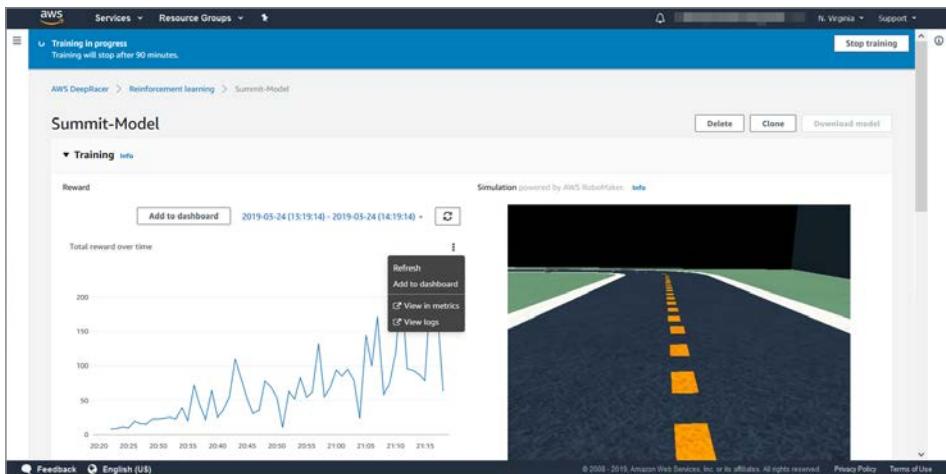
## Настройка условий остановки обучения

Перед обучением модели нужно определить, как долго будет длиться обучение. Это ограничение позволяет прекратить обучение в нужный момент, тем более что за время обучения мы платим деньги.

Установите время 60 минут, а затем щелкните на кнопке **Start training** (Запустить обучение). Если обнаружится ошибка, то нам сообщат, где она. После начала обучения может потребоваться до шести минут для запуска Amazon SageMaker, AWS Robomaker, AWS Lambda, AWS Step Function, которые используются в обучении. Помните, что, щелкнув на кнопке **Stop**, обучение можно прервать раньше, если мы увидим, что модель сошлась (как объясняется в следующем разделе).

## Шаг 3: обучение модели

После начала обучения модель можно выбрать из списка в консоли DeepRacer и оценивать прогресс обучения количественно, по графику изменения общего вознаграждения с течением времени, и качественно, наблюдая за поведением автопилота в симуляторе «от первого лица» (рис. 17.10).



**Рис. 17.10.** Тренировочный график и видеопоток симуляции AWS DeepRacer

Сначала автопилот будет постоянно отклоняться в стороны, но по мере обретения навыков вождения его характеристики будут улучшаться, а кривая графика вознаграждений пойдет вверх. Кроме того, выкатившись за пределы трека, автопилот будет возвращать машину обратно. Это можно заметить по пилообразной форме графика вознаграждений.



*FAQ: почему график вознаграждений пилообразный?*

Первоначально агент активно исследует доступные ему возможности, а по мере накопления опыта начинает использовать его. Поскольку в части своих решений агент выбирает случайные действия, иногда его решения могут быть неверными, и тогда он выкатывается за пределы трека. По этой причине в начале обучения график вознаграждений имеет резко пилообразную форму с большими перепадами, но по мере обучения модели становится более гладким.

Лучший источник подробной информации об обучении модели — логи. Дальше мы покажем, как можно программно анализировать логи, чтобы получить более полное представление об обучении модели. А пока загляните в файлы логов

Amazon SageMaker и AWS RoboMaker, которые доступны в Amazon CloudWatch. Для их просмотра наведите курсор на график вознаграждений и выберите три точки, которые появятся под кнопкой **Refresh** (Обновить). Затем выберите пункт **View logs** (Просмотр логов) в открывшемся меню. Поскольку обучение модели займет целый час, сейчас самое время перейти к следующему разделу и узнать чуть больше об обучении с подкреплением.

## Шаг 4: оценка качества модели

В обучении с подкреплением лучший способ оценить качество модели — запустить ее так, чтобы она использовала только накопленный опыт и не предпринимала никаких случайных действий. Сначала протестируем ее на похожей на учебную трассе и посмотрим, сможет ли она воспроизвести изученное поведение. Затем опробуем модель на другой трассе, чтобы проверить ее способность к обобщению. После обучения можно приступить к оценке модели. На странице информации о модели, где мы следили за обучением, щелкните на кнопке **Start evaluation** (Запустить оценку). Затем выберите трассу, где будет оцениваться качество модели, а также количество кругов. В нашем случае — трасса «te:Invent 2018» и 5 кругов. Щелкните на кнопке **Start** (Пуск). После этого откроется страница (на рис. 17.11) с обобщенными результатами попыток модели проехать по трассе и завершить круг.

Отличная работа! Мы благополучно создали свой первый автопилот, используя методику обучения с подкреплением.



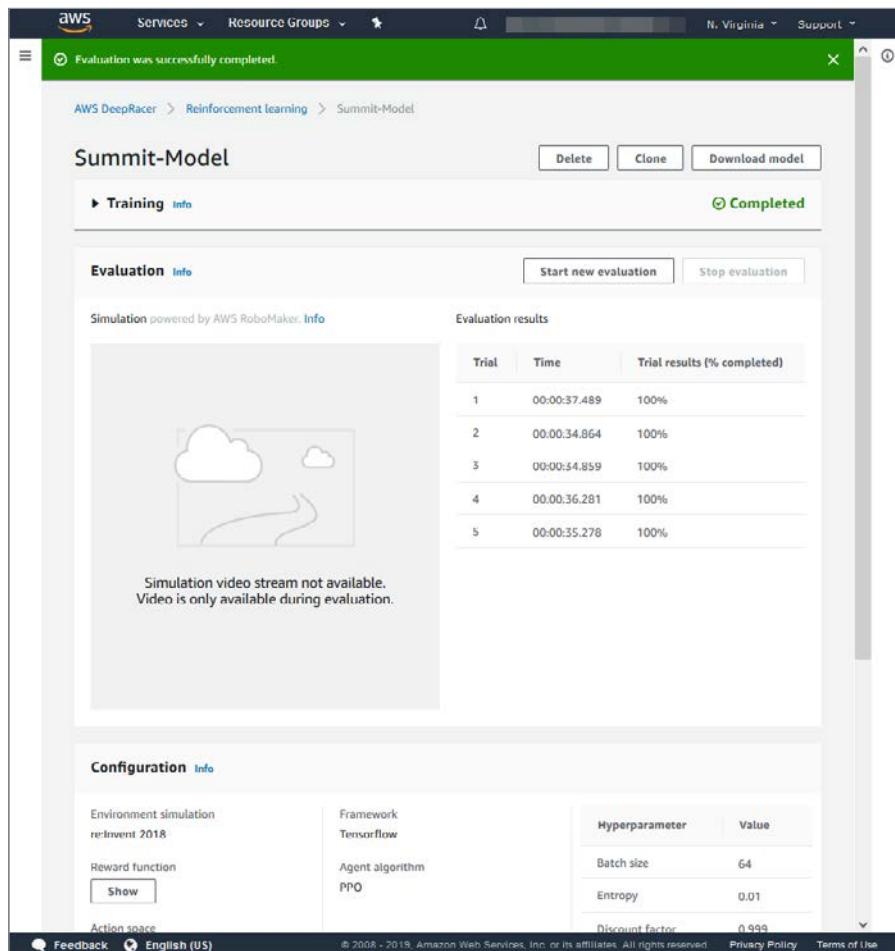
*FAQ: почему иногда, запустив оценку, можно видеть, что не все попытки выполнены на 100%?*

В процессе инференса и перемещения по треку из-за погрешностей вычислений в симуляторе автомобиль может оказываться немного в разных местоположениях при выполнении одного и того же действия. Например, поворот влево на 15 градусов может фактически дать поворот на 14,9 градуса. На практике мы наблюдали только очень небольшие отклонения в симуляторе, но они могут накапливаться в процессе оценки. Хорошо обученная модель умеет возвращаться к осевой линии из крайних позиций, а недостаточно обученная модель — нет.

Оценив качество обучения модели, перейдем к ее улучшению и посмотрим, как сократить время прохождения круга. Но для этого нужно глубже вникнуть в теорию обучения с подкреплением и понять, как протекает обучение.



Консоль AWS DeepRacer позволяет создавать модели, обучать их (до шести часов), оценивать и бесплатно отправлять в AWS DeepRacer League для участия в состязаниях.



**Рис. 17.11.** Страница оценки модели в консоли AWS DeepRacer

## Обучение с подкреплением на практике

Рассмотрим, как действует обучение с подкреплением. В этом разделе мы обсудим некоторые теоретические основы, внутренние особенности и практические идеи, связанные с нашим проектом беспилотного автомобиля.

### Как происходит обучение с подкреплением?

Во-первых, важно понимать разницу между *исследованием* и *использованием*. Как и ребенок, система обучения с подкреплением учится, исследуя и узна-

вая, что хорошо, а что плохо. Родители направляют ребенка, сообщают ему об ошибках, дают оценку принятых им решений — хорошие они или плохие и насколько хорошие или плохие. Ребенок запоминает решения, которые принимал в определенных ситуациях, и пытается повторно применить, или использовать, эти решения в подходящие моменты. По сути, ребенок пытается получить максимальное одобрение от своих родителей. В начале жизни ребенок более внимателен к подсказкам родителей и склонен к обучению. Повзрослев, он перестает их слушать, а для принятия решений использует накопленный опыт.

Как показано на рис. 17.12, на каждом временном шаге  $t$  автопилот DeepRacer (агент) получает свое текущее наблюдение, состояние ( $S_t$ ), и на его основе выбирает действие ( $A_t$ ). По результатам оценки выбранного действия агент получает вознаграждение  $R_{t+1}$  и переходит в состояние  $S_{t+1}$ . Этот процесс продолжается на протяжении всего эпизода.



**Рис. 17.12.** Краткое представление теоретической основы обучения с подкреплением

В контексте DeepRacer агент исследует свою среду, выбирая случайные действия, а функция вознаграждения, как родитель, сообщает агенту, насколько хорошими были действия, предпринятые для данного состояния. Обычно оценка действия для данного состояния выражается числом, при этом чем выше оценка, тем ближе выбранное действие к оптимальному, а чем ниже — тем дальше. Система фиксирует всю эту информацию для каждого шага, а именно *текущее состояние, предпринятые действия, величину вознаграждения и следующее состояние* ( $s, a, r, s'$ ), в то, что мы называем буфером воспроизведения или опыта, который является обычным времененным буфером в памяти. Идея заключается в возможности обучения автопилота на оценках его решений. Ключевой момент здесь — более высокая доля исследовательских действий в начале обучения и постепенное увеличение доли использования накопленного опыта.

