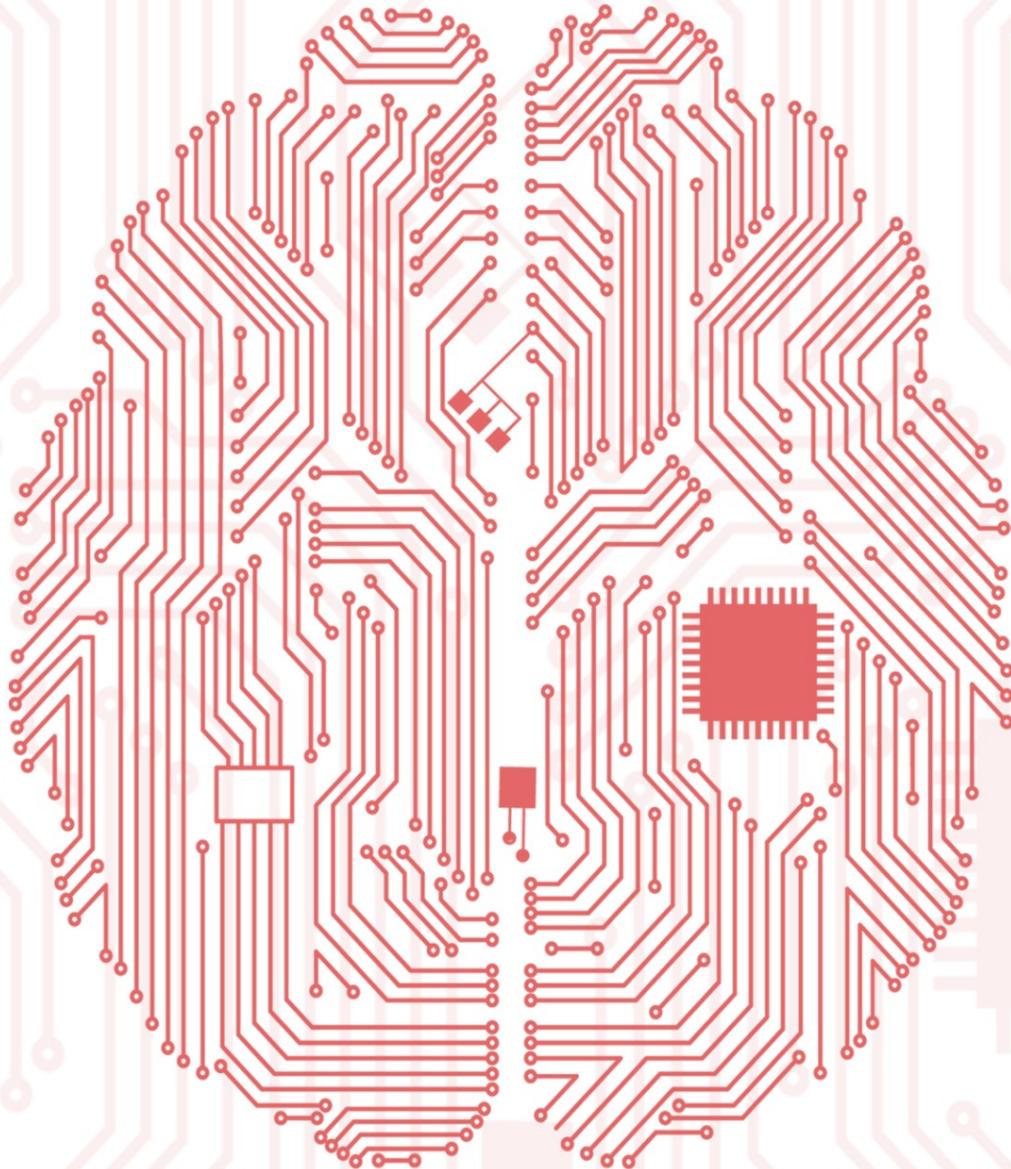


DEEP LEARNING FOR COMPUTER VISION



WITH PYTHON

Dr. Adrian Rosebrock

 PyImageSearch

Глубокое обучение для компьютерного зрения с Питон

Набор «Практик»

Доктор Адриан Роузброк

1-е издание (1.2.1)

Copyright с 2017 Адриан Роузброк, PyImageSearch.com

ИЗДАНО PYIMAGESearch

PYIMAGESearch.COM

Содержание этой книги, если не указано иное, защищено авторским правом с Adrian Rosebrock, 2017, PyImageSearch.com. Все права защищены. Книги, подобные этой, стали возможными благодаря времени, вложенному авторами. Если вы получили эту книгу, но не купили ее, рассмотрите возможность создания будущих книг , купив копию на странице <https://www.pyimagesearch.com/deep-learning-computer-vision> .
[питон-книга/](#) сегодня.

Первый тираж, сентябрь 2017 г.

Моему отцу Джо; моя жена Триша;
и семейные гончие, Джози и Джемма.
Без их постоянной любви и поддержки
эта книга была бы невозможна.

Содержание

1	Введение .	13
2	Введение .	15
3	Тренировочные сети с использованием нескольких графических процессоров .	17
3.1	Сколько графических процессоров мне нужно?	17
3.2	Повышение производительности при использовании нескольких графических процессоров	18
3.3	Резюме	19
4	Что такое ImageNet? .	21
4.1	Набор данных ImageNet	21
4.1.1	ILSVRC .	21
4.2	Получение ImageNet 4.2.1	23
Запрос доступа к вызову ILSVRC.		23
4.2.2	Программная загрузка изображений.	23
4.2.3	Использование внешних служб.	24
4.2.4	Комплект для разработки ImageNet .	24
4.2.5	Вопросы авторского права ImageNet .	25
4.3	Резюме	27
5	Подготовка набора данных ImageNet .	29
5.1	Понимание файловой структуры ImageNet 5.1.1	29
«Тестовый» каталог ImageNet .		30
5.1.2	Директория ImageNet «поезд» .	31
5.1.3	Директория ImageNet «val» .	32
5.1.4	Директория ImageNet «ImageSets» .	33

5.1.5 Директория ImageNet «DevKit»	34
файл конфигурации	37
5.2 Создание набора данных ImageNet	
5.2.1 Ваш первый	37
5.2.2 Наша вспомогательная утилита ImageNet	42
5.2.3 Создание файлов списка и среднего значения	46
5.2.4 Создание компактных файлов записи	50
5.3 Резюме	52
6 Обучение AlexNet на ImageNet	53
6.1 Внедрение AlexNet 6.2	54
Обучение AlexNet	58
6.2.1 Что насчет тренировочных участков?	59
6.2.2 Реализация сценария обучения. 6.3	60
Оценка AlexNet 6.4	65
Эксперименты AlexNet 6.4.1	67
AlexNet: эксперимент №1 .	68
6.4.2 AlexNet: Эксперимент №2.	70
6.4.3 AlexNet: эксперимент №3.	71
6.5 Резюме	74
7 Обучение VGGNet на ImageNet	75
7.1 Реализация VGGNet 7.2	76
Обучение VGGNet 7.3 Эксперименты	81
7.3 VGGNet 7.5 Резюме	85
86	88
8 Обучение GoogLeNet работе с ImageNet	89
8.1 Понимание GoogLeNet 8.1.1	89
Начальный модуль .	90
8.1.2 Архитектура GoogLeNet	90
8.1.3 Внедрение GoogLeNet.	91
8.1.4 Обучение GoogLeNet. 8.2 Оценка	95
GoogLeNet 8.3 Эксперименты GoogLeNet №1 .	99
99	99
100	100
8.3.2 GoogLeNet: Эксперимент №2.	101
8.3.3 GoogLeNet: эксперимент №3 .	102
8.4 Резюме	103
9 Обучение ResNet на ImageNet	105
9.1 Понимание ResNet 9.2	105
Внедрение ResNet 9.3	106
Обучение ResNet Оценка	112
9.4	116

9.5 Эксперименты ResNet	116
9.5.1 ResNet: эксперимент №1 .	116
9.5.2 ResNet: Эксперимент №2.	116
9.5.3 ResNet: эксперимент №3.	117
9.6 Резюме	120
10 Обучение SqueezeNet на ImageNet	121
10.1 Понимание SqueezeNet 10.1.1	121
Пожарный модуль .	121
10.1.2 Архитектура SqueezeNet.	123
10.1.3 Реализация SqueezeNet.	124
10.2 Обучение SqueezeNet	128
10.3 Оценка SqueezeNet 10.4	132
Эксперименты SqueezeNet 10.4.1	132
SqueezeNet: эксперимент №1 .	132
10.4.2 SqueezeNet: Эксперимент №2.	134
10.4.3 SqueezeNet: эксперимент №3.	135
10.4.4 SqueezeNet: эксперимент №4.	136
10.5 Резюме	139
11 Практический пример: Распознавание эмоций	141
11.1 Задача распознавания выражений лица Kaggle 11.1.1 Набор данных FER13.	141
11.1.2 Создание набора данных FER13.	142
11.2 Реализация VGG-подобной сети 11.3	147
Обучение нашего распознавателя выражений лица	150
11.3.1 EmotionVGGNet: эксперимент №1 .	153
11.3.2 EmotionVGGNet: Эксперимент №2.	153
11.3.3 EmotionVGGNet: эксперимент №3.	154
11.3.4 EmotionVGGNet: эксперимент №4.	155
11.4 Оценка нашего распознавателя выражения лица 11.5	157
Обнаружение эмоций в режиме реального времени	159
11.6 Резюме	163
12 Практический пример: коррекция ориентации изображения	165
12.1 Набор данных CVPR для помещений	165
12.1.1 Создание набора данных .	166
12.2 Извлечение признаков	170
12.3 Обучение классификатора коррекции ориентации	173
12.4 Корректировка ориентации 12.5 Резюме	175
	177
13 Практический пример: идентификация автомобиля	179
13.1 Набор данных Stanford Cars	179
13.1.1 Создание набора данных Stanford Cars.	180

13.2 Тонкая настройка VGG в наборе данных Stanford Cars	187
13.2.1 Тонкая настройка VGG: эксперимент №1.	192
13.2.2 Точная настройка VGG: Эксперимент №2.	193
13.2.3 Тонкая настройка VGG: Эксперимент №3.	194
13.3 Оценка нашего классификатора транспортных	195
средств 13.4 Визуализация результатов классификации	197
транспортных средств 13.5 Резюме	201
14 Практический пример: прогнозирование возраста и пола	203
14.1 Этика гендерной идентификации в машинном обучении 14.2	203
Набор данных Adience 14.2.1 Создание набора данных Adience.	204
.	205
.	219
.	221
.	224
.	227
.	230
.	230
14.7.2 Гендерные результаты.	231
14.8 Визуализация результатов	233
14.8.1 Визуализация результатов изнутри аудитории	234
14.8.2 Понимание выравнивания лица	238
14.8.3 Применение предсказания возраста и пола к вашим собственным изображениям	240
14.9 Резюме	244
15 более быстрых R-CNN	247
15.1 Обнаружение объектов и глубокое обучение	247
15.1.1 Измерение производительности детектора объектов :	248
15.2 (более быстрая) архитектура R-CNN	250
15.2.1 Краткая история R-CNN	250
15.2.2 Базовая сеть.	254
15.2.3 Анкеры.	255
15.2.4 Региональная сеть предложений (RPN)	257
15.2.5 Объединение области интереса (ROI)	258
15.2.6 Сверточная нейронная сеть на основе регионов	259
15.2.7 Полный конвейер обучения	260
15.3 Резюме	260
16 Обучение более быстрой R-CNN с нуля	261
16.1 Набор данных о дорожных знаках	261
LISA 16.2 Установка API обнаружения объектов TensorFlow	262
16.3 Обучение быстрой R-CNN 16.3.1 Структура каталога	263
проекта	263
16.3.2 Конфигурация	265
16.3.3 Класс аннотаций TensorFlow.	267

16.3.4 Создание набора данных LISA + TensorFlow.	269
16.3.5 Важный этап подготовки к обучению .	274
16.3.6 Настройка более быстрого R-CNN.	275
16.3.7 Обучение более быстрой R-CNN.	280
16.3.8 Предложения по работе с TFOD API.	282
16.3.9 Экспорт графика замороженной модели .	286
16.3.10 Более быстрый R-CNN для изображений и видео.	286
16.4 Резюме	290
17 детекторов одиночного выстрела (SSD)	293
17.1 Общие сведения об однократных детекторах (SSD)	293
17.1.1 Мотивация.	293
17.1.2 Архитектура.	294
17.1.3 Мультибокс, Приоры и Фиксированные Приоры.	295
17.1.4 Методы обучения.	296
17.2 Резюме	297
18 Обучение SSD с нуля	299
18.1 Набор данных автомобиля	299
18.2 Обучение SSD 18.2.1	300
Структура и конфигурация каталогов .	300
18.2.2 Создание набора данных о транспортном средстве .	302
18.2.3 Обучение SSD.	307
18.2.4 Результаты SSD .	310
18.2.5 Возможные проблемы и ограничения.	311
18.3 Резюме	312
19 Выводы	313
19.1 Куда теперь?	314

Сопутствующий веб-сайт

Спасибо, что приобрели книгу «Глубокое обучение компьютерному зрению с помощью Python» ! В дополнение к этой книге я создал сопутствующий веб-сайт, который включает в себя:

- Актуальные инструкции по настройке среды разработки • Инструкции по использованию предварительно настроенной виртуальной машины Ubuntu VirtualBox и Образ машины Amazon (AMI)
- Дополнительный материал , который я не смог уместить в этой книге •

Часто задаваемые вопросы (FAQ) и предлагаемые исправления и решения Кроме того, вы можете использовать функцию «Проблемы» на сопутствующем веб-сайте, чтобы сообщать о любых ошибках, опечатках или проблемах, с которыми вы сталкиваетесь, когда работа по книге. Я не ожидаю много проблем; тем не менее, это совершенно новая книга, поэтому я и другие читатели были бы признательны за сообщение о любых проблемах, с которыми вы столкнетесь. Оттуда я могу обновлять книгу и избавляться от ошибок.

Чтобы создать учетную запись на сопутствующем веб-сайте, просто используйте эту ссылку: <http://pyimg.co/fnkxk> . Потратите секунду, чтобы создать учетную запись сейчас, чтобы иметь доступ к дополнительным материалам.

пока вы работаете с книгой.

1. Введение

Добро пожаловать в Практический пакет глубокого обучения для компьютерного зрения с Python! Этот том должен стать следующим логическим шагом в вашем глубоком обучении компьютерному зрению после завершения Starter Bundle.

На этом этапе вы должны иметь четкое представление об основах параметризованного обучения, нейронных сетей и сверточных нейронных сетей (CNN). Вы также должны чувствовать себя относительно комфортно, используя библиотеку Keras и язык программирования Python для обучения собственных сетей глубокого обучения.

Цель набора «Практик» — развить ваши знания, полученные из набора «Начальный», и представить более продвинутые алгоритмы, концепции и профессиональные приемы — эти приемы будут рассмотрены в трех отдельных частях книги.

Первая часть будет посвящена методам, которые так или иначе используются для повышения точности вашей классификации. Один из способов повысить точность вашей классификации — применить методы трансферного обучения, такие как тонкая настройка или использование вашей сети в качестве средства извлечения признаков.

Мы также рассмотрим ансамблевые методы (т. е. обучение нескольких сетей и объединение результатов) и то, как эти методы могут дать вам хороший прирост классификации с небольшими дополнительными усилиями. Методы регуляризации, такие как увеличение данных, используются для создания дополнительных обучающих данных — почти во всех ситуациях увеличение данных улучшает способность вашей модели к обобщению.

Более продвинутые алгоритмы оптимизации, такие как Adam [1], RMSprop [2] и другие, также могут использоваться для некоторых наборов данных, чтобы снизить потери. После того, как мы рассмотрим эти методы, мы рассмотрим оптимальный путь их применения, чтобы гарантировать, что вы получите максимальную пользу с наименьшими усилиями.

Затем мы переходим ко второй части пакета Practitioner Bundle, в которой основное внимание уделяется большим наборам данных и более экзотическим сетевым архитектурам. До сих пор мы работали только с наборами данных, которые поместились в основную память нашей системы, но что, если наш набор данных слишком велик, чтобы поместиться в ОЗУ? Что мы делаем тогда? Мы рассмотрим этот вопрос в главе ?? когда мы работаем с HDF5.

Учитывая, что мы будем работать с большими наборами данных, мы также сможем обсудить более сложные сетевые архитектуры с использованием AlexNet, GoogLeNet, ResNet и более глубоких вариантов VGGNet. Эти сетевые архитектуры будут применяться к более сложным наборам данных и соревнованиям, включая

Kaggle: испытание на распознавание собак против кошек [3], а также испытание cs231n Tiny ImageNet [4], точно такое же задание, в котором соревнуются студенты Stanford CNN. 25-е место в таблице лидеров Kaggle Dogs vs. Cats и первое место в соревновании cs231n для нашего типа техники.

В заключительной части этой книги рассматриваются приложения глубокого обучения для компьютерного зрения за пределами классификации изображений, включая базовое обнаружение объектов, глубокое сновидение и нейронный стиль, генеративно-состязательные сети (GAN) и суперразрешение изображений. Опять же, методы, описанные в этом томе, должны быть намного более продвинутыми, чем в Starter Bundle — здесь вы начнете отделять себя от новичка в области глубокого обучения и превратитесь в настоящего практикующего специалиста по глубокому обучению.

Чтобы начать свое превращение в эксперта по глубокому обучению, просто переверните страницу.

2. Введение

Добро пожаловать в ImageNet Bundle Deep Learning for Computer Vision with Python, последний том серии. Этот том должен быть наиболее продвинутым с точки зрения содержания, охватывая методы, которые позволят вам воспроизвести результаты самых современных публикаций, статей и докладов. Чтобы упростить эту работу, я разделил ImageNet Bundle на две части.

В первой части мы подробно изучим набор данных ImageNet и узнаем, как обучать современные глубокие сети, включая AlexNet, VGGNet, GoogLeNet, ResNet и SqueezeNet, с нуля, добиваясь максимально возможной точности, как и их соответствующие оригинальные произведения. Для достижения этой цели нам потребуется использовать все наши навыки из набора «Начальный набор» и «Набор специалиста» .

Нам нужно убедиться, что мы понимаем основы сверточных нейронных сетей, особенно типы слоев и регуляризацию, поскольку мы реализуем некоторые из этих более «экзотических» архитектур. К счастью, вы уже видели более поверхностные реализации этих более глубоких архитектур в пакете Practitioner Bundle, поэтому реализация таких сетей, как VGGNet, GoogLeNet и ResNet, покажется вам знакомой.

Нам также необходимо убедиться, что нам удобно присматривать за процессом обучения, поскольку мы можем легко адаптировать наши сетевые архитектуры к набору данных ImageNet, особенно в более поздние эпохи. Изучение того, как правильно отслеживать графики потерь и точности, чтобы определить, следует ли и когда обновлять обновления параметров, является приобретенным навыком, поэтому, чтобы помочь вам быстрее развить этот навык и обучить глубокие архитектуры на больших сложных наборах данных, я написал каждую из этих глав. как «экспериментальные журналы» , применяющие научный метод.

Внутри каждой главы для данной сети вы найдете:

1. Точный процесс, который я использовал при обучении сети.
2. Частные результаты.
3. Изменения, которые я решил внести в следующем эксперименте.

Таким образом, каждая глава читается как «история» : вы узнаете, что сработало для меня, что не сработало и, в конечном счете, что дало наилучшие результаты и позволило мне воспроизвести работу данной публикации. Прочитав эту книгу, вы сможете использовать эти знания для обучения своих собственных сетевых архитектур с нуля в ImageNet, не теряя времени и недель (или даже месяцев) на настройку параметров.

Вторая часть этой книги посвящена кейс-стади — реальным применениям глубокого обучения и компьютерного зрения для решения конкретной проблемы. Сначала мы начнем с обучения CNN с нуля распознаванию эмоций/мимики людей в видеопотоках в реальном времени. Оттуда мы будем использовать трансферное обучение через извлечение признаков для автоматического определения и исправления ориентации изображения. Второе тематическое исследование по трансферному обучению (на этот раз с помощью тонкой настройки) позволит нам распознавать более 164 марок и моделей автомобилей на изображениях. Модель, подобная этой, может позволить вам создать «интеллектуальную» систему рекламных щитов на шоссе, которая отображает целевую информацию или рекламу для водителя в зависимости от типа транспортного средства, которым он управляет. Наш последний пример продемонстрирует, как научить CNN правильно предсказывать возраст и пол человека на фотографии.

Наконец, я хочу напомнить вам, что техники, описанные в этом томе, гораздо более продвинуты, чем набор для начинающих и набор для практикующих. Оба предыдущих тома дали вам необходимые знания, необходимые для успешного чтения этой книги, но на этом этапе вы отделите себя от практикующего специалиста по глубокому обучению и настоящего мастера глубокого обучения. Чтобы начать свое окончательное превращение в эксперта по глубокому обучению, просто переверните страницу.

3. Тренировочные сети с использованием нескольких графических процессоров

Обучение глубоких нейронных сетей на крупномасштабных наборах данных может занять много времени, даже отдельные эксперименты могут занять несколько дней. Чтобы ускорить процесс обучения, мы можем использовать несколько графических процессоров. Хотя серверные части, такие как Theano и TensorFlow (и, следовательно, Keras), поддерживают обучение с несколькими GPU, процесс настройки эксперимента с несколькими GPU сложен и нетривиален. Я ожидаю, что в будущем этот процесс изменится к лучшему и станет значительно проще.

Поэтому для глубоких нейронных сетей и больших наборов данных я настоятельно рекомендую использовать библиотеку mxnet [5], которую мы будем использовать для большинства экспериментов в оставшейся части этой книги. Библиотека глубокого обучения mxnet (написанная на C++) обеспечивает привязку к языку программирования Python и специализируется на распределенном многомашинном обучении архитектуры нейронных сетей на массивных наборах данных (таких как ImageNet).

С библиотекой mxnet также очень легко работать — учитывая ваш опыт использования библиотеки Keras из предыдущих глав этой книги, вы обнаружите, что работать с mxnet легко, просто и даже вполне естественно.

Важно отметить, что все нейронные сети в ImageNet Bundle можно обучать с использованием одного графического процессора — единственный нюанс — это время. Некоторым сетям, таким как AlexNet и SqueezeNet, требуется всего несколько дней для обучения на одном графическом процессоре. Для других архитектур, таких как VGGNet и ResNet, обучение на одном графическом процессоре может занять больше месяца.

В первой части этой главы я выделю обсуждаемые нами сетевые архитектуры, которые можно легко обучить на одном графическом процессоре, и какие архитектуры должны использовать несколько графических процессоров, если это вообще возможно. Затем, во второй половине этой главы, мы рассмотрим некоторые приrostы производительности, которых можно ожидать при обучении сверточных нейронных сетей с использованием нескольких графических процессоров.

3.1 Сколько графических процессоров мне нужно?

Если бы вы спросили любого опытного специалиста по глубокому обучению, сколько графических процессоров вам нужно для обучения достаточно глубокой нейронной сети на большом наборе данных, он бы почти всегда ответил: «Чем больше, тем лучше». Преимущество использования нескольких GPU очевидно — распараллеливание. Чем больше графических процессоров мы

может бросить на проблему, тем быстрее мы можем обучить данную сеть. Однако у некоторых из нас при работе с этой книгой может быть только один графический процессор. В связи с этим возникают вопросы: • Является ли использование только одного графического процессора бесполезным занятием? • Является ли чтение этой главы пустой тратой времени? • Была ли покупка пакета ImageNet неудачной инвестицией?

Ответ на все эти вопросы — решительное «нет» — вы в надежных руках, и знания, которые вы здесь изучите, будут применимы к вашим собственным проектам глубокого обучения. Однако вам нужно управлять своими ожиданиями и осознавать, что вы пересекаете порог, который отделяет образовательные проблемы глубокого обучения от сложных реальных приложений. Теперь вы вступаете в мир современного глубокого обучения, где эксперименты могут занять дни, недели или даже, в некоторых редких случаях, месяцы — этот график совершенно и совершенно нормальный.

Независимо от того, есть ли у вас один GPU или восемь GPU, вы сможете воспроизвести производительность сетей, описанных в этой главе, но опять же, имейте в виду оговорку о времени. Чем больше у вас графических процессоров, тем быстрее будет проходить обучение. Если у вас один графический процессор, не расстраивайтесь — просто наберитесь терпения и поймите, что это часть процесса. Основная цель ImageNet Bundle — предоставить вам фактические тематические исследования и подробную информацию о том, как обучать современные глубокие нейронные сети на сложном наборе данных ImageNet (а также несколько дополнительных приложений). Независимо от того, один у вас графический процессор или восемь, вы сможете извлечь уроки из этих тематических исследований и использовать эти знания в своих собственных приложениях.

Читателям, использующим один графический процессор, я настоятельно рекомендую потратить большую часть времени на обучение AlexNet и SqueezeNet работе с набором данных ImageNet. Эти сети более мелкие и могут быть обучены намного быстрее на системе с одним GPU (порядка 3-6 дней для AlexNet и 7-10 дней для SqueezeNet, в зависимости от вашей машины). Более глубокие сверточные нейронные сети, такие как GoogLeNet, также можно обучить на одном графическом процессоре, но это может занять до 7-14 дней.

Меньшие варианты ResNet также можно обучать на одном графическом процессоре, но для более глубокого версии, описанную в этой книге, я бы рекомендовал использовать несколько графических процессоров.

Единственная сетевая архитектура, которую я не рекомендую пытаться обучать с использованием одного графического процессора, — это VGGNet. Мало того, что настройка гиперпараметров сети может быть сложной задачей (как мы увидим позже в этой книге), так еще и сеть работает чрезвычайно медленно из-за своей глубины и количество полно связанных узлов. Если вы решили обучать VGGNet с нуля, имейте в виду, что на обучение сети может уйти до 14 дней даже при использовании четырех графических процессоров.

Опять же, как я упоминал ранее в этом разделе, сейчас вы перешагиваете порог от специалиста по глубокому обучению до эксперта по глубокому обучению. Наборы данных, которые мы изучаем, большие и сложные, а сети, которые мы будем обучать на этих наборах данных, имеют большую глубину. По мере увеличения глубины увеличивается и количество вычислений, необходимых для выполнения прямого и обратного прохода. Потратите секунду сейчас, чтобы заявить о своих ожиданиях, что эти эксперименты нельзя оставить на ночь и собрать результаты на следующее утро — ваши эксперименты займут больше времени. Это факт, который должен принять каждый исследователь глубокого обучения.

Но даже если вы тренируете свои собственные современные модели глубокого обучения на одном графическом процессоре, не беспокойтесь. Те же методы, которые мы используем для нескольких графических процессоров, можно применить и к одному графическому процессору. Единственная цель ImageNet Bundle — дать вам знания и опыт, необходимые для успешного применения глубокого обучения в ваших собственных проектах.

[3.2 Повышение производительности при использовании нескольких графических процессоров](#)

В идеальном мире, если одна эпоха для данного набора данных и сетевой архитектуры занимает N секунд для завершения на одном графическом процессоре, то мы ожидаем, что та же эпоха с двумя графическими процессорами завершится за $N/2$ секунд. Однако это ожидание не соответствует действительности. Производительность обучения сильно зависит от шины PCIe в вашей системе, конкретной архитектуры, которую вы обучаете, количества слоев в сети и того, связана ли ваша сеть посредством вычислений или связи.

3.3 Резюме

В целом, обучение с двумя графическими процессорами увеличивает скорость примерно в 1,8 раза. При использовании четырех графических процессоров производительность увеличивается примерно в 2,5–3,5 раза в зависимости от вашей системы [6]. Таким образом, обучение не уменьшается линейно с количеством графических процессоров в вашей системе. Архитектуры, которые связаны вычислениями (большие размеры пакетов увеличиваются с количеством графических процессоров), будут лучше масштабироваться с несколькими графическими процессорами, в отличие от сетей, которые полагаются на связь (т. е. меньшие размеры пакетов), где задержка начинает играть роль в снижении производительности.

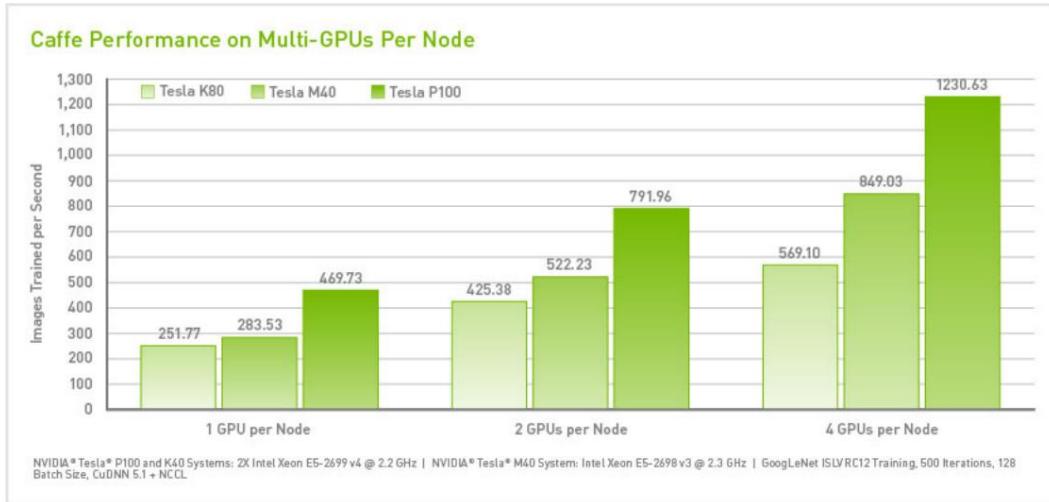


Рисунок 3.1: По оси X указано количество графических процессоров (1–4), а по оси Y — количество изображений, обучаемых в секунду. При увеличении от одного до двух графических процессоров мы можем наблюдать увеличение производительности примерно в 1,82 раза. Переход от одного GPU к четырем дает прирост 2,71.

Чтобы глубже изучить масштабирование графического процессора, давайте посмотрим на официальные тесты, выпущенные NVIDIA, на рис. 3.1. Здесь мы видим три типа графических процессоров (Tesla K80, Tesla M40 и Tesla P400), которые используются для обучения GoogLeNet на наборе данных ImageNet с использованием библиотеки глубокого обучения Caffe [7]. По оси X откладывается количество графических процессоров (один, два и четыре соответственно), а по оси Y — количество изображений, обучаемых в секунду (прямой и обратный проход). В среднем мы наблюдаем увеличение производительности примерно в 1,82 раза при переключении с одного GPU на два GPU. При сравнении одного графического процессора с четырьмя графическими процессорами производительность увеличивается до 2,71x.