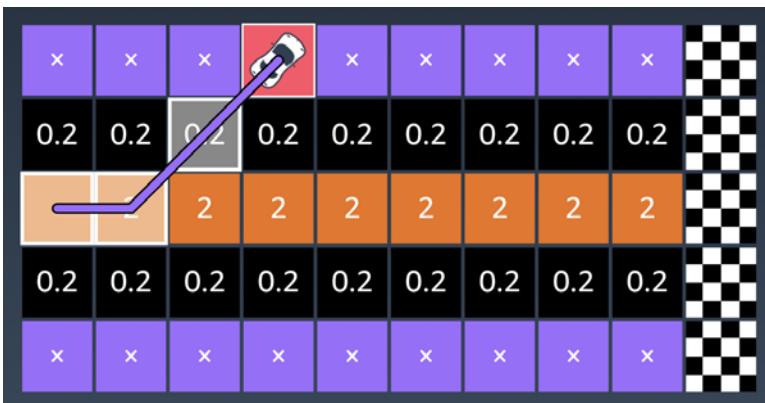
В симуляторе DeepRacer входные изображения отбираются с частотой 15 FPS. На каждом шаге (кадре) автомобиль переходит из одного состояния в другое. Каждое изображение представляет текущее состояние автомобиля, и в итоге после обучения модель будет пытаться использовать его для выбора действия, которое следует предпринять. Принимая решения, можно выбирать случайные действия или использовать опыт, накопленный моделью. Со временем процесс обучения уравновешивает исследование и эксплуатацию. На первых порах доля исследовательских действий больше, потому что модель еще не успела накопить достаточно опыта, но по мере обучения доля исследовательских действий уменьшается и увеличивается доля использования накопленного опыта. Этот процесс показан на рис. 17.13. Этот переход может следовать линейной, экспоненциальной или любой похожей стратегии, которая обычно настраивается на основе предполагаемой скорости обучения. В практике обучения с подкреплением обычно используется стратегия экспоненциального убывания доли исследовательских действий.



**Рис. 17.13.** Процесс обучения в DeepRacer

После выполнения действия, выбранного случайно или на основе прежнего опыта, автомобиль переходит в новое состояние. С помощью функции вознаграждения вычисляется оценка, и результат присваивается выбранному действию. Этот процесс повторяется для каждого состояния, пока не будет достигнуто конечное состояние, то есть когда автомобиль выкатится за пределы трека или завершит круг, после чего автомобиль будет установлен на старт и заезд повторится. Шаг — это переход из одного состояния в другое, и на каждом шаге записывается кортеж (состояние, действие, вознаграждение, новое состояние). Коллекция шагов от начального до конечного состояния называется *эпизодом*.

Рассмотрим пример миниатюрной гоночной трассы на рис. 17.14 с функцией вознаграждения, побуждающей автопилот придерживаться осевой линии — это самый короткий и быстрый путь от старта до финиша. Эпизод состоит из четырех шагов: на шагах 1 и 2 автомобиль следует вдоль осевой линии, а на шаге 3 выполняет поворот на 45 градусов и продолжает движение в этом направлении, пока на шаге 4 не выкатывается за пределы трека.



**Рис. 17.14.** Исследовательское поведение агента в течение эпизода

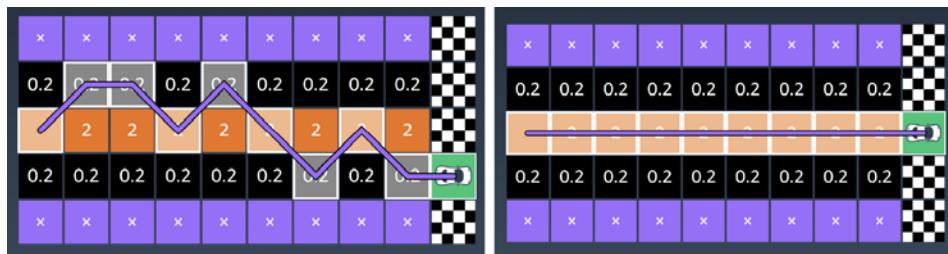
Записи этих эпизодов можно рассматривать как опыт или обучающие данные для нашей модели. Периодически система обучения с подкреплением выбирает случайное подмножество записей из буфера в памяти и обучает глубокую нейронную сеть (нашу модель глубокого обучения с подкреплением), чтобы произвести модель, которая лучше ориентируется в средах с нашей руководящей и направляющей функцией вознаграждения. Другими словами, модель должна получить максимальное совокупное вознаграждение. Со временем, точнее с увеличением количества эпизодов, модель будет улучшаться. Разумеется, огромное значение имеет функция вознаграждения, которая должна быть четко определена и направлять агента к цели. С плохой функцией вознаграждения модель не научится правильному поведению.

Давайте отвлечемся и попробуем понять, что такое «плохая функция вознаграждения». Наша первая функция позволяла автопилоту двигаться в любом направлении (влево, вправо, вперед, назад), не делая различий между направлениями, и в конце концов модель научилась накапливать вознаграждение, просто катаясь взад-вперед. Мы устранили эту проблему, стимулировав автопилот двигаться только вперед. К счастью для разработчиков, команда DeepRacer упростила задачу, позволив машине двигаться только прямо, поэтому вам даже не придется думать о таком поведении.

Теперь вернемся к обучению с подкреплением. *Как убедиться, что модель становится лучше?* Вспомним понятия «исследование» и «использование опыта». Мы уже говорили, что на начальном этапе велика доля исследовательских действий, которая постепенно уменьшается, и увеличивается доля действий, выбираемых на основе опыта. Это означает, что определенный процент времени в процессе обучения система обучения с подкреплением занимается исследованием, то есть совершает случайные действия. По мере обучения в буфер опыта для каждого состояния записываются все принятые решения (действия), включая

те, которые привели к наибольшему или наименьшему вознаграждению. Модель, по сути, использует эту информацию, чтобы выявить и выбрать наилучшее действие, которое следует предпринять для данного состояния. Предположим, что первоначально для состояния  $S$  модель предприняла действие «ехать прямо» и получила вознаграждение 0,5, но в следующий раз, оказавшись в состоянии  $S$ , она выполнила действие «поворнуть вправо» и получила вознаграждение в размере 1. Если в буфере памяти появится несколько записей для одного и того же состояния  $S$  с величиной вознаграждения «1» за одно и то же действие «поворнуть вправо», то в итоге модель выучит, что «поворнуть вправо» является оптимальным действием для этого состояния. Со временем модель будет меньше исследовать и больше использовать накопленный опыт, и процентное отношение между ними изменяется линейно или экспоненциально для лучшего обучения.

Ключевым моментом здесь является то, что если модель совершает оптимальное действие, то система учится выбирать именно его. При выборе другого, менее оптимального, действия система продолжит искать лучшее действие для данного состояния. С практической точки зрения может быть множество путей к цели, но для примера с мини-трассой на рис. 17.13 самый быстрый путь — прямой, вдоль осевой линии, потому что любые повороты только замедляют движение. В период обучения в самых первых эпизодах, когда модель только начинает учиться, можно заметить, что иногда она достигает финишной черты (цели), никуда не сворачивая, а иногда может поехать не по оптимальному пути, как показано на рис. 17.15 (слева), двигаясь зигзагами и поэтому затрачивая больше времени, чтобы добраться до финиша, и получая совокупное вознаграждение, равное всего лишь 9. Но, поскольку система все еще выделяет часть времени на исследования, агент может найти лучший путь. По мере накопления опыта агент учится находить оптимальный путь и приближается к оптимальной политике, обеспечивающей достижение цели с общим совокупным вознаграждением 18, как показано на рис. 17.15 (справа).



**Рис. 17.15.** Иллюстрация разных путей к цели

С количественной точки зрения в каждом эпизоде должна наблюдаться тенденция к увеличению суммарного вознаграждения. Если модель принимает оптимальные решения, машина должна оставаться в пределах трека и двигаться

по установленной траектории, накапливая вознаграждения. Но даже после эпизодов с высоким суммарным вознаграждением на графике могут быть провалы, что особенно характерно на ранних эпизодах, когда у модели все еще может быть установлена высокая доля исследовательских действий.

## Теория обучения с подкреплением

Теперь, разобравшись с тем, как работает и обучается наша модель, особенно в контексте AWS DeepRacer, рассмотрим некоторые формальные определения и общую теорию обучения с подкреплением. Эти знания пригодятся вам, когда вы будете решать другие практические задачи.

### Марковский процесс принятия решений

Марковский процесс принятия решений (Markov Decision Process, MDP) — это дискретный стохастический процесс перехода между состояниями, который используется для моделирования принятия решений в управлении. Характерное свойство марковских процессов — каждое последующее состояние зависит исключительно от предыдущего. Это очень удобное свойство, потому что для перехода между состояниями вся необходимая информация доступна в текущем состоянии. Теоретические результаты обучения с подкреплением зависят от возможности сформулировать проблему в терминах MDP, поэтому важно понимать, как смоделировать проблему в форме MDP, чтобы потом решить ее с помощью обучения с подкреплением.

### С использованием и без использования модели среды

Здесь под «моделью» подразумевается выученное представление среды. Применение модели полезно, потому что позволяет выучить динамику среды и обучить агента с использованием модели среды взамен реальной среды.

Но создать модель среды непросто — проще использовать представление реального мира в симуляторе и изучать восприятие и динамику одновременно, в процессе обучения агента, а не одно за другим в последовательности. В этой главе рассмотрим только обучение с подкреплением без использования моделей среды.

### На основе оценки

За каждое предпринятое действие агент получает соответствующее назначенное функцией вознаграждение. Для любой пары состояние-действие полезно знать ее оценку (вознаграждение). Если бы существовала функция вычисления максимального вознаграждения, мы бы нашли максимальное вознаграждение, которое может быть достигнуто в любом состоянии, и просто выбрали соответствующее действие. Например, в игре «крестики-нолики  $3 \times 3$ » количество игровых си-

туаций ограничено, поэтому можно построить таблицу поиска и получить наилучший ход в любой конкретной ситуации. А вот в шахматах, учитывая размер доски и сложность игры, создание такой таблицы поиска обойдется слишком дорого с вычислительной точки зрения и для хранения потребуется огромный объем памяти. Поэтому в сложных средах непрактично поддерживать таблицу пар «состояние-действие» и определять функцию, способную отобразить эти пары в их оценке. Вместо этого используется нейронная сеть, параметризованная функцией оценки, и эта сеть применяется для аппроксимации оценки каждого действия с учетом наблюдаемого состояния. Пример алгоритма, основанного на оценке, — Deep Q-Learning.

## На основе политики

Политика — это набор правил навигации в среде, которую пытается изучить агент. Проще говоря, функция политики сообщает агенту, какое действие лучше всего предпринять в текущем состоянии. Алгоритмы обучения с подкреплением на основе политик, например REINFORCE и Policy Gradients, находят оптимальную политику без отображения оценки в состояния. В обучении с подкреплением нейронная сеть параметризуется политикой, то есть мы позволяем нейронной сети изучить оптимальную функцию политики.

## На основе политики или оценки — почему не обе вместе?

Исследователи продолжают спорить, какой подход к обучению с подкреплением лучше — на основе политики или оценки. Новейшие архитектуры стараются выучить и функцию оценки, и функцию политики. Такой подход в обучении с подкреплением получил название «актор-критик» (actor critic).

Здесь под актором подразумевается политика, а под критиком — функция оценки. Актор отвечает за выбор действий, а критик — за оценку этих действий. Актор отображает состояния в действия, а критик отображает пары «состояние-действие» в их оценке. В парадигме «актор-критик» две сети (актор и критик) обучаются отдельно с использованием метода градиентного восхождения для корректировки весов глубокой нейронной сети. (Вы наверняка помните, что наша цель — максимизировать совокупное вознаграждение, поэтому используется метод поиска глобального максимума.) По мере прохождения эпизодов актор все чаще выбирает действия, которые приводят к большим вознаграждениям, а критик лучше оценивает ценность этих действий. Обучающий сигнал для актора и критика исходит исключительно из функции вознаграждения.

Ценность данной пары «состояние-действие» называется *Q-оценкой* и обозначается как  $Q(s, a)$ . Можно разложить Q-оценку на две части: саму оценку и количественную меру фактора, согласно которой данное действие оценивается лучше других. Эта мера называется *функцией преимущества* (advantage function). Преимущество можно рассматривать как разницу между фактическим

и ожидаемым вознаграждением для данной пары «состояние-действие». Чем больше разница, тем больше отклонение от оптимального действия.

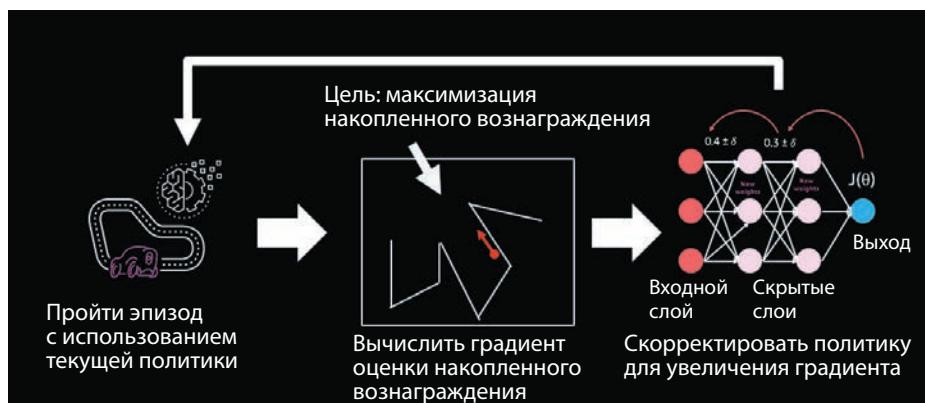
Учитывая, что иногда получение оценки является слишком сложной задачей, можно сосредоточиться на изучении функции преимущества. Это позволит оценивать действие не только по тому, насколько оно хорошо, но и по тому, насколько лучше оно могло бы быть. Это сильно упрощает поиск оптимальной политики по сравнению с другими, более простыми методами на основе градиента политики, потому что, как правило, сети на основе политики имеют высокую дисперсию.

### Отложенные вознаграждения и понижающий коэффициент ( $\gamma$ )

За каждый переход из одного состояния в другое выдается вознаграждение, величина которого зависит от предпринятого действия. Но влияние этих вознаграждений может быть нелинейным. В одних задачах может быть более важным немедленное вознаграждение, а в других — отложенное. Например, в алгоритме торговли акциями будущие вознаграждения могут иметь более высокую неопределенность, поэтому при построении такого алгоритма нужно предусмотреть соответствующий понижающий коэффициент. Понижающий коэффициент ( $\gamma$ ) — это коэффициент с величиной между  $[0,1]$ , где значения, близкие к нулю, означают, что вознаграждения в ближайшем будущем более важны, а значения, близкие к единице, означают, что агенту лучше сосредоточиться на действиях, которые максимизируют вознаграждения в отдаленном будущем.

## Алгоритм обучения с подкреплением в AWS DeepRacer

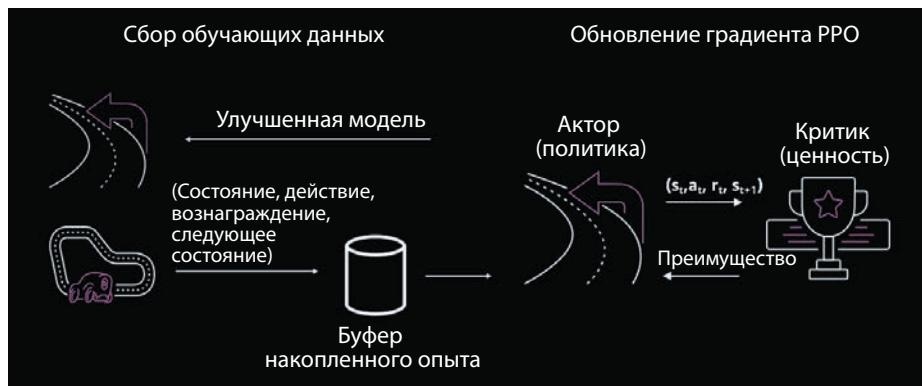
Рассмотрим один из простейших примеров оптимизирующих алгоритмов обучения с подкреплением: стандартный (vanilla) градиент политики.



**Рис. 17.16.** Обучение с использованием стандартного алгоритма градиента политики

Модель глубокого обучения с подкреплением можно представить как состоящую из двух частей: модуля выделения признаков и сети политики. Модуль выделения признаков извлекает признаки из входных изображений и передает их в сеть политики, которая принимает решения, например предсказывает, какое действие лучше подходит для данного текущего состояния. Учитывая, что на вход подаются изображения, для извлечения признаков используются сверточные слои. Поскольку наша цель — изучить политику, мы параметризуем сеть простейшей функцией политики, которую можно изучить с помощью полно связного слоя. Входные сверточные слои принимают изображение, а сеть политики использует признаки, извлеченные из изображения, и прогнозирует действие. Так состояние отображается в действие. По мере обучения модель все лучше справляется с извлечением признаков из входного пространства и их отображением в действия, попутно оптимизируя политику выбора наилучшего действия для каждого состояния. Наша цель — получить как можно более высокое суммарное вознаграждение. Для этого веса модели обновляются так, чтобы максимизировать суммарное вознаграждение в будущем, а для этого мы повышаем вероятность выбора действия, которое приведет к более высокому суммарному вознаграждению в будущем. Ранее при обучении нейронных сетей мы использовали стохастический градиентный спуск или его разновидности; в обучении с подкреплением мы стремимся максимизировать суммарное вознаграждение. По этой причине используем градиентное восхождение, корректирующее веса в направлении самого большого градиента вознаграждения.

В DeepRacer используется расширенный вариант алгоритма оптимизации, называемый алгоритмомproxимальной оптимизации политики (Proximal Policy Optimization, PPO), краткое описание которого приводится на рис. 17.17.



**Рис. 17.17.** Обучение с использованием алгоритма PPO

Слева на рис. 17.17 находится симулятор, использующий последнюю политику (модель) для получения нового опыта ( $s, a, r, s'$ ). Опыт записывается в буфер

воспроизведения опыта, который передается в алгоритм PPO после прохождения заданного количества эпизодов.

Справа на рис. 17.17 алгоритм PPO обновляет модель. Алгоритм PPO — это алгоритм оптимизации политики, но он использует подход «актор-критик», описанный выше. Мы вычисляем градиент PPO и меняем политику в направлении получения наибольшего вознаграждения. Большие шаги при движении в этом направлении могут вызвать слишком большую дисперсию в результатах обучения, маленькие — замедлить обучение до бесконечности. Алгоритм PPO повышает стабильность политики (актора), ограничивая количество обновлений политики на каждом этапе обучения. Для этого используется обрезанная суррогатная целевая функция (clipped surrogate-objective function), которая предотвращает слишком частое обновление политики и тем самым решает проблему большой дисперсии, характерную для многих методов оптимизации политики. Обычно при использовании PPO соотношение новой и старой политик сохраняется на уровне [0,8, 1,2]. Критик говорит актору, насколько хорошим было выбранное действие и как актору следует скорректировать свою сеть. После обновления политики новая модель отправляется в симулятор, чтобы получить дополнительный опыт.

## Кратко о порядке глубокого обучения с подкреплением на примере DeepRacer

Для решения любой задачи с использованием обучения с подкреплением мы должны выполнить следующие шаги:

1. Определить цель.
2. Выбрать входное состояние.
3. Определить пространство действий.
4. Сконструировать функцию вычисления вознаграждения.
5. Определить архитектуру нейронной сети.
6. Выбрать алгоритм оптимизации для обучения с подкреплением (DQN, PPO и т. д.).

Что бы мы ни создавали — беспилотный автомобиль или роботизированный манипулятор для захвата объектов, фундаментальный способ обучения с подкреплением от этого не меняется. Это огромное преимущество, потому что дает возможность сосредоточиться на более высоком уровне абстракции. Чтобы решить задачу с использованием обучения с подкреплением, прежде всего следует сформулировать ее как MDP, определить форму входного состояния и перечень действий, доступных агенту в данной среде, а также функцию вознаграждения. С практической точки зрения определение функции вознаграждения может оказаться одной из самых сложных задач и часто является самой

важной, потому что именно она влияет на политику, изучаемую агентом. После определения всех факторов, зависящих от среды, нужно выбрать архитектуру глубокой нейронной сети, отображающей входные данные в действия, а затем — алгоритм обучения с подкреплением (основанный на оценке или на политике либо на том и другом — на подходе «актор-критик»). После выбора алгоритма можно сосредоточиться на высокоуровневых параметрах, управляющих алгоритмом. Когда мы сами видим машину, то склонны больше внимания уделять элементам управления, а знание устройства двигателя мало влияет на стиль вождения. Точно так же, зная параметры, с помощью которых можно управлять алгоритмом, мы сможем обучить модель с подкреплением.

Рассмотрим это на практическом примере. Сформулируем нашу задачу для модели DeepRacer:

1. Цель: проехать круг за кратчайшее время.
2. Вход: черно-белые изображения  $120 \times 160$ .
3. Действия: дискретные действия, сочетающие значение скорости и угол поворота.
4. Вознаграждение: автопилот получает вознаграждение за то, что остается на треке, за высокую скорость и за отсутствие множества корректировок направления (зигзагообразного движения).
5. Архитектура нейронной сети: три сверточных слоя + полносвязный слой (вход  $\rightarrow$  сверточный слой  $\rightarrow$  сверточный слой  $\rightarrow$  сверточный слой  $\rightarrow$  полносвязный слой  $\rightarrow$  выход).
6. Алгоритм оптимизации: PPO.

## Шаг 5: улучшение модели обучения с подкреплением

Теперь улучшим модель, а заодно получим дополнительные знания об обучении моделей. Посмотрим, как сделать это в консоли, которая предлагает возможность изменения параметров алгоритма обучения с подкреплением и гиперпараметров нейронной сети.

### Настройки алгоритма

В этом разделе описываются гиперпараметры, которые используются алгоритмом обучения с подкреплением. Гиперпараметры позволяют повысить эффективность обучения.

### Гиперпараметры нейронной сети

В табл. 17.2 перечислены доступные гиперпараметры для настройки нейронной сети. Значения по умолчанию экспериментально доказали свою работоспособ-

ность, но разработчик может попробовать изменить размер пакета, количество эпох и скорость обучения — гиперпараметры, которые оказались наиболее важными для получения высококачественных моделей, то есть моделей, получающих максимальную отдачу от функции вознаграждения.