Производительность будет продолжать расти по мере добавления в систему большего количества графических процессоров, но опять же, имейте в виду, что скорость обучения не будет линейно увеличиваться с количеством графических процессоров — если вы обучаете сеть с использованием одного графического процессора, а затем снова обучаете ее с использованием четырех графических процессоров, не НЕ ожидайте, что время, необходимое для обучения сети, уменьшится в четыре раза. Тем не менее, за счет обучения моделей глубокого обучения с большим количеством графических процессоров можно получить прирост производительности, поэтому, если они у вас есть, обязательно используйте их.

3.3 Резюме

В этой главе мы обсудили концепцию обучения архитектур глубокого обучения с использованием нескольких графических процессоров. Для выполнения большинства экспериментов, описанных в этой книге, мы будем использовать библиотеку mxnet, оптимизированную для обучения на нескольких графических процессорах. Учитывая ваш опыт использования библиотеки Keras в предыдущих главах этой книги, вы обнаружите, что использование mxnet является естественным с очень похожими именами функций и классов.

Оттуда мы обсудили основные ожидания при обучении сетей с использованием одного графического процессора по сравнению с несколькими графическими процессорами. Да, обучение глубокой сети на большом наборе данных с одним GPU займет больше времени,

но не расстраивайтесь — те же методы, которые вы используете для экземпляров с одним GPU, применимы и к экземплярам с несколькими GPU. Имейте в виду, что сейчас вы переходите порог от специалиста по глубокому обучению до эксперта по глубокому обучению — эксперименты, которые мы здесь проводим, будут более сложными и потребуют больше времени и усилий. Установите это ожидание сейчас, как это делают все исследователи глубокого обучения в своей карьере.

В главе 6 мы обучим нашу первую сверточную нейронную сеть, AlexNet, на наборе данных ImageNet, повторяя работу Крижевского и др. в своей основополагающей работе в 2012 году [8], которая навсегда изменила ландшафт классификации изображений.

4. Что такое ImageNet?

В этой главе мы обсудим набор данных ImageNet и связанную с ним программу ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9]. Эта задача является де-факто эталоном для оценки алгоритмов классификации изображений. В таблице лидеров ILSVRC доминируют сверточные нейронные сети и методы глубокого обучения с 2012 года, когда Крижевский и др. опубликовали свою основополагающую работу AlexNet [8].

С тех пор методы глубокого обучения продолжают увеличивать разрыв в точности между CNN и другими традиционными методами классификации компьютерного зрения. Нет никаких сомнений в том, что CNN являются мощными классификаторами изображений и в настоящее время прочно вошли в литературу по компьютерному зрению и машинному обучению. Во второй половине этой главы мы рассмотрим, как получить набор данных ImageNet, необходимый для того, чтобы вы воспроизвели результаты современных нейронных сетей позже в этой главе.

4.1 Набор данных ImageNet

В сообществах компьютерного зрения и глубокого обучения вы можете столкнуться с некоторой контекстуальной путаницей в отношении того, что такое ImageNet, а что нет. На самом деле ImageNet — это проект, направленный на маркировку и классификацию изображений по всем 22 000 категорий на основе определенного набора слов и фраз. На момент написания этой статьи в проекте ImageNet насчитывается более 14 миллионов изображений.

Итак, как организована ImageNet? Чтобы упорядочить такое огромное количество данных, ImageNet фактически следует иерархии WordNet [10]. Каждое осмысленное слово/фраза внутри WordNet называется «набором синонимов» или сокращенно синсетом. В рамках проекта ImageNet изображения классифицируются в соответствии с этими синсетами; цель проекта — иметь более 1000 изображений на синсет.

4.1.1 ИЛСВРК

В контексте компьютерного зрения и глубокого обучения, всякий раз, когда вы слышите, как люди говорят о сети изображений, они, вероятно, имеют в виду задачу ImageNet Large Scale Visual Recognition Challenge [9] или просто ILSVRC для краткости. Целью трека классификации изображений в этом задании является обучение модели, которая может правильно классифицировать изображение по 1000 отдельных категорий объектов, некоторые из которых считаются детализированной классификацией, а другие — нет. Изображения внутри набора данных ImageNet

были собраны путем компиляции предыдущих наборов данных и изучения популярных онлайн-сайтов. Затем эти изображения были вручную помечены, аннотированы и помечены.

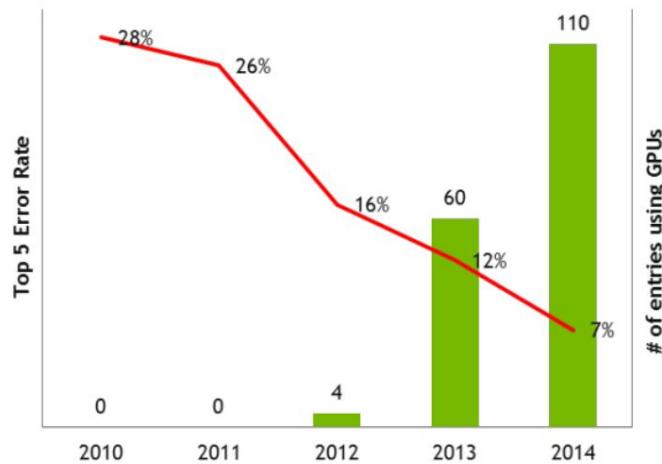


Рисунок 4.1: С тех пор как в 2012 году была представлена основополагающая архитектура AlexNet, методы сверточной нейронной сети доминировали в задаче ILSVRC как с точки зрения точности, так и с точки зрения количества записей. (Изображение предоставлено NVIDIA [11])

С 2012 года в таблице лидеров по задачам ILSVRC доминируют подходы, основанные на глубоком обучении, точность рангов 1 и 5 увеличивается с каждым годом (рис. 4.1). Модели обучаются на 1,2 миллионах обучающих изображений, а также на 50 000 изображений для проверки (50 изображений на синсет) и на 100 000 изображений для тестирования (100 изображений на синсет).

Эти 1000 категорий изображений представляют различные классы объектов, с которыми мы можем столкнуться в повседневной жизни, например, виды собак, кошек, различные предметы домашнего обихода, типы транспортных средств и многое другое. Вы можете найти полный список категорий объектов в вызове ILSVRC на официальной странице документации ImageNet (<http://pyimg.co/1ogm0>).

В главу 5 Starter Bundle я включил рисунок, демонстрирующий некоторые проблемы, связанные с набором данных ImageNet из программы ImageNet Large Scale Visual Recognition Challenge. В отличие от общих классов «птица», «кошка» и «собака», ImageNet включает более мелкие классы по сравнению с предыдущими эталонными наборами данных классификации изображений, такими как PASCAL VOC [12]. В то время как PASCAL VOC ограничил «собаку» только одной категорией, ImageNet вместо этого включает 120 различных пород собак. Это требование классификации пальцев подразумевает, что наши сети глубокого обучения должны не только распознавать изображения как «собаку», но и быть достаточно разборчивыми, чтобы определить, какой вид собаки.

Кроме того, изображения в ImageNet сильно различаются по масштабу объекта, количеству экземпляров, беспорядку/окклюзии изображения, деформируемости, текстуре, цвету, форме и реальному размеру. Этот набор данных, мягко говоря, сложен, и в некоторых случаях даже людям трудно правильно маркировать. Из-за сложного характера этого набора данных модели глубокого обучения, которые хорошо работают в ImageNet, вероятно, будут хорошо обобщаться для изображений за пределами набора проверки и тестирования — это точная причина, по которой мы применяем трансферное обучение и к этим моделям.

Мы обсудим дополнительные примеры изображений и конкретных классов в главе 5, когда начнем изучать набор данных ImageNet и писать код для подготовки изображений к обучению. Однако до этого времени я настоятельно рекомендую вам уделить 10-20 минут просмотру синсетов (<http://pyimg.co/1ogm0>) в своем веб-браузере, чтобы почувствовать масштаб и трудности, связанные с правильной классификацией этих изображений.

4.2 Получение ImageNet

Набор данных задачи классификации ImageNet довольно большой: 138 ГБ для обучающих изображений, 6,3 ГБ для проверочных изображений и 13 ГБ для тестовых изображений. Прежде чем вы сможете загрузить ImageNet, вам сначала необходимо получить доступ к вызову ILSGRC и загрузить изображения и соответствующие метки классов. Этот раздел поможет вам получить набор данных ImageNet.

4.2.1 Запрос доступа к вызову ILSVRC

ILSVRC Challenge — это совместная работа Принстонского и Стэнфордского университетов и, следовательно, академический проект. ImageNet не владеет авторскими правами на изображения и предоставляет доступ к необработанным файлам изображений только для некоммерческих исследовательских и/или образовательных целей (хотя этот вопрос подлежит обсуждению — см. Раздел 4.2.5 ниже). Если вы попадаете в этот лагерь, вы можете просто зарегистрировать учетную запись на веб-сайте ILSVRC (<http://pyimg.co/fy844>).

Однако обратите внимание, что ImageNet не принимает свободно доступные адреса электронной почты, такие как Gmail, Yahoo и т. д. Вместо этого вам нужно будет указать адрес электронной почты вашего университета или государственного/исследовательского учреждения. Как показано на рис. 4.2, мне просто нужно было указать адрес электронной почты своего университета, оттуда я смог подтвердить свой адрес электронной почты, а затем принять Условия доступа.

Here you can request access to the original images. Click [here](#) for details of how it works.

*Below is the information you have provided. Please make sure it is true and correct. Please provide your email address at the organization you are affiliated with, for example, `yourname@princeton.edu`. **We will not approve requests based on freely available email addresses such as gmail, hotmail, etc.** If necessary, you can [update](#) your information before submitting a request.*

Email: [REDACTED] @umbc.edu
 Full Name: Dr. Adrian Rosebrock
 Organization: PyImageSearch

[Submit Request](#)

Рисунок 4.2: Если у вас есть адрес электронной почты, связанный с университетом или исследовательской организацией, обязательно используйте его при регистрации в ImageNet и соответствующем конкурсе ILSVRC.

После того, как вы примете Условия доступа, вы получите доступ к странице загрузки исходных изображений — щелкните ссылку данных изображения ILSVRC 2015. Оттуда убедитесь, что вы загрузили комплект разработчика, ZIP-файл, содержащий README, информацию о разделении обучения/тестирования, файлы из черного списка, которые не следует использовать для обучения, и т. д. (рис. 4.3).

Затем вам нужно загрузить набор данных CLS-LOC, который содержит 1,2 миллиона изображений в наборе данных ImageNet (рис. 4.3). Имейте в виду, что это большой файл, и в зависимости от вашего интернет-соединения (и стабильности image-net.org) загрузка может занять пару дней. Мое личное предложение состоит в том, чтобы использовать программу командной строки wget для загрузки архива, что позволит вам перезапустить загрузку с того места, где вы остановились, на случай возникновения проблем с подключением (которых, вероятно, будет несколько). Объяснение того, как использовать wget, выходит за рамки этой книги, поэтому обратитесь к следующей странице за инструкциями по перезапуску загрузки с помощью wget (<http://pyimg.co/97u59>).

После загрузки архива .tar следующим шагом будет его распаковка, что также требует больших вычислительных затрат, поскольку вам нужно разархивировать 1,2 миллиона изображений — я бы посоветовал оставить вашу систему для решения этой задачи на ночь.

4.2.2 Программная загрузка изображений

Если вам отказано в

доступе к необработанным данным изображения ILSVRC, не беспокойтесь — есть и другие способы получения данных, хотя эти методы несколько более утомительны. Имейте в виду, что ImageNet

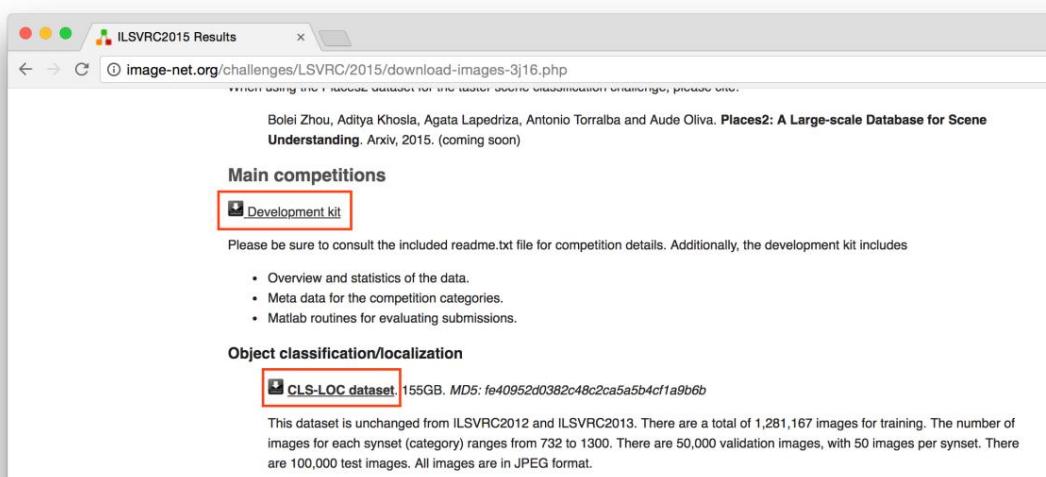


Рисунок 4.3: Чтобы загрузить весь набор данных ImageNet, вам необходимо загрузить комплект для разработки вместе с большим архивом .tar, содержащим 1,2 миллиона изображений. (выделены красными прямоугольниками).

не «владеет» изображениями внутри набора данных, поэтому они могут свободно распространять URL-адреса изображений. URL-адреса каждого изображения (файл .txt с одним URL-адресом в строке) в наборе данных можно найти здесь:

<http://pyimg.co/kw64x> Опять

же, вам нужно будет использовать wget для загрузки изображений. Распространенная проблема, с которой вы можете столкнуться здесь, заключается в том, что некоторые URL-адреса изображений могут иметь 404 ошибки после исходного веб-сканирования, и у вас не будет к ним доступа. Таким образом, загрузка изображений программным способом может быть довольно обременительной, утомительной, и этот метод я не рекомендую. Но не волнуйтесь – есть еще один способ получить ImageNet.

4.2.3 Использование внешних служб Из-за

огромного размера набора данных ImageNet и необходимости его глобального распространения этот набор данных хорошо подходит для распространения через BitTorrent. На веб-сайте AcademicTorrents.com можно загрузить как набор для обучения, так и набор для проверки (<http://pyimg.co/asdyi>). [13]. Скриншот веб-страницы можно найти на рисунке 4.4.

Набор для тестирования не включен в торрент, поскольку у нас не будет доступа к оценочному серверу ImageNet, чтобы представить наши прогнозы по данным тестирования. Имейте в виду, что даже если вы используете внешние службы, такие как AcademicTorrents, для загрузки набора данных ImageNet, вы по-прежнему неявно связаны Условиями доступа. Вы можете использовать ImageNet для исследования и разработки собственных моделей, но вы не можете переупаковывать ImageNet и использовать его для получения прибыли — это строго академический набор данных, предоставленный совместным предприятием Стенфорда и Принстона. Уважайте научное сообщество и не нарушайте Условия доступа.

4.2.4 Комплект для разработки ImageNet

Пока вы загружаете фактический набор данных ImageNet, убедитесь, что вы загрузили ImageNet Development Kit (<http://pyimg.co/wijj7>) который мы в дальнейшем будем называть просто «DevKit» .

Я также разместил зеркало DevKit здесь: <http://pyimg.co/ounw6> DevKit содержит:

- Обзор и статистика набора данных. • Метаданные для категорий (позволяющие нам создать сопоставление имени файла изображения с меткой класса). • Подпрограммы MATLAB для оценки (которые нам не понадобятся).

4.2 Получение ImageNet

25

Type	Name	Files	Added	Size	DLS	DLs	
	ImageNet LSVRC 2012 Validation Set (Object Detection)	1	2015-10-16	6.74GB	584	26+	1
	ImageNet LSVRC 2012 Training Set (Object Detection)	1	2015-10-16	147.90GB	646	21+	5
	ImageNet LSVRC 2013 Validation Set (Object Detection)	1	2015-10-15	2.71GB	160	12+	0
	ImageNet LSVRC 2014 Training Set (Object Detection)	1	2015-10-15	50.12GB	199	11+	2

We are a community-maintained distributed repository for datasets and scientific knowledge
[About](#) - [Terms](#) [Send Feedback](#)

Рисунок 4.4: Скриншот веб-сайта AcademicTorrents для ImageNet. Обязательно загрузите файлы «Проверочный набор ImageNet LSVRC 2012 (обнаружение объектов)» и «Обучающий набор ImageNet LSVRC 2012 (обнаружение объектов)», обведенные красным.

DevKit представляет собой небольшую загрузку размером всего 7,4 МБ, которая должна завершиться в течение нескольких секунд. После загрузки DevKit разархивируйте его и найдите время, чтобы ознакомиться со структурой каталога, включая лицензию (КОПИРОВАНИЕ) и файл readme.txt. Мы подробно рассмотрим DevKit в нашей следующей главе, когда мы создадим набор данных ImageNet и подготовим его для обучения CNN.

4.2.5 Вопросы авторского права ImageNet

На первый взгляд может показаться, что набор данных ImageNet и связанная с ним задача ILSVRC — это минное поле претензий на авторские права — кому именно принадлежит набор данных ImageNet? Чтобы ответить на этот вопрос, давайте разобьем проблему на три конкретных класса активов:

- Актив №1: сами изображения. • Актив №2: предварительно скомпилированный набор данных ILSVRC. • Ресурс №3: веса выходной модели, полученные путем обучения сети на ILSVRC.

Во-первых, сами необработанные изображения принадлежат человеку/организации, которые сделали снимок — они владеют полными авторскими правами на эти изображения. Проект ImageNet действует с теми же ограничениями, что и поисковые системы, такие как Google, Bing и т. д. — им разрешено предоставлять ссылки на оригинальные изображения, защищенные авторским правом, при условии, что авторские права сохраняются. Это положение объясняет, почему веб-сайту ImageNet разрешено предоставлять URL-адреса исходных изображений в наборе данных, не требуя от вас регистрации и создания учетной записи — вы несете ответственность за их фактическую загрузку.

Этот процесс кажется довольно четким; однако вода начинает мутиться, когда мы смотрим на настоящую задачу ILSVRC. Поскольку конечный пользователь больше не несет ответственности за загрузку каждого изображения по одному (и вместо этого может загружать весь архив набора данных), мы сталкиваемся с проблемами авторского права — почему пользователь может загружать предварительно скомпилированный архив (потенциально) изображения защищены авторским правом?

Разве это не нарушает авторские права человека, сделавшего исходную фотографию? Это предмет споров между художественными и научными сообществами, но в настоящее время нам разрешено

загрузите архивы изображений ILSGRC в соответствии с Условиями доступа, которые мы принимаем при участии в ИЛСВРК:

1. Вы можете свободно использовать набор данных ImageNet для академических и некоммерческих целей.
2. Вы не можете распространять данные ILSVRC как часть вашего конечного продукта.

На первоначальный вопрос об авторском праве нет прямого ответа, но он несколько умиротворен ограничениями, наложенными на предварительно скомпилированные архивы наборов данных. Кроме того, веб-сайт ImageNet предоставляет заявки на удаление DMCA для правообладателей, которые хотят удалить свои изображения из набора данных.

Наконец, давайте рассмотрим актив № 3, сериализованные веса данной модели, полученные после обучения сверточной нейронной сети на наборе данных ImageNet — эти веса модели также защищены авторским правом?

Ответ немного неясен, но, насколько мы сейчас понимаем закон, нет ограничений на открытый выпуск изученных весов моделей [14]. Поэтому мы можем свободно распространять наши обученные модели по своему усмотрению, при условии, что мы помним о духе добросовестного использования и надлежащего указания авторства.

Причина, по которой нам разрешено распространять наши собственные модели (и даже авторские права на них с использованием наших собственных ограничений), заключается в параметризованном обучении (Starter Bundle, глава 8) — наша CNN не хранит «внутренние копии» необработанных изображений (таких как алгоритм k-NN будет). Поскольку модель не хранит исходные изображения (полностью или частично), сама модель не связана теми же заявлениями об авторских правах, что и исходный набор данных ImageNet. Таким образом, мы можем свободно распространять веса нашей модели или размещать на них дополнительные авторские права (например, конечный пользователь может свободно использовать нашу существующую архитектуру, но должен повторно обучить сеть с нуля на исходном наборе данных, прежде чем использовать его в коммерческом приложении).

А как насчет моделей, обученных в ImageNet, которые используются в коммерческих приложениях? Нарушают ли модели, обученные на наборе данных ImageNet и используемые в коммерческих приложениях, Условия доступа? Согласно формулировке Условий доступа, да, технически эти коммерческие приложения могут нарушить договор.

С другой стороны, не было возбуждено ни одного судебного дела против компании/стартапа глубокого обучения, которые обучали свои собственные сети с нуля, используя набор данных ImageNet. Имейте в виду, что авторское право не имеет силы, если оно не соблюдается — в отношении ImageNet такое принуждение никогда не применялось.

Короче говоря: это серая зона в сообществе глубокого обучения. Существует большое количество стартапов в области глубокого обучения, которые полагаются на CNN, обученные на наборе данных ImageNet (названия компаний намеренно опущены) — их доход основан исключительно на производительности этих сетей. На самом деле, без ImageNet и ILSVRC у этих компаний не было бы набора данных, необходимого для создания их продукта (если только они не инвестировали миллионы долларов и много лет собирали и аннотировали набор данных самостоятельно).

Мое анекдотическое мнение

Мое анекдотическое мнение состоит в том, что существует негласный набор правил, регулирующих добросовестное использование набора данных ImageNet. Я полагаю, что эти правила выглядят следующим образом (хотя наверняка найдутся многие, кто со мной не согласится):

- Правило № 1: Вам необходимо каким-либо образом получить набор данных ILSVRC и принять (либо явным образом

или неявно) Условия доступа.

- Правило № 2: после получения данных, связанных с заданием ILSVRC, вам необходимо обучить свою собственную сверточную нейронную сеть на наборе данных. Вы можете использовать существующие сетевые архитектуры, такие как AlexNet, VGGNet, ResNet и т. д., при условии, что вы обучаете сеть с нуля на наборе данных ILSVRC. Вам не нужно разрабатывать новую сетевую архитектуру.
- Правило № 3: После того, как вы получили веса своих моделей, вы можете распространять их со своими собственными ограничениями, включая открытый доступ, использование с указанием авторства и даже ограниченную коммерческую доступность.

4.3 Резюме

коммерческое использование.

Правило номер три будет горячо оспариваться, и я уверен, что получу несколько электронных писем по этому поводу, но суть в том, что, хотя правила неясны, в суд не было подано ни одного судебного иска о том, как веса сети получены из ILSVRC можно использовать, в том числе в коммерческих приложениях. Опять же, имейте в виду, что авторские права действительны только в том случае, если они действительно соблюдаются — простое владение авторскими правами не служит формой защиты.

Кроме того, использование моделей глубокого обучения, обученных на ILSVRC, является как юридической, так и экономической проблемой — индустрия компьютерных наук переживает огромный бум приложений для глубокого обучения. Если бы был принят радикальный закон, ограничивающий коммерческое использование CNN, обученных с нуля на данных изображений, защищенных авторским правом (даже если нет копий исходных данных из-за параметризованного обучения), мы бы убили часть экономики, переживающей высокие темпы роста и оценки . в миллиардах долларов. Этот стиль сильно ограничительного, широкого законодательного органа может очень легко стать катализатором новой зимы ИИ (Стартовый набор, глава 2).

Для получения дополнительной информации о том, «кто чем владеет» в сообществе глубокого обучения (наборы данных, веса моделей и т. д.), посмотрите «Глубокое обучение против больших данных: кто чем владеет?» , отличная статья Томаша Малисевича на эту тему [15].].

4.3 Резюме

В этой главе мы рассмотрели набор данных ImageNet и связанную с ним задачу ILSVRC, де-факто тест, используемый для оценки алгоритмов классификации изображений. Затем мы рассмотрели несколько методов получения набора данных ImageNet.

В остальных главах этой книги я буду предполагать, что у вас нет доступа к набору для тестирования и связанному с ним оценочному серверу ImageNet; поэтому мы получим наш собственный тестовый набор из обучающих данных. Это гарантирует, что мы сможем оценить наши модели локально и получить разумный прокси для точности нашей сети.

Потратите время сейчас, чтобы начать загрузку набора данных ImageNet на свой компьютер. Я бы рекомендовал использовать официальный веб-сайт ILSVRC для загрузки данных ImageNet, так как этот метод является самым простым и надежным. Если у вас нет доступа к адресу электронной почты университета, правительства или исследовательской организации, не стесняйтесь обращаться к своим коллегам за доступом, но опять же имейте в виду, что вы по-прежнему связаны Условиями доступа, независимо от того, как вы получаете данные. (даже если скачиваете через AcademicTorrents).

По моему неподтвержденному мнению, веса моделей, полученные в результате обучения на наборе данных ILSVRC , можно использовать по своему усмотрению; однако имейте в виду, что это все еще предмет спора. Прежде чем развертывать коммерческое приложение, использующее модель, обученную в ImageNet, я бы посоветовал вам проконсультироваться с надлежащим юрисконсультом.

В нашей следующей главе мы изучим набор данных ImageNet, поймем его файловую структуру и напишем вспомогательные utilites Python, чтобы облегчить нам загрузку изображений с диска и подготовку их к обучению.

5. Подготовка набора данных ImageNet

После того, как вы загрузили набор данных ImageNet, вы можете быть немного ошеломлены. Теперь у вас есть более 1,2 миллиона изображений, находящихся на диске, ни у одного из них нет «удобочитаемых» имен файлов, нет очевидного способа извлечь из них метки классов, и совершенно неясно, как вы должны обучать пользовательский сверточный алгоритм. Нейронная сеть на этих изображениях — во что вы ввязались?

Не беспокойся, я тебя прикрою. В этой главе мы начнем с понимания файловой структуры ImageNet, включая как необработанные изображения, так и комплект для разработки (т. е. «DevKit»). Оттуда мы напишем вспомогательный служебный скрипт Python, который позволит нам анализировать имена файлов ImageNet + метки классов, создавая хороший выходной файл, который сопоставляет заданное имя входного файла с его соответствующей меткой (одно имя файла и метка в строке).

Наконец, мы будем использовать эти выходные файлы вместе с инструментом mxnet im2rec, который возьмет наши сопоставления и создаст эффективно упакованные файлы записей (.rec), которые можно использовать при обучении моделей глубокого обучения на наборах данных, слишком больших для размещения в основной памяти. Как мы увидим, этот формат .rec не только более компактен, чем HDF5, но и более эффективен при вводе-выводе, что позволяет нам быстрее обучать наши сети.

Методы и инструменты, которые мы применяем в этой главе, позволяют нам обучать наши собственные CNN с нуля на наборе данных ImageNet в последующих главах. В последующих главах, таких как наши тематические исследования по марке автомобиля и идентификации модели наряду с предсказанием возраста и пола, мы снова будем использовать те же самые инструменты, чтобы помочь нам создать наши наборы данных изображений.

Обязательно уделите пристальное внимание этому набору данных и не торопитесь при работе с ним. Код, который мы будем писать, не обязательно является «кодом глубокого обучения», а скорее полезными служебными скриптами, которые облегчат нам возможность обучать сети в дальнейшем.

5.1 Понимание файловой структуры ImageNet

Давайте продолжим и начнем с понимания файловой структуры ImageNet. Я предполагаю, что вы закончили загрузку файла ILSVRC2015_CLS-LOC.tar.gz, и, вероятно, вам придется перезапустить загрузку по крайней мере два-три раза (когда я лично загрузил массивный 166-гигабайтный архив, он

потребовалось два перезапуска и в общей сложности 27,15 часов для загрузки). Затем я распаковал архив с помощью следующей команды:

```
$ tar -xvf ILSVRC2015_CLS-LOC.tar.gz
```

Я бы посоветовал запустить эту команду прямо перед сном, чтобы к тому времени, когда вы проснетесь утром, она была полностью распакована. Имейте в виду, что только в обучающем наборе более 1,2 миллиона изображений, поэтому процесс разархивирования займет немного времени.

После завершения распаковки архива у вас появится каталог с именем ILSVRC2015:

```
$ ls ILSVRC2015
```

Давайте продолжим и изменим каталог на ILSVRC2015 и перечислим содержимое, где вы найдете три подкатаога:

```
$ cd ILSVRC2015 $ ls
```

Наборы изображений данных аннотаций

Во-первых, у нас есть каталог Annotations . Этот каталог используется только для локализации вызовов (т. е. обнаружение объекта), поэтому мы можем игнорировать этот каталог.

Каталог данных важнее. Внутри данных мы найдем подкаталог с именем CLS-LOC:

```
$ ls данных/
CLS-LOC
```

Здесь мы можем найти «сплиты» для обучения, тестирования и проверки:

```
$ ls Данные/CLS-LOC/
испытательный поезд val
```

Я взял слово «расщепление» в кавычки, так как еще предстоит проделать работу, чтобы получить эти данные в таком формате, чтобы мы могли обучить на них сверточную нейронную сеть и получить современные результаты классификации. . Давайте продолжим и рассмотрим каждый из этих подкаталогов по отдельности.

[5.1.1 «Тестовый» каталог ImageNet](#)

Каталог test содержит (как следует из названия) 100 000 изображений (100 точек данных для каждого из 1000 классов) для нашего тестового разделения:

```
$ ls -l Данные/CLS-LOC/тест/ | голова -n 10
всего 13490508
-rw-r--r-- 1 adrian adrian 33889 1 июля 2012 ILSVRC2012_test_00000001.JPG -rw-r--r-- 1 adrian adrian 122117 1
июля 2012 ILSVRC2012_test_00000002.JPG -rw-r--r-- 1 adrian adrian 26831 1 июля 2012 г.
ILSVRC2012_test_00000003.JPG -rw-r--r-- 1 adrian adrian 124722 1 июля 2012 г. ILSVRC2012_test_00000004.JPG
-rw-r--r-- 1 adrian adrian 98627 1 июля 2011 г. 0J050000000 ILSVRC2020
```

```
-rw-r--r-- 1 adrian adrian 211157 1 июля 2012 ILSVRC2012_test_00000006.JPG -rw-r--r-- 1 adrian adrian 219906 1
июля 2012 ILSVRC2012_test_00000007.JPG -rw-r--r-- 1 adrian adrian 181734 1 июля 2012 г.
ILSVRC2012_test_00000008.JPG -rw-r--r-- 1 adrian adrian 10696 1 июля 2012 г. ILSVRC2012_test_00000009.JPG
```

Однако мы не могли использовать эти изображения непосредственно для наших экспериментов. Напомним, что вызов ILSVRC является стандартом де-факто для алгоритмов классификации изображений. Чтобы эта задача была честной (и чтобы никто не обманывал), метки тестового набора остаются закрытыми.