**Таблица 17.2.** Описание и порядок настройки гиперпараметров глубокой нейронной сети

Параметр	Описание	Советы
Размер пакета	Количество образцов принятия решений, выбираемых случайным образом из буфера опыта и используемых для обновления весов нейронной сети. Если, к примеру, в буфере 5120 образцов, а размер пакета задать равным 512, то без учета случайности выбора мы получим 10 пакетов опыта, каждый из которых будет использоваться для обновления весов нейронной сети во время обучения	Используйте большие пакеты, чтобы обеспечить более стабильные и плавные обновления весов нейронной сети, но помните, что при этом обучение может протекать медленнее
Количество эпох	Эпоха — это один проход через все пакеты, на которых обновляются веса нейронной сети. Десять эпох — это десятикратное повторение процесса обучения на всех пакетах с корректировкой весов нейронной сети после каждого пакета	Чем больше эпох, тем стабильнее обновления весов, но тем дольше длится обучение. С пакетами небольшого размера можно использовать меньшее количество эпох
Скорость обучения	Скорость обучения управляет величиной изменения весов нейронной сети, то есть насколько следует изменить наши веса политики, чтобы получить максимальное суммарное вознаграждение	Чем выше скорость, тем меньше длится обучение, но может возникнуть проблема несходимости. Чем меньше скорость обучения, тем стабильнее сходимость, но на обучение может уйти много времени
Доля исследовательских действий	Относится к методу определения баланса между исследованием и использованием опыта. Иначе говоря, этот метод будет определять, когда следует прекратить исследования (выбирать действия случайно) и использовать накопленный опыт	При использовании дискретного пространства действий, как в нашем случае, всегда следует выбирать «CategoricalParameters»
Энтропия	Степень неопределенности или случайности, добавляемая в распределение вероятностей в пространстве действий. Этот параметр способствует выбору случайных действий для более широкого изучения пространства состояний и действий	

Таблица 17.2 (окончание)

Параметр	Описание	Советы
Понижающий коэффициент	Коэффициент, определяющий вклад вознаграждений в отдаленном будущем в общее суммарное вознаграждение. Чем больше понижающий коэффициент, тем дальше модель будет заглядывать для определения ожидаемого суммарного вознаграждения и тем медленнее будет идти процесс обучения. С коэффициентом 0,9, выбирая действие для текущего состояния, наша модель автопилота будет учитывать размер вознаграждения за следующие 10 шагов. С коэффициентом 0,999 она будет учитывать размер вознаграждения примерно за 1000 следующих шагов	Рекомендуемые значения понижающего коэффициента: 0,99, 0,999 и 0,9999
Тип потерь	Тип потерь определяет тип целевой функции (функции стоимости), используемой для обновления весов сети. Функции потерь Хьюбера (Huber) и среднеквадратичной ошибки проявляют похожее поведение при небольших изменениях весов. Но при больших изменениях функция потерь Хьюбера дает меньшие приращения по сравнению с функцией среднеквадратичной ошибки	Когда наблюдается проблема несходимости, используйте функцию потерь Хьюбера. Если сходимость модели хорошая и желательно сократить время на ее обучение, используйте функцию среднеквадратичной ошибки
Количество эпизодов между сессиями обучения	Этот параметр определяет, сколько опыта должно быть получено перед каждой итерацией обучения модели. Для сложных задач, имеющих несколько локальных максимумов, требуется больший буфер опыта, чтобы представить для обучения большее количество некоррелированных точек данных. В этом случае обучение будет продолжаться дольше, зато более стабильно	Рекомендуемые значения: 10, 20 и 40

## Взгляд внутрь обучения модели

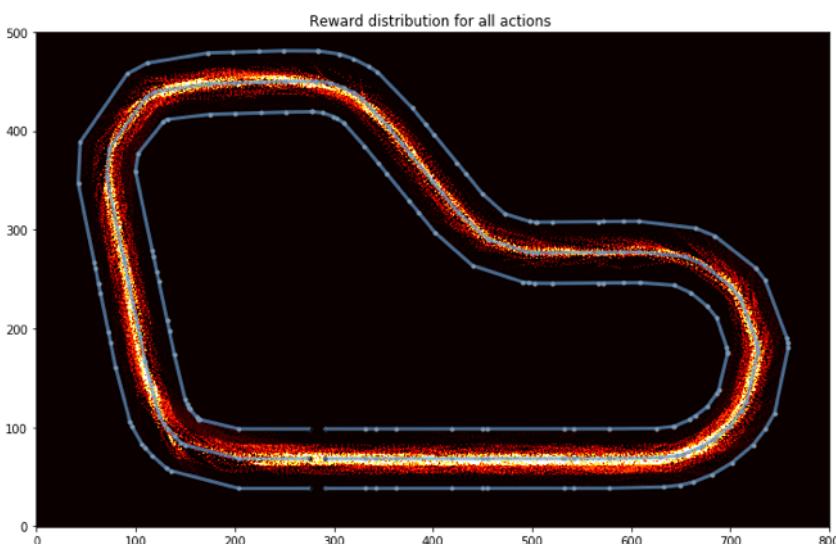
После обучения модели график изменения вознаграждений во времени, как на рис. 17.10, помогает получить представление о том, как продвигалось обучение, и о точке, в которой модель начинает сходиться. Но он не дает представления о сходимости политики, о поведении функции вознаграждения и об участках, на которых можно было бы увеличить скорость автомобиля. Чтобы получить больше информации, мы разработали блокнот Jupyter, который анализирует логи и формирует предложения по улучшению. В этом разделе рассмотрим некоторые из наиболее полезных инструментов визуализации, которые помогают заглянуть внутрь процесса обучения.

В файл лога записывается каждый шаг, совершенный автомобилем. Для каждого шага сохраняются координаты X и Y, угол отклонения от осевой линии, угол по-

ворота рулевого колеса, положение педали газа, пройденное расстояние от линии старта, предпринятое действие, вознаграждение, ближайшая путевая точка и т. д.

## Тепловая карта

При использовании сложных функций вознаграждения хочется видеть, как распределается вознаграждение вдоль трека, то есть где на треке и какое вознаграждение получил автопилот. Для визуализации этого распределения можно создать тепловую карту (рис. 17.18). Учитывая, что функция вознаграждения, которую мы использовали, давала максимальное вознаграждение за близость к осевой линии, можно видеть, что именно эта область окрашена в яркий цвет, а узкие полосы по обе стороны от осевой линии окрашены красным цветом, что указывает на меньшее вознаграждение. Наконец, остальная часть трека темная, что указывает на отсутствие или незначительность вознаграждения, когда машина находилась в этих местах. Чтобы сгенерировать свою тепловую карту для анализа распределения вознаграждений, используйте код из репозитория <https://oreil.ly/n-w7G>.

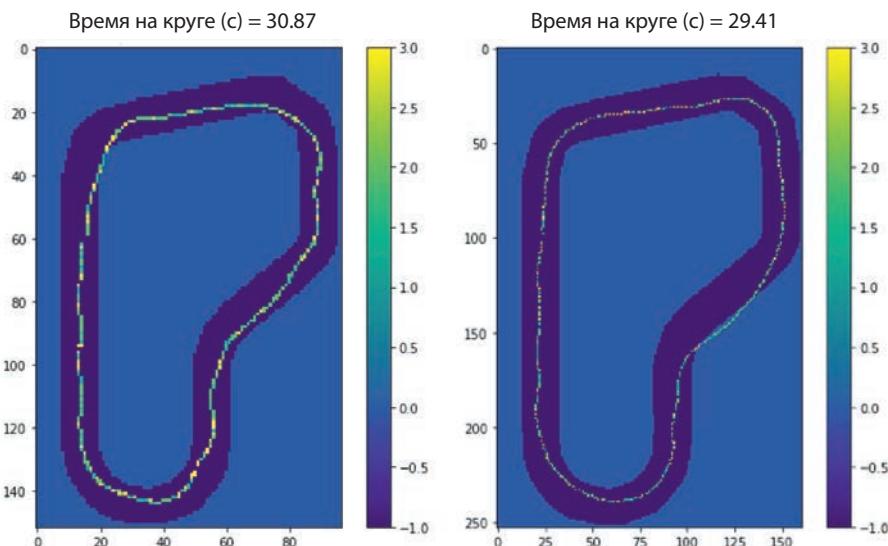


**Рис. 17.18.** Термография для примера с функцией вознаграждения, которая дает максимальное вознаграждение за близость к осевой линии

## Оптимизация скорости движения автомобиля

После оценки модели мы получаем результаты прохождения каждого круга. В этот момент может быть интересно узнать, какой путь прошла машина, где она потерпела неудачу или, может быть, где она излишне замедлилась. Чтобы выяснить все это, можно использовать код из блокнота (<https://oreil.ly/tLOtk>)

и с его помощью построить тепловую карту распределения скоростей. В следующем примере показан путь, по которому машина движется по трассе, а также ее скорость в различных точках. Это позволяет выявить участки трассы, где можно оптимизировать скорость. Беглого взгляда на рис. 17.19 (слева) достаточно, чтобы понять, что машина ехала слишком медленно по прямому участку трассы. Мы могли бы побудить модель двигаться быстрее, предусмотрев дополнительное вознаграждение за высокую скорость на этом участке трассы.



**Рис. 17.19.** Тепловая карта скорости в оценочном заезде; слева — оценочный круг с исходной функцией вознаграждения, справа — быстрейший круг с модифицированной функцией вознаграждения

На рис. 17.19 (слева) кажется, что иногда автомобиль движется зигзагообразно, поэтому одним из улучшений может быть уменьшение величины вознаграждения за слишком частые повороты. В следующем примере кода вознаграждение умножается на коэффициент 0,8, если количество поворотов превышает пороговое значение. Также можно стимулировать автопилот ехать быстрее, увеличив вознаграждение на 20 %, если машина едет со скоростью 2 м/с или больше. После обучения с этой новой функцией вознаграждения машина действительно едет быстрее. На рис. 17.19 (справа) можно видеть, что машина едет намного стабильнее; она почти идеально следует вдоль осевой линии и финиширует примерно на две секунды быстрее, чем в первой попытке. И это результат довольно скромных усилий усовершенствовать модель. Используя эти инструменты, можно продолжить обучать модель и определить лучшее время круга. Все подсказки включены в пример функции вознаграждения:

```
def reward_function(params):
    ...

Пример штрафования за слишком частые повороты рулем. Это помогает
избавиться от зигзагообразной манеры езды и увеличить скорость
    ...

# Прочитать входные параметры
distance_from_center = params['distance_from_center']
track_width = params['track_width']
steering = abs(params['steering_angle']) # Нужен только абсолютный угол
speed = params['speed'] # в м/с

# Вычислить три маркера, определяющие три зоны на треке
marker_1 = 0.1 * track_width
marker_2 = 0.25 * track_width
marker_3 = 0.5 * track_width

# Дать более высокое вознаграждение за нахождение в центральной зоне
if distance_from_center <= marker_1:
    reward = 1
elif distance_from_center <= marker_2:
    reward = 0.5
elif distance_from_center <= marker_3:
    reward = 0.1
else:
    reward = 1e-3 # вероятно, потерпел аварию или выехал за пределы трека

# Штраф за превышение порогового количества поворотов,
# измените это число в зависимости от настройки пространства действий
ABS_STEERING_THRESHOLD = 15

# Уменьшить вознаграждение, если агент слишком активно пользуется
# рулевым управлением
if steering > ABS_STEERING_THRESHOLD:
    reward *= 0.8

# Стимулировать более быструю езду
if speed >= 2:
    reward *= 1.2
```

## Гонки на автомобиле AWS DeepRacer

А теперь перенесем все то, что мы узнали, из виртуального мира в реальный и погоняем на настоящем беспилотнике. Игрушечном, конечно же.

Если у вас есть автомобиль AWS DeepRacer, следуйте инструкциям по тестированию своей модели в реальном мире. Для тех, кому это интересно, отмечу, что модели AWS DeepRacer можно купить на Amazon.

## Создание трека

Теперь, имея обученную модель, можно оценить ее на реальном треке с использованием физической модели AWS DeepRacer. Для начала построим импрови-

зированную гоночную трассу. Чтобы было проще, построим только часть, но, следуя инструкциям ниже, можно построить и полноценную трассу.

Для этого понадобятся следующие материалы:

#### Для границ трека

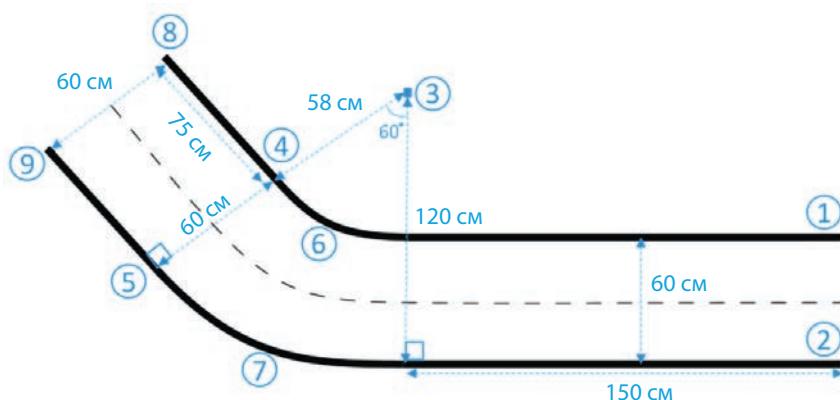
Трек можно проложить, используя ленту шириной около пяти сантиметров белого или близкого к белому цвета, которая будет играть роль границ на темном фоне. В виртуальной среде ширина границ трека составляет пять сантиметров. Выложите белую ленту на темной поверхности, обозначив границы трека. В роли такой ленты можно использовать, например, жемчужно-белую клейкую ленту (<https://oreil.ly/2x6dl>) или (менее липкий) малярный скотч (<https://oreil.ly/Hn0AD>).

#### Для дорожного полотна

В роли дорожного полотна можно использовать твердый пол темного цвета (деревянный, с ковровым покрытием, бетонный или асфальтовый). Последний лучше всего имитирует дорожное покрытие в реальном мире.

### Шаблон трека для AWS DeepRacer с одним поворотом

Этот простой шаблон трека состоит из двух прямолинейных сегментов, соединенных дугообразным сегментом, как показано на рис. 17.20. Модели, обученные на таком треке, заставляют автомобиль двигаться по прямой или делать повороты в одном направлении. Указанные угловые размеры поворотов предполагают, что при прокладке трека допускается использовать приблизительные размеры.



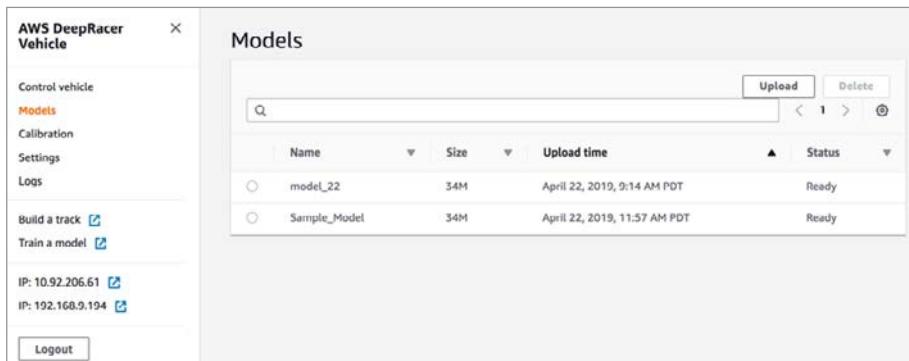
**Рис. 17.20.** Шаблон тестового трека

## Запуск модели на автомобиле AWS DeepRacer

Чтобы выполнить заезд на AWS DeepRacer в беспилотном режиме, нужно выгрузить в него как минимум одну модель AWS DeepRacer. Для этого выберите обученную модель в консоли AWS DeepRacer и загрузите все файлы модели из хранилища Amazon S3 на диск (локальный или сетевой), к которому можно получить доступ с компьютера. На странице модели есть удобная кнопка для загрузки модели.

Чтобы выгрузить модель в автомобиль, сделайте следующее:

1. В главной панели навигации в консоли устройства выберите пункт **Models** (Модели), как показано на рис. 17.21.



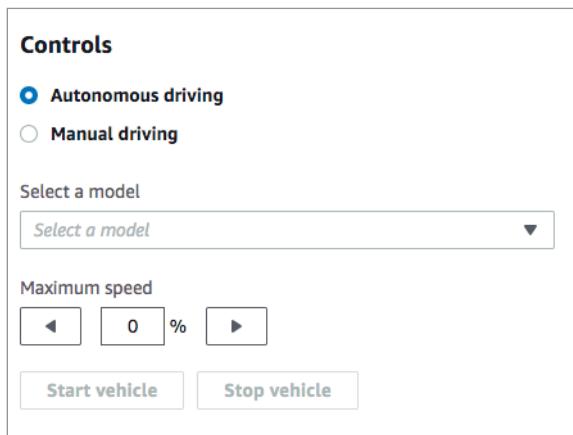
**Рис. 17.21.** Меню выгрузки модели в консоли AWS DeepRacer

2. На странице **Models** (Модели) щелкните на кнопке **Upload** (Выгрузить), которая находится над списком моделей.
3. В диалоге выбора файлов перейдите к диску или ресурсу, на который вы загрузили файлы модели, и выберите модель для выгрузки.
4. После успешной выгрузки модели она добавится в список моделей и может быть загружена для инференса.

## Автономное вождение AWS DeepRacer

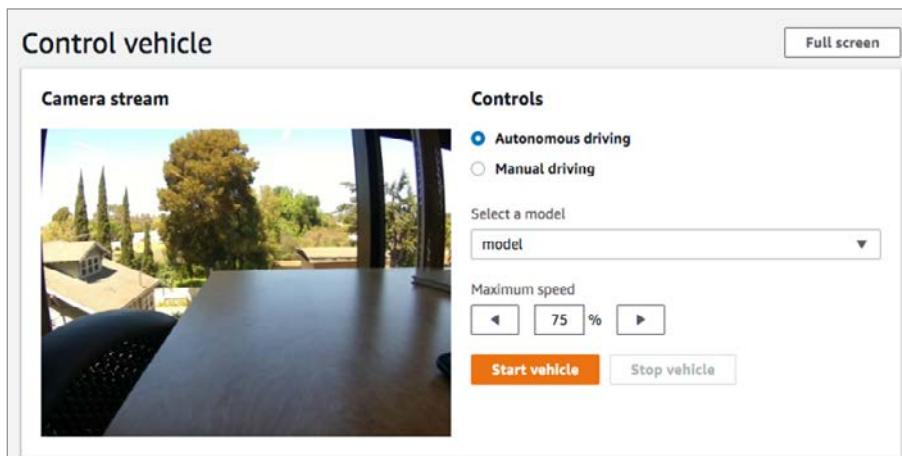
Чтобы начать заезд в беспилотном режиме, поставьте автомобиль на трек и выполните следующие действия:

1. Следуя инструкциям (<https://oreil.ly/OwzSz>), выполните вход в консоль управления автомобилем, а затем на странице **Control vehicle** (Управление автомобилем) в разделе **Controls** (Органы управления) выберите **Autonomous driving** (Автономное вождение), как показано на рис. 17.22.



**Рис. 17.22.** Выбор режима вождения AWS DeepRacer

2. В раскрывающемся списке **Select a model** (Выбор модели; рис. 17.23) выберите выгруженную модель, а затем щелкните на кнопке **Load model** (Загрузить модель). После этого модель будет загружена для инференса. Это займет около 10 секунд.
3. Отрегулируйте параметр **Maximum speed** (Максимальная скорость), чтобы сделать максимальную скорость во время заезда меньше той, что использовалась при обучении. (Это нужно, чтобы учесть трение о поверхность реальной трассы, которое делает невозможным движение с максимальной скоростью, использовавшейся во время обучения. Возможно, придется поэкспериментировать, чтобы найти оптимальную настройку.)



**Рис. 17.23.** Меню выбора модели в консоли AWS DeepRacer

4. Щелкните на кнопке **Start vehicle** (Запустить автомобиль), чтобы начать заезд в беспилотном режиме.
5. Наблюдайте за автомобилем на физическом треке или в потоковом плеере в консоли.
6. Чтобы остановить автомобиль, щелкните на кнопке **Stop vehicle** (Остановить автомобиль).

## Перенос Sim2Real

Симуляторы удобнее, чем реальный мир. В симуляторе легче создавать ситуации — например, столкновение машины с человеком или с другой машиной, которые нежелательны в реальном мире :). Но в большинстве случаев симулятор не имеет такой же визуальной точности, как реальный мир. Кроме того, он может не передавать или неточно передавать физику реального мира. Эти факторы могут повлиять на моделирование разных ситуаций в симуляторе, и в результате даже при большом успехе в симуляторе агент может допускать серьезные ошибки в реальном мире. Вот несколько распространенных подходов, помогающих преодолеть ограничения, характерные для симуляторов:

### *Идентификация системы*

Постройте математическую модель реальной среды и откалибруйте физическую систему, чтобы добиться максимальной реалистичности.

### *Адаптация моделируемой среды*

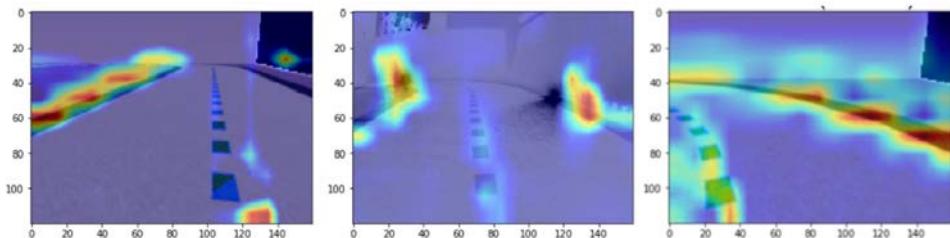
Отобразите моделируемую среду в реальность или наоборот, используя регуляризацию, генеративно-состязательные сети или перенос обучения.

### *Рандомизация моделируемой среды*

Создайте несколько моделируемых сред со случайными свойствами и обучайте модель на данных из всех этих сред.