Сначала человек/команда/организация тренирует свой алгоритм, используя разделы обучения и тестирования. Как только они удовлетворены результатами, на тестовом наборе делаются прогнозы. Прогнозы из тестового набора затем автоматически загружаются на сервер оценки ImageNet, где они сравниваются с метками достоверности. Ни в коем случае ни один из конкурентов не имеет доступа к этикеткам, подтверждающим достоверность результатов тестирования. Затем оценочный сервер ImageNet возвращает их общую точность.

Некоторые читатели этой книги могут иметь доступ к оценочному серверу ImageNet, и в этом случае я призываю вас изучить этот формат подробнее и подумать о том, чтобы представить свои собственные прогнозы. Однако многие другие читатели получат ImageNet без непосредственной регистрации учетной записи на веб-сайте ImageNet. Любой вариант вполне приемлем (при условии, что вы соблюдаете лицензионные соглашения, упомянутые в главе 4), но у вас не будет доступа к пробному серверу. Поскольку я хочу, чтобы эта глава оставалась открытой и доступной для всех, независимо от того, как вы получили ImageNet, мы проигнорируем тестовый каталог и создадим свой собственный тестовый набор, выбирая обучающие данные, как мы это делали для задач Tiny ImageNet в главе 11 и глава 12 комплекта для практикующих.

5.1.2 Директория ImageNet «поезд»

Каталог поездов ImageNet состоит из набора подкаталогов:

```
$ ls -l Данные/CLS-LOC/поезд/ | голова -n 10
всего 60020
drwxr-xr-x 2 adrian adrian 69632 Sep 29 2014 n01440764 drwxr-xr-x 2 adrian
adrian 69632 Sep 29 2014 n01443537 drwxr-xr-x 2 adrian adrian 57344 Sep 29
2014 n01484850 drwxr-xr-x 2 adrian adrian 57344 Sep 29 2014 n01491361 drwxr-
xr-x 2 adrian adrian 61440 Sep 29 2014 n01494475 drwxr-xr-x 2 adrian adrian
61440 Sep 29 2014 n01496331 drwxr-xr-x 2 adrian adrian 53248 Sep 29 2014
n01498041 drwxr-xr-x 2 adrian adrian 53248 29 сентября 2014 г. n01514668
drwxr-xr-x 2 Адриан Адриан 61440 29 сентября 2014 г. n01514859
```

Сначала имена этих подкаталогов могут показаться нечитаемыми. Однако вспомните из главы 4 об ImageNet, что набор данных организован в соответствии с идентификаторами WordNet [10], называемыми наборами синонимов или просто «наборами синонимов» для краткости. Синсет сопоставляется с конкретным понятием/объектом, таким как золотая рыбка, белоголовый орлан, самолет или акустическая гитара. Таким образом, в каждом из этих странно помеченных подкаталогов вы найдете примерно 732-1300 изображений на класс.

Например, идентификатор WordNet n01440764 состоит из 1300 изображений линя, типа Европейские пресноводные рыбы, близкие к семейству голльянов (рис. 5.1):

```
$ ls -l Данные/CLS-LOC/train/n01440764/*.JPEG | туалет -л
1300
$ ls -l Данные/CLS-LOC/train/n01440764/*.JPEG | head -n 5 adrian 13697 10 июня
2012 г. Data/CLS-LOC/train/n01440764/n01440764_10026.JPG
```

Адриан 9673 10 июня 2012 г. Data/CLS-LOC/train/n01440764/n01440764_10027.JPG adrian 67029 10 июня 2012 г. Data/CLS-LOC/train/n01440764/n01440764_10029.JPG adrian 146489/train/train/train/10 июня 2012 г. n01440764/n01440764_10040.JPG 6350 10 июня 2012 г. Данные/CLS-LOC/train/n01440764/n01440764_10042.JPG
Адриан

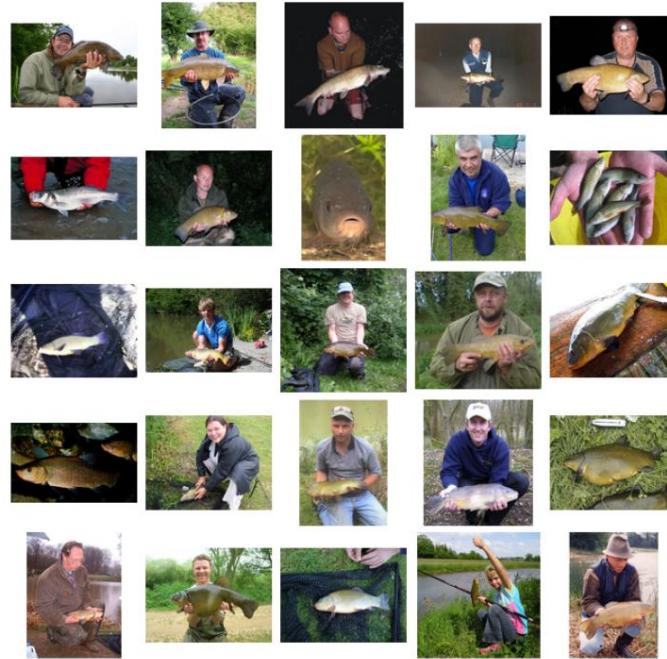


Рисунок 5.1: Выборка из 25 изображений из син-набора n01440764 . Эти изображения являются примерами линя, разновидности европейской пресноводной рыбы.

Учитывая, что идентификаторы WordNet обучающих изображений встроены в имя файла, наряду с файлом train_cls.txt , который мы рассмотрим далее в этой главе, нам будет довольно просто связать данное обучающее изображение с его классом. метка.

5.1.3 Директория ImageNet «val»

Как и каталог test , каталог val содержит 50 000 изображений для каждого класса (по 50 изображений для каждого из 1000 классов):

```
$ ls -l Данные/CLS-LOC/val/*.JPEG | туалет -l
50000
```

Каждое из этих 50 000 изображений хранится в «плоском» каталоге, что означает, что никакие дополнительные подкаталоги не используются, чтобы помочь нам связать данное изображение с меткой класса : идентифицирующая информация, встроенная в пути к файлам (например, идентификатор WordNet и т. д.). К счастью, позже в этой главе мы рассмотрим файл с именем val.txt , который предоставляет нам сопоставления имени файла с меткой класса.

```
$ ls -l Данные/CLS-LOC/val/ | голова -n 10
всего 6648996
```

```
-rw-r--r-- 1 adrian adrian 109527 12 июня 2012 ILSVRC2012_val_00000001.jpeg -rw-r--r-- 1 adrian adrian 140296
12 июня 2012 ILSVRC2012_val_00000002.jpeg -rw-r--r-- 1 adrian adrian 122660 12 июня 2012 г.
ILSVRC2012_VAL_00000003.jpeg -RW-R--r-- 1 ADRIAN ADRIAN 84885 12 июня 2012 г. ILSVRC2012_VAL_00000004.jpeg
-RW-R-R-1 ADRIAN ADRIAN 130340 JUN 12 2012 -r-- 1 adrian adrian 151397 Jun 12 2012
ILSVRC2012_val_00000006.jpeg -rw-r--r-- 1 adrian adrian 165863 Jun 12 2012 ILSVRC2012_val_00000007.jpeg -rw-
r--r-- 1 adrian adrian 107423 Jun 12 2012 ILSVRC2012_val_00000008.jpeg -rw-r--r-- 1 adrian adrian 114708 12
июня 2012 г. ILSVRC2012_val_00000009.jpeg
```

5.1.4 Директория ImageNet «ImageSets»

Теперь, когда мы прошли через подкаталоги train, test и val , давайте вернемся на уровень выше к папкам Annotations и Data . Здесь вы увидите каталог с именем ImageSets . Давайте изменим каталог на ImageSets и исследуем его:

```
$ ls
Аннотации Data ImageSets $ cd
ImageSets/ $ ls
CLS-LOC
$ cd CLS-LOC $ ls
test.txt train_cls.txt
train_loc.txt val.txt
```

Мы можем игнорировать файл test.txt , так как мы будем создавать собственное тестовое разделение на основе обучающих данных. Однако нам нужно взглянуть как на train_cls.txt (где «cls» означает «классификация»), так и на val.txt . Эти файлы содержат базовые имена файлов для обучающих изображений (1 281 167) вместе с проверочными изображениями (50 000). Вы можете проверить этот факт с помощью следующей команды:

```
$ wc -l train_cls.txt val.txt
train_cls.txt
      50000 знач.txt
всего 1331167
```

Всего в этом файле содержится 1 331 167 изображений для работы. Исследуя файл train_cls.txt , вы можете увидеть, что его содержимое представляет собой просто имя файла базового изображения (без расширения файла) и уникальный целочисленный идентификатор с одной строкой в строке:

```
$ head -n 10 train_cls.txt n01440764/
n01440764_10026 1 n01440764/
n01440764_10027 2 n01440764/
n01440764_10029 3 n01440764/
n01440764_10040 4 n01440764/
n01440764_10042 5 n01440764/
n01440764_10043 6 n01440764/
n01440764_10048 7 n01440764/
n01440764_10066 8 n01440764/
n01440764_10074 9 n01440764/
n01440764_10095 10
```

Базовое имя файла изображения позволит нам получить полное имя файла изображения. Уникальное целое число просто счетчик, который увеличивается, по одному на строку.

То же самое верно и для val.txt:

```
$ head -n 10 val.txt
ILSVRC2012_val_00000001 1
ILSVRC2012_val_00000002 2
ILSVRC2012_val_00000003 3
ILSVRC2012_val_00000004 4
ILSVRC2012_val_00000005 5
ILSVRC2012_val_00000006 6
ILSVRC2012_val_00000007 7
ILSVRC2012_val_00000008 8
ILSVRC2012_val_00000009 9
ILSVRC2012_val_00000010 10
```

Уникальный целочисленный идентификатор для изображения не слишком полезен, за исключением случаев, когда нам нужно определить «черные списки» изображений. Изображения, помеченные кураторами набора данных ImageNet как «занесенные в черный список», имеют слишком двусмысленные метки классов, и поэтому их не следует учитывать в процессе оценки. Позже в этой главе мы пройдемся по всем изображениям из черного списка и удалим их из нашего проверочного набора, изучив уникальный целочисленный идентификатор, связанный с каждым проверочным изображением.

Преимущество использования файлов train_cls.txt и val.txt заключается в том, что нам не нужно перечислять содержимое подкаталогов для обучения и проверки с помощью paths.list_images — вместо этого мы можем просто перебрать каждую из строк в этих файлах .txt . Мы будем использовать оба этих файла позже в этой главе, когда будем преобразовывать необработанные файлы ImageNet в формат .rec , подходящий для обучения с помощью mxnet.

[5.1.5 Каталог ImageNet «DevKit»](#) Помимо загрузки

самих необработанных изображений в главе 4, вы также загрузили ILSVRC 2015 DevKit. Этот архив содержит фактические индексные файлы, которые нам нужны для сопоставления имен файлов изображений с соответствующими метками классов. Вы можете разархивировать файл ILSVRC2015_devkit.tar.gz с помощью следующей команды:

```
$ tar -xvf ILSVRC2015_devkit.tar.gz
```

Внутри каталога ILSVRC2015 вы найдете каталог, который мы ищем — devkit:

```
$ cd ILSVRC2015 $
ls
девкит
```

Технически вы можете разместить этот файл в любом месте вашей системы (поскольку мы создадим файл конфигурации Python, чтобы указать на важные пути); однако лично мне нравится хранить его в подкаталогах Annotations, Data и ImageSets для организационных целей. Я бы посоветовал вам скопировать каталог devkit, чтобы он также жил с нашими каталогами Annotations, Data и ImageSets:

```
$ cp -R ~/home/ILSVRC2015/devkit/raid/datasets/imagenet/
```

 Точные пути, которые вы укажете здесь, будут зависеть от вашей системы. Я просто показываю примеры команд, которые я запускал в своей личной системе. Фактические команды, которые вы используете, будут такими же, но вам нужно будет обновить соответствующие пути к файлам.

Давайте продолжим и посмотрим на содержимое devkit:

```
$ cd devkit/ $ ls
КОПИРОВАТЬ
данные оценки readme.txt
```

Вы можете прочитать файл COPYING для получения дополнительной информации о копировании и распространении набора данных ImageNet и связанного программного обеспечения для оценки. Файл readme.txt содержит информацию о Задача ILSVRC, в том числе о том, как структурирован набор данных (мы предоставляем более подробный обзор набора данных в этой главе). Каталог оценки , как следует из названия, содержит подпрограммы MATLAB для оценки прогнозов, сделанных на наборе тестов — поскольку мы будем получать собственный набор тестов, мы можем игнорировать этот каталог.

Самое главное, у нас есть каталог данных . Внутри данных вы найдете ряд метафайлов, как в формате MATLAB, так и в текстовом формате (.txt):

```
$ cd data/ $
ls -l total 2956
ILSVRC2015_clsloc_validation_blacklist.txt
ILSVRC2015_clsloc_validation_ground_truth.mat
ILSVRC2015_clsloc_validation_ground_truth.txt
ILSVRC2015_det_validation_blacklist.txt
ILSVRC2015_det_validation_ground_truth.mat
ILSVRC2015_vid_validation_ground_truth.mat map_clsloc.txt
map_det.txt map_vid.txt meta_clsloc.mat meta_det.mat
meta_vid.mat
```

Из этого каталога нас больше всего интересуют следующие три файла: • map_clsloc.txt
• ILSVRC2015_clsloc_validation_ground_truth.txt • ILSVRC2015_clsloc_validation_blacklist.txt
Файл map_clsloc.txt сопоставляет наши идентификаторы WordNet с удобочитаемыми метками классов и, следовательно, является самым простым методом преобразования идентификатор WordNet для метки, которую человек может интерпретировать. Перечислив первые несколько строк этого файла, мы можем увидеть сами сопоставления:

```
$ head -n 10 map_clsloc.txt n02119789
1 kit_fox n02100735 2
английский_сеттер n02110185 3
сибирский_хаски n02096294 4
австралийский_терьер n02102040 5
английский_спрингер n02066245 6
серый_кит n02509815
```

n02124075 8 Egypt_cat n02417914

9 Козерог

n02123394 10 персидская_кошка

Здесь мы видим, что WordNet ID n02119789 сопоставляется с меткой класса `kit_fox`. Идентификатор WordNet n02096294 соответствует австралийскому терьеру, виду собак. Это сопоставление продолжается для всех 1000 классов в наборе данных ImageNet. Как подробно описано в разделе 5.1.3 выше, изображения в каталоге `val` не содержат никакой информации о метке класса, встроенной в имя файла; однако у нас есть файл `val.txt` в каталоге `ImageSets`. В файле `val.txt` перечислены (частичные) имена файлов изображений для набора проверки. В файле `val.txt` ровно 50 000 записей (по каждой строке). Внутри `ILSVRC2015_clsloc_validation_ground_truth.txt` также есть 50 000 записей (по одной на строку).

Давайте посмотрим на эти записи:

```
$ head -n 10 ILSVRC2015_clsloc_validation_ground_truth.txt 490
```

361

171

822

297

482

13

704

599

164

Как мы видим, в каждой строке указано одно целое число. Взяв первую строку `val.txt` и первую строку `ILSVRC2015_clsloc_validation_ground_truth.txt` мы получаем:

```
(ILSVRC2012_val_00000001, 490)
```



Рисунок 5.2: Это изображение змеи, но что это за змея? Чтобы выяснить это, нам нужно изучить метки достоверности для проверки, установленной внутри ImageNet.

Если бы мы открыли файл `ILSVRC2012_val_00000001.JPG`, то увидели бы изображение на рис. 5.2. Ясно, что это какая-то змея, но какая? Если мы изучим `map_clsloc.txt`, мы увидим, что идентификатор метки класса с номером 490 — это идентификатор WordNet n01751748, который является `sea_snake`:

```
$ grep '490' map_clsloc.txt n01751748
490 sea_snake
```

Поэтому нам нужно использовать `val.txt`, и `ILSVRC2015_clsloc_validation_ground_truth.txt`.
для создания нашего набора проверки.

Давайте также рассмотрим содержимое файла `ILSGRC2015_clsloc_validation_blacklist.txt`:

```
$ head -n 10 ILSVRC2015_clsloc_validation_blacklist.txt
36
50
56
103
127
195
199
226
230
235
```

Как я упоминал ранее, некоторые файлы проверки считаются слишком двусмысленными в их метке класса. Поэтому организаторы ILSVRC пометили эти изображения как «занесенные в черный список», подразумевая, что их не следует включать в проверочный набор. При создании нашего проверочного набора нам нужно проверить идентификаторы проверочных изображений в этом наборе черного списка — если мы обнаружим, что данное изображение принадлежит этому набору, мы проигнорируем его и исключим из проверочного набора.

Как видите, для создания набора данных ImageNet требуется много файлов. Нам нужны не только сами необработанные изображения, но и ряд файлов `.txt`, используемых для построения сопоставлений исходного имени файла обучения и проверки с соответствующей меткой класса. Это был бы сложный и трудоемкий процесс для выполнения вручную, поэтому в следующем разделе я покажу вам свой класс `ImageNetHelper`, который я лично использую при создании набора данных ImageNet.

5.2 Создание набора данных ImageNet

Общая цель создания набора данных ImageNet состоит в том, чтобы мы могли обучать сверточные нейронные сети с нуля на нем. Поэтому мы рассмотрим создание набора данных ImageNet в контексте его подготовки к CNN. Для этого мы сначала определим файл конфигурации, в котором хранятся все соответствующие пути к изображениям, пути к открытому тексту и любые другие параметры, которые мы хотим включить.

Оттуда мы определим класс Python с именем `ImageNetHelper`, который позволит нам быстро и легко построить:

1. Наши файлы `.lst` для обучения, тестирования и проверки. Каждая строка в файле `.lst` содержит уникальный идентификатор изображения, метку класса и полный путь к входному изображению. Затем мы сможем использовать эти файлы `.lst` в сочетании с инструментом `mxnet im2rec` для преобразования файлов изображений в эффективно упакованный файл записи.
2. Наши средние значения красного, зеленого и синего каналов для тренировочного набора, которые мы позже будем использовать при выполнении нормализации среднего.

5.2.1 Ваш первый файл конфигурации ImageNet

Каждый раз при обучении CNN в ImageNet мы создаем проект со следующей структурой каталогов:

```
-- mx_imagenet_alexnet | |--  
    конфигурация
```

```

| |--- __init__.py
| |--- imagenet_alexnet_config.py
| --- имиджнет
| --- вывод/
| --- build_imagenet.py
| --- test_alexnet.py
| --- train_alexnet.py

```

Как следует из каталога и имен файлов, этот файл конфигурации предназначен для AlexNet. Внутри config мы разместили два файла:

1. __init__.py
2. imagenet_alexnet_config.py

Файл __init__.py превращает конфигурацию в пакет Python, который можно импортировать через оператор импорта в наши собственные скрипты — этот файл позволяет нам использовать синтаксис/библиотеки Python в реальную конфигурацию, что значительно упрощает процесс настройки нейронной сети для ImageNet.

Фактические конфигурации ImageNet затем сохраняются в файле imagenet_alexnet_config.py.

Вместо ввода полного пути к набору данных ImageNet (например, /raid/datasets/imagenet/)

Я решил создать символическую ссылку (часто называемую «символической ссылкой» или «ярлыком») с псевдонимом imagenet. Это избавило меня от множества нажатий клавиш и ввода длинных путей. В следующем примере вы можете см. (1) полный путь к каталогу списков и (2) версию символической ссылки:

- /raid/наборы данных/имиджнет/справки
- imagenet/lists (указывает на полный путь /raid выше)

Для создания собственных символьических ссылок (в среде Unix) вы можете использовать команду ln .

Приведенный ниже пример команды создаст символьскую ссылку с именем imagenet в моей текущей рабочей папке. каталог, который ссылается на полный набор данных ImageNet на моем диске /raid:

```
$ ln -s /raid/datasets/imagenet imagenet
```

Вы можете изменить приведенную выше команду на свои собственные пути.

Независимо от того, где вы решите хранить свой базовый каталог набора данных ImageNet, не торопитесь.

Теперь создадим два подкатаога — lists и rec:

```

$ mkdir imagenet/справки
$ mkdir imagenet/rec

```

В приведенной выше команде я предполагаю, что вы создали символьскую ссылку с именем imagenet , чтобы указать на базовый каталог набора данных. Если нет, укажите полный путь. Подкаталоги lists и rec будут использоваться далее в этой главе.

Сценарий build_imagenet.py будет отвечать за построение сопоставлений из входных данных. файл изображения для вывода метки класса. Скрипт train_alexnet.py будет использоваться для обучения AlexNet. с нуля на ImageNet. Наконец, сценарий test_alexnet.py будет использоваться для оценки производительность AlexNet на нашем тестовом наборе.

Последние два сценария будут рассмотрены в главе 6, поэтому пока давайте просто рассмотрим imagenet_alexnet_config.py — этот файл практически не изменится для всех экспериментов ImageNet , которые мы проводим в этой книге. Поэтому нам важно найти время, чтобы понять, как это файл структурирован.

Откройте imagenet_alexnet_config.py и вставьте следующий код:

```

1 # импортируем необходимые пакеты 2 из
пути импорта ОС
3
4 # определить базовый путь к месту хранения набора данных
ImageNet 5 # devkit на диске)
6 BASE_PATH = "/raid/наборы данных/imagenet/ILSVRC2015"

```

Строка 2 импортирует единственный нужный нам пакет Python, подмодуль пути . Подмодуль пути содержит специальную переменную с именем path.sep — это разделитель пути для вашей операционной системы. На машинах Unix разделителем пути является / — пример пути к файлу может выглядеть как путь/к/вашему/файлу.txt. Однако в Windows разделителем пути является \, что делает путь к файлу примера \to\your\file.txt. Мы хотели бы, чтобы наша конфигурация работала независимо от операционной системы, поэтому мы будем использовать переменную path.sep всякий раз, когда это удобно.

Затем в строке 6 определяется BASE_PATH , где на диске находится наш набор данных ImageNet. Этот Каталог должен содержать четыре каталога Annotations, Data, devkit и ImageSets.

Из-за размера набора данных ImageNet я решил хранить все файлы, связанные с ImageNet, на системном диске RAID (поэтому вы видите, что BASE_PATH начинается с /raid). Вы должны изменить BASE_PATH и указать , где в вашей системе хранится набор данных ImageNet. Не стесняйтесь использовать каталог наборов данных , который мы использовали в предыдущих примерах Deep Learning for Computer Vision с Python — структура проекта наборов данных будет работать нормально, если у вас достаточно места в основном разделе для хранения всего набора данных.

Еще раз хочу обратить внимание, что вам нужно будет обновить переменную BASE_PATH на вашем собственную систему – пожалуйста, найдите время, чтобы сделать это сейчас.

Из нашего BASE_PATH мы можем получить еще три важных пути:

```

8 # на основе базового пути получить базовый путь к изображениям, наборы образов 9 #
путь и путь к комплекту разработки 10 IMAGES_PATH = path.sep.join([BASE_PATH, "Data/CLS-
LOC"])
11 IMAGE_SETS_PATH = path.sep.join([BASE_PATH, "ImageSets/CLS-LOC/"])
12 DEVKIT_PATH = path.sep.join([BASE_PATH, "комплект разработчика/данные"])

```

IMAGES_PATH объединяется с BASE_PATH , чтобы указать на каталог, содержащий наши необработанные изображения для тестов, обучения и val . IMAGE_SETS_PATH указывает на каталог , содержащий важные файлы train_cls.txt и val.txt , в которых явно перечислены имена файлов для каждого набора. Наконец, как следует из названия, DEVKIT_PATH — это базовый путь к местонахождению нашего DevKit , в частности наших файлов с открытым текстом, которые мы будем анализировать в Разделе 5.1.5.

Говоря о DevKit, давайте определим WORD_IDS, путь к файлу map_clsloc.txt , который сопоставляет 1000 возможных идентификаторов WordNet с (1) уникальными идентификационными целыми числами и (2) удобочитаемыми метками.

```

14 # определить путь, который сопоставляет 1000 возможных идентификаторов WordNet с
15 # целыми числами метки класса 16 WORD_IDS = path.sep.join([DEVKIT_PATH, "map_clsloc.txt"])

```

Чтобы построить наш тренировочный набор, нам нужно определить TRAIN_LIST, путь, который содержит 1,2 миллиона (частичных) имен файлов изображений для обучающих данных:

```
18 # определить пути к тренировочному файлу, который сопоставляет (частично) 19 #
имя файла изображения с меткой целочисленного класса 20 TRAIN_LIST =
path.sep.join([IMAGE_SETS_PATH, "train_cls.txt"])
```

Далее нам нужно определить некоторые конфигурации проверки:

```
22 # определить пути к именам файлов проверки вместе с 23 # файлом, который содержит
метки проверки подлинности 24 VAL_LIST = path.sep.join([IMAGE_SETS_PATH, "val.txt"])
```

```
25 VAL_LABELS = path.sep.join([DEVKIT_PATH,
26     "ILSVRC2015_clsloc_validation_ground_truth.txt"])
27
28 # определяем путь к файлам проверки, занесенным в черный список 29 VAL_BLACKLIST
= path.sep.join([DEVKIT_PATH,
30     "ILSVRC2015_clsloc_validation_blacklist.txt"])
```

Переменная VAL_LIST указывает на файл val.txt в каталоге ImageSets . Напоминаем, что в файле val.txt перечислены (частичные) имена файлов изображений для 50 000 файлов проверки. Чтобы получить достоверные метки для данных проверки, нам нужно определить путь VAL_LABELS — это позволит нам связать имена отдельных файлов изображений с метками классов. Наконец, файл VAL_BLACKLIST содержит уникальные целочисленные идентификаторы файлов проверки, которые были занесены в черный список. Когда мы создадим набор данных ImageNet, мы позаботимся о том, чтобы эти изображения не были включены в данные проверки.

В следующем блоке кода мы определяем переменную NUM_CLASSES, а также NUM_TEST_IMAGES:

```
32 # поскольку у нас нет доступа к данным тестирования, нам нужно 33 # взять
несколько изображений из обучающих данных и использовать их вместо этого
34 NUM_CLASSES = 1000
35 NUM_TEST_IMAGES = 50 * NUM_CLASSES
```

Для набора данных ImageNet существует 1000 возможных классов изображений, поэтому для NUM_CLASSES установлено значение 1000. Чтобы получить наш тестовый набор, нам нужно выбрать изображения из обучающего набора. Мы установим NUM_TEST_IMAGES равным $50 \times 1000 = 50\,000$ изображений. Как я упоминал ранее, мы будем использовать инструмент mxnet im2rec для преобразования наших необработанных файлов изображений на диске в файл записи, подходящий для обучения с использованием библиотеки mxnet.

Чтобы выполнить это действие, нам сначала нужно определить путь MX_OUTPUT , а затем получить несколько других переменных:

```
37 # определить путь к выходным данным для обучения, проверки и тестирования 38 #
справки 39 MX_OUTPUT = "/raid/datasets/imagenet"
```

```
40 TRAIN_MX_LIST = path.sep.join([MX_OUTPUT, "lists/train.lst"])
41 VAL_MX_LIST = path.sep.join([MX_OUTPUT, "lists/val.lst"])
42 TEST_MX_LIST = path.sep.join([MX_OUTPUT, "lists/test.lst"])
```

Все файлы, которые выводятся либо (1) нашими вспомогательными утилитами Python, либо (2) двоичным файлом im2rec , будут храниться в базовом каталоге MX_OUTPUT . Основываясь на том, как я организую наборы данных (подробно описанном в главе 6 Starter Bundle), я решил включить все выходные файлы в каталог imagenet , в котором также хранятся необработанные изображения, DevKit и т. д. Вы должны хранить выходные файлы там, где вы чувствуете себя комфортно — я просто привожу пример того, как я организую наборы данных на своей машине.

Как я уже упоминал, после применения наших служебных скриптов Python у нас останется три файла — train.lst, val.lst и test.lst — эти файлы будут содержать (целочисленные) идентификаторы меток классов и полный путь к именам файлов изображений для каждого из наших разделений данных (строки 40-42). Затем инструмент im2rec возьмет эти файлы .lst в качестве входных данных и создаст файлы .rec , в которых вместе хранятся фактические необработанные изображения + метки классов, аналогично созданию набора данных HDF5 в главе 10 пакета Practitioner Bundle:

```
44 # определить путь к выходным данным обучения, проверки и тестирования 45 #
записи изображений 46 TRAIN_MX_REC = path.sep.join([MX_OUTPUT, "rec/train.rec"])
```

```
47 VAL_MX_REC = path.sep.join([MX_OUTPUT, "rec/val.rec"])
48 TEST_MX_REC = path.sep.join([MX_OUTPUT, "rec/test.rec"])
```

Разница здесь в том, что эти файлы записей гораздо более компактны (поскольку мы можем хранить изображения в виде сжатых файлов JPEG или PNG вместо необработанных растровых изображений массива NumPy). Кроме того, эти файлы записей предназначены для использования исключительно с библиотекой mxnet, что позволяет нам получить более высокую производительность, чем исходные наборы данных HDF5.

 Я решил включить подкаталоги lists и rec в каталог imagenet в организационных целях — настоятельно рекомендую вам сделать то же самое. Если вы следите моей структуре каталогов, пожалуйста, найдите время, чтобы создать свои списки и подкаталоги recs прямо сейчас. Если вы подождете, пока мы просмотрим и выполним build_imagenet.py , вы можете забыть создать эти подкаталоги, что приведет к ошибке сценария. Но не волнуйтесь! Вы можете просто вернуться к созданию списков и повторному выполнению сценария.