В контексте DeepRacer моделируемая среда является приблизительным представлением реального мира, и физический движок симулятора может использовать некоторые ухищрения для адекватного моделирования возможной физики автомобиля. Но прелест глубокого обучения с подкреплением в том, что для идеального результата не требуется идеальное моделирование всего и вся. Чтобы ослабить влияние больших различий в восприятии, влияющих на автомобиль, можно предпринять два важных шага: а) вместо изображений RGB использовать черно-белые изображения, чтобы уменьшить различия в восприятии между симулятором и реальным миром, и б) намеренно использовать неглубокий механизм извлечения признаков, например неглубокую сверточную сеть с несколькими слоями. Это будет стимулировать сеть изучать не все моделируемые среды целиком, а только важные признаки. Например, на гоночной трассе авто-

мобиль учится удерживаться на треке, ориентируясь на белые полосы по краям. Взгляните на рис. 17.24, иллюстрирующий использование метода под названием GradCAM для создания тепловой карты наиболее важных частей изображения, помогающих автомобилю ориентироваться на треке.



**Рис. 17.24.** Тепловые карты GradCAM, помогающие автомобилю ориентироваться на треке

## Что изучать дальше

Вы можете продолжить приключение и поучаствовать в гоночных состязаниях в виртуальном и физическом мире. Есть несколько возможных вариантов.

### Лига DeepRacer

AWS DeepRacer проводит состязания в физическом мире на саммитах AWS и ежемесячные гонки в виртуальном мире. Чтобы поучаствовать в гонке на текущей виртуальной трассе и побороться за призы, посетите сайт лиги: <https://console.aws.amazon.com/deepracer/home?region=us-east-1#leaderboards>.

### AWS DeepRacer с расширенными возможностями

Мы понимаем, что опытные разработчики захотят получить больший контроль над моделируемыми средами, а также возможность самим определять архитектуры нейронных сетей. Для этого мы предлагаем среду на основе блокнота Jupyter (<https://oreil.ly/Xto3S>), с помощью которой вы сможете подготовить компоненты, необходимые для обучения собственных моделей для AWS DeepRacer.

### Олимпиада автопилотов с искусственным интеллектом

На конференции NeurIPS 2018 была учреждена олимпиада автопилотов с искусственным интеллектом (AI Driving Olympics, AI-DO), в которой принимают

участие беспилотные автомобили, управляемые ИИ. Первоначально в эти всемирные соревнования были включены задачи следования по полосе движения и управления автопарком на платформе Duckietown (рис. 17.25). Эта платформа состоит из двух компонентов: Duckiebot (миниатюрные роботизированные такси) и Duckietown (миниатюрная городская среда с дорогами и указателями). Перед Duckiebot стоит единственная задача — перевозить жителей городка Duckietown (уточек). Оборудованные крошечными камерами и выполняющие вычисления на Raspberry Pi, автомобильчики Duckiebot поставляются с простым в использовании ПО, которое позволяет всем, от старшеклассников до учёных-исследователей, относительно быстро запустить свой код. Это соревнование распространилось и на другие ведущие академические конференции по ИИ и теперь включает задачи, решаемые с использованием платформ Duckietown и DeepRacer.



**Рис. 17.25.** Городская среда Duckietown на олимпиаде AI Driving Olympics

## DIY Robocars

Митап DIY Robocars Meetup (<https://oreil.ly/SJOfT>), впервые состоявшийся в Окленде (Калифорния), теперь насчитывает более 50 групп по всему миру. Это веселые и увлекательные сообщества, присоединившись к которым можно поработать с другими энтузиастами в области беспилотных и роботизированных автомобилей. Многие из этих сообществ проводят ежемесячные соревнования и являются отличными площадками для участия в гонках с физическими моделями автомобилей.

## Roborace

Разбудите своего внутреннего Михаэля Шумахера. Автоспорт обычно считается состязанием моторов, но автогонки Roborace превратили его в состязание интеллектов. Roborace организует соревнования между электрическими беспилотными гоночными автомобилями, например элегантным Robocar, созданным Дэниелом Саймоном (Daniel Simon), известным своим футуристическим проектом Tron Legacy Light Cycle (рис. 17.26). Мы уже не говорим о миниатюрных автомобилях. Это полноразмерные электромобили весом 1350 кг и длиной 4,8 метра, способные развивать скорость до 320 км/ч. И, что самое заманчивое, для участия в гонках не нужно обладать обширными техническими знаниями.



**Рис. 17.26.** Роботизированный автомобиль Roborace, созданный Дэниелом Саймоном  
(источник: Roborace)

Настоящие звезды здесь — разработчики ИИ. Все команды получают одинаковые автомобили, поэтому все преимущество сосредоточено в софте ИИ. Автопилот может анализировать вывод бортовых датчиков — камер, радаров, лидаров, сонаров и ультразвуковых датчиков. Для высокопроизводительных вычислений автомобили оснащены мощной платформой NVIDIA DRIVE, обладающей вычислительной мощностью порядка нескольких терафлопс в секунду. Все, что от вас требуется, — это алгоритм, который позволит машине не выкатиться за пределы трека, избежать аварий и, конечно же, как можно быстрее приехать к финишу.

Roborace предоставляет симулятор гонок, в котором доступны точные виртуальные модели реальных автомобилей. Затем претендентам предлагается пройти виртуальные квалификационные заезды, победители которых уже могут участвовать в реальных гонках, в том числе на трассах Формулы Е. К 2019 году ведущие гоночные команды уже сократили отставание от лучших людей-гон-

щиков до 5–10 %. Скоро на пьедестале почёта будут стоять разработчики, победившие профессиональных гонщиков. Подобные соревнования способствуют появлению инноваций, и мы надеемся, что знания и опыт, полученные в этих гонках, можно будет перенести в индустрию беспилотных автомобилей, чтобы сделать их безопаснее, надежнее и в то же время эффективнее.

### От создателя

Развитие обучения с подкреплением в будущем.

Не будем рассматривать обучение с подкреплением в обычном смысле, когда предполагается, что обучение происходит с нуля, без использования предварительно обученных моделей. Мне нравится более широкий термин «интерактивное обучение». Он предполагает активное обучение, когда активно собираются данные на основе ограничений текущей обучаемой модели. Я представляю себе обучение с использованием физики в робототехнике, когда соединяются физика и классическое управление и добавляется обучение для повышения эффективности при сохранении стабильности и безопасности.

Анима Анандкумар (Anima Anandkumar), директор по исследованиям в NVIDIA, профессор Калтека

## Итоги

В предыдущей главе мы узнали, как обучить модель для беспилотного автомобиля, управляя им вручную внутри симулятора. В этой главе мы познакомились с обучением с подкреплением и узнали, как сформулировать главную проблему, волнующую каждого: как научить беспилотный автомобиль ездить. Мы использовали магию обучения с подкреплением, чтобы исключить человека из цепочки и научить автопилот водить в симуляторе. Но зачем ограничивать себя рамками виртуального мира? Мы перенесли полученные знания в физический мир и устроили заезд на настоящей машине. И на это потребовался всего час!

Глубокое обучение с подкреплением — относительно новая и очень интересная область. Некоторые из недавних расширений обучения с подкреплением открывают новые сферы, в которых его применение может привести к масштабной автоматизации. Например, иерархическое обучение с подкреплением позволяет моделировать принятие детализированных решений. Meta-RL позволяет моделировать принятие обобщенных решений в разных средах. Эти новшества приближают нас к имитации человеческого поведения. Неудивительно, что многие исследователи машинного обучения считают, что обучение с подкреплением способно приблизить нас к созданию универсального искусственного интеллекта и открыть возможности, которые раньше считались научной фантастикой.

## ПРИЛОЖЕНИЕ

---

# Краткое введение в сверточные нейронные сети

В названии нашей книги есть слова «реальные проекты», и это значит, что мы сфокусировались именно на практических аспектах глубокого обучения. Это приложение — справочник, а не полноценное исследование теоретических аспектов глубокого обучения. Чтобы погрузиться в некоторые из обсуждаемых тем, посмотрите раздел «Что изучать дальше» в конце приложения, где приводятся ссылки на другие источники.

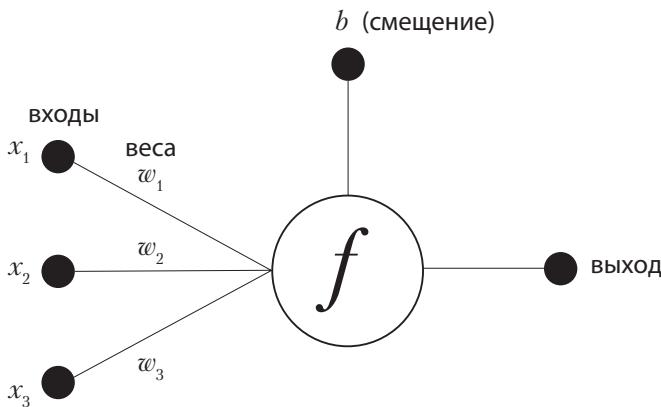
## Машинное обучение

- Машинное обучение изучает закономерности на основе данных и делает прогнозы на основе данных, которые модель машинного обучения прежде не видела.
- Есть три вида машинного обучения: обучение с учителем (с использованием размеченных данных), обучение без учителя (с использованием неразмеченных данных) и обучение с подкреплением (с выполнением действий и получением обратной связи от среды).
- К задачам обучения с учителем относятся классификация (результатом является одна из многих категорий или классов) и регрессия (результатом является числовое значение).
- Есть разные методы машинного обучения с учителем, включая наивный байесовский алгоритм, метод опорных векторов, деревья решений, алгоритм  $k$ -ближайших соседей, нейронные сети и др.

## Персептрон

- Персептрон, как показано на рис. П.1, представляет собой простейшую форму нейронной сети — однослойную нейронную сеть с одним нейроном.

- Персептрон вычисляет взвешенную сумму своих входов, то есть принимает входные значения, умножает каждое на соответствующий вес, прибавляет член смещения и генерирует числовой результат.



**Рис. П.1.** Пример персептрана

- Поскольку фактически персептрон представляет собой линейное уравнение, он хорошо моделирует только линейные или близкие к линейным зависимости. Для задачи регрессии прогноз можно представить в виде прямой, а для задачи классификации — в виде прямой, разделяющей плоскость на две части.
- Большинство практических задач носят нелинейный характер, и поэтому персептрон не может достаточно хорошо моделировать данные, что влечет за собой низкое качество прогнозов.

## ФУНКЦИИ АКТИВАЦИИ

- Функции активации преобразуют линейный вход в нелинейный выход. Сигмоида (рис. П.2) — это одна из разновидностей функций активации. Другими примерами функций активации могут служить гиперболический тангенс и ReLU, изображенная на рис. П.3.
- Для задач классификации желательно иметь на выходе вероятность каждой категории. Это можно реализовать с помощью сигмоидной функции (в задачах бинарной классификации) в концевом персептроне, которая преобразует значения в диапазон от 0 до 1 и потому лучше подходит для представления вероятностей.
- Функции активации помогают воспроизвести биологическую механику возбуждения нейронов, которая в простейшем случае может присутствовать

вать или отсутствовать, то есть входы активируют нейроны. Ступенчатая функция (функция Хэвисайда) — еще одна из форм функций активации, значение которой равно единице, когда нейрон активирован, и нулю в противном случае. Один из недостатков ступенчатой функции — потеря величины. Функция ReLU, напротив, возвращает величину активации, если персептрон активен, и ноль в противном случае. Из-за его простоты и низких вычислительных требований эту функцию предпочтительнее использовать для вывода данных, не связанных с вероятностями.



Рис. П.2. График сигмоидной функции

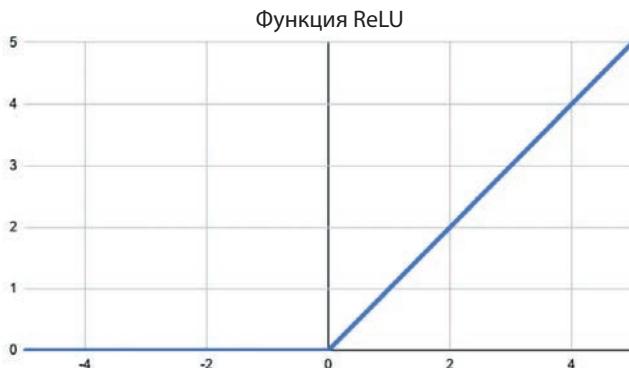
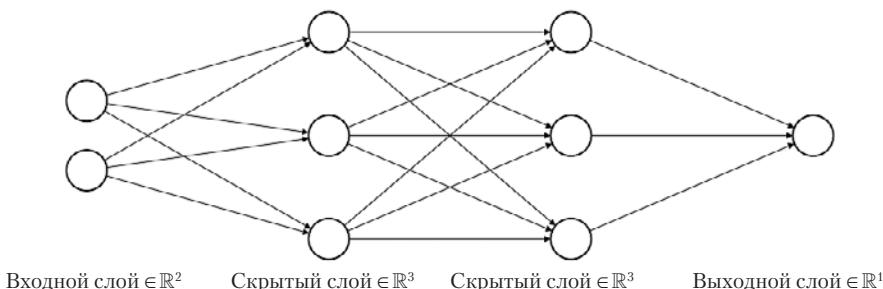


Рис. П.3. График функции ReLU

## Нейронные сети

- Комбинация персептрона с нелинейной функцией активации улучшает прогнозирующую способность. Объединение нескольких персептронов с нелинейными функциями активации еще больше улучшает ее.

- Нейронная сеть, как показано на рис. П.4, состоит из нескольких слоев, каждый из которых содержит несколько персепtronов. Слои соединяются последовательно, выход каждого предыдущего слоя передается на вход следующего. Эта передача данных изображена в виде соединений, при этом каждый нейрон в предыдущем слое имеет связь с каждым нейроном в следующем слое (поэтому такие слои называют полносвязанными). Каждому из этих соединений присвоен свой коэффициент — его вес.



**Рис. П.4.** Пример многослойной нейронной сети

- Каждый нейрон хранит связанные с ним веса и смещения, а на выходе — нелинейную (необязательно) функцию активации.
- Поскольку информация распространяется в одном направлении, без каких-либо циклов, эта сетевая структура также называется нейронной сетью прямого распространения или многослойной сетью прямого распространения.
- Слои между входом и выходом скрыты от конечного пользователя и потому называются скрытыми слоями.
- Нейронные сети обладают мощной способностью моделировать закономерности в данных (то есть представлять базовые отношения между данными и их метками). Универсальная теорема аппроксимации (*Universal approximation theorem*) утверждает, что нейронная сеть с единственным скрытым слоем может представлять любую непрерывную функцию в определенном диапазоне, действуя как универсальный аппроксиматор. То есть технически однослойная сеть может моделировать любую существующую задачу. Но на практике такие сети плохо реализуемы из-за необходимости иметь огромное количество нейронов для моделирования задач. Альтернативное решение — построить сеть с несколькими слоями, содержащую меньше нейронов в каждом слое. Это решение хорошо зарекомендовало себя на практике.
- Выход одного скрытого слоя — это результат применения нелинейной функции к линейной комбинации входов. Выход второго скрытого слоя является

результатом применения нелинейной функции к линейной комбинации входов, которые сами являются результатом применения нелинейной функции к линейной комбинации их входов. Каждый слой дополнительно опирается на прогнозирующую способность предыдущего слоя. Поэтому чем больше слоев, тем лучше способность нейронной сети моделировать реальные нелинейные зависимости. С каждым новым слоем глубина сети увеличивается, поэтому сети с большим количеством слоев также называют *глубокими нейронными сетями*.

## Обратное распространение ошибки

- Обучение сети включает итеративное изменение весов и смещений до тех пор, пока сеть не начнет делать прогнозы с минимальной ошибкой.
- Обратное распространение ошибки (backpropagation) помогает обучать нейронные сети. Этот процесс включает в себя вычисление прогноза, измерение, насколько далеким получился прогноз от истинного значения, и распространение ошибки по сети в обратном направлении для коррекции весов, чтобы уменьшить величину ошибки в последующих итерациях. Этот процесс повторяется до тех пор, пока ошибка сети не перестанет уменьшаться.
- Нейронные сети обычно инициализируются случайными весами.
- Качество модели оценивается с помощью функций потерь. Для задач регрессии (с числовыми прогнозами) в роли функции потерь обычно используется среднеквадратичная ошибка (MSE). Из-за возведения величины ошибки в квадрат применение среднеквадратичной ошибки лучше справляется с исправлением множества мелких ошибок, чем нескольких больших. Для задач классификации (предсказания категорий) в роли функции потерь обычно используется перекрестная энтропия. Точность используется больше как понятный человеку показатель качества, но для обучения сети применяется функция перекрестной энтропии как более чувствительная.

## Недостатки нейронных сетей

- Слой нейронной сети с  $n$  нейронами и  $i$  входами требует  $n \times i$  весов и вычислительных операций, потому что каждый нейрон подключен к каждому входу. Входное изображение обычно содержит большое количество пикселей. Для обработки изображения с шириной  $w$ , высотой  $h$  и  $c$  каналами потребуется выполнить  $w \times h \times c \times n$  вычислительных операций. Для обработки стандартного изображения, снятого на камеру смартфона (обычно  $4032 \times 3024$  с 3 каналами цвета — красный, зеленый и синий), если первый

скрытый слой содержит 128 нейронов, потребуется более 4 миллиардов весов и вычислительных операций. Это обстоятельство делает непрактичным применение нейронных сетей для большинства задач компьютерного зрения. В идеале желательно преобразовать изображение в более компактное представление, которое затем можно было бы классифицировать с помощью нейронной сети.

- Пиксели изображения обычно связаны отношениями с другими пикселями, находящимися поблизости. Если зафиксировать эту связь, можно лучше понять содержание изображения. Но при передаче изображения в нейронную сеть связь между пикселями теряется, так как изображение превращается в одномерный массив.

## Желаемые свойства классификатора изображений

- Эффективность в вычислительном отношении.
- Фиксация отношений между различными частями изображения.
- Устойчивость к различиям в местоположениях, размерах и углах поворота объектов, присутствующих на изображении, и другим факторам, таким как шум. То есть классификатор изображений должен быть инвариантным к геометрическим преобразованиям (которые часто упоминаются с использованием таких терминов, как трансляционная инвариантность, вращательная инвариантность, позиционная инвариантность, сдвиг-инвариантность, масштабная инвариантность и т. д.).
- Сверточная нейронная сеть (Convolutional Neuron Network, CNN) удовлетворяет многим из этих желаемых свойств (одним по своей природе, другим — с увеличением объема данных).

## Свертка

- Сверточный фильтр — это матрица, которая скользит по изображению (по рядам пикселей, слева направо) и умножается на пиксели, перекрываемые фильтром для получения результата. Такие фильтры широко используются в Photoshop для создания эффекта размытия, повышения резкости, яркости, затемнения и т. д.
- Сверточные фильтры используют для обнаружения вертикальных, горизонтальных и диагональных краев. То есть, когда на изображении присутствуют резкие границы, для них фильтры генерируют более высокие значения на выходе. Как побочный эффект они «фильтруют» вход, если тот не содержит границы.

- Первоначально сверточные фильтры создавались вручную для выполнения определенной задачи. В контексте нейронных сетей сверточные фильтры можно использовать в роли строительных блоков, параметры которых (веса матриц) могут вычисляться автоматически в обучающем конвейере.
- Сверточный фильтр активируется каждый раз, когда обнаруживает входной шаблон, которому был обучен. Поскольку фильтр выполняет обход всего изображения, он может найти свой шаблон в любом месте изображения. Благодаря этому сверточные фильтры делают классификатор инвариантным к местоположению объектов.

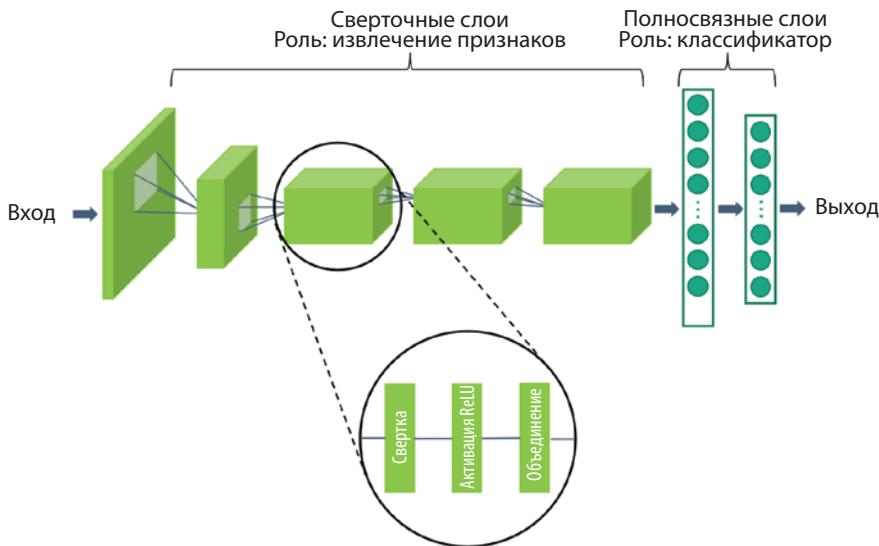
## Объединение

- Операция объединения (pooling) уменьшает пространственный размер входных данных. Она просматривает наименьшие входы и выполняет операцию агрегирования, сокращая вывод до одного числа. Обычно в роли операции агрегирования используется поиск максимума или среднего всех значений, также известных как *объединение по максимальному значению* (max pooling) и *объединение по среднему значению* (average pooling) соответственно. Например, операция объединения по максимальному значению  $2 \times 2$  заменит каждую входную (неперекрывающуюся) подматрицу  $2 \times 2$  одним значением на выходе. Подобно свертке, окно объединения пересекает всю входную матрицу, объединяя блоки из нескольких значений в отдельные значения. Если входная матрица имеет размер  $2N \times 2N$ , то после применения операции объединения с окном  $2 \times 2$  на выходе получится матрица  $N \times N$ .
- В примере выше результат будет одинаковым независимо от того, где матрица  $2 \times 2$  обнаружит максимум.
- Операция объединения по максимальному значению используется чаще, чем объединение по среднему, потому что действует как подавитель шума. Поскольку в каждой операции используется только максимальное значение, более низкие значения, которые обычно являются шумом, игнорируются.

## Структура сверточной сети

- В общем случае сверточная сеть (как показано на рис. П.5) состоит из двух частей — модуля извлечения признаков, за которым следует классификатор. Модуль извлечения признаков сжимает входное изображение до более компактного представления в форме вектора признаков, с которым затем работает классификатор.

- Модуль извлечения признаков в сверточной сети — это повторяющаяся структура из сверточных, активационных и объединяющих слоев. Исходные данные проходят через эту структуру последовательно, слой за слоем.