При создании нашего набора данных нам нужно будет вычислить DATASET_MEAN для каждого из каналов RGB , чтобы выполнить нормализацию среднего:

```
50 # определяем путь к набору данных mean 51
DATASET_MEAN = "output/imagenet_mean.json"
```

Эта конфигурация просто хранит путь, по которому средства будут сериализованы на диск в формате JSON. При условии, что вы проводите все эксперименты на одном и том же компьютере (или, по крайней мере, на компьютерах с одинаковой структурой каталогов ImageNet), единственное конфигурации, которые вам придется редактировать от эксперимента к эксперименту, — это приведенные ниже:

```
53 # определить размер пакета и количество устройств, используемых для обучения
54 ПАКЕТ_РАЗМЕР = 128
55 ЧИСЛО_УСТРОЙСТВ = 8
```

Строка 54 определяет BATCH_SIZE, в котором изображения будут передаваться по сети во время обучения. Для AlexNet мы будем использовать размер мини-пакета 128. В зависимости от того, насколько глубока данная CNN, мы можем уменьшить этот размер пакета. Атрибут NUM_DEVICES контролирует количество устройств (будь то процессоры, графические процессоры и т. д.), используемых при обучении данной нейронной сети. Вы должны настроить эту переменную в зависимости от количества устройств, доступных для обучения на вашем компьютере.

5.2.2 Наша вспомогательная утилита ImageNet

Теперь, когда мы создали пример файла конфигурации, давайте перейдем к ImageNetHelper . class, который мы будем использовать для создания файлов .lst для разделов обучения, тестирования и проверки соответственно. Этот класс является важным вспомогательным средством, поэтому мы обновим наш модуль pyimagesearch и сохраним его в файл с именем imagenethelper.py внутри подмодуля utils:

```
-- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- ИО
| |--- НН
| |--- предварительная обработка
| |--- утилиты
| | |--- __init__.py
| | |--- captchahelper.py
| | |--- imagenethelper.py
| | |--- Ranked.py
```

Откройте imagenethelper.py, и мы определим служебный класс:

```
1 # импортируем необходимые пакеты
2 импортировать numpy как np
3 импорт ОС
4
5 класс ImageNetHelper:
6     def __init__(я, конфигурация):
7         # сохранить объект конфигурации
8         self.config = конфигурация
9
10        # создаем сопоставления меток и черный список проверки
11        self.labelMappings = self.buildClassLabels()
12        self.valBlacklist = self.buildBlacklist()
```

Строка 6 определяет конструктор нашего ImageNetHelper. Конструктор требует только **одного** параметра: объект с именем config. Конфиг на самом деле является файлом imagenet_alexnet_config . которые мы определили в предыдущем разделе. Передав этот файл как объект ImageNetHelper , мы можем получить доступ ко всем нашим путям к файлам и дополнительным конфигурациям. Затем мы строим сопоставления меток и черный список проверки в строках 11 и 12. Мы рассмотрим как buildClassLabels , так и Методы buildBlacklist далее в этом разделе.

Начнем с buildClassLabels:

```
14     деф buildClassLabels (я):
15         # загрузить содержимое файла, который отображает идентификаторы WordNet
16         # в целые числа, затем инициализируем словарь сопоставлений меток
17         строки = открыть(self.config.WORD_IDS).read().strip().split("\n")
18         меткаMappings = {}
```

В строке 17 мы читаем все содержимое файла WORD_IDS , который сопоставляет идентификаторы WordNet с (1) **的独特** целое число, представляющее этот класс, и (2) удобочитаемые метки. Затем мы определяем словарь labelMappings , который принимает идентификатор WordNet в качестве ключа и метку целочисленного класса в качестве ценность.

Теперь, когда весь файл WORD_IDS загружен в память, мы можем перебрать каждый из ряды:

```

20      # цикл по меткам
21      для строки в строке :
22          # разделить строку на идентификатор WordNet, пометить целое число и
23          # удобочитаемая метка
24          (wordID, метка, hrLabel) = row.split(" ")
25
26          # обновить словарь сопоставлений меток, используя идентификатор слова
27          # в качестве ключа и метки в качестве значения, вычитая '1'
28          # из метки, так как MATLAB имеет один индекс, а Python
29          # имеет нулевой индекс
30          labelMappings[wordID] = int(метка) - 1
31
32      # вернуть словарь сопоставлений меток
33      возвращаемые меткиMappings

```

Для каждой строки мы разбиваем ее на 3 кортежа (поскольку каждая запись в строке отделена пробелом), состоящий из:

1. Идентификатор WordNet (wordID).
2. Уникальный целочисленный идентификатор метки класса (метки).
3. Удобочитаемое имя идентификатора WordNet (hrLabel).

Теперь, когда у нас есть эти значения, мы можем обновить наш словарь labelMappings . Ключ к словарю — это идентификатор WordNet, wordID . Наше значение — это метка , из которой вычтено значение 1 .

Почему мы вычитаем единицу? Имейте в виду, что инструменты ImageNet, предоставленные ILSVRC, были построен с помощью MATLAB. Язык программирования MATLAB является одноиндексным (это означает, что он начинается считая с 1), в то время как язык программирования Python имеет нулевой индекс (мы начинаем считать с 0). Поэтому, чтобы преобразовать индексы MATLAB в индексы Python, мы просто вычитаем значение 1 с этикетки.

Словарь labelMappings возвращается вызывающей функции в строке 33.

Далее у нас есть функция buildBlacklist:

```

35      def buildBlacklist (я):
36          # загрузить список идентификаторов изображений из черного списка и преобразовать их в
37          # множество
38          строки = открыть (self.config.VAL_BLACKLIST).read()
39          строки = набор (строки.strip().split("\n"))
40
41          # вернуть идентификаторы изображений из черного списка
42          возвращаемые строки

```

Эта функция довольно проста. В строке 38 мы читаем все содержимое файла VAL_BLACKLIST файл. Файл VAL_BLACKLIST содержит уникальные целочисленные имена файлов проверки (по одному на строку), которые мы должны исключить из набора проверки из-за неоднозначных меток.

Мы просто разбиваем строку на список (разделяя символ новой строки \n) и преобразуем строки в заданный объект . Установленный объект позволит нам определить, является ли данное проверочное изображение частью черный список за O(1) раз.

Наша следующая функция отвечает за прием конфигурации TRAIN_LIST и IMAGES_PATH .

Чтобы построить набор путей к изображениям и связанных с ними меток целочисленных классов для обучающего набора:

```

44     дeф buildTrainingSet (я):
45         # загрузить содержимое входного файла обучения, в котором перечислены
46         # частичный идентификатор изображения и номер изображения, затем инициализируем
47         # список путей к изображениям и меток классов
48         строки = открыть (self.config.TRAIN_LIST).read().strip()
49         строки = строки.split("\n")
50         пути = []
51         метки = []

```

В строке 48 мы загружаем все содержимое файла TRAIN_LIST и разбиваем его на строки по Стока 49. Напомним, что файл TRAIN_LIST содержит частичные пути к файлам изображений — образец Файл train_cls.txt можно увидеть ниже:

```

n01440764/n01440764_10969 91
n01440764/n01440764_10979 92
n01440764/n01440764_10995 93
n01440764/n01440764_11011 94
n01440764/n01440764_11018 95
n01440764/n01440764_11044 96
n01440764/n01440764_11063 97
n01440764/n01440764_11085 98
n01440764/n01440764_1108 99
n01440764/n01440764_1113 100

```

Наша задача будет заключаться в создании обоих списков для (1) полного пути к изображению и (2) соответствующей метки класса. (строки 50 и 51). Чтобы построить списки, нам нужно перебрать каждую или каждую строку по отдельности:

```

53     # цикл по строкам во входном обучающем файле
54     для строки в строке :
55         # разбить строку на частичный путь и изображение
56         # номер (номер изображения порядковый и
57         # по сути бесполезен для нас)
58         (partialPath, imageName) = row.strip().split(" ")
59
60         # создаем полный путь к тренировочному образу, затем
61         # получить идентификатор слова из пути и использовать его для определения
62         # метка целочисленного класса
63         путь = os.path.sep.join([self.config.IMAGES_PATH,
64             "поезд", "{}.JPEG".format(partialPath)])
65         wordID = partialPath.split("/")[0]
66         метка = self.labelMappings[wordID]

```

Каждая строка состоит из двух записей: частичного пути к файлу обучающего изображения (например, n01440764/n01440764_10026) и imageName. Переменная imageName — это просто бухгалтерский счетчик. – это бесполезно при построении обучающей выборки; мы будем игнорировать это. Строки 63 и 64 отвечает за построение полного пути к тренировочному изображению с учетом IMAGES_PATH и частичный путь.

Этот путь состоит из трех компонентов:

1. IMAGES_PATH, где живут все наши каталоги train, test и val.
2. Жестко закодированная строка обучения , указывающая, что мы создаем путь к файлу для обучения.

изображение.

3. PartialPath, который является подкаталогом и базовым именем файла самого изображения.

Мы добавляем расширение файла .JPEG , чтобы создать окончательный путь к изображению. Пример пути может быть видно ниже:

```
/raid/datasets/imagenet/ILSVRC2015/Data/CLS-LOC/train/n02097130/n02097130_4602.JPEG
```

При отладке сценариев Python, используемых для создания этих файлов .lst , убедитесь, что вы проверяете пути к файлам, прежде чем продолжить. Я бы предложил использовать простую команду ls , чтобы проверить и посмотреть, файл существует. Если ls возвращает и сообщает, что путь к файлу не существует, значит, у вас есть ошибка в вашей конфигурации.

В строке 65 мы извлекаем wordID. wordID является подкаталогом partialPath , поэтому нам просто нужно разделить символ / , и мы можем извлечь идентификатор слова. Как только у нас есть wordID, мы можем найти соответствующую метку целочисленного класса в labelMappings (строка 66).

Учитывая путь и метку, мы можем обновить списки путей и меток соответственно:

```
68     # обновить соответствующие пути и списки меток
69     paths.append(путь)
70     labels.append(метка)
71
72     # вернуть кортеж путей к изображениям и соответствующий целочисленный класс
73     # ярлыки
74     возврат (np.array (пути), np.array (метки))
```

Строка 74 возвращает вызывающей функции набор из двух путей и меток . Эти значения будут позже быть записаны на диск в виде файла .lst с помощью нашего (будет определено) сценария build_dataset.py.

Последняя функция, которая нам нужна для создания нашего пути, это buildValidationSet, которая отвечает за для создания наших путей к проверочным изображениям и меток классов проверки:

```
76     def buildValidationSet (сам):
77         # инициализируем список путей к изображениям и меток классов
78         пути = []
79         метки = []
80
81         # загрузить содержимое файла, в котором перечислены частичные
82         # имена файлов проверочных изображений
83         valFilenames = open(self.config.VAL_LIST).read()
84         valFilenames = valFilenames.strip().split("\n")
85
86         # загрузить содержимое файла, который содержит *фактический*
87         # достоверные метки целочисленного класса для проверочного набора
88         valLabels = открытый(self.config.VAL_LABELS).read()
89         valLabels = valLabels.strip().split("\n")
```

Наша функция buildValidationSet очень похожа на нашу функцию buildTrainingSet, только с несколько дополнительных дополнений. Для начала мы инициализируем наш список путей к изображениям и меток классов (строки 78 и 79). Мы загружаем содержимое VAL_LIST , которое содержит частичные имена файлов проверки. файлы (строки 83 и 84). Чтобы построить наши метки классов, нам нужно прочитать содержимое VAL_LABELS — этот файл содержит метку целочисленного класса для каждой записи в VAL_LIST.

Имея как valFilenames, так и valLabels, мы можем создать наши списки путей и меток:

```

91         # цикл по данным проверки
92         for (строка, метка) в zip(valFilenames, valLabels):
93             # разбить строку на частичный путь и номер изображения
94             (partialPath, imageName) = row.strip().split(" ")
95
96             # если номер изображения находится в черном списке, то мы
97             # следует игнорировать это проверочное изображение
98             если imageName в self.valBlacklist:
99                 Продолжать
100
101            # создать полный путь к проверочному изображению, затем
102            # обновить соответствующие списки путей и меток
103            путь = os.path.sep.join([self.config.IMAGES_PATH, "val",
104                "{}.JPEG".format(partialPath)])
105            paths.append(путь)
106            labels.append(int(метка) - 1)
107
108            # вернуть кортеж путей к изображениям и соответствующий целочисленный класс
109            # ярлыки
110            возврат (np.array(пути), np.array(метки))

```

В строке 92 мы перебираем каждое из valFilenames и valLabels. Распаковываем ряд в строке 94 , чтобы извлечь partialPath вместе с imageName. В отличие от тренировочного набора, Здесь важен imageName — мы проверяем строки 98 и 99 , чтобы увидеть, находится ли imageName в черный список установлен, и если это так, мы игнорируем его.

Оттуда строки 103 и 104 создают путь к файлу проверки. Мы обновляем пути список в строке 105. Затем список меток обновляется в строке 106 , где мы снова позаботимся чтобы вычесть 1 из метки , так как она имеет нулевой индекс. Наконец, строка 110 возвращает двойку пути проверки и метки к вызывающей функции.

Теперь, когда наш ImageNetHelper определен, мы можем перейти к созданию файлов .lst . который будет загружен в im2rec.

5.2.3 Создание файлов списка и среднего значения

Как и в наших предыдущих сценариях build_*.py в предыдущих главах, сценарий build_imagenet.py будет очень похоже. На высоком уровне мы:

1. Соберите обучающую выборку.
2. Создайте проверочный набор.
3. Создайте тестовую выборку путем выборки обучающей выборки.
4. Прокрутите каждый из наборов.
5. Запишите путь к образу + соответствующую метку класса на диск.

Давайте продолжим и начнем работать над build_imagenet.py прямо сейчас:

1 # импортируем необходимые пакеты
2 из конфига импортировать imagenet_alexnet_config как конфиг
3 из sklearn.model_selection импорта train_test_split
4 из pyimagesearch.utils импортировать ImageNetHelper
5 импортировать numpy как np
6 индикатор выполнения импорта
7 импортировать json
8 импорт cv2

Строка 2 импортирует наш модуль `imagenet_alexnet_config` и присваивает ему псевдоним `config`. Затем мы импортируем функцию `train_test_split` из `scikit-learn`, чтобы мы могли построить тестовое разделение из нашего тренировочного набора. Мы также импортируем наш недавно определенный класс `ImageNetHelper`, чтобы помочь нам в создании набора данных `imageNet`.

Затем мы можем построить наши пути обучения и проверки + метки классов:

```
10 # инициализируем хелпер ImageNet и используем его для построения набора 11 #
данных для обучения и тестирования 12 print("[INFO] загрузка путей к изображениям...")
13 inh = ImageNetHelper(config) 14 (trainPaths, trainLabels) = inh.buildTrainingSet() 15
(valPaths, valLabels) = inh.buildValidationSet()
```

Затем нам нужно выбрать `NUM_TEST_IMAGES` из `trainPaths` и `trainLabels`, чтобы построить нашу тестовую разбивку:

```
17 # выполнить стратифицированную выборку из обучающей выборки для построения
18 # тестовой выборки 19 print("[INFO] построение разбиений...") 20 split =
train_test_split(trainPaths, trainLabels,
21           test_size=config.NUM_TEST_IMAGES, stratify=trainLabels, random_state=42)
22
23 (trainPaths, testPaths, trainLabels, testLabels) = разделить
```

Отсюда наш код выглядит почти идентично всем нашим предыдущим сценариям «создания набора данных» в Глубокое обучение компьютерному зрению с помощью Python:

```
25 # создать список, объединяющий обучение, проверку и тестирование 26 # пути к
изображениям вместе с соответствующими им метками и выходным списком 27 # файлы
28 наборы данных = [
29     ("train", trainPaths, trainLabels, config.TRAIN_MX_LIST), ("val", valPaths,
30     valLabels, config.VAL_MX_LIST), ("test", testPaths, testLabels,
31     config.TEST_MX_LIST)]
32
33 # инициализируем список средних красных, зеленых и синих каналов
34 (R, G, B) = ([], [], [])
```

Строки 28-31 определяют список наборов данных . Каждая запись в списке наборов данных представляет собой 4-кортеж, состоящий из четырех

- значения: 1. Тип разделения (т. е. обучение, тестирование или проверка).
- 2. Пути изображений.
- 3. Метки изображений.
- 4. Путь к выходному файлу `.lst`, который требуется `mxnet`.

Мы также инициализируем средние значения канала RGB в строке 34. Затем давайте пройдемся по каждой записи в списке наборов данных:

```
36 # цикл по кортежам набора данных 37
для (dType, paths, labels, outputPath) в наборах данных:
38     # открываем выходной файл для записи
```

```

39     print("[INFO] здание {}".format(outputPath))
40     f = открыть(выходной путь, "w")
41
42     # инициализируем индикатор выполнения
43     widgets = ["Список построек:", progressbar.Percentage(), " ",
44                progressbar.Bar(), " ", progressbar.ETA()]
45     pbar = progressbar.ProgressBar(maxval = len(пути),
46                                    виджеты=виджеты).start()

```

Строка 40 открывает указатель файла на наш outputPath. Затем мы создаем виджет ProgressBar . в строках 43-46. Индикатор выполнения, конечно, не требуется, но я считаю полезным предоставить ETA . информацию при создании наборов данных (и вычисления могут занять некоторое время).

Теперь нам нужно перебрать каждое из отдельных изображений и меток в разделении:

```

48     # перебираем каждое отдельное изображение + метки
49     для (i, (путь, метка)) в перечислении (zip(пути, метки)):
50         # записываем индекс изображения, метку и выходной путь в файл
51         row = "\t".join([str(i), str(label), path])
52         f.write("{}\n".формат(строка))

53
54         # если мы строим обучающий набор данных, то вычисляем
55         # среднее значение каждого канала в изображении, затем обновите
56         # соответствующие списки
57         если dType == "поезд":
58             изображение = cv2.imread(путь)
59             (b, g, r) = cv2.mean(изображение)[:3]
60             R.добавить(r)
61             G. добавить(g)
62             B. добавить(b)

63
64         # обновить индикатор выполнения
65         pbar.update(я)

```

Для каждого пути и метки мы записываем три значения в выходной файл .lst:

1. Индекс i (это просто уникальное целое число, которое mxnet может связать с изображением в набор).
2. Метка целочисленного класса.
3. Полный путь к файлу образа.

Каждое из этих значений отделено табуляцией с одним набором значений в строке. Образец такого выходной файл можно увидеть ниже:

```

0 35 /raid/datasets/imagenet/ILSVRC2015/Data/CLS-LOC/train/n02097130/n02097130_4602.JPG
1 640 /raid/datasets/imagenet/ILSVRC2015/Data/CLS-LOC/train/n02276258/n02276258_7039.JPG
2 375 /raid/datasets/imagenet/ILSVRC2015/Data/CLS-LOC/train/n03109150/n03109150_2152.JPG
3 121 /raid/datasets/imagenet/ILSVRC2015/Data/CLS-LOC/train/n02483708/n02483708_3226.JPG
4 977 /raid/datasets/imagenet/ILSVRC2015/Data/CLS-LOC/train/n04392985/n04392985_22306.JPG

```

Инструмент im2rec в mxnet возьмет эти файлы .lst и создаст наборы данных .rec . На строках 57-62 мы проверяем, является ли наш набор данных типом поезда , и если да, мы вычисляем среднее значение RGB . для изображения и обновить соответствующие списки каналов.

Наш последний блок кода обрабатывает очистку указателей файлов и сериализацию средств RGB на диск:

```

67      # закрыть выходной файл
68      pbar.finish() f.close()
69
70
71 # создать словарь средних значений, затем сериализовать значения в файл 72 # JSON 73 print("[INFO] сериализующие средства...")
74 D = {"R": np.mean(R), "G": np.mean(G), "B": np.mean(B)} 75 f =
75 f = open(config.DATASET_MEAN, "w") 76 f.write(json.dumps(D)) 77 f.close()

```

Чтобы выполнить скрипт build_dataset.py, просто выполните следующую команду:

```

$ time python build_imagenet.py [INFO]
загрузка путей к изображениям...
[INFO] построение шпагатов...
[INFO] сборка /raid/datasets/imagenet/lists/train.lst...
Список построек: 100% ##### | Время: 1:47:35 [ИНФО] сборка /
raid/datasets/imagenet/lists/val.lst...
Список построек: 100% ##### | Время: 0:00:00 [ИНФО] сборка /
raid/datasets/imagenet/lists/test.lst...
Список построек: 100% ##### | Время: 0:00:00 [INFO]
сериализация означает...

настоящий      107 м 39,276 с
пользователь    75м26.044с
система        2 мин 35,237 с

```

Как видите, тренировочный набор занял больше всего времени — 107 мин 39 с из-за того, что каждое из 1,2 миллиона изображений нужно было загрузить с диска и вычислить их среднее значение. Файлы .lst для тестирования и проверки были записаны на диск за считанные секунды из-за того, что все, что нам нужно было сделать, это записать пути к изображениям и метки в файл (дополнительный ввод-вывод не требовался).

Чтобы убедиться, что файлы .lst для обучения, тестирования и проверки успешно созданы, проверьте содержимое каталога MX_OUTPUT. Ниже вы можете найти содержимое моего каталога MX_OUTPUT/lists (где я храню файлы .lst):

```

$ ls /raid/datasets/imagenet/lists/ test.lst train.lst
val.lst

```

Подсчет строк в каждом из файлов показывает, что в нашем тестовом наборе содержится 50 000 изображений, в нашем проверочном наборе — 50 000 изображений, а в нашем тестовом наборе — 1 231 167 изображений:

```

$ wc -l /raid/наборы данных/imagenet/справки/*.lst
50000 /raid/datasets/imagenet/lists/test.lst 1231167 /raid/datasets/
imagenet/lists/train.lst 48238 /raid/datasets/imagenet/lists/val.lst
1329405 всего

```

Имея эти файлы .lst, давайте создадим наш упакованный набор данных, используя mxnet im2rec.

5.2.4 Создание компактных файлов записей

Теперь, когда у нас есть файлы .lst, создание файла .rec для каждого из отдельных файлов обучения, тестирования и проверки не составляет труда. Ниже вы можете найти мой пример команды для создания файла train.rec с использованием двоичного файла mxnet im2rec:

```
$ ~/mxnet/bin/im2rec /raid/datasets/imagenet/lists/train.lst "" \
/raid/datasets/imagenet/rec/train.rec \ resize=256 encoding='.jpg' \ качество=100
```

В моей системе я скомпилировал библиотеку mxnet в своем домашнем каталоге; поэтому полный путь к im2rec — /mxnet/im2rec. Вам может понадобиться изменить этот путь, если вы скомпилировали mxnet в другом месте на вашем компьютере. Другое альтернативное решение — просто поместить двоичный файл im2rec в системный PATH.

Первый аргумент im2rec — это путь к нашему выходному файлу train.lst — этот путь должен соответствовать переменной TRAIN_MX_LIST в imagenet_alexnet_config.py. Второй аргумент — пустая пустая строка. Этот параметр указывает корневой путь к нашим файлам изображений. Поскольку мы уже получили полный путь к изображениям на диске внутри наших файлов .lst, мы можем просто оставить этот параметр пустым. Затем у нас есть последний обязательный параметр для im2rec — это путь к нашей базе данных файлов выходных записей, где будут храниться наши сжатые изображения и метки классов. Этот параметр должен соответствовать переменной TRAIN_MX_REC в файле imagenet_alexnet_config.py.

Затем мы передаем im2rec три необязательных аргумента: 1.

resize: Здесь мы указываем, что размер нашего изображения должен быть изменен до 256 пикселей по кратчайшему измерению. Это изменение размера не только уменьшает входное разрешение изображения (тем самым уменьшая размер выходного файла .rec), но также позволяет нам выполнять увеличение данных во время обучения. 2. кодировка: Здесь мы можем указать кодировку JPEG или PNG. Мы будем использовать кодировку JPEG для экономии места на диске, так как это формат с потерями. PNG — это формат без потерь, но в результате файлы .rec будут намного больше. Кроме того, нас не должно беспокоить небольшое количество артефактов JPEG. Артефакты JPEG существуют в реальных изображениях и даже могут рассматриваться как тип увеличения данных.

3. качество: это значение является качеством изображения в формате JPEG или PNG. Для изображений JPEG это значение находится в диапазоне от 1 до 100 (чем больше значение, тем меньше сжатие/артефактов вводится, но в результате увеличивается размер файла). Для изображений PNG это значение будет находиться в диапазоне от 1 до 9. В этом примере мы предоставим значение JPEG, равное 100, чтобы не вводить дополнительные артефакты JPEG в наш набор данных.

Как видно из приведенного ниже вывода, преобразование файлов .lst в упакованные файлы записей занимает много времени, поскольку каждое из 1,2 миллиона изображений необходимо загрузить с диска, изменить их размер и сохранить в наборе данных:

```
[17:25:21] tools/im2rec.cc:126: новый размер изображения: короткий край 256 [17:25:21] tools/im2rec.cc:139: кодировка .jpg [17:25:21] tools/im2rec.cc:185: запись в вывод: /raid/datasets/imagenet/rec/train.rec [17:25:21] tools/im2rec.cc:187: вывод: /raid/datasets/imagenet/rec/train.rec [17:25:21] tools/im2rec.cc:200: Качество кодирования JPEG: 100 [17:25:26] tools/im2rec.cc:295: обработано 1000 изображений, прошло 5,81205 с [17:25:33] tools/im2rec.cc:295: обработано 2000 изображений, прошло 12,213 с [17:25:39] tools/im2rec.cc:295: обработано 3000 изображений, прошло 18,6334 с
```

...

```
[19:22:42] tools/im2rec.cc:295: обработано 1230000 изображений, прошло 7041,57 с
[19:22:49] tools/im2rec.cc:295: обработано 1231000 изображений, прошло 7048,79 с
[19:22:51] tools/im2rec.cc:298: Всего: обработано 1231167 изображений, затрачено 7050,07 с.
```

На моей машине весь этот процесс занял чуть менее двух часов.

Затем я могу повторить процесс для набора проверки:

```
$ ~/mxnet/bin/im2rec /raid/datasets/imagenet/lists/val.lst "" \
    /raid/datasets/imagenet/rec/val.rec resize=256 encoding='.jpg' \ quality=100
[06:40:10] tools/im2rec.cc:126: Новый размер изображения: короткий край
256 [06: 40:10] tools/im2rec.cc:139: Кодировка.jpg [06:40:10] tools/im2rec.cc:185:
Запись в вывод: /raid/datasets/imagenet/rec/val.rec [06: 40:10] tools/im2rec.cc:187:
Вывод: /raid/datasets/imagenet/rec/val.rec [06:40:10] tools/im2rec.cc:200: Качество кодирования
JPEG: 100 [06:40 :15] tools/im2rec.cc:295: обработано 1000 изображений, прошло 5,9966 с [06:40:22]
tools/im2rec.cc:295: обработано 2000 изображений, прошло 12,0281 с [06:40:27] tools/im2rec .cc:295:
обработано 3000 изображений, прошло 17,2865 с.
```

...

```
[06:44:36] tools/im2rec.cc:295: обработано 47000 изображений, прошло 266,616 с
[06:44:42] tools/im2rec.cc:295: обработано 48000 изображений, прошло 272,019 с
[06:44:43] ] tools/im2rec.cc:298: Всего: обработано 48238 изображений, затрачено 273,292 с.
```

А также тестовый набор:

```
$ ~/mxnet/bin/im2rec /raid/datasets/imagenet/lists/test.lst "" \
    /raid/datasets/imagenet/rec/test.rec resize=256 encoding='.jpg' \ quality=100
[06:47:16] tools/im2rec.cc:139: Кодировка.jpg [06:47:16 ] tools/im2rec.cc:185:
запись в вывод: /raid/datasets/imagenet/rec/test.rec [06:47:16] tools/im2rec.cc:187:
вывод: /raid/datasets/imagenet/rec /test.rec [06:47:16] tools/im2rec.cc:200: Качество кодирования
JPEG: 100 [06:47:22] tools/im2rec.cc:295: обработано 1000 изображений, прошло 6,32423 с [06:47 :28]
tools/im2rec.cc:295: обработано 2000 изображений, прошло 12,3095 с [06:47:35] tools/im2rec.cc:295:
обработано 3000 изображений, прошло 19,0255 с
```

...

```
[06:52:47] tools/im2rec.cc:295: обработано 49000 изображений, прошло 331,259 с
[06:52:55] tools/im2rec.cc:295: обработано 50000 изображений, прошло 339,409 с
[06:52:55] ] tools/im2rec.cc:298: Всего: обработано 50000 изображений, затрачено 339,409 с.
```

После того, как мы сгенерировали наборы данных с помощью im2rec для обучения, тестирования и проверки , давайте взглянем на размеры файлов:

```
$ ls -l /raid/datasets/imagenet/rec/ всего 105743748
```

```
-rw-rw-r-- 1 adrian adrian 4078794388 12 декабря 2016 г. test.rec -rw-rw-r-- 1 adrian adrian 100250277132
12 декабря 2016 г. 2016 вал.рек
```

Здесь мы видим, что файл train.rec является самым большим, его размер составляет чуть более 100 ГБ. Файлы test.rec и val.rec весят примерно 4 ГБ каждый. Преимущество здесь в том, что мы смогли существенно сжать весь наш набор данных ImageNet в эффективно упакованные файлы записей. Сжатие не только сэкономило нам место на диске, но и значительно ускорило процесс обучения за счет того, что потребуется выполнять меньше операций ввода-вывода.

Этот подход отличается от подхода HDF5, где нам нужно было хранить необработанное растровое изображение массива NumPy для каждого изображения. Если бы мы построили обучающую разбивку для ImageNet с использованием HDF5 (используя изображения 256× 256× 3), результирующий файл был бы более 1,9 ТБ, что на 1831% больше! Из-за этого увеличения настоятельно рекомендуется использовать формат mxnet .rec при работе с большими наборами данных.

5.3 Резюме

В этой главе мы узнали, как подготовить набор данных ImageNet. Мы начали с изучения структуры каталогов обоих необработанных изображений в ImageNet, а затем метафайлов в DevKit. Оттуда мы создали наш первый файл конфигурации ImageNet Python — все эксперименты, выполненные в ImageNet, будут производными от этого исходного файла конфигурации. Здесь мы указали пути к необработанным входным изображениям, метафайлам, количеству меток классов, выходным средним значениям сериализации RGB и т. д. При условии, что вы используете одну и ту же машину для запуска всех экспериментов ImageNet (или, по крайней мере, идентичную структуру каталогов на всех машинах, на которых вы проводите эксперименты). on), этот файл конфигурации будет редко (если вообще когда-либо) нуждаться в обновлении.

После создания файла конфигурации мы определили класс ImageNetHelper . Этот класс включает четыре утилиты, которые облегчают нам создание списка mxnet (называемого файлами .lst). Каждый файл .lst содержит по три значения в каждой строке:

1. Уникальный целочисленный идентификатор изображения.
2. Метка класса изображения.
3. Полный путь к образу на диске.

Получив файл .lst для каждого разделения обучения, тестирования и проверки, мы затем применили двоичный файл im2rec (предоставленный mxnet) для создания файлов записей с эффективной поддержкой для каждого разделения . Эти файлы записи очень компактны и очень эффективны. Как мы увидим в следующей главе, когда мы будем обучать AlexNet с нуля на ImageNet, мы будем использовать эти файлы .rec с помощью специального ImageRecordIter. Подобно нашему HDF5DatasetGenerator, класс ImageRecordIter позволит нам:

1. Перебрать все изображения в заданном файле .rec в мини-пакетах.
2. Примените увеличение данных к каждому изображению в каждом пакете.

Чтобы увидеть, как все части сочетаются друг с другом, чтобы мы могли обучать сверточную нейронную сеть с нуля в ImageNet, перейдите к следующей главе.

6. Обучение AlexNet на ImageNet

В нашей предыдущей главе мы подробно обсудили набор данных ImageNet; в частности, структура каталогов набора данных и используемые вспомогательные метафайлы предоставляют метки классов для каждого изображения. Оттуда мы определили два набора файлов:

1. Файл конфигурации, позволяющий нам легко создавать новые эксперименты при обучении Convolutional. Нейронные сети на ImageNet.
2. Набор служебных скриптов для подготовки набора данных к преобразованию из необработанных изображений, находящихся на диск в эффективно упакованный файл записи `txhnet`.

Используя двоичный файл `im2rec`, предоставленный `txhnet`, вместе с файлами `.lst`, которые мы создали с помощью наших служебных сценариев, мы смогли создать файлы записей для каждого из наших обучающих, тестовых и проверочных наборов. Прелест этого подхода в том, что файлы `.rec` нужно создать только один раз — мы можем повторно использовать эти файлы записей для любого эксперимента по классификации ImageNet, который мы хотим провести.

Во-вторых, сами файлы конфигурации также можно использовать повторно. Хотя мы создали наш первый файл конфигурации с учетом AlexNet, реальность такова, что мы будем использовать один и тот же файл конфигурации для VGGNet, GoogLeNet, ResNet и SqueezeNet — единственный аспект файла конфигурации, который необходимо изменить . при обучении новой сети на ImageNet являются: 1. Название сетевой архитектуры (которое заложено в имени файла конфигурации).

2. Размер партии.

3. Количество графических процессоров для обучения сети (если применимо).

В этой главе мы сначала реализуем архитектуру AlexNet с помощью библиотеки `txhnet`. Мы уже однажды реализовывали AlexNet в главе 10 пакета Practitioner Bundle с помощью Keras. Как вы увидите, существует много параллелей между `txhnet` и Keras, что делает чрезвычайно простым перенос реализации между двумя библиотеками. Оттуда я продемонстрирую, как обучить AlexNet на наборе данных ImageNet.

Эта глава и все другие главы в этом пакете, демонстрирующие, как обучать заданную сетевую архитектуру на наборе данных ImageNet, рассматриваются как нечто среднее между конкретным примером и лабораторным журналом. Для каждой главы этой книги я провел от десятков до сотен экспериментов, чтобы получить соответствующие результаты. Я хочу поделиться с вами своим ходом мыслей при обучении глубоких современных нейронных сетей на сложном наборе данных ImageNet, чтобы вы могли набраться опыта, наблюдая за мной.

получить неоптимальные результаты, а затем настроить несколько параметров, чтобы повысить точность воспроизведения ультрасовременное исполнение. Делиться «историей» о том, как обучалась сеть, а не только окончательный результат, поможет вам в ваших собственных экспериментах по глубокому обучению. Наблюдая за другими и тогда обучение на собственном опыте является оптимальным способом быстро овладеть методами, необходимыми для успешная работа с большими наборами данных изображений и глубокое обучение.

6.1 Внедрение AlexNet

Первым шагом в обучении AlexNet на ImageNet является реализация архитектуры AlexNet с использованием библиотека mxnet. Мы уже рассмотрели основополагающую архитектуру Крижевского и др. [8] в главе 10. пакета Practitioner Bundle, где мы обучали AlexNet решению задачи Kaggle Dogs vs. Cats, поэтому эта сеть не должна казаться новой и незнакомой. Тем не менее, я включил Таблицу 6.1, представляющую структура сети как вопрос полноты.

Теперь мы реализуем эту архитектуру с помощью Python и mxnet. Как личное предпочтение, Мне нравится, когда мои реализации mxnet CNN отделены от моих реализаций Keras CNN.

Поэтому я создал подмодуль с именем mxconv внутри модуля nn pyimagesearch:

```
-- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- nn
| | |--- __init__.py
| | |--- конв.
| | |--- mxconv
| | --- предварительная обработка
| | --- утилиты
```

Все сетевые архитектуры, реализованные с использованием mxnet, будут жить внутри этого подмодуля (поэтому имя модуля начинается с текста mx). Создайте новый файл с именем mxalexnet.py внутри mxconv. для хранения нашей реализации класса MxAlexNet:

```
-- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- nn
| | |--- __init__.py
| | |--- конв.
| | |--- mxconv
| | | |--- __init__.py
| | | |--- mxalexnet.py
| | --- предварительная обработка
| | --- утилиты
```

Оттуда мы можем начать реализацию AlexNet:

```
1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3
4 класс MxAlexNet:
5     @статический метод
6         деф- сборка (классы):
```

Тип слоя	Выходной размер	Размер фильтра/шаг
ВХОДНОЕ ИЗОБРАЖЕНИЕ	227× 227× 3	
КОНВ.	55× 55× 96	11× 11/4× 4, K = 96
ДЕЙСТВОВАТЬ	55× 55× 96	
БН	55× 55× 96	
БАССЕЙН	27× 27× 96	3× 3/2× 2
ВЫБЫВАТЬ	27× 27× 96	
КОНВ.	27× 27× 256	5× 5, K = 256
ДЕЙСТВОВАТЬ	27× 27× 256	
БН	27× 27× 256	
БАССЕЙН	13× 13× 256	3× 3/2× 2
ВЫБЫВАТЬ	13× 13× 256	
КОНВ.	13× 13× 384	3× 3, K = 384
ДЕЙСТВОВАТЬ	13× 13× 384	
БН	13× 13× 384	
КОНВ.	13× 13× 384	3× 3, K = 384
ДЕЙСТВОВАТЬ	13× 13× 384	
БН	13× 13× 384	
КОНВ.	13× 13× 256	3× 3, K = 256
ДЕЙСТВОВАТЬ	13× 13× 256	
БН	13× 13× 256	
БАССЕЙН	13× 13× 256	3× 3/2× 2
ВЫБЫВАТЬ	6× 6× 256	
ФК	4096	
ДЕЙСТВОВАТЬ	4096	
БН	4096	
ВЫБЫВАТЬ	4096	
ФК	4096	
ДЕЙСТВОВАТЬ	4096	
БН	4096	
ВЫБЫВАТЬ	4096	
ФК	1000	
СОФТМАКС	1000	

Таблица 6.1: Сводная таблица архитектуры AlexNet. Размеры выходного тома включены для каждого слой, а также размер сверточного фильтра/размер пула, когда это уместно.

```

7      # ввод данных
8      данные = mx.sym.Variable("данные")

```

Строка 2 импортирует нашу единственную необходимую библиотеку, mxnet для удобства имеет псевдоним mx . Для читателей аналогично OpenCV и библиотеке cv2 , вся функциональность аккуратно организована и содержится внутри одиночный импорт, преимущества которого заключается в том, что наш блок кода импорта остается аккуратным и чистым, а также требует, чтобы код архитектуры был немного более подробным.

Затем мы определяем метод сборки для MxAlexNet в строке 6, стандарт, который мы разработали. на протяжении всей этой книги. Метод build отвечает за построение сети архитектуру и возвращая ее вызывающей функции.

Однако, изучив структуру, вы заметите, что нам нужен только один аргумент, классы, общее количество меток классов в нашем наборе данных. Количество аргументов отличается от реализации CNN в Keras, где нам также необходимо указать пространственные размеры, включая ширину, высота, глубина. Почему нам не нужно передавать эти значения в библиотеку mxnet? В виде мы узнаем, причина, по которой mxnet явно не требует пространственных измерений, заключается в том, что Класс ImageRecordIter (класс, отвечающий за чтение изображений из наших файлов записей) автоматически определяет пространственные размеры наших изображений — нет необходимости явно передавать пространственные параметры. размеры в класс.

Строка 8 определяет переменную mxnet с именем data — это очень важная переменная, поскольку она представляет входные данные в нейронную сеть. Без этой переменной наша сеть не смогла бы получать входы, поэтому мы не сможем обучить или оценить его.

Далее реализуем первый набор слоев CONV => RELU => POOL:

```

10     # Блок №1: первый набор слоев CONV => RELU => POOL
11     conv1_1 = mx.sym.Свертка (данные = данные, ядро = (11, 11),
12         шаг=(4, 4), num_filter=96)
13     act1_1 = mx.sym.LeakyReLU(data=conv1_1, act_type="elu")
14     bn1_1 = mx.sym.BatchNorm (данные = act1_1)
15     pool1 = mx.sym.Pooling(data=bn1_1, pool_type="max",
16         ядро = (3, 3), шаг = (2, 2))
17     do1 = mx.sym.Dropout (данные = pool1, p = 0,25)

```

Как видите, API mxnet аналогичен API Keras. Названия функций изменены немного, но в целом легко увидеть, как имена функций mxnet сопоставляются с именами функций Keras (например, Conv2D в Keras — это просто Convolution в mxnet). Каждый уровень в mxnet требует, чтобы вы передали аргумент данных — этот аргумент данных является входом для слоя. Использование Keras и последовательного модель, ввод будет автоматически выводиться. И наоборот, mxnet дает вам гибкость легко строить структуры графов, где входные данные для одного слоя не обязательно являются выходными данными для предыдущий слой. Эта гибкость особенно удобна, когда мы начинаем определять более экзотические архитектуры. такие как GoogleNet и ResNet.

Строки 11 и 12 определяют наш первый слой CONV . Этот слой принимает наши входные данные в качестве входных данных, затем применяет размер ядра 11×11 пикселей с шагом 4×4 и изучением num_filter=96 фильтры.

Первоначальная реализация AlexNet использовала стандартные слои ReLU; тем не менее, я часто находил что ELU работают лучше, особенно в наборе данных ImageNet; поэтому мы применим ELU к Стока 13. Активация ELU реализована внутри класса LeakyReLU , который принимает наш первый Слой CONV , conv1_1 в качестве входных данных. Затем мы указываем act_type="elu" , чтобы указать, что мы хотим использовать вариант ELU семейства Leaky ReLU. После применения активации мы хотим выполнить нормализация партии в строках 14 с помощью класса BatchNorm . Обратите внимание, что мы не обязаны поставлять

ось канала для нормализации активаций – ось канала определяется mxnet автоматически.

Чтобы уменьшить пространственные размеры ввода, мы можем применить класс Pooling к строкам 15 и 16. Класс Pooling принимает выходные данные bn1_1 в качестве входных данных, а затем применяет максимальное объединение с размером ядра 3×3 и шаг 2×2 . Dropout применяется к строке 17, чтобы уменьшить переоснащение.

Учитывая, что это ваше первое знакомство с реализацией набора слоев в mxnet, я бы порекомендовал вернуться и перечитать этот раздел еще раз или два. Существуют очевидные параллели между соглашениями об именах в mxnet и Keras, но убедитесь, что вы понимаете их сейчас, особенно data и как мы должны явно определить вход текущего слоя как выход предыдущий слой.

Остальную часть реализации AlexNet я объясню менее утомительно:

1. Вы уже однажды реализовали AlexNet в пакете Practitioner Bundle.
2. Сам код говорит сам за себя, и объяснение каждой строки кода до тонкоты стать утомительным.

Наш следующий набор слоев состоит из другого блока слоев CONV => RELU => POOL:

```

19      # Блок №2: второй набор слоев CONV => RELU => POOL
20      conv2_1 = mx.sym.Convolution(данные = do1, ядро = (5, 5),
21          pad=(2, 2), num_filter=256)
22      act2_1 = mx.sym.LeakyReLU(data=conv2_1, act_type="elu")
23      bn2_1 = mx.sym.BatchNorm(данные = act2_1)
24      pool2 = mx.sym.Pooling(data=bn2_1, pool_type="max",
25          ядро = (3, 3), шаг = (2, 2))
26      do2 = mx.sym.Dropout(данные = pool2, p = 0,25)

```

Здесь наш слой CONV изучает 256 фильтров, каждый размером 5×5 . Однако, в отличие от Keras, который может автоматически определить необходимое количество отступов (т. е. padding="same"), нам нужно явно указать введите значение пэда , как указано в таблице выше. Предоставление pad=(2, 2) гарантирует, что ввод и выходные пространственные размеры одинаковы.

После слоя CONV применяется еще одна активация ELU, за которой следует BN. Максимальный пул применяется к выходу BN, уменьшая размер пространственного входа до 13×13 пикселей. Опять отсев применяется для уменьшения переобучения.

Чтобы изучить более глубокие и богатые функции, мы наложим несколько слоев CONV => RELU поверх каждого. Другие:

```

28      # Блок №3: (CONV => RELU) * 3 => POOL
29      conv3_1 = mx.sym.Convolution(данные = do2, ядро = (3, 3),
30          pad=(1, 1), num_filter=384)
31      act3_1 = mx.sym.LeakyReLU(data=conv3_1, act_type="elu")
32      bn3_1 = mx.sym.BatchNorm(данные = act3_1)
33      conv3_2 = mx.sym.Convolution(данные = bn3_1, ядро = (3, 3),
34          pad=(1, 1), num_filter=384)
35      act3_2 = mx.sym.LeakyReLU(data=conv3_2, act_type="elu")
36      bn3_2 = mx.sym.BatchNorm(данные = act3_2)
37      conv3_3 = mx.sym.Convolution(данные = bn3_2, ядро = (3, 3),
38          pad=(1, 1), num_filter=256)
39      act3_3 = mx.sym.LeakyReLU(data=conv3_3, act_type="elu")
40      bn3_3 = mx.sym.BatchNorm(данные = act3_3)
41      pool3 = mx.sym.Pooling(data=bn3_3, pool_type="max",
42          ядро = (3, 3), шаг = (2, 2))
43      do3 = mx.sym.Dropout(данные = pool3, p = 0,25)

```

Наш первый слой CONV изучает 384 фильтра 3×3 , используя размер заполнения (1,1), чтобы обеспечить ввод пространственные измерения соответствуют выходным пространственным измерениям. Активация применяется немедленно после свертки с последующей нормализацией партии. Наш второй слой CONV также изучает 384, фильтры 3×3 , снова с последующей активацией и нормализацией партии. Окончательный CONV в слое set уменьшает количество изученных фильтров до 256; однако сохраняет тот же размер файла 3×3 .

Выход окончательного CONV затем проходит через активацию и нормализацию партии. А операция POOL снова используется для уменьшения пространственных размеров тома. Выпадение следует POOL, чтобы помочь уменьшить переоснащение.

Следуя таблице 6.1 выше, следующим шагом в реализации AlexNet является определение двух набора слоев FC:

```

45      # Блок №4: первый набор слоев FC => RELU
46      сгладить = mx.sym.Flatten (данные = do3)
47      fc1 = mx.sym.FullyConnected (данные = сглаживание, num_hidden = 4096)
48      act4_1 = mx.sym.LeakyReLU(data=fc1, act_type="elu")
49      bn4_1 = mx.sym.BatchNorm (данные = act4_1)
50      do4 = mx.sym.Dropout (данные = bn4_1, p = 0,5)
51
52      # Блок №5: второй набор слоев FC => RELU
53      fc2 = mx.sym.FullyConnected (данные = do4, num_hidden = 4096)
54      act5_1 = mx.sym.LeakyReLU(data=fc2, act_type="elu")
55      bn5_1 = mx.sym.BatchNorm (данные = act5_1)
56      do5 = mx.sym.Dropout (данные = bn5_1, p = 0,5)

```

Каждый из слоев FC включает в себя 4096 скрытых модулей, за каждым из которых следует активация, нормализация и более агрессивное отсев 50%. Обычно используется отсев 40-50% в слоях FC, так как именно там соединения CNN наиболее плотные, и переоснащение, скорее всего, происходит.

Наконец, мы применяем наш классификатор softmax, используя предоставленное количество классов:

```

58      # классификатор softmax
59      fc3 = mx.sym.FullyConnected (данные = do5, num_hidden = классы)
60      модель = mx.sym.SoftmaxOutput (данные = fc3, имя = "softmax")
61
62      # вернуть архитектуру сети
63      модель возврата

```

Изучив эту реализацию, вы можете быть удивлены тем, насколько похожа библиотека mxnet находится в Keras. Хотя имена функций и не идентичны, их очень легко сопоставить друг с другом, как и имена функций. Параметры. Возможно, единственное неудобство заключается в том, что теперь мы должны явно вычислить наш отступ. вместо того, чтобы полагаться на автоматический вывод Keras. В противном случае реализация Сверточные нейронные сети с mxnet так же просты, как и Keras.

6.2 Обучение AlexNet

Теперь, когда мы реализовали архитектуру AlexNet в mxnet, нам нужно определить сценарий драйвера. Отвечает за фактическое обучение сети. Подобно Keras, обучение сети с помощью mxnet довольно просто, хотя есть два ключевых отличия:

1. Обучающий код mxnet немного более подробный из-за того, что мы хотели бы использовать

несколько графических процессоров (если возможно).

2. Библиотека mxnet не предоставляет удобного метода для построения графика потерь/точности с течением времени. и вместо этого записывает прогресс обучения в терминал.

Поэтому нам нужно использовать пакет ведения журнала Python, чтобы зафиксировать этот вывод и сохранить его на диск. Затем мы вручную проверяем выходные журналы прогресса обучения, а также пишем служебные сценарии Python для анализа журналов и построения графика обучения/потери. Это немного утомительнее, чем использование Keras; тем не менее, преимущество обучения сетей значительно быстрее благодаря тому, что (1) mxnet представляет собой скомпилированную библиотеку C++ с привязками Python и (2) несколько графических процессоров, стоит компромисса.

Я включил пример журнала обучения mxnet ниже:

Начните обучение с [gpu(0), gpu(1), gpu(2), gpu(3), gpu(4)],

графический процессор (5), графический процессор (6), графический процессор (7)

Эпоха[0] Пакет [1000] Скорость: 1677,33 выборки/сек Train-accuracy=0,004186
 Эпоха[0] Пакет [1000] Скорость: 1677,33 выборки/сек Train-top_k_accuracy_5=0,0181 Эпоха[0]
 Пакет [1000] Скорость: 1677,33 выборки /сек Train-cross-entropy=6,748022 Эпоха[0] Сброс
 Итератора Данных Эпоха[0] Стоимость времени=738,577 Контрольная точка сохранена в
 "imagenet/checkpoints/alexnet-0001.params"

Эпоха[0] Validation-accuracy=0,008219
 Epoch[0] Validation-top_k_accuracy_5=0,031189
 Epoch[0] Validation-cross-entropy=6,629663 Epoch[1]
 Пакет [1000] Скорость: 1676,29 выборок/сек Train-accuracy=0,028924 Epoch[1] Пакет
 [1000] Скорость: 1676,29 выборок/сек Train-top_k_accuracy_5=0,0967 Эпоха[1] Пакет [1000]
 Скорость: 1676,29 выборок/сек Перекрестная энтропия поезда=5,883830 Эпоха[1] Сброс
 итератора данных Эпоха[1] Время cost=734.455 Сохраненная контрольная точка в
 "imagenet/checkpoints/alexnet-0002.params"

Эпоха[1] Validation-accuracy=0,052816
 Epoch[1] Validation-top_k_accuracy_5=0,150838
 Epoch[1] Validation-cross-entropy=5,592251 Epoch[2]
 Пакет [1000] Скорость: 1675,09 выборок/сек Train-accuracy=0,073691 Epoch[2] Пакет
 [1000] Скорость: 1675,09 выборок/сек Train-top_k_accuracy_5=0,2045 Эпоха[2] Пакет [1000]
 Скорость: 1675,09 выборок/сек Перекрестная энтропия поезда=5,177066 Эпоха[2] Сброс
 итератора данных Эпоха[2] Время cost=733,579 Контрольная точка сохранена в "imagenet/
 checkpoints/alexnet-0003.params"

Эпоха[2] Проверка-точность=0,094177
 Эпоха[2] Проверка-top_k_accuracy_5=0,240031
 Эпоха[2] Проверка-кросс-энтропия=5,039742

Здесь вы можете видеть, что я тренирую AlexNet на восьми графических процессорах (достаточно одного графического процессора, но я использовал восемь, чтобы быстрее собирать результаты). После каждого установленного количества пакетов (которое мы определим позже) потери при обучении, точность ранга 1 и ранга 5 записываются в файл. По завершении эпохи итератор обучающих данных сбрасывается, создается файл контрольных точек и сериализуются веса модели, а также отображаются потери проверки, точность ранга 1 и ранга 5. Как мы видим, после первой эпохи AlexNet получает 3% точности ранга 1 для обучающих данных и 6% точности ранга 1 для проверочных данных.

6.2.1 Что насчет тренировочных участков?

Журнал тренировок достаточно легко читать и интерпретировать; тем не менее, сканирование простого текстового файла не компенсирует отсутствие визуализации — фактическая визуализация графика потерь и точности с течением времени может позволить нам принимать более обоснованные решения относительно того, нужно ли нам корректировать скорость обучения, применять больше регуляризации и др.

Библиотека mxnet (к сожалению) не поставляется с готовым инструментом для анализа журналов и построения графика обучения, поэтому я создал отдельный инструмент Python для выполнения этой задачи . Вместо того, чтобы еще больше раздувать эту главу служебным кодом (который просто сводится к использованию базового программирования и регулярных выражений для анализа журнала), я решил осветить эту тему непосредственно в блоге PyImageSearch — вы можете узнать больше о сценарии plot_log.py здесь. : <http://pyimg.co/ycyao> Однако на данный момент просто поймите, что этот сценарий используется для анализа журналов обучения mxnet и построения графика наших соответствующих потерь и точности обучения и проверки.

6.2.2 Реализация сценария обучения

Теперь, когда мы определили архитектуру AlexNet, нам нужно создать скрипт Python для фактического обучения сети на наборе данных ImageNet. Для начала откройте новый файл, назовите его train_alexnet.py и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2
из config import imagenet_alexnet_config as config 3 из
pyimagesearch.nn.mxconv import MxAlexNet 4 import mxnet as mx 5
import argparse 6 import logging 7 import json 8 import os
```

Строки 2-8 импортируют необходимые пакеты Python. Обратите внимание, как мы импортируем наш `imagenet_alexnet_config` с псевдонимом `config`, чтобы мы могли получить доступ к нашим конфигурациям обучения, специфичным для ImageNet. Затем мы импортируем нашу реализацию архитектуры AlexNet в mxnet в строке 3. Как я упоминал ранее в этой главе, mxnet записывает ход обучения в файл; поэтому нам нужен пакет протоколирования в строке 6 , чтобы захватить выходные данные mxnet и сохранить их непосредственно в файл, который мы сможем позже проанализировать.

Отсюда давайте разберем наши аргументы командной строки:

```
10 # построить разбор аргумента и разобрать аргументы 11 ap =
argparse.ArgumentParser() 12 ap.add_argument("-c", "--checkpoints",
required=True,
13         help="путь к выходному каталогу контрольных точек")
14 ap.add_argument("-p", "--prefix", required=True,
15         help="имя префикса модели") 16
ap.add_argument("-s", "--start-epoch", type=int, default=0,
17         help="эпоха для возобновления обучения")
18 args = vars(ap.parse_args())
```

Наш скрипт `train_alexnet.py` требует двух переключателей, за которыми следует третий необязательный. Переключатель `-checkpoints` управляет путем к выходному каталогу, в котором веса нашей модели будут сериализованы после каждой эпохи. В отличие от Keras, где нам нужно явно указать, когда модель будет сериализована на диск, mxnet делает это автоматически после каждой эпохи.

Аргумент командной строки `-prefix` — имя архитектуры, которую вы обучаете. В нашем случае мы будем использовать префикс `alexnet`. Имя префикса будет включено в имя файла каждого сериализованного файла гирь.

Наконец, мы также можем указать `--start-epoch`. При обучении AlexNet на ImageNet мы неизбежно заметим признаки стагнации обучения или переобучения. В этом случае нам нужно нажать `ctrl +`

с вне сценария, настроить нашу скорость обучения и продолжить обучение. Указав `--start -epoch`, мы можем возобновить обучение с определенной предыдущей эпохи, которая была сериализована на диск.

При обучении нашей сети с нуля мы будем использовать следующую команду, чтобы начать процесс обучения:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet
```

Однако, если мы хотим возобновить обучение с определенной эпохи (в данном случае с эпохи 50), мы предоставит эту команду:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet \ --start-epoch 50
```

Помните об этом процессе при обучении AlexNet на наборе данных ImageNet.

Как я упоминал ранее, mxnet использует ведение журнала для отображения прогресса обучения. Вместо того, чтобы отображать журнал обучения на стандартный вывод, мы должны вместо этого захватить журнал и сохранить его на диск, чтобы мы могли проанализировать его и просмотреть позже:

```
20 # установить уровень логирования и выходной файл
21 logging.basicConfig(level=logging.DEBUG,
22     filename="training_{epoch}.log".format(args["start_epoch"]), filemode="w")
23
```

Строки 21 и 22 создают файл с именем `training_{epoch}.log` на основе значения `--start-epoch`.

Наличие уникального имени файла для каждой начальной эпохи упростит отображение точности и потерь во времени, как обсуждалось в сообщении блога PyImageSearch выше.

Затем мы можем загрузить наши средства RGB с диска и вычислить размер пакета:

```
25 # загрузите средства RGB для тренировочного набора, затем определите размер пакета 26 #
```

```
27 означает = json.loads(open(config.DATASET_MEAN).read()) 28
batchSize = config.BATCH_SIZE * config.NUM_DEVICES
```

Раньше размер пакета всегда был фиксированным, поскольку мы использовали только один ЦП/ГП/устройство для обучения наших сетей. Однако теперь, когда мы начинаем исследовать мир нескольких графических процессоров, размер нашего пакета на самом деле составляет `BATCH_SIZE * NUM_DEVICES`. Причина, по которой мы умножаем наш первоначальный размер пакета на общее количество процессоров/графических процессоров/устройств, на которых мы обучаем нашу сеть, заключается в том, что каждое устройство параллельно анализирует отдельный пакет изображений. После того, как все устройства обработали свой пакет, mxnet обновляет соответствующие веса в сети. Таким образом, размер нашего пакета фактически увеличивается с количеством устройств, которые мы используем для обучения, благодаря распараллеливанию.

Конечно, нам нужно получить доступ к нашим обучающим данным, чтобы обучить AlexNet на ImageNet:

```
30 # создать итератор обучающего изображения
```

—

32

33

34

35

```

36     rand_mirror = Верно,
37     повернуть=15,
38     max_shear_ratio=0,1,
39     mean_r = означает ["R"],
40     mean_g = означает ["G"],
41     mean_b = означает ["B"],
42 preprocess_threads=config.NUM_DEVICES * 2)

```

Строки 31-42 определяют `ImageRecordIter`, ответственный за чтение наших пакетов изображений из наш файл записи тренировок. Здесь мы указываем, что каждое выходное изображение из итератора должно быть изменено. до 227× 227 пикселей при увеличении данных, включая случайное кадрирование, случайное искажение, случайное вращение и сдвиг. Вычитание среднего также применяется внутри итератора.

Чтобы ускорить обучение (и гарантировать, что наша сеть не ждет новых образцов из итератора данных), мы можем предоставить значение для `preprocess_threads` — генерирует N потоков, которые опрашивают пакеты изображений для итератора и применить увеличение данных. Я нашел хорошее эмпирическое правило: установите количество `preprocess_threads` в два раза больше количества устройств, которые вы используете для обучения сеть.

Точно так же, как нам нужен доступ к нашим обучающим данным, нам также нужен доступ к нашим проверочным данным:

```

44 # создать итератор проверочного изображения
45 valIter = mx.io.ImageRecordIter(
46     path_imgrec=config.VAL_MX_REC,
47     data_shape=(3, 227, 227),
48     batch_size = размер партии,
49     mean_r = означает ["R"],
50     mean_g = означает ["G"],
51     mean_b = означает ["B"])

```

Итератор данных проверки идентичен итератору обучающих данных, за исключением того, что увеличение данных не применяется (но мы применяем нормализацию среднего вычитания).

Далее давайте определим наш оптимизатор:

```

53 # инициализировать оптимизатор
54 opt = mx.optimizer.SGD(learning_rate=1e-2, импульс=0,9, wd=0,0005,
55     rescale_grad=1,0 / размер партии)

```

Как и в оригинальной статье Крижевского, мы будем обучать AlexNet с использованием SGD с начальным скорость обучения $1e^{-2}$, импульсный член $\gamma = 0,9$ и L2-весовая регуляризация (т. е. «весовой распад») 0,0005. Параметр `rescale_grad` в оптимизаторе SGD очень важен, так как он масштабирует градиенты по размеру партии. Без этого масштабирования наша сеть не сможет обучаться.

Следующий блок кода обрабатывает определение пути к нашему выходному каталогу путей контрольных точек, а также с инициализацией параметров аргумента и вспомогательных параметров сети:

```

57 # построить путь контрольных точек, инициализировать аргумент модели и
58 # вспомогательные параметры
59 checkpointsPath = os.path.sep.join([args["checkpoints"],
60     аргументы["предфикс"]])
61 argParams = Нет
62 дополнительных параметра = нет

```

В случае, если мы обучаем AlexNet с самой первой эпохи, нам нужно построить сеть архитектура:

```
64 # если не указана начальная эпоха конкретной модели, то
65 # инициализируем сеть
66 , если args["start_epoch"] <= 0:
67     # построить архитектуру LeNet
68     print("[INFO] строим сеть...")
69     модель = MxAlexNet.build(config.NUM_CLASSES)
```

В противном случае, если мы перезапускаем обучение с определенной эпохи, нам нужно загрузить сериализованный веса с диска и извлечь параметры аргумента, вспомогательные параметры и «символ» модели (т. е. то, что mxnet называет скомпилированной сетью):

```
71 # в противном случае указана конкретная контрольная точка
72 еще:
73     # загружаем чекпойнт с диска
74     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
75     модель = mx.model.FeedForward.load(путь к контрольным точкам,
76                                         аргументы["начало_эпохи"])
77
78     # обновить модель и параметры
79     argParams = модель.arg_params
80     auxParams = модель.aux_params
81     модель = модель.символ
```

Независимо от того, начинаем ли мы обучение с первой первой эпохи или перезапускаем обучения из определенной эпохи, нам нужно инициализировать объект FeedForward, представляющий сеть, которую мы хотим обучить:

```
83 # компилируем модель
84 модель = mx.model.FeedForward(
85     ctx=[mx.gpu(1), mx.gpu(2), mx.gpu(3)],
86     символ = модель,
87     инициализатор=mx.initializer.Xavier(),
88     arg_params = параметры аргумента,
89     aux_params = вспомогательные параметры,
90     оптимизатор=выбор,
91     число_эпох = 90,
92     begin_epoch=args["start_epoch"])
```

Параметр ctx управляет контекстом нашего обучения. Здесь мы можем предоставить список графических процессоров, ЦП или устройства, используемые для обучения сети. В данном случае я использую три графических процессора для обучения AlexNet; однако вам следует изменить эту строку в зависимости от количества графических процессоров, доступных в вашей системе. Если у вас есть только один графический процессор, тогда строка 85 будет выглядеть так:

```
85     ctx=[mx.gpu(0)],
```

Инициализатор управляет методом инициализации веса для всех весов позже в сети — здесь мы используем инициализацию Xavier (также известную как Glorot), метод инициализации по умолчанию для

большинство сверточных нейронных сетей (и та, которая используется по умолчанию для Keras). Мы позволим AlexNet обучаться в течение максимум 100 эпох, но опять же, это значение, скорее всего, будет корректироваться по мере обучения вашей сети и мониторинга процесса — вполне может быть меньше эпох или больше эпох.

Наконец, параметр `begin_epoch`s определяет, с какой эпохи мы должны возобновить обучение. Этот параметр важен, так как он позволяет mxnet поддерживать в порядке свои внутренние учетные переменные.

Точно так же, как Keras предоставляет нам обратные вызовы для мониторинга эффективности обучения, так же и mxnet:

```
94 # инициализировать обратные вызовы и метрики оценки
95 batchEndCBs = [mx.callback.Speedometer(batchSize, 500)] 96
epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)] 97 metrics =
[mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5), mx.metric.CrossEntropy()]
98
```

В строке 95 мы определяем обратный вызов спидометра , который вызывается в конце каждого пакета. Этот обратный вызов предоставит нам обучающую информацию после каждого batchSize * 500 пакетов. Вы можете уменьшить или увеличить это значение в зависимости от того, как часто вы хотите получать обновления тренировок. Меньшее значение приведет к большему количеству обновлений в журнале, а большее значение будет означать меньшее количество обновлений в журнале.

Строка 96 определяет наш обратный вызов `do_checkpoint` , который вызывается в конце каждой эпохи. Этот обратный вызов отвечает за сериализацию весов нашей модели на диск. Опять же, в конце каждой эпохи наши сетевые веса будут сериализованы на диск, что позволит нам перезапустить обучение с определенной эпохи , когда это необходимо.

Наконец, строки 97 и 98 инициализируют наш список обратных вызовов метрик . Мы будем отслеживать точность ранга 1 (Accuracy), точность ранга 5 (TopKAccuracy), а также категориальную кросс-энтропийную потерю.

Осталось только обучить нашу сеть:

```
100 # обучить сеть
101 print("[ИНФО] тренировочная сеть...")
102 model.fit(
103     X=trainIter,
104     eval_data=valIter,
105     eval_metric=метрики,
106     batch_end_callback=batchEndCB,
107     epoch_end_callback=epochEndCB)
```

Вызов `.fit` модели запускает (или перезапускает) процесс обучения. Здесь нам нужно предоставить итератор обучающих данных, итератор проверки и любые обратные вызовы. После вызова `.fit` mxnet начнет записывать результаты в наш выходной файл журнала, чтобы мы могли просмотреть процесс обучения.

Хотя код, необходимый для обучения сети с использованием mxnet, может показаться немного многословным, имейте в виду , что это буквально схема обучения любой сверточной нейронной сети на наборе данных ImageNet. Когда мы внедряем и обучаем другие сетевые архитектуры, такие как VGGNet, GoogLeNet и т. д., нам просто нужно:

1. Измените строку 2 , чтобы правильно настроить файл конфигурации.
2. Обновите `data_shape` в `trainIter` и `valIter` (только если для сети требуются другие пространственные размеры входного изображения).
3. Обновите оптимизатор `SGD` в строках 54 и 55.
4. Измените имя инициализируемой модели в строке 69.

Помимо этих трех изменений, для нашего сценария буквально не требуется никаких других обновлений при обучении различных CNN в наборе данных ImageNet. Я специально закодировал наш сценарий обучения таким образом, чтобы он был переносимым и расширяемым. Будущие главы ImageNet Bundle потребуют гораздо меньше кода — мы просто реализуем новую сетевую архитектуру, внесем несколько изменений в оптимизатор и

конфигурационный файл, и все будет запущено в течение нескольких минут. Я надеюсь, что вы будете использовать этот же скрипт при реализации и экспериментировании со своими собственными сетевыми архитектурами глубокого обучения.

6.3 Оценка AlexNet

В этом разделе мы узнаем, как оценить свёрточную нейронную сеть, обученную на наборе данных ImageNet. В этой главе конкретно обсуждается AlexNet; однако, как мы увидим далее в этой книге, тот же сценарий можно использовать и для оценки VGGNet, GoogLeNet и т. д., просто изменив файл импорта конфигурации.

Чтобы увидеть, как мы можем изменить конфигурацию, откройте новый файл, назовите его `test_alexnet.py`, и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2
из конфига импортируем imagenet_alexnet_config as config 3
импортируем mxnet как mx 4 импортируем argparse 5 импортируем
json 6 импортируем os
```

Строки 2-6 импортируют необходимые пакеты Python. Обратите особое внимание на строку 2 , где мы импортируем наш файл конфигурации. Как упоминалось выше, тот же самый набор сценариев можно использовать для оценки других сетей, просто изменив строку 2 , чтобы она соответствовала файлу конфигурации для вашей соответствующей сети.

Оттуда мы можем проанализировать наши аргументы командной строки:

```
8 # построить разбор аргумента и разобрать аргументы 9 ap =
argparse.ArgumentParser() 10 ap.add_argument("-c", "--checkpoints",
required=True,
11     help="путь к выходному каталогу контрольных точек")
12 ap.add_argument("-p", "--prefix", required=True,
13     help="имя префикса модели") 14
ap.add_argument("-e", "--epoch", type=int, required=True, help="epoch # для
15     загрузки") 16 args = vars(ap.parse_args())
```

Наш скрипт требует три переключателя, каждый из которых подробно описан ниже: 1. `-checkpoints`: это путь к нашему выходному каталогу контрольных точек во время обучения. процесс.

2. `-prefix`: Префикс – это имя нашей фактической CNN. Когда мы запускаем `test_alexnet.py` скрипт, мы укажем значение `alexnet` для `-prefix`.

3. `-epoch`: Здесь мы указываем эпоху нашей сети, которую мы хотим использовать для оценки. Например , если мы остановили наше обучение после эпохи 100, то мы бы использовали 100-ю эпоху для оценки нашей сети на данных тестирования.

Эти три аргумента командной строки необходимы, так как все они используются для создания пути к сериализованные веса моделей, находящиеся на диске.

Чтобы оценить нашу сеть, нам нужно создать `ImageRecordIter` для циклического просмотра данных тестирования:

```
18 # загрузить средства RGB для тренировочного набора
19 mean = json.loads(open(config.DATASET_MEAN).read())
```

```

20
21 # создать итератор тестового изображения -
_ ["G"], mean_b=означает["B"])
23
24
25
26
27
28

```

Строка 19 загружает средние значения красных, зеленых и синих пикселей по всему тренировочному набору точно так же, как мы делали это в процессе обучения. Эти значения будут вычтены из каждого из отдельных каналов RGB в изображении во время тестирования перед передачей по сети для получения наших выходных классификаций. Напомним, что вычитание среднего является формой нормализации данных и, следовательно, должно выполняться на всех трех обучающих, тестовых и проверочных наборах.

Строки 22-28 определяют testIter, используемый для циклического перебора пакетов изображений в тестовом наборе. Нет в этом случае необходимости выполнить аугментацию данных, поэтому мы просто предоставим: 1. Путь к файлу записи тестирования.

2. Предполагаемые пространственные размеры изображений (3 канала, ширина и высота 227× 227 , соответственно).
3. Размер партии, используемый при оценке – этот параметр менее важен при тестировании, поскольку во время торговли, так как мы просто хотим получить наши выходные прогнозы.
4. Значения RGB, используемые для вычитания/нормализации среднего.

Наши веса AlexNet для каждой эпохи сериализуются на диск, поэтому следующим шагом будет определение пути к каталогу выходных контрольных точек с использованием префикса (имя сети) и эпохи (конкретные веса, которые мы хотим загрузить):

```

30 # загрузить контрольную точку с
диска 31 print("[INFO] loading model...")
32 checkpointsPath = os.path.sep.join([args["checkpoints"],
33         args["prefix"]])
34 model = mx.model.FeedForward.load(checkpointsPath,
35         аргументы["эпоха"])

```

Строки 32 и 33 определяют путь к нашему выходному каталогу -checkpoints с использованием префикса модели -. Как мы знаем в процессе обучения, выходные сериализованные веса сохраняются с использованием следующего соглашения об именах файлов:

каталог контрольных точек/префикс-epoch.params

Переменная checkpointsPath содержит часть checkpoints_directory/prefix
путь к файлу. Затем мы используем строки 34 и 35 , чтобы:

1. Получите остальную часть пути к файлу, используя указанный номер эпохи.
2. Загрузите сериализованные параметры с диска.

Теперь, когда наши предварительно обученные веса загружены, нам нужно закончить инициализацию модели:

```

37 # компилируем модель
38 model =
39         mx.model.FeedForward( ctx=[mx.gpu(0)],

```

```
40     символ=модель.символ,
41     arg_params=model.arg_params,
42     aux_params=model.aux_params)
```

Строка 38 определяет модель как нейронную сеть FeedForward . Во время оценки мы будем использовать только один графический процессор (хотя вы, безусловно, можете использовать более одного графического процессора или даже свой ЦП). Затем параметры аргумента (arg_params) и вспомогательные параметры (aux_params) устанавливаются путем доступа к их соответствующим значениям из модели, загруженной с диска.

Делать прогнозы на тестовом наборе тривиально просто:

```
44 # делать прогнозы на основе тестовых данных 45
print("[INFO] прогнозирование на тестовых данных...") 46
metrics = [mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5)] 47 (rank1 , rank5) =
model.score(testIter, eval_metric=метрика)
48
49 # отображение точности rank-1 и rank-5 50 print("[INFO]
rank-1: {:.2f}%".format(rank1 * 100)) 51 print("[INFO] rank-5: { :.2f}
%".format(rank5 * 100))
```

Строка 46 определяет список интересующих нас метрик — ранг-1 и ранг-5 соответственно. Затем мы вызываем метод .score модели для вычисления точности ранга 1 и ранга 5. Метод .score требует, чтобы мы передали объект ImageRecordIter для тестового набора, за которым следует список метрик , которые мы хотим вычислить. При вызове .score mxnet перебирает все пакеты изображений в тестовом наборе и сравнивает эти оценки. Наконец, строки 50 и 51 отображают точность нашего терминала.

На данный момент у нас есть все ингредиенты, необходимые для обучения AlexNet на наборе данных ImageNet. У нас есть:

1. Скрипт для обучения сети.
2. Скрипт для построения графика потерь и точности при обучении с течением времени.
3. Скрипт для оценки сети.

Последний шаг — начать эксперименты и применить научный метод для получения AlexNet. веса модели, которые воспроизводят характеристики Крижевского и соавт.

6.4 Эксперименты AlexNet

При написании глав этой книги, особенно связанных с обучением современным сетевым архитектурам в ImageNet, я хотел предоставить больше, чем просто код и примеры результатов. Вместо этого я хотел показать настоящую «историю» о том, как специалист по глубокому обучению проводит различные эксперименты, чтобы получить желаемый результат. Таким образом, почти каждая глава в ImageNet Bundle содержит раздел, подобный этому, где я создаю гибрид лабораторного журнала и тематического исследования. В оставшейся части этого раздела я опишу эксперименты, которые я провел, детализирую результаты, а затем опишу изменения, которые я сделал для повышения точности AlexNet.

 При оценке и сравнении производительности AlexNet мы обычно используем реализацию BVLC AlexNet , предоставленную Caffe [16], а не исходную реализацию AlexNet. Это сравнение вызвано рядом причин, в том числе различными аугментациями данных, используемыми Крижевским и др. и использование (теперь устаревших) уровней нормализации локального ответа (LRN). Кроме того, версия AlexNet «CaffeNet» , как правило, более доступна для научного сообщества. В оставшейся части этого раздела я сравню свои результаты с тестом CaffeNet, но все же вернусь к оригинальному тесту Крижевского и др. бумага.

Скорость обучения эпохи	
1	50 1e -2
51	65 1e -3
66	80 1e -4
81	90 1e -5

Таблица 6.2: График скорости обучения, использованный при обучении AlexNet в ImageNet для эксперимента №1.

6.4.1 AlexNet: эксперимент №1

В своем первом эксперименте AlexNet на ImageNet я решил эмпирически продемонстрировать, почему мы размещаем слои пакетной нормализации после активации, а не до активации. Я тоже использую стандартный ReLU, а не ELU, чтобы получить базовый уровень производительности модели (Крижевский и др. использовали ReLU в своих опытах). Таким образом, я изменил файл mxalexnet.py , подробно описанный ранее в этой главе, на отражают изменения пакетной нормализации и активации, образец которых можно увидеть ниже:

```

10      # Блок №1: первый набор слоев CONV => RELU => POOL
11      conv1_1 = mx.sym.Convolution(data=donnees, kernel=(11, 11),
12          stride=(4, 4), num_filter=96)
13      bn1_1 = mx.sym.BatchNorm(data=conv1_1)
14      act1_1 = mx.sym.Activation(data=bn1_1, act_type="relu")
15      pool1 = mx.sym.Pooling(data=act1_1, pool_type="max",
16          kernel=(3, 3), stride=(2, 2))
17      do1 = mx.sym.Dropout(data=pool1, p = 0,25)

```

Обратите внимание, что мой пакетный слой нормализации теперь перед активацией, и я использую ReLU. функции активации. Я включил Таблицу 6.2, чтобы отразить номер моей эпохи и связанное с ней обучение. ставки ниже — мы рассмотрим, почему я предпочитаю снижать скорость обучения в каждую соответствующую эпоху в оставшаяся часть этого раздела.

Я начал тренировать AlexNet с использованием SGD с начальной скоростью обучения 1e - 2, импульсным термином. 0,9 и снижение веса L2 0,0005. Команда для запуска тренировочного процесса выглядела так:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet
```

Я разрешил своей сети обучаться, отслеживая прогресс примерно каждые 10 эпох. Один из Худшие ошибки, которые я вижу у новых практиков глубокого обучения, — это проверка своих тренировочных графиков. довольно часто. В большинстве случаев вам нужен контекст 10-15 эпох, прежде чем вы сможете принять решение что сеть действительно переоснащается, недоучивается и т. д. После эпохи 70 я нарисовал свою тренировочную потерю и точность (рис. 6.1, вверху слева). На данный момент точность валидации и обучения была существенно снижена. застопорился на уровне 49-50% , что является явным признаком того, что скорость обучения может быть снижена для дальнейшего улучшения ТОЧНОСТЬ.

Таким образом, я обновил свою скорость обучения до 1e - 3 , отредактировав строки 53 и 54 в train_alexnet.py:

```

53 # инициализировать оптимизатор
54 opt = mx.optimizer.SGD(learning_rate=1e-3, импульс=0,9, wd=0,0005,
55             rescale_grad=1.0 / размер партии)

```

Обратите внимание, как скорость обучения уменьшилась с 1e-2 до 1e - 3, но все остальные SGD параметры остались прежними. Затем я возобновил обучение с эпохи 50, используя следующие команды:

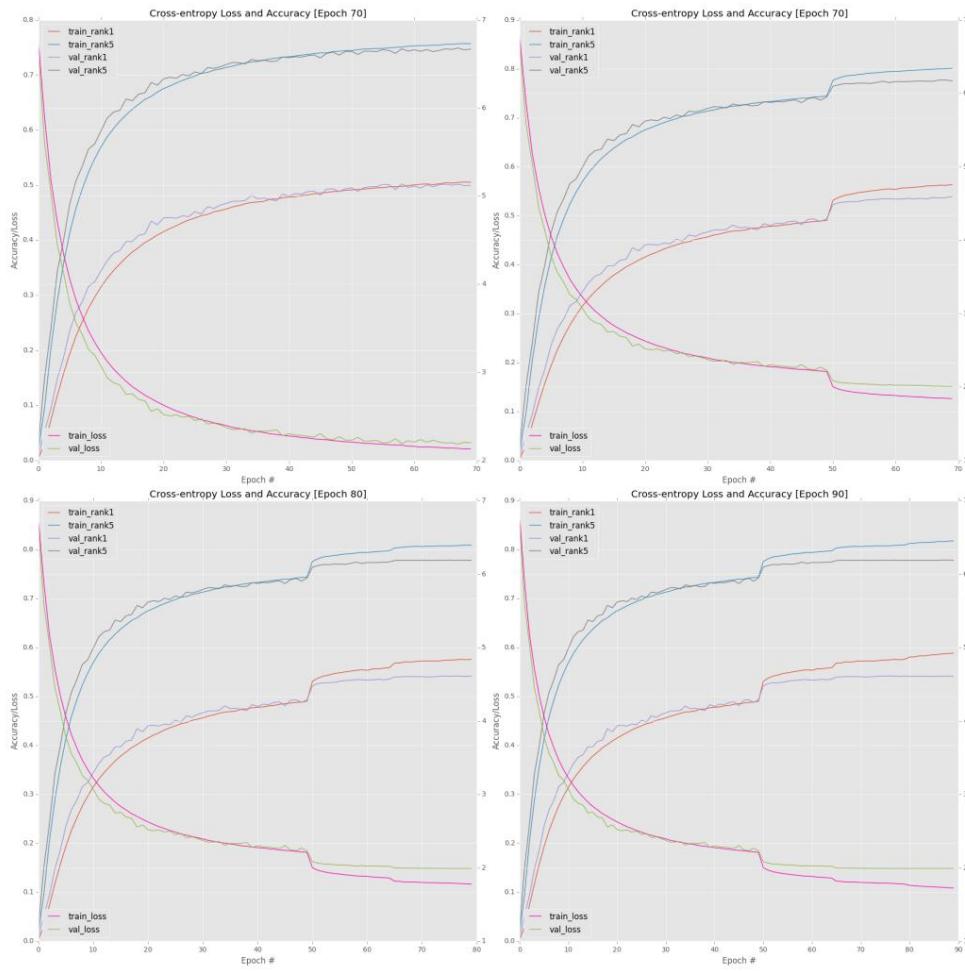


Рисунок 6.1: Вверху слева: обучение AlexNet до эпохи 70 со скоростью обучения $1e^{-2}$. Обратите внимание, как точность ранга 1 колеблется около 49%. Я прекратил обучение после 70-й эпохи и решил возобновить обучение в 50-й эпохе. Вверху справа: перезапуск обучения с 50-й эпохи со скоростью обучения $1e^{-3}$. Уменьшение α на порядок позволяет сети «перескакивать» к более высокой точности/меньшим потерям. Внизу слева: перезапуск обучения с эпохи 65 с $\alpha = 1e^{-4}$. Внизу справа: эпохи 80-90 при $\alpha = 1e^{-5}$.

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet \ --start-epoch 50
```

Опять же, я продолжал отслеживать прогресс AlexNet до эпохи 70 (рис. 6.1, вверху справа). Первый ключевой вывод, который вы должны изучить из этого графика, заключается в том, как снижение моей скорости обучения с $1e-2$ до $1e-3$ вызвало резкое повышение точности и резкое падение потерь сразу после эпохи 50 - это повышение точности и снижение потерь нормально, когда вы обучаете глубокие нейронные сети на больших наборах данных. Снижая скорость обучения, мы позволяем нашей сети опускаться в более низкие области потерь, поскольку ранее скорость обучения была слишком велика, чтобы оптимизатор мог найти эти области. Имейте в виду, что цель обучения сети глубокого обучения не обязательно состоит в том, чтобы найти глобальный минимум или даже локальный минимум; а просто найти область, где потери достаточно низки.

Однако ближе к более поздним эпохам я начал замечать стагнацию в потерях/точности при проверке (хотя точность/потери при обучении продолжали улучшаться). Этот застой, как правило, является явным признаком

что начинает происходить переобучение, но разрыв между проверкой и потерей обучения более чем приемлем, поэтому я не слишком беспокоился. Я обновил свою скорость обучения до $1e-4$ (опять же, отредактировав строки 53 и 54 в train_alexnet.py) и перезапустил обучение с эпохи 65:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet \ --start-epoch 65
```

Потери/точность валидации немного улучшились, но на данный момент скорость обучения начинает становиться слишком маленькой — более того, мы начинаем подгонять к обучающим данным (рис. 6.1, внизу слева).

Наконец, я позволил своей сети обучаться еще 10 эпох (80–90), используя скорость обучения $1e-5$:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet \ --start-epoch 80
```

Рисунок 6.1 (внизу справа) содержит результирующий график для последних десяти эпох. Дальнейшее обучение после эпохи 90 не требуется, поскольку потери/точность при проверке перестали улучшаться, в то время как потери при обучении продолжают снижаться, подвергая нас риску переобучения. В конце эпохи 90 я получил 54,14 % точности ранга 1 и 77,90% точности ранга 5 на данных проверки. Эта точность очень разумна для первого эксперимента, но не совсем то, что я ожидал бы от производительности на уровне AlexNet, которая, по данным эталонной модели BVLC CaffeNet, составляет примерно 57% точности ранга 1 и 80% точности ранга 5.

 Я намеренно пока не оцениваю свои эксперименты на тестовых данных. Я знаю, что нужно провести больше экспериментов, и я стараюсь оценивать на тестовом наборе только тогда, когда уверен, что получил высокопроизводительную модель. Помните, что ваш тестовый набор следует использовать очень экономно — вы не хотите, чтобы он соответствовал вашему тестовому набору; в противном случае вы полностью лишите свою модель возможности обобщать данные за пределами выборок в вашем наборе данных.

6.4.2 AlexNet: эксперимент № 2 Цель

этого эксперимента — развить предыдущий и продемонстрировать, почему мы размещаем слои пакетной нормализации после активации. Я сохранил активацию ReLU, но изменил порядок нормализации пакетов, как показано в следующем блоке кода:

```
10      # Блок №1: первый набор слоев CONV => RELU => POOL
11      conv1_1 = mx.sym.Convolution(data=data, kernel=(11, 11),
12          шаг=(4, 4), num_filter=96)
13      act1_1 = mx.sym.Activation(data=conv1_1, act_type="relu") bn1_1 =
14          mx.sym.BatchNorm(data=act1_1) pool1 = mx.sym.Pooling(data=bn1_1,
15          pool_type="max", kernel=(3, 3), шаг = (2, 2)) do1 = mx.sym.Dropout
16          (данные = pool1, p = 0,25)
17
```

Опять же, я использовал те же самые параметры оптимизатора SGD с начальной скоростью обучения $1e-2$, импульсом 0,9 и уменьшением веса L2 0,0005. Таблица 6.3 включает мою эпоху и соответствующий график скорости обучения. Я начал обучать AlexNet с помощью следующей команды:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet
```

Скорость обучения эпохи 1-65	
1e-2	66-85
1e-3	86-100
1e-4	

Таблица 6.3: График скорости обучения, используемый при обучении AlexNet в ImageNet для эксперимента № 2 и эксперимента № 3.

Примерно в эпоху 65 я заметил, что потери при проверке и точность не изменились (рис. 6.2, вверху слева). Поэтому я прекратил тренировку, скорректировал скорость обучения до $1e^{-3}$, а затем перезапустил тренировку с 65-й эпохи:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet \ --start-epoch 65
```

Опять же, мы можем увидеть характерный скачок точности за счет снижения скорости обучения, когда точность/потери проверки выходят на плато (рис. 6.2, вверху справа). В эпоху 85 я снова снизил скорость обучения, на этот раз с $1e^{-3}$ до $1e^{-4}$, и позволил сети обучаться еще 15 эпох, после чего потери/точность проверки перестали улучшаться (рис. 6.2, внизу).

Изучая журналы своего эксперимента, я заметил, что моя точность ранга 1 составила 56,72%, а точность ранга 5 — 79,62%, что намного лучше, чем в моем предыдущем эксперименте по размещению слоя пакетной нормализации перед активацией. Кроме того, эти результаты находятся в пределах статистического диапазона того, как выглядит истинная производительность на уровне AlexNet.

6.4.3 AlexNet: эксперимент №3 Учитывая,

что мой предыдущий эксперимент продемонстрировал нормализацию размещения пакетов после того, как активация дала лучшие результаты, я решил заменить стандартные активации ReLU активациями ELU. По моему опыту, замена ReLU на ELU часто может повысить точность вашей классификации на 1-2% в наборе данных ImageNet. Поэтому мой блок CONV => RELU теперь выглядит так:

```
10      # Блок №1: первый набор слоев CONV => RELU => POOL
11      conv1_1 = mx.sym.Convolution(data=data, kernel=(11, 11),
12          шаг=(4, 4), num_filter=96)
13      act1_1 = mx.sym.LeakyReLU(data=conv1_1, act_type="elu") bn1_1 =
14          mx.sym.BatchNorm(data=act1_1) pool1 = mx.sym.Pooling(data=bn1_1,
15          pool_type="max", kernel=(3, 3), шаг=(2, 2)) do1 = mx.sym.Dropout
16          (данные = pool1, p = 0,25)
```

Обратите внимание, как слой нормализации партии размещается после активации вместе с ELU, заменяющими ReLU. Во время этого эксперимента я использовал те же параметры оптимизатора SGD, что и в двух предыдущих испытаниях. Я также следовал тому же графику скорости обучения, что и во втором эксперименте (таблица 6.3).

Чтобы воспроизвести мой эксперимент, вы можете использовать следующие команды:

```
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet
...
$ python train_alexnet.py --checkpoints контрольные точки --prefix alexnet \
```

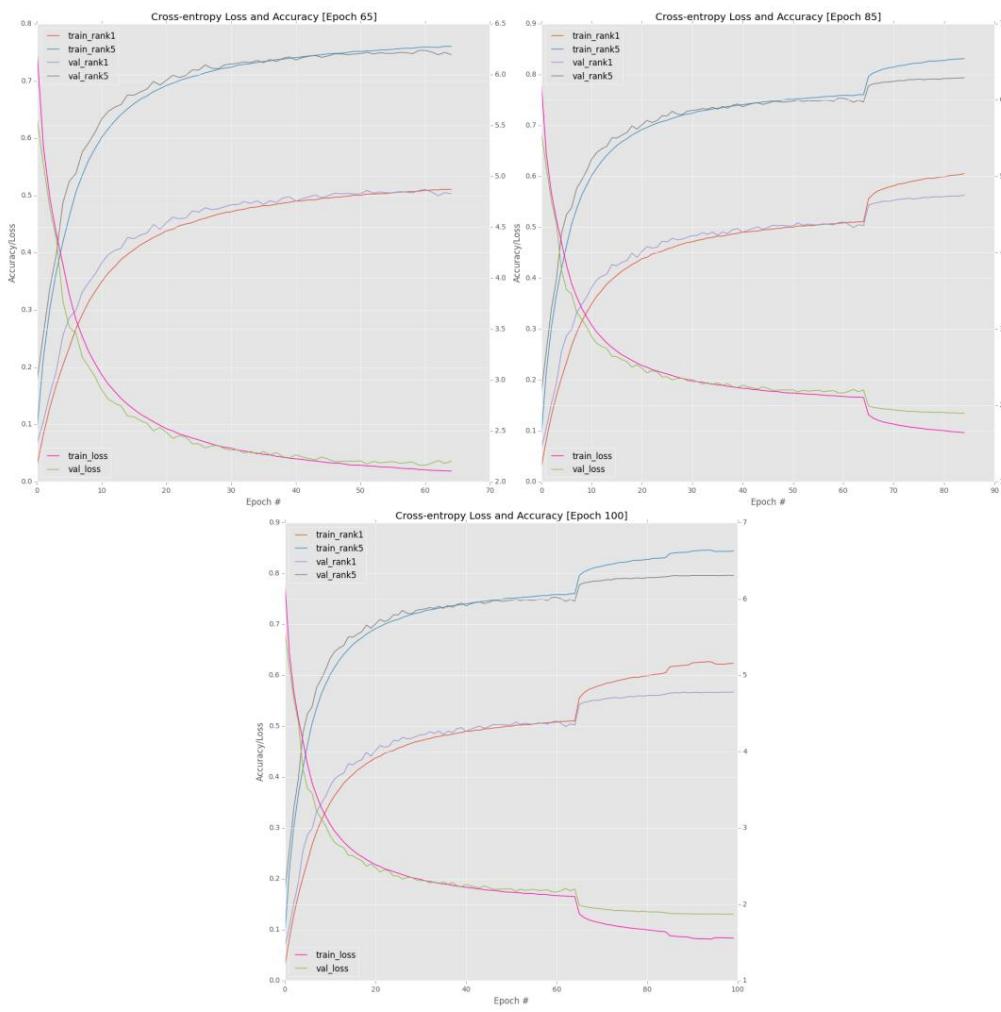


Рисунок 6.2: Вверху справа: первые 65 эпох со скоростью обучения $1e^{-2}$ при размещении активации перед нормализацией партии. Вверху слева: уменьшение α до $1e^{-3}$ вызывает резкое увеличение точности и уменьшение потерь; однако потери при обучении уменьшаются значительно быстрее, чем потери при проверке. Внизу: окончательное уменьшение α до $1e^{-4}$ для эпох 85–100.

--начало эпохи 65

```
***  
$ python train_alexnet.py --checkpoints контрольные точки \ --  
prefix alexnet --start-epoch 85
```

Первая команда начинает обучение с первой эпохи с начальной скоростью обучения $1e^{-2}$. Вторая команда перезапускает обучение в эпоху 65, используя скорость обучения $1e^{-3}$. И последняя команда перезапускает обучение на 85-й эпохе со скоростью обучения $1e^{-4}$.

Полный график потерь/точности обучения и проверки можно увидеть на рисунке 6.3. Опять же, вы можете видеть четкие характерные следы корректировки скорости обучения на порядок на эпохах 65 и 85, при этом скачки становятся менее выраженным по мере снижения скорости обучения. Я не хотел тренироваться после эпохи 100, так как AlexNet явно начинает подгонять данные обучения, в то время как точность/потери проверки остаются неизменными. Чем больше этому разрыву позволяет расти, тем хуже становится переоснащение, поэтому мы применяем критерии регуляризации «ранней остановки», чтобы предотвратить

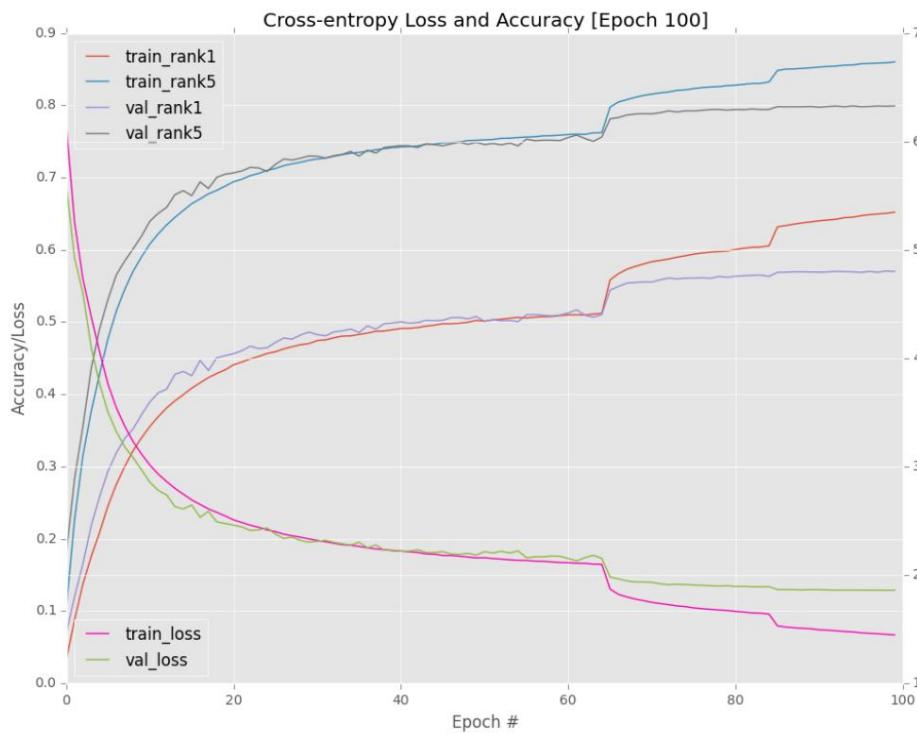


Рисунок 6.3: В нашем последнем эксперименте AlexNet + ImageNet мы заменяем ReLU на ELU и получаем точность проверки ранга 1/ранг 5 57,00%/75,52% и точность тестирования ранга 1/ранг 5 59,80%/81,75%.

далее переоснащение.

Изучив точность для сотой эпохи, я обнаружил, что получил точность ранга 1 57,00% и точность ранга 5 79,52% в наборе данных проверки. Этот результат лишь незначительно лучше, чем мой второй эксперимент, но что очень интересно, так это то, что происходит, когда я оцениваю тестовый набор с помощью сценария `test_alexnet.py`:

```
$ python test_alexnet.py --checkpoints контрольные точки --prefix alexnet \
--epoch 100
[INFO] загрузка модели...
[INFO] прогнозирование на основе тестовых данных...
[ИНФО] ранг-1: 59,80%
[ИНФО] ранг-5: 81,74%
```

Я суммировал результаты в Таблице 6.4. Здесь вы можете видеть, что я получил 59,80% точности ранга 1 и 81,75% ранга 5 на тестовом наборе, что, безусловно, выше того, что большинство независимых статей и публикаций сообщают о точности на уровне AlexNet. Для вашего удобства я включил веса для этого эксперимента AlexNet в загружаемый вами пакет ImageNet.

В целом, цель этого раздела — дать вам представление о типах экспериментов, которые вам необходимо провести, чтобы получить достаточно производительную модель в наборе данных ImageNet. На самом деле, мой лабораторный журнал включал 25 отдельных экспериментов для AlexNet + ImageNet, слишком много, чтобы включить их в эту книгу. Вместо этого я выбрал те, которые наиболее характерны для важных изменений, которые я внес в сетевую архитектуру и оптимизатор. Имейте в виду, что для большинства задач глубокого обучения вы будете использовать

	Набор для
тестирования Точность	1-го ранга
59,80 % Точность 5- го ранга	81,75 %

Таблица 6.4: Оценка AlexNet на тестовом наборе ImageNet. Наши результаты превосходят стандартную эталонную модель Caffe, используемую для оценки AlexNet.

10-100 (а в некоторых случаях и больше) экспериментов, прежде чем вы получите модель, которая хорошо работает как на ваших данных проверки, так и на данных тестирования.

Глубокое обучение не похоже на другие области программирования, где вы пишете функцию один раз, и она работает вечно. Вместо этого есть много ручек и рычагов, которые нужно настроить. Как только вы настроите параметры, вы будете вознаграждены хорошо работающей CNN, но до тех пор наберитесь терпения и запишите свои результаты! Замечание о том, что работает, а что нет, бесценно — эти заметки позволят вам обдумать свои эксперименты и определить новые направления для дальнейшего развития.

6.5 Резюме

В этой главе мы реализовали архитектуру AlexNet с помощью библиотеки mxnet, а затем обучили ее набору данных ImageNet. Эта глава была довольно длинной из-за того, что нам нужно было всесторонне рассмотреть архитектуру AlexNet, сценарий обучения и сценарий оценки. Теперь, когда мы определили наши сценарии Python для обучения и оценки, мы сможем повторно использовать их в будущих экспериментах, что значительно упростит обучение и оценку — основной задачей для нас будет реализация реальной сетевой архитектуры.

Проведенные нами эксперименты позволили нам сделать два важных вывода: 1.

Нормализация партии после активации (а не до) приведет к более высокому точности классификации/меньшие потери в большинстве ситуаций.

2. Замена ReLU на ELU может немного повысить точность классификации.

В целом, мы смогли получить 59,80% точности ранга 1 и 81,75% точности ранга 5 на ImageNet, превосходя стандартную эталонную модель Caffe, используемую для оценки AlexNet.

7. Обучение VGGNet на ImageNet

В этой главе мы узнаем, как с нуля обучить сетевую архитектуру VGG16 набору данных ImageNet. Семейство сверточных нейронных сетей VGG было впервые представлено Симоняном и Зиссерманом в их статье 2014 года «Очень глубокие сверточные сети для крупномасштабного распознавания изображений» [17].

Эта сеть отличается своей простотой, поскольку в ней используются только сверточные слои 3×3 , уложенные друг на друга с возрастающей глубиной. Уменьшение пространственных размеров томов достигается за счет использования max pooling. За двумя полно связанными слоями, каждый из которых состоит из 4096 узлов (и отсева между ними), следует классификатор softmax.

Сегодня VGG часто используется для передачи обучения, поскольку сеть демонстрирует способность обобщать наборы данных, на которых она не обучалась, выше среднего (по сравнению с другими типами сетей, такими как GoogLeNet и ResNet). В большинстве случаев, если вы читаете публикацию или лабораторный журнал, в котором применяется трансферное обучение, он, вероятно, использует VGG в качестве базовой модели.

К сожалению, обучение VGG с нуля — это, по меньшей мере, боль. Сеть очень медленно обучается, а вес самой сетевой архитектуры довольно велик (более 500 МБ). Это единственная сеть в пакете ImageNet, которую я бы рекомендовал вам не тренировать, если у вас нет доступа как минимум к четырем графическим процессорам. Из-за глубины сети и полностью связанных слоев фаза обратного распространения мучительно медленная.

В моем случае обучение VGG на восьми графических процессорах заняло 10 дней — с любым менее чем четырьмя графическими процессорами обучение VGG с нуля, вероятно, займет чрезмерно много времени (если только вы не проявите терпение). Тем не менее, специалисту по глубокому обучению важно понимать историю глубокого обучения, особенно концепцию предварительного обучения и то, как мы позже научились избегать этой дорогостоящей операции, оптимизируя наши весовые функции инициализации.

Опять же, эта глава включена в пакет ImageNet, поскольку семейство сетей VGG является критическим аспектом глубокого обучения; однако, пожалуйста, не считайте необходимость обучать эту сеть с нуля — я включил файл весов, полученный в результате моего эксперимента, в эту главу, чтобы вы могли использовать его в своих собственных приложениях. Используйте эту главу в качестве образовательного справочника, чтобы вы могли учиться у VGG, применяя его в своих проектах. В частности, в этой главе будет освещено правильное использование функции активации PReLU и инициализации MSRA.

7.1 Внедрение VGGNet

При внедрении VGG Симонян и Зиссерман пробовали варианты VGG с увеличенной глубиной.

Таблица 1 их публикации включена в рисунок 7.1 ниже, чтобы подчеркнуть их эксперименты.

В частности, нас больше всего интересуют конфигурации A, B, D и E. Ранее в этой книге вы уже использовали обе конфигурации D и E — это архитектуры VGG16 и VGG19.

Глядя на эти архитектуры, вы заметите две закономерности: во-

первых, сеть использует только фильтры 3×3 . Во-вторых, по мере увеличения глубины сети количество изученных фильтров также увеличивается, а точнее, количество фильтров удваивается каждый раз, когда применяется максимальное объединение для уменьшения размера тома. Идея удвоения количества фильтров каждый раз, когда вы уменьшаете пространственные измерения, имеет историческое значение в литературе по глубокому обучению, и вы даже увидите эту закономерность сегодня.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
FC-4096					
FC-4096					
FC-1000					
soft-max					

Рисунок 7.1: Повторение таблицы 1 Симоняна и Зиссермана [17]. В этой главе мы реализуем конфигурацию D, обычно называемую VGG16.

Причина, по которой мы выполняем это удвоение фильтров, состоит в том, чтобы гарантировать, что ни один блок одного слоя не будет более смещенным, чем другие. Более ранние уровни в сетевой архитектуре имеют меньше фильтров, но их пространственные объемы также намного больше, что означает наличие «большего количества (пространственных) данных» для изучения.

Однако мы знаем, что применение операции максимального объединения уменьшит объемы входных пространственных данных.

Если мы уменьшим пространственные объемы без увеличения количества фильтров, то наши слои станут несбалансированными и потенциально смещенными, а это означает, что более ранние слои в сети могут влиять на нашу выходную классификацию больше, чем более глубокие слои в сети. Для борьбы с этим дисбалансом мы учитываем соотношение размера тома и количества фильтров. Если мы уменьшим размер входного объема на 50-75%, то удвоим количество фильтров в следующем наборе слоев CONV, чтобы сохранить баланс.

Проблема с обучением таких глубоких архитектур заключается в том, что Симонян и Зиссерман сочли обучение VGG16 и VGG19 чрезвычайно сложным из-за их глубины. Если бы эти архитектуры были

Случайно инициализированные и обученные с нуля, они часто изо всех сил пытались научиться и получить какую-либо начальную «тягу» — сети были просто слишком глубоки для базовой случайной инициализации. Поэтому для тренировки более глубоких вариантов ВГГ Симонян и Зиссерман придумали хитрую концепцию под названием предварительная тренировка.

Предварительное обучение — это практика сначала обучения меньших версий вашей сетевой архитектуры с меньшим количеством весовых уровней, а затем использования этих конвергентных сетевых весов в качестве инициализации для более крупных и глубоких сетей. В случае с VGG авторы сначала обучили конфигурацию A, VGG11. VGG11 удалось достичь уровня достаточно низких потерь, но не достойная современной точности.

Затем веса из VGG11 использовались в качестве инициализаций для конфигурации B, VGG13. Слои conv3-64 и conv3-128 (выделены жирным шрифтом на рис. 7.1) в VGG13 были инициализированы случайнym образом, а остальные слои были просто скопированы из предварительно обученной сети VGG11. Используя инициализацию, Симонян и Зиссерман смогли успешно обучить VGG13, но все еще не достигли современной точности.

Этот предтренировочный шаблон продолжился конфигурацией D, которую мы обычно знаем как VGG16. На этот раз три новых слоя были инициализированы случайнym образом, а остальные слои были скопированы из VGG13. Затем сеть обучалась с использованием этих «подогретых предварительно обученных» слоев, что позволяло случайнym инициализированным слоям сходиться и изучать различительные шаблоны. В конечном итоге VGG16 смог очень хорошо справиться с задачей классификации ImageNet.

В качестве заключительного эксперимента Симонян и Зиссерман еще раз применили предобучение к конфигурации E, VGG19. Эта очень глубокая архитектура скопировала веса из предварительно обученной архитектуры VGG16, а затем добавила еще три дополнительных сверточных слоя. После обучения было обнаружено, что VGG19 добился наивысшей точности классификации в своих экспериментах; однако размер модели (574 МБ) и количество времени, которое потребовалось для обучения и оценки сети, сделали ее менее привлекательной для практиков глубокого обучения.

Если предварительная тренировка кажется болезненным и утомительным процессом, то это потому, что так оно и есть. Обучение небольших вариаций вашей сетевой архитектуры, а затем использование конвергентных весов в качестве инициализации ваших более глубоких версий сети — это умный трюк; однако для этого требуется обучение и настройка гиперпараметров для N отдельных сетей, где N — ваша окончательная сетевая архитектура вместе с количеством предыдущих (меньших) сетей, необходимых для получения конечной модели. Выполнение этого процесса занимает очень много времени, особенно для более глубоких сетей с множеством полносвязных уровней, таких как VGG.

Хорошая новость заключается в том, что мы больше не проводим предварительное обучение при обучении очень глубоких сверточных нейронных сетей — вместо этого мы полагаемся на хорошую функцию инициализации. Вместо чисто случайных инициализаций веса мы теперь используем Xavier/Glorot [18] или MSRA (также известную как He et al. инициализация). Благодаря работе Мишкина и Мтаса в их статье 2015 года All you need is good init [19] и He et al. в работе «Углубление в выпрямители: превосходство производительности на уровне человека в классификации ImageNet» [20] мы обнаружили, что можем полностью пропустить этап предварительной подготовки и сразу перейти к более глубоким вариациям сетевых архитектур.

После того, как эти статьи были опубликованы, Симонян и Зиссерман пересмотрели свои эксперименты и обнаружили, что эти «более умные» схемы инициализации и функции активации смогли воспроизвести их предыдущую работу без использования утомительного предварительного обучения.

Поэтому всякий раз, когда вы тренируете VGG16 или VGG19, убедитесь,

что вы:

1. Замените все ReLU на PReLU.
2. Используйте MSRA (также называемую «He et al. Initialization») для инициализации весовых слоев в вашем сеть.

Дополнительно рекомендуется использовать пакетную нормализацию после активации функций в сети. Применение пакетной нормализации не обсуждалось в оригинальной статье Симоняна и Зиссермана, но, как обсуждалось в других главах, пакетная нормализация может стабилизировать ваше обучение и уменьшить общее количество эпох, необходимых для получения модели с достаточной производительностью.

Теперь, когда мы узнали об архитектуре VGG, давайте приступим к реализации конфигурации. Д от Симоняна и Зиссермана, основополагающая архитектура VGG16:

```
--- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- нн
| | |--- __init__.py
| | |--- конв.
| | |--- mxconv
| | | |--- __init__.py
| | |--- mxalexnet.py
| | |--- mxvggnet.py
| |--- предварительная обработка
| |--- утилиты
```

Из приведенной выше структуры проекта вы заметите, что я создал файл с именем `mxvggnet.py` в подмодуль `mxconv` `pyimagesearch` — в этом файле находится наша реализация VGG16 собирается жить. Откройте `mxvggnet.py`, и мы начнем реализовывать сеть:

```
1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3
4 класс MxVGGNet:
5     @статический метод
6     деф- сборка (классы):
7         # ввод данных
8         данные = mx.sym.Variable("данные")
```

Строка 2 импортирует нашу библиотеку `mxnet` , а строка 4 определяет класс `MxVGGNet` . Как и все остальные CNN, которые мы реализовали в этой книге, мы создадим метод сборки в строке 6 , который отвечает за для построения фактической сетевой архитектуры. Этот метод принимает единственный параметр, количество классов, которые наша сеть должна различать. Затем строка 8 инициализирует все важные переменные данных, фактические входные данные для CNN.

Глядя на рисунок 7.1 выше, мы видим, что наш первый блок слоев должен включать (`CONV => RELU`) * 2 => БАССЕЙН. Давайте продолжим и определим этот блок слоя сейчас:

```
10     # Блок №1: (CONV => RELU) * 2 => POOL
11     conv1_1 = mx.sym.Convolution (данные = данные, ядро = (3, 3),
12         pad=(1, 1), num_filter=64, name="conv1_1")
13     act1_1 = mx.sym.LeakyReLU(data=conv1_1, act_type="prelu",
14         имя="act1_1")
15     bn1_1 = mx.sym.BatchNorm(data=act1_1, name="bn1_1")
16     conv1_2 = mx.sym.Convolution (данные = bn1_1, ядро = (3, 3),
17         pad=(1, 1), num_filter=64, name="conv1_2")
18     act1_2 = mx.sym.LeakyReLU(data=conv1_2, act_type="prelu",
19         имя="act1_2")
20     bn1_2 = mx.sym.BatchNorm(data=act1_2, name="bn1_2")
21     pool1 = mx.sym.Pooling(data=bn1_2, pool_type="max",
22         ядро = (2, 2), шаг = (2, 2), имя = «пул1» )
23     do1 = mx.sym.Dropout (данные = pool1, p = 0,25)
```

Слои CONV в этом блоке изучают 64 фильтра, каждый размером 3×3 . Вариант ReLU с утечкой, PReLU (описано в главе 10 Starter Bundle) используется в качестве нашей функции активации. После каждого PReLU затем мы применяем слой пакетной нормализации (не включеный в исходную статью, но полезный, когда стабилизирующее обучение ВГГ). В конце блока слоев используется объединяющий слой для уменьшения пространственные размеры объема с использованием ядра 2×2 и шага 2×2 , уменьшающие размер объема. Мы также применяем отсев с небольшим процентом (25%), чтобы помочь бороться с переоснащением.

Второй слой, установленный в VGG16, также применяется (CONV => ReLU) * 2 => POOL, только на этот раз Изучается 128 фильтров (опять же, каждый 3×3). Количество фильтров удвоилось с тех пор, как мы уменьшил объем пространственного ввода с помощью операции максимального объединения:

```

25      # Блок #2: (CONV => ReLU) * 2 => POOL
26      conv2_1 = mx.sym.Convolution(данные = do1, ядро = (3, 3),
27          pad=(1, 1), num_filter=128, name="conv2_1")
28      act2_1 = mx.sym.LeakyReLU(data=conv2_1, act_type="prelu",
29          имя="act2_1")
30      bn2_1 = mx.sym.BatchNorm(data=act2_1, name="bn2_1")
31      conv2_2 = mx.sym.Convolution(данные = bn2_1, ядро = (3, 3),
32          pad=(1, 1), num_filter=128, name="conv2_2")
33      act2_2 = mx.sym.LeakyReLU(data=conv2_2, act_type="prelu",
34          имя="act2_2")
35      bn2_2 = mx.sym.BatchNorm(data=act2_2, name="bn2_2")
36      pool2 = mx.sym.Pooling(data=bn2_2, pool_type="max",
37          ядро = (2, 2), шаг = (2, 2), имя = «пул2» )
38      do2 = mx.sym.Dropout(данные = pool2, p = 0,25)

```

Каждый слой CONV в этом блоке слоев отвечает за изучение 128 фильтров. И снова PReLU функция активации применяется после каждого слоя CONV, за которым следует слой пакетной нормализации. Максимум объединение используется для уменьшения пространственных размеров объема с последующим небольшим количеством отсевов, чтобы уменьшить влияние переобучения.

Блок третьего уровня VGG добавляет дополнительный слой CONV, подразумевая, что всего три CONV. операции до максимального объединения:

```

40      # Блок №3: (CONV => ReLU) * 3 => POOL
41      conv3_1 = mx.sym.Convolution(данные = do2, ядро = (3, 3),
42          pad=(1, 1), num_filter=256, name="conv3_1")
43      act3_1 = mx.sym.LeakyReLU(data=conv3_1, act_type="prelu",
44          имя="act3_1")
45      bn3_1 = mx.sym.BatchNorm(data=act3_1, name="bn3_1")
46      conv3_2 = mx.sym.Convolution(данные = bn3_1, ядро = (3, 3),
47          pad=(1, 1), num_filter=256, name="conv3_2")
48      act3_2 = mx.sym.LeakyReLU(data=conv3_2, act_type="prelu",
49          имя="act3_2")
50      bn3_2 = mx.sym.BatchNorm(data=act3_2, name="bn3_2")
51      conv3_3 = mx.sym.Convolution(данные = bn3_2, ядро = (3, 3),
52          pad=(1, 1), num_filter=256, name="conv3_3")
53      act3_3 = mx.sym.LeakyReLU(data=conv3_3, act_type="prelu",
54          имя="act3_3")
55      bn3_3 = mx.sym.BatchNorm(data=act3_3, name="bn3_3")
56      pool3 = mx.sym.Pooling(data=bn3_3, pool_type="max",
57          ядро = (2, 2), шаг = (2, 2), имя = «пул3» )
58      do3 = mx.sym.Dropout(данные = pool3, p = 0,25)

```

Здесь количество фильтров, изученных каждым слоем CONV, увеличивается до 256, но размер фильтров остается 3×3 . Также используется тот же шаблон PReLU и пакетной нормализации.

Четвертый блок VGG16 также применяет три слоя CONV, уложенных друг на друга, но на этот раз общее количество фильтров снова удваивается до 512:

```

60      # Блок №4: (CONV => RELU) * 3 => POOL
61      conv4_1 = mx.sym.Convolution(данные = do3, ядро = (3, 3),
62          pad=(1, 1), num_filter=512, name="conv4_1")
63      act4_1 = mx.sym.LeakyReLU(data=conv4_1, act_type="prelu",
64          имя="act4_1")
65      bn4_1 = mx.sym.BatchNorm(data=act4_1, name="bn4_1")
66      conv4_2 = mx.sym.Convolution(данные = bn4_1, ядро = (3, 3),
67          pad=(1, 1), num_filter=512, name="conv4_2")
68      act4_2 = mx.sym.LeakyReLU(data=conv4_2, act_type="prelu",
69          имя="act4_2")
70      bn4_2 = mx.sym.BatchNorm(data=act4_2, name="bn4_2")
71      conv4_3 = mx.sym.Convolution(данные = bn4_2, ядро = (3, 3),
72          pad=(1, 1), num_filter=512, name="conv4_3")
73      act4_3 = mx.sym.LeakyReLU(data=conv4_3, act_type="prelu",
74          имя="act4_3")
75      bn4_3 = mx.sym.BatchNorm(data=act4_3, name="bn4_3")
76      pool4 = mx.sym.Pooling(data=bn4_3, pool_type="max",
77          ядро = (2, 2), шаг = (2, 2), имя = «пул3» )
78      do4 = mx.sym.Dropout(данные = pool4, p = 0,25)

```

Интересно, что окончательный набор слоев CONV в VGG не увеличивает количество фильтров и остается на уровне 512. Что касается того, почему это число не удвоилось, я не совсем уверен, но мне кажется, либо:

1. Переход с 512 фильтров на 1024 привел к тому, что в сети появилось слишком много параметров, которые вызвал переоснащение VGG16.
2. Обучение сети с использованием 1024 фильтров в конечных блоках CONV было просто слишком сложным . союзник дорогой.

Несмотря на это, окончательный набор слоев CONV в VGG16 также изучает 512 фильтров, каждый 3×3 :

```

80      # Блок №5: (CONV => RELU) * 3 => POOL
81      conv5_1 = mx.sym.Convolution(данные = do4, ядро = (3, 3),
82          pad=(1, 1), num_filter=512, name="conv5_1")
83      act5_1 = mx.sym.LeakyReLU(data=conv5_1, act_type="prelu",
84          имя="act5_1")
85      bn5_1 = mx.sym.BatchNorm(data=act5_1, name="bn5_1")
86      conv5_2 = mx.sym.Convolution(данные = bn5_1, ядро = (3, 3),
87          pad=(1, 1), num_filter=512, name="conv5_2")
88      act5_2 = mx.sym.LeakyReLU(data=conv5_2, act_type="prelu",
89          имя="act5_2")
90      bn5_2 = mx.sym.BatchNorm(данные = act5_2, имя = "bn5_2")
91      conv5_3 = mx.sym.Convolution(данные = bn5_2, ядро = (3, 3),
92          pad=(1, 1), num_filter=512, name="conv5_3")
93      act5_3 = mx.sym.LeakyReLU(data=conv5_3, act_type="prelu",
94          имя="act5_3")
95      bn5_3 = mx.sym.BatchNorm(data=act5_3, name="bn5_3")
96      pool5 = mx.sym.Pooling(data=bn5_3, pool_type="max",
97          ядро = (2, 2), шаг = (2, 2), имя = «пул5» )
98      do5 = mx.sym.Dropout(данные = pool5, p = 0,25)

```

Наш первый набор слоев FC включает 4096 узлов и также следует шаблону применения PReLU. и нормализация партии:

```

100      # Блок №6: FC => слои RELU
101      слгладить = mx.sym.Flatten (данные = do5, имя = "сгладить")
102      fc1 = mx.sym.FullyConnected(data=flatten, num_hidden=4096,
103          имя="fc1")
104      act6_1 = mx.sym.LeakyReLU(data=fc1, act_type="prelu",
105          имя="act6_1")
106      bn6_1 = mx.sym.BatchNorm (данные = act6_1, имя = "bn6_1")
107      do6 = mx.sym.Dropout (данные = bn6_1, p = 0,5)

```

Dropout применяется с гораздо большей вероятностью 50% в слоях FC , поскольку соединения здесь гораздо более плотные и склонны к переоснащению. Точно так же мы добавляем еще один набор слоев FC в точно так же ниже:

```

109      # Блок №7: FC => слои RELU
110      fc2 = mx.sym.FullyConnected (данные = do6, num_hidden = 4096,
111          имя = "ФК2")
112      act7_1 = mx.sym.LeakyReLU(data=fc2, act_type="prelu",
113          имя="act7_1")
114      bn7_1 = mx.sym.BatchNorm(data=act7_1, name="bn7_1")
115      do7 = mx.sym.Dropout (данные = bn7_1, p = 0,5)

```

Последний блок кода в VGG16 создает уровень FC для общего числа классов, а затем применяет классификатор softmax:

```

117      # классификатор softmax
118      fc3 = mx.sym.FullyConnected (данные = do7, num_hidden = классы,
119          имя="fc3")
120      модель = mx.sym.SoftmaxOutput (данные = fc3, имя = "softmax")
121
122      # вернуть архитектуру сети
123      модель возврата

```

Хотя для реализации VGG16, безусловно, требуется намного больше кода (по сравнению с AlexNet), все довольно просто. Кроме того, используя схему, такую как рис. 7.1 выше существенно облегчает процесс. Когда вы реализуете свою собственную последовательную сеть Для таких архитектур я предлагаю написать первый набор слоев CONV , а затем скопировать и вставить их . для следующего блока, убедившись, что вы:

1. Измените входные данные каждого слоя (т. е. аргумент данных) соответствующим образом.
2. Увеличьте количество фильтров в слоях CONV в соответствии с вашим планом.

Это поможет уменьшить любые ошибки, которые могут быть внедрены в ваш код из-за написания определение каждого слоя вручную.

7.2 Обучение VGGNet

Теперь, когда мы внедрили VGG16, мы можем обучить его набору данных ImageNet. Но сначала давайте определить структуру нашего проекта:

```
-- mx_imagenet_vggnet | |--  
  конфигурация | |-- imagenet_vggnet_config.py  
  | |-- вывод | |-- train_vggnet.py |
```

Структура проекта практически идентична структуре AlexNet из предыдущей главы. Нам нужен скрипт с именем `train_vggnet.py` для фактического обучения сети. Затем сценарий `test_vggnet.py` будет отвечать за оценку VGG16 в ImageNet. Наконец, файл `imagenet_vggnet_config.py` содержит наши конфигурации для эксперимента.

При определении структуры этого проекта я просто скопировал весь каталог `mx_imagenet_alexnet`, а затем переименовал файлы в `vggnet` вместо `alexnet`. Сравнив `imagenet_vggnet_config.py` с `imagenet_alexnet_config.py`, вы заметите, что конфигурации идентичны (поскольку пути к нашим файлам набора данных ImageNet не меняются), за одним исключением:

```
53 # определить размер пакета и количество устройств, используемых для обучения
54 ПАКЕТ_РАЗМЕР = 32
55 ЧИСЛО_УСТРОЙСТВ = 8
```

Здесь я уменьшил `BATCH_SIZE` для 128 до 32. Из-за глубины и размера VGG16 (и, следовательно, объема памяти, который он потребляет на нашем графическом процессоре), мы не сможем передавать столько пакетов изображений по сети за один раз. время. Чтобы разместить VGG16 на моем графическом процессоре (Titan X, 12 ГБ), размер пакета пришлось уменьшить до 32. Если вы решите обучать VGG16 с нуля на своем собственном графическом процессоре, вам, возможно, придется дополнительно уменьшить `BATCH_SIZE`, если вы этого не сделаете. не иметь столько памяти графического процессора.

Во-вторых, я также увеличил количество графических процессоров, которые буду использовать для обучения VGGNet, до восьми. Как я упоминал в начале этой главы, VGG16 — единственная глава в ImageNet Bundle, которую я не рекомендую обучать с нуля, если только у вас нет четырех-восьми графических процессоров. Даже с таким количеством графических процессоров на обучение сети может уйти 10-20 дней.

Теперь, когда мы обновили наш файл конфигурации, давайте также обновим `train_vggnet.py`. Опять же, вспомните из главы 6 об AlexNet, что мы написали код `train_alexnet.py` так, что он требует минимальных изменений при обучении новой сетевой архитектуры в ImageNet. Для полноты картины я рассмотрю весь файл, выделив сделанные нами обновления; однако более исчерпывающий обзор сценария обучения см. в главе 6, где подробно рассматривается весь шаблон.

```
1 # импортируем необходимые пакеты 2 из config
import imagenet_vggnet_config as config 3 из pyimagesearch.nn.mxconv import MxVGGNet
4 import mxnet as mx 5 import argparse 6 import logging 7 import json 8 import os
```

В строке 2 мы импортируем наш файл `imagenet_vggnet_config` из структуры нашего проекта — этот файл конфигурации был изменен по сравнению с импортом `imagenet_alexnet_config` из предыдущей главы. Мы также импортируем `MxVGGNet`, нашу реализацию архитектуры VGG16. Это единственные два изменения, которые необходимо внести в наш раздел импорта.

Далее давайте проанализируем наши аргументы командной строки и создадим наш файл журнала, чтобы mxnet мог регистрировать обучение. перейти к нему:

```

10 # построить разбор аргумента и разобрать аргументы
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-c", "-checkpoints", required=True,
13                  help="путь к выходному каталогу контрольных точек")
14 ap.add_argument("-p", "--prefix", required=True,
15                  help="имя префикса модели")
16 ap.add_argument("-s", "--start-epoch", type=int, default=0,
17                  help="эпоха для возобновления обучения")
18 аргументов = переменные (ap.parse_args())
19

20 # установить уровень логирования и выходной файл
21 logging.basicConfig(level=logging.DEBUG,
22                     имя_файла="training_{}.log".format(args["start_epoch"]),
23                     режим файла = "w")
24

25 # загрузите средства RGB для тренировочного набора, затем определите пакет
26 # размер
27 означает = json.loads(open(config.DATASET_MEAN).read())
28 batchSize = config.BATCH_SIZE * config.NUM_DEVICES

```

Строка 27 загружает средние значения RGB, поэтому мы можем применить нормализацию данных среднего вычитания, в то время как строка 28 вычисляет наш batchSize на основе общего количества устройств, которые мы используем для обучения VGG16. Далее нам нужно построить итератор обучающих данных:

```

30 # построить итератор обучающего изображения
31 trainIter = mx.io.ImageRecordIter(
32     path_imgrec=config.TRAIN_MX_REC,
33     data_shape=(3, 224, 224),
34     batch_size = размер партии,
35     rand_crop = Верно,
36     rand_mirror = Верно,
37     повернуть=15,
38     max_shear_ratio=0,1,
39     mean_r = означает ["R"],
40     mean_g = означает ["G"],
41     mean_b = означает ["B"],
42     preprocess_threads=config.NUM_DEVICES * 2)

```

А также итератор данных проверки:

```

44 # создать итератор проверочного изображения
45 valIter = mx.io.ImageRecordIter(
46     path_imgrec=config.VAL_MX_REC,
47     data_shape=(3, 224, 224),
48     batch_size = размер партии,
49     mean_r = означает ["R"],
50     mean_g = означает ["G"],
51     mean_b = означает ["B"])

```

Теперь мы готовы инициализировать наш оптимизатор:

```
53 # инициализировать оптимизатор
54 opt = mx.optimizer.SGD(learning_rate=1e-2, импульс=0,9, wd=0,0005,
55             rescale_grad=1.0 / размер партии)
```

Следуя статье Симоняна и Зиссермана, мы будем использовать оптимизатор SGD с начальная скорость обучения $1e^{-2}$, момент импульса 0,9 и уменьшение веса L2 0,0005. Особый необходимо тщательно пересмасштабировать градиенты в зависимости от размера партии.

Оттуда мы можем построить путь к нашему каталогу checkpointsPath , где находится модель. веса будут сериализованы после каждой эпохи:

```
57 # построить путь контрольных точек, инициализировать аргумент модели и
58 # вспомогательные параметры
59 checkpointsPath = os.path.sep.join([args["checkpoints"],
60         аргументы["предфикс"]])
61 argParams = Нет
62 дополнительных параметра = нет
```

Затем мы можем определить, обучаем ли мы нашу модель с самой первой эпохи или (2) перезапуск обучения с определенной эпохи:

```
64 # если не указана начальная эпоха конкретной модели, то
65 # инициализируем сеть
66 , если args["start_epoch"] <= 0:
67     # построить архитектуру LeNet
68     print("[INFO] строим сеть...")
69     модель = MxVGGNet.build(config.NUM_CLASSES)
70
71 # в противном случае указана конкретная контрольная точка
72 еще:
73     # загружаем чекпойнт с диска
74     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
75     модель = mx.model.FeedForward.load(путь к контрольным точкам,
76             аргументы["начало_эпохи"])
77
78     # обновить модель и параметры
79     argParams = модель.arg_params
80     auxParams = модель.aux_params
81     модель = модель.символ
```

Строки 66-69 обрабатывают, если мы обучаем VGG16 без предварительной контрольной точки. В этом случае мы создайте экземпляр класса MxVGGNet в строке 69. В противном случае в строках 72-81 предполагается, что мы загружаем конкретная контрольная точка с диска и перезапуск обучения (предположительно после настройки скорости обучения к оптимизатору SGD).

Наконец, мы можем скомпилировать модель:

```
83 # компилируем модель
84 модель = mx.model.FeedForward(
85     ctx=[mx.gpu(i) для i в диапазоне (0, config.NUM_DEVICES)],
```

```

86     СИМВОЛ = модель,
87     инициализатор=mx.initializer.MSRAPrelu(),
88     arg_params = параметры аргумента,
89     aux_params = вспомогательные параметры,
90     оптимизатор=выбор,
91     число_эпох = 80,
92     begin_epoch=args["start_epoch"])

```

Наша модель будет обучаться с помощью ctx NUM_DEVICES графических процессоров — вы должны изменить эту строку . в зависимости от количества графических процессоров, которые вы используете в своей системе. Также обратите особое внимание на Line 87 , где мы определяем инициализатор — мы будем использовать MSRAPrelu в качестве метода инициализации, точно такой же метод, предложенный He et al. [20] для обучения очень глубоких нейронных сетей. Без этого метод инициализации, VGG16 будет трудно сойтись во время обучения.

Далее давайте определим наш набор обратных вызовов и метрик оценки:

```

94 # инициализировать обратные вызовы и метрики оценки
95 batchEndCBs = [mx.callback.Speedometer(batchSize, 250)]
96 epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)]
97 метрик = [mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5),
98           mx.metric.CrossEntropy()]

```

Поскольку VGG16 — очень медленная сеть для обучения, я предпочитаю, чтобы в журнале регистрировалось больше обновлений обучения. подать; поэтому я уменьшаю количество пакетов обновлений спидометра (ранее 500 для AlexNet), чтобы чаще регистрировать обновления обучения в журнале. Затем мы будем отслеживать точность ранга 1, точность ранга 5 и категориальная кросс-энтропия.

Наш последний блок кода обрабатывает обучение VGG16:

```

100 # обучить сеть
101 print("[INFO] обучающая сеть...")
102 модель.подходит(
103     X=поезд,
104     eval_data=значение,
105     eval_metric = показатели,
106     batch_end_callback=batchEndCB,
107     epoch_end_callback=epochEndCBs)

```

Как видно из реализации train_vggnet.py, отличий очень мало .

между приведенным здесь кодом и файлом train_alexnet.py из предыдущей главы — это Именно сходство — вот почему я использую этот же шаблон при обучении своих собственных CNN в ImageNet. Я могу просто скопируйте файл, настройте несколько операторов импорта, создайте экземпляр моей сети, а затем, при желании, настроить параметры оптимизатора и инициализатора. Проделав этот процесс несколько раз, вы может запустить новый проект ImageNet менее чем за десять минут.

7.3 Оценка VGGNet

Для оценки VGGNet мы будем использовать сценарий test_vggnet.py , упомянутый в структуре нашего проекта . над. Обратите внимание, что этот скрипт идентичен test_alexnet.py из предыдущей главы. Там не были ли внесены какие-либо изменения в сценарий, так как сценарии test_*.py в этой главе предназначены для шаблоны, которые можно применять и повторно применять к любой CNN, обученной на ImageNet.

Поскольку код идентичен, я не буду рассматривать здесь `test_vggnet.py`. Пожалуйста, обратитесь к Главе 6 на сайте `test_alexnet.py` для подробного ознакомления с кодом. Кроме того, вы можете использовать часть этой книги, посвященную загрузке кода, для проверки проекта и просмотра содержимого файла `test_vggnet.py`. Опять же, содержимое этих файлов идентично, поскольку они являются частью нашей структуры для обучения и оценки CNN, обученных в ImageNet.

7.4 Эксперименты VGGNet

При обучении VGG16 было важно, чтобы я учитывал эксперименты, проведенные другими исследователями, включая Симоняна и Зиссермана [17], He et al. [21, 22], Мишкин и соавт. [19]. Благодаря этим работам я смог избежать дополнительных дорогостоящих экспериментов и применил следующие рекомендации: 1.

Пропустить предварительное обучение в пользу лучших методов инициализации.

2. Используйте MSRA/He et al. инициализация.

3. Используйте функции активации PReLU.

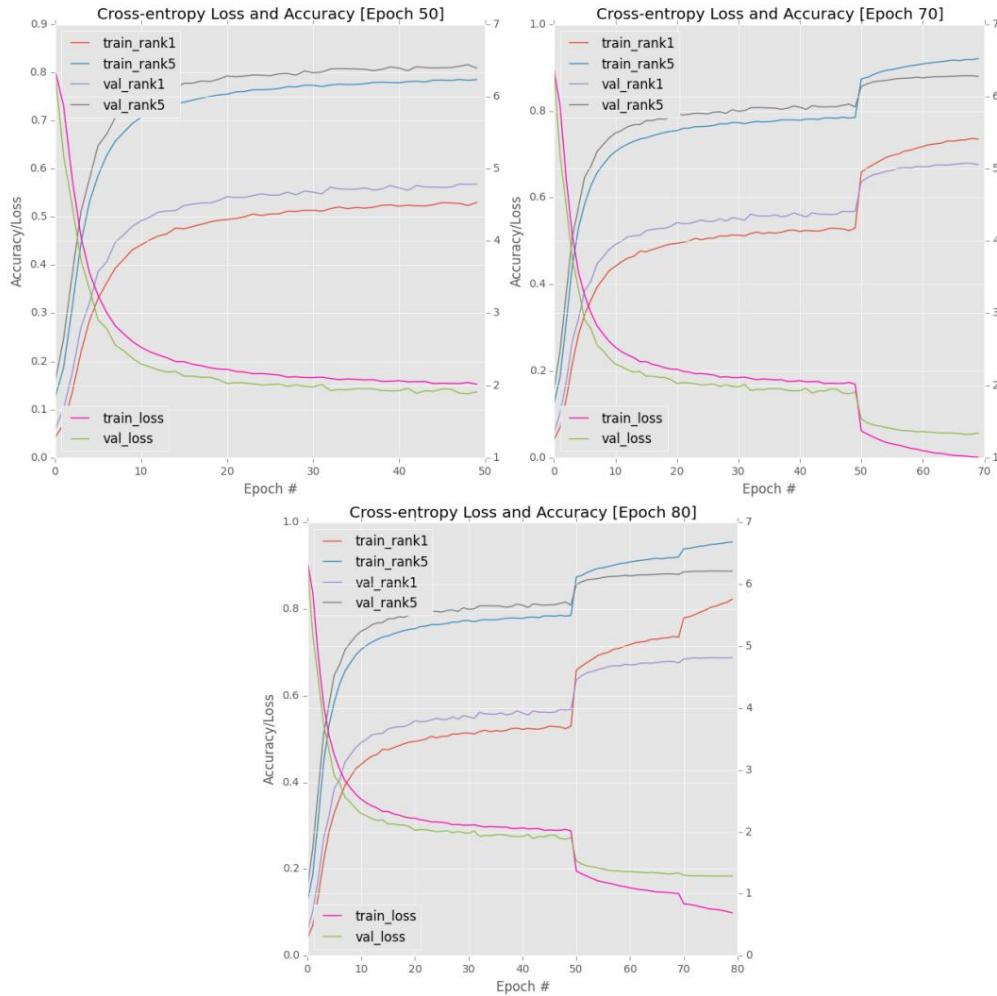


Рисунок 7.2: Верхний левый: обучение VGGNet до эпохи 50 со скоростью обучения $1e^{-2}$. Вверху справа: перезапуск обучения с эпохи 50 со скоростью обучения $1e^{-2}$. Уменьшение α на порядок позволяет сети «перескакивать» к более высокой точности/меньшим потерям. Внизу: перезапуск обучения с эпохи 70 с $\alpha = 1e^{-4}$, всего еще 10 эпох.

Поскольку я следовал этим руководствам, мне нужно было провести только один эксперимент с VGG16, чтобы воспроизвести результаты Симоняна и Зиссермана — этот единственный эксперимент воспроизвел их результаты почти идентично. В этом эксперименте я использовал оптимизатор SGD для обучения VGG16 с начальной скоростью обучения 1e-2, моментом 0,9 и регуляризацией веса L2 0,0005. Для ускорения обучения я использовал инстанс Amazon EC2 с восемью графическими процессорами. Я бы не рекомендовал пытаться обучать VGG на машине с менее чем четырьмя графическими процессорами, если только вы не очень терпеливы.

Я запустил процесс обучения VGG с помощью следующей команды:

```
$ python train_vggnet.py --checkpoints контрольные точки --prefix vggnet
```

Я позволил сети обучаться до эпохи 50, когда точность как обучения, так и проверки , казалось, застопорилась (рис. 7.2, вверху слева). Затем я выбрал ctrl + c из скрипта train_vggnet.py и снизил скорость обучения с 1e-2 до 1e-3:

```
53 # инициализируем оптимизатор
54 opt = mx.optimizer.SGD(learning_rate=1e-3, импульс=0,9, wd=0,0005,
55             rescale_grad=1.0 / размер партии)
```

Затем обучение возобновлялось с помощью следующей команды:

```
$ python train_vggnet.py --checkpoints контрольные точки --prefix vggnet \ --start-
epoch 50
```

На рисунке 7.2 (вверху справа) вы можете увидеть результаты снижения скорости обучения в течение 20 эпох. Сразу же вы можете увидеть огромный скачок как в обучении, так и в точности проверки. Потери при обучении и валидации также уменьшаются с обновлением скорости обучения, как это часто бывает при выполнении сдвигов порядка величины, когда точность/потери достигают насыщения в глубоких сверточных нейронных сетях , обученных на больших наборах данных, таких как ImageNet.

В прошлую эпоху 70 я еще раз отметил стагнацию потерь при проверке / точности, в то время как потери при обучении продолжали снижаться — этот индикатор показывает, что начинает происходить переоснащение. Фактически, мы можем видеть, что это расхождение начинает проявляться после 60-й эпохи на графике точности: точность обучения продолжает расти , в то время как точность проверки остается прежней.

Чтобы выжить из VGG все до последнего кусочка производительности, который я мог (без слишком ужасной переобучения), я еще раз снизил скорость обучения с 1e-3 до 1e-4 и перезапустил обучение на эпохе 70:

```
$ python train_vggnet.py --checkpoints контрольные точки --prefix vggnet \ --start-
epoch 70
```

Затем я позволил сети продолжить обучение еще 10 эпох до эпохи 80, где я применил критерии ранней остановки (рис. 7.2, внизу). В этот момент точность/потери при проверке не изменились, в то время как разрыв между потерями при обучении и потерями при проверке начал сильно расходиться, указывая на то, что мы немного переоснащаемся, и дальнейшее обучение только повредит способности модели к обобщению. В конце 80-х годов VGG16 достигал точности проверки 68,77% ранга 1 и 88,78% ранга 5. Затем я оценил 80-ю эпоху на тестовом наборе, используя следующую команду:

Скорость обучения эпохи	
1–50	1e-2
51–70	1e-3
71–	
80	1e-4

Таблица 7.1: График скорости обучения, используемый при обучении VGGNet в ImageNet.

```
$ python test_vggnet.py -checkpoints контрольные точки --prefix vggnet \
--epoch 80
[INFO] загрузка модели...
[INFO] прогнозирование на основе тестовых данных...
[ИНФО] ранг-1: 71,42%
[ИНФО] ранг-5: 90,03%
```

Как показывает мой вывод, VGG16 достиг точности 71,42% ранга 1 и 90,03% ранга 5, что почти идентично исходной статье VGGNet Симоняна и Зиссермана. Для полноты картины я включил свой график скорости обучения в Таблицу 7.1 для читателей, которые хотят воспроизвести эти результаты.

Самым большим недостатком VGG16 (помимо того, сколько времени требуется для обучения) является результирующий размер модели, который весит более 533 МБ. Если бы вы создавали приложение для глубокого обучения и намеревались поставлять размер модели вместе со своим приложением, у вас уже был бы пакет > 500 МБ для распространения. Кроме того, все программное обеспечение в какой-то момент обновляется, что требует от вас повторной упаковки и повторного распространения большого пакета размером 500 МБ, вероятно, через сетевое соединение. Для высокоскоростных широкополосных подключений этот большой размер модели может не быть проблемой. Но для устройств с ограниченными ресурсами, таких как встроенные устройства, сотовые телефоны и даже беспилотные автомобили, этот размер модели 500 МБ может стать огромным бременем. В таких ситуациях мы предпочитаем очень маленькие размеры моделей.

К счастью, все остальные модели, которые мы обсудим в этом комплекте, существенно меньше, чем VGGNet. Веса для нашей высокоточной модели ResNet составляют 102 МБ. GoogLeNet еще меньше — 28 МБ. А сверхмаленький и эффективный размер модели SqueezeNet составляет всего 4,9 МБ, что делает ее идеальной для любого типа глубокого обучения с ограниченными ресурсами.

7.5 Резюме

В этой главе мы реализовали архитектуру VGG16 с помощью библиотеки mxnet и обучили ее с нуля на наборе данных ImageNet. Вместо того, чтобы использовать утомительный и трудоемкий процесс предварительного обучения для обучения меньших версий нашей сетевой архитектуры и последующего использования этих предварительно обученных весов в качестве инициализации нашей более глубокой архитектуры, мы вместо этого пропустили этот шаг, полагаясь на работу He et al. др. и Мишкин и др.: 1. Мы заменили стандартные

активации ReLU на PReLU.

2. Мы заменили инициализацию веса Glorot/Xavier на MSRA/He et al. инициализация.

Этот процесс позволил нам воспроизвести работу Симоняна и Зиссермана в одном эксперименте. Всякий раз, когда вы обучаете VGG-подобные архитектуры с нуля, обязательно рассмотрите возможность использования PReLU и инициализации MSRA, когда это возможно. В некоторых сетевых архитектурах вы не заметите влияния на производительность при использовании PReLU + MSRA, но с VGG влияние будет существенным.

В целом, наша версия VGG16 получила точность 71,42% при ранге 1 и 90,03% при ранге 5 на тестовом наборе ImageNet, что является самой высокой точностью, которую мы когда-либо видели в этом пакете. Кроме того, архитектура VGG показала себя хорошо подходящей для задач обобщения. В следующей главе мы более подробно рассмотрим микроархитектуры, в том числе модель GoogLeNet, которая подготовит почву для более специализированных микроархитектур, включая ResNet и SqueezeNet.

8. Обучение GoogLeNet работе с ImageNet

В этой главе мы собираемся реализовать полную архитектуру GoogLeNet, представленную Szegedy et al. в своей статье 2014 года «Углубление сверток» [23]. И GoogLeNet, и VGGNet показали лучшие результаты в конкурсе ImageNet Large Scale Visual Recognition Challenge (ILSVRC) в 2014 году, при этом GoogLeNet немного опередил VGGNet и занял первое место. GoogLeNet также имеет дополнительное преимущество, заключающееся в том, что он значительно меньше, чем VGG16 и VGG19, с размером модели всего 28,12 МБ по сравнению с VGG, который составляет более 500 МБ.

При этом было показано, что семейство сетей VGG: 1. Достигает более высокой точности классификации (на практике).
2. Лучше обобщайте.

Обобщение, в частности, заключается в том, почему VGG чаще используется в задачах трансферного обучения, таких как извлечение признаков и особенно тонкая настройка. Более поздние воплощения GoogLeNet (которые просто называются Inception N, где N — номер версии, выпущенной Google) расширяют исходную реализацию GoogLeNet и модуль Inception, обеспечивая дополнительную точность — при этом мы по-прежнему склонны использовать VGG для трансферного обучения. .

Наконец, также стоит упомянуть, что многие исследователи (включая меня) столкнулись с трудностями при воспроизведении исходных результатов Szegedy et al. Независимые эксперименты ImageNet, такие как таблица лидеров, поддерживаемая vlffeat (библиотека с открытым исходным кодом для компьютерного зрения и машинного обучения), сообщают, что VGG16 и VGG19 значительно превосходят GoogLeNet [24]. Точно неясно, почему это так, но, как мы узнаем в оставшейся части этой главы, наша реализация GoogLeNet не превосходит VGG, чего мы ожидали от оригинальной публикации и испытания ILSVRC 2014. Несмотря на это, для нас по-прежнему важно изучить эту основополагающую архитектуру и способы ее обучения на наборе данных ImageNet.

8.1 Понимание Google

Мы начнем этот раздел с краткого обзора модуля Inception, нового вклада Szegedy et al. в их основополагающей работе. Оттуда мы рассмотрим полную архитектуру GoogLeNet, которая использовалась в конкурсе ILSVRC 2014. Наконец, мы реализуем GoogLeNet, используя Python и мкснет.

8.1.1 Начальный модуль

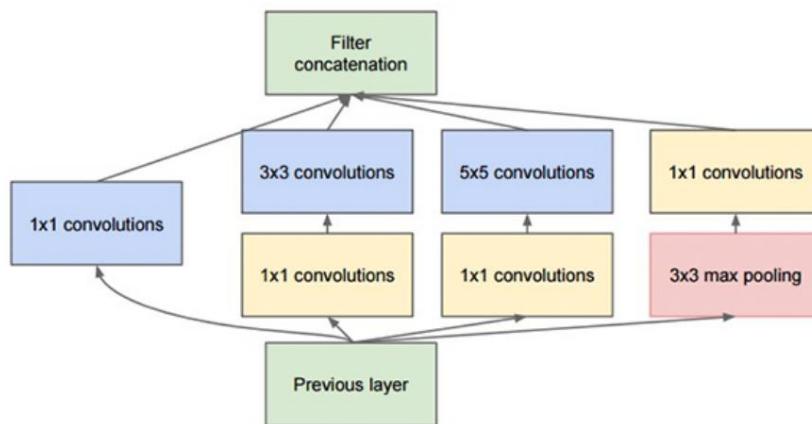


Рисунок 8.1: Исходный модуль Inception, используемый в GoogLeNet. Модуль Inception действует как «многоуровневый экстрактор признаков», вычисляя свертки 1×1 , 3×3 и 5×5 в одном и том же модуле сети. Рисунок из Szegedy et al., 2014 [23].

Начальный модуль представляет собой четырехветвевую микроархитектуру, используемую внутри архитектуры GoogLeNet (рис. 8.1). Основная цель начального модуля — изучить многомасштабные функции (фильтры 1×1 , 3×3 и 5×5), а затем позволить сети «решить», какие веса являются наиболее важными, на основе алгоритма оптимизации.

Первая ветка модуля Inception полностью состоит из фильтров 1×1 . Вторая ветвь применяет свертки 1×1 , за которыми следуют фильтры 3×3 . Меньшее количество фильтров 1×1 изучается как форма уменьшения размерности, тем самым уменьшая количество параметров в общей архитектуре. Третья ветвь идентична второй, только вместо изучения фильтров 3×3 она изучает фильтры 5×5 .

Четвертая и последняя ветвь в начальном модуле называется ветвью проекции пула. Проекция пула применяет максимальное объединение 3×3 , за которым следует серия сверток 1×1 . Причина этой ветви заключается в том, что современные сверточные нейронные сети примерно в 2014 году активно использовали максимальное объединение. Было высказано предположение, что для достижения высокой точности в сложном наборе данных ImageNet сетевая архитектура должна применять максимальный пул, поэтому мы видим его в модуле Inception. Теперь мы знаем, что максимальный пул не является обязательным требованием в сетевой архитектуре, и вместо этого мы можем уменьшить размер тома исключительно с помощью слоев CONV [21, 25]; однако в то время это было преобладающей мыслью.

Выходные данные четырех ветвей объединяются по измерению канала, образуя стек фильтров, а затем передаются на следующий уровень в сети. Для более подробного ознакомления с начальным модулем см. главу 12 пакета «Практик».

8.1.2 Архитектура GoogLeNet На рис. 8.2

показана архитектура GoogLeNet, которую мы будем реализовывать, включая количество фильтров для каждой из четырех ветвей начального модуля. После каждой свертки неявно подразумевается (т. е. не показано в таблице для экономии места), что применяется пакетная нормализация, за которой следует активация ReLU. Обычно мы размещаем нормализацию партии после активации, но опять же, мы будем придерживаться исходной реализации GoogLeNet, насколько это возможно.

Мы начинаем со свертки 7×7 с шагом 2×2 , где мы изучаем в общей сложности 64 фильтра. Операция максимального объединения немедленно вызывается с размером ядра 3×3 и шагом

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
convolution	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	2		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	2	64	96	128	16	32	32
inception (3b)		28×28×480	2	128	128	192	32	96	64
max pool	3×3/2	14×14×480	0						
inception (4a)		14×14×512	2	192	96	208	16	48	64
inception (4b)		14×14×512	2	160	112	224	24	64	64
inception (4c)		14×14×512	2	128	128	256	24	64	64
inception (4d)		14×14×528	2	112	144	288	32	64	64
inception (4e)		14×14×832	2	256	160	320	32	128	128
max pool	3×3/2	7×7×832	0						
inception (5a)		7×7×832	2	256	160	320	32	128	128
inception (5b)		7×7×1024	2	384	192	384	48	128	128
avg pool	7×7/1	1×1×1024	0						
dropout (40%)		1×1×1024	0						
linear		1×1×1000	1						
softmax		1×1×1000	0						

Рисунок 8.2: Полная архитектура GoogLeNet, предложенная Szegedy et al. Мы будем внедрять эту точную архитектуру и пытаются воспроизвести их точность.

2× 2. Эти слои CONV и POOL уменьшают размер входного тома с 224× 224× 3 до вплоть до 56× 56× 64 (обратите внимание, как сильно уменьшились пространственные размеры изображения).

Оттуда применяется еще один слой CONV , где мы изучаем 192 фильтра 3× 3 . Затем слой POOL следует, уменьшая наши пространственные размеры до 28× 28× 192. Далее укладываем два модуля Inception (названные 3a и 3b), за которым следует еще один POOL.

Чтобы изучить более глубокие и богатые функции, мы затем складываем пять начальных модулей (4a-4e), за которыми снова следуют БАССЕЙН . Применяются еще два начальных модуля (5a и 5b), оставляя нам размер выходного тома.

7× 7× 1024. Чтобы избежать использования полносвязных слоев, мы можем вместо этого использовать глобальное среднее значение. объединение, где мы усредняем объем 7× 7 до 1× 1× 1024. Отсев может быть применен к уменьшить переоснащение. Сегеди и др. рекомендуется использовать показатель отсева 40%; однако 50% склонны быть стандартом для большинства сетей — в этом случае мы будем следовать исходной реализации как максимально закрыть и использовать 40%. После отсева применяем единственный полносвязный слой в вся архитектура, где количество узлов равно общему количеству меток классов (1000), за которыми следует классификатором softmax.

8.1.3 Внедрение GoogLeNet

Используя приведенную выше таблицу, теперь мы можем реализовать GoogLeNet с помощью Python и библиотеки mxnet . Создайте новый файл с именем mxgooglenet.py внутри подмодуля nn.mxconv pyimagesearch, таким образом, мы можем отделить наши реализации Keras CNN от наших реализаций mxnet CNN (что я очень рекомендую):

```
--- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- нн
| | |--- __init__.py
```

```

| |--- конв.
| |--- mxconv
| | |--- __init__.py
| | |--- mxalexnet.py
| | |--- mxgooglenet.py
| | |--- mxvggnet.py
| --- предварительная обработка
| --- утилиты

```

Оттуда откройте файл, и мы начнем реализацию GoogLeNet:

```

1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3
4 класс MxGoogLeNet:
5     @статический метод
6     def conv_module (данные, K, kX, kY, pad=(0, 0), шаг=(1, 1)):
7         # определить шаблон CONV => BN => RELU
8         conv = mx.sym.Convolution (данные = данные, ядро = (kX, kY),
9             num_filter=K, pad=pad, step=шаг)
10        bn = mx.sym.BatchNorm (данные = конв.)
11        act = mx.sym.Activation(data=bn, act_type="relu")
12
13        # вернуть блок
14        ответный акт

```

В строке 6 мы определяем наш метод `conv_module`, удобную функцию, используемую для применения слоя. последовательность CONV => BN => RELU. Эта функция принимает входные данные (т. е. входные данные из предыдущий слой), размер ядра K, размер ядра kX и kY, количество заполнения нулями , и, наконец, шаг свертки. Строки 8-11 используют эти входные данные для построения нашего CONV => BN.

=> RELU, которую мы затем возвращаем вызывающей функции в строке 14.

Причина, по которой мы определяем `conv_module` , — просто для удобства — в любое время, когда нам нужно применить шаблон CONV => BN => RELU (которых будет много), мы просто делаем вызов `conv_module`.

Кроме того, это помогает очистить наш код, поскольку нам не придется явно записывать каждый Свертка, BatchNorm и активация.

Затем мы можем определить наш `inception_module` , который идентичен нашей реализации в Глава 11 пакета Practitioner Bundle, только сейчас мы используем mxnet в качестве реализующей библиотеки. а не Керас:

```

16     @статический метод
17     def inception_module (данные, num