**Рис. П.5.** Обобщенное представление сверточной сети

- После преобразования входные данные передаются в слои классификации, которые на самом деле являются просто нейронными сетями. В этом контексте нейронную сеть классификации обычно называют полносвязным слоем.
- Так же как в нейронных сетях, нелинейность, вносимая функциями активации, помогает сверточной сети изучать сложные признаки. Чаще всего на практике используется функция активации ReLU.
- Как подмечено на практике, более ранние слои (то есть находящиеся ближе к входным слоям) обнаруживают более простые элементы, такие как кривые и границы. При обнаружении совпадения слой активируется и передает сигнал последующим слоям. Конечные слои объединяют эти сигналы и обнаруживают более сложные признаки, такие как человеческий глаз, нос или уши. В итоге эти сигналы могут складываться и использоваться для обнаружения всего объекта, такого как человеческое лицо.
- Сверточные сети обучаются с использованием обратного распространения ошибки, как и многослойные сети персептронов.
- В отличие от традиционного машинного обучения, это позволяет автоматизировать выбор признаков. Кроме того, сжатие входных данных до менее

объемных векторов признаков повышает эффективность процесса с точки зрения вычислений. И то и другое — важные аргументы в пользу глубокого обучения.

## Что изучать дальше

Темы в этой книге рассмотрены достаточно полно, но если вы все же решите глубже вникнуть в основы, то настоятельно рекомендуем следующие источники:

<https://www.deeplearning.ai>

Этот набор курсов глубокого обучения, разработанных Эндрю Ыном, охватывает широкий круг тем, включая теоретические основы каждой концепции.

<https://course.fast.ai>

Этот курс Джереми Ховарда и Рэйчел Томас предлагает более практический подход к глубокому обучению с использованием PyTorch в качестве основной среды глубокого обучения. Библиотека fast.ai, используемая в этом курсе, помогла многим студентам создать высокоэффективные модели, написав всего несколько строк кода.

«*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*»<sup>1</sup>

В этой книге Орельен Жерон (Aurélien Géron) рассматривает машинное обучение и такие темы, как метод опорных векторов, деревья решений, случайные леса и т. д., а затем переходит к глубокому обучению, рассказывая о сверточных и рекуррентных сетях. Он объясняет теоретические основы, а также приводит практические примеры применения этих методов.

---

<sup>1</sup> Жерон Орельен. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. 2-е изд. — Примеч. пер.

---

# Об авторах

## Основные авторы

**Анирад Коул (Anirudh Koul)** — известный эксперт в области ИИ, спикер UN/TEDx и бывший старший научный сотрудник в Microsoft AI & Research, где он основал Seeing AI — после iPhone считается самой используемой технологией среди слабовидящих. Анирад возглавляет отдел искусственного интеллекта и исследований в компании Aira, признанной журналом *Time* одной из лучших компаний 2018 года в своей сфере. Он раскрыл новые возможности для миллиардов пользователей и имеет более чем десятилетний опыт прикладных исследований с использованием петабайтных наборов данных. Активно занимается разработкой технологий с использованием методов ИИ в сфере дополненной реальности, робототехники, речи, повышения производительности и доступности. Его работы, представленные на саммите AI for Good, которые в IEEE назвали «судьбоносными», удостоены наград от CES, FCC, MIT, Cannes Lions и Американского совета общества слепых; демонстрировались на мероприятиях ООН, Всемирного экономического форума, Белого дома, Палаты лордов, Netflix и National Geographic и заслужили похвалу мировых лидеров, включая Джастина Трюдо (Justin Trudeau) и Терезу Мэй (Theresa May).

**Сиддха Ганджу (Siddha Ganju)**, исследователь ИИ, которого журнал *Forbes* включил в свой список «30 Under 30», является архитектором беспилотных автомобилей в NVIDIA. В качестве советника по ИИ в NASA FDL она помогла построить автоматизированный конвейер обнаружения метеоритов для проекта CAMS в NASA, с помощью которого была обнаружена комета. Ранее, работая в Deep Vision, Сиддха разрабатывала модели глубокого обучения для краевых устройств с ограниченными ресурсами. Занималась созданием самых разных систем, от поиска ответов на визуальные вопросы до генеративно-состязательных сетей и сбора информации из петабайтных массивов данных в CERN, сведения о которых публиковались на ведущих конференциях, включая CVPR и NeurIPS. Была членом жюри нескольких международных технических конкурсов, включая CES.

**Мехер Казам (Meher Kasam)** — опытный разработчик программного обеспечения, приложениями которого ежедневно пользуются десятки миллионов людей. В настоящее время занимается разработкой программного обеспечения для iOS в компании Square, а прежде работал в Microsoft и Amazon. Участвовал в создании ряда приложений для iPhone, от Square Point of Sale до Bing. В период работы в Microsoft руководил разработкой мобильного приложения Seeing AI, получившего широкое признание и награды от Mobile World Congress, CES, FCC, Американского совета общества слепых и многих других организаций. Хакер в душе, склонный к быстрому созданию прототипов, он выиграл несколько хакатонов и воплотил найденные решения во многие широко используемые продукты. Также является судьей международных конкурсов, в том числе Global Mobile Awards и Edison Awards.

## Приглашенные авторы

### Глава 17

**Сунил Маллья (Sunil Mallya)** — ведущий специалист по глубокому обучению в AWS, специализирующийся на глубоком обучении и обучении с подкреплением. Сунил работает с клиентами AWS в рамках различных инициатив по трансформации и инновациям в разных отраслях, создавая модели для передовых приложений машинного обучения. В свое время руководил научными разработками в AWS DeepRacer. Перед тем как присоединиться к AWS, Сунил был соучредителем компании Neon Labs, специализирующейся на анализе изображений с использованием методов машинного обучения и нейробиологии, а также на выработке рекомендаций по кадрам из видео. Принимал участие в разработке крупномасштабных систем с малой задержкой в Zynga и страстно увлечен бессерверными вычислениями. Имеет степень магистра информатики Брауновского университета.

### Глава 16

**Адитья Шарма (Aditya Sharma)** — руководитель проектов в Microsoft, где он работает в группе Azure Autonomous Driving. Целью его работы является оказание помощи компаниям, занимающимся созданием беспилотных транспортных средств, в масштабировании их деятельности с применением облачных технологий и за счет этого значительного сокращения времени выхода на рынок. Является ведущим руководителем проекта Project Road Runner, в рамках которого был создан сборник рецептов по автоматическому управлению автомобилями. Он также возглавляет отдел глубокого обучения и робототехники в Microsoft Garage. Адитья имеет ученую степень, полученную в институте робототехники университета Карнеги — Меллона.

## Глава 16

**Митчелл Сприн (Mitchell Spryn)** окончил университет Алабамы, где получил степени бакалавра электротехники и физики. Во время учебы в университете Митчелл участвовал в различных исследовательских проектах, начиная от беспроводной передачи энергии и заканчивая автономной робототехникой. В настоящее время работает в Microsoft, где специализируется на распределенных реляционных базах данных. Помимо работы с базами данных Митчелл продолжает участвовать в проектах в области робототехники, таких как симулятор AirSim и сборник рецептов *Autonomous Driving Cookbook* (<https://oreil.ly/CUU1v>).

## Глава 15

**Сэм Стерквал (Sam Sterckval)**, бельгийский инженер-электронщик, во время учебы в колледже открыл в себе вторую страсть (помимо электроники) — искусственный интеллект. После окончания учебы Сэм стал соучредителем компании Edgise, в которой сумел объединить эти две страсти, создавая краевые устройства, способные эффективно выполнять сложные модели ИИ. Сэм, которого в Бельгии считают крупным специалистом по ИИ для краевых устройств, помогает компаниям создавать когнитивные решения, характеризующиеся минимальной задержкой, высокой защищенностью и невысокой стоимостью.

## Глава 10

**Зайд Аляфеи (Zaid Alyafeai)** известен своими популярными блогами и блокнотами в Google Colab на TensorFlow.js, а также использованием технологии глубокого обучения в браузере для реализации интересных сценариев. Он также пишет статьи для официального блога TensorFlow на сайте Medium и участвует во множестве проектов на GitHub, которые все вместе получили более 1000 звезд. Когда Зайд не занимается созданием современных веб-приложений, то проводит исследования в области ИИ в качестве аспиранта в KFUPM в Саудовской Аравии. Автор справочника по передовым методам интеграции ([https://oreil.ly/TW\\_kA](https://oreil.ly/TW_kA)).

---

# Иллюстрация на обложке

На обложке книги «Искусственный интеллект и компьютерное зрение. Реальные проекты на Python, Keras и TensorFlow» изображен американский сокол-сапсан (*Falco peregrinus anatum*), или утиный ястреб, подвид сапсана, который в настоящее время обитает в Скалистых горах. Этот подвид вымер в восточной части Северной Америки, но, к счастью, существуют жизнеспособные гибридные популяции, выведенные в результате повторного заселения вида.

Соколы-сапсаны во всем мире, независимо от подвидов, имеют общие характеристики: это большие и быстро летающие хищные птицы с темным оперением на голове и крыльях, светлым пестрым брюхом, желтым клювом и лапами и большими глазами с характерной вертикальной полосой черного цвета под глазами. Соколов легко отличить издалека по изогнутым, резко очерченным крыльям. Самки заметно крупнее самцов, но и те и другие в среднем размером с американскую ворону и издают отчетливый резкий крик «как-как-как», защищая территорию или птенцов.

Сапсаны обосновались на всех континентах, кроме Антарктиды. Их видовой успех обусловлен способностью адаптироваться к роли ловчих птиц, а также к различным условиям гнездования и охоты. Самая известная характеристика сапсана — его охотничья привычка пикировать с большой высоты на птицу, находящуюся на земле или около земли. В момент, когда сапсан настигает добычу, он развивает скорость 200 миль в час (330 км/час), а сила удара такова, что убивает добычу мгновенно. Это не только самая быстрая птица на земле, но и самый быстрый представитель животного мира, и именно поэтому сапсан является любимой птицей сокольников всего мира.

Соколы-сапсаны стали символом экологического движения США в 1960-х годах, когда широкое использование пестицида ДДТ уничтожило птиц на большей части ареала их обитания до того, как это химическое вещество было запрещено (ДДТ попадал в организм птиц через их добычу и негативно воздействовал на эмбриональное развитие потомства). В 1970 году Агентство по охране окружающей среды США объявило этого сокола вымирающим видом. Однако после запрета ДДТ, за которым последовала программа разведения в неволе, которая вернула этот вид в их прежний ареал в восточной части США, вид восстановился. После установки ящиков для гнезд в верхних частях городских небоскребов популяция быстро закрепились, потому что соколы могли взращивать птенцов, используя в пищу голубей и других многочисленных городских птиц. В 1999 году сокол-сапсан был исключен из списка видов, находящихся под угрозой исчезновения. В настоящее время американскому сапсану присвоен природоохранный статус «вызывающий наименьшее беспокойство», тем не менее многие животные, изображенные на обложках книг издательства O'Reilly, находятся под угрозой исчезновения; они все важны для нашего мира.

Иллюстрацию для обложки нарисовала Карен Монтгомери (Karen Montgomery) на основе черно-белой гравюры из журнала *British Birds*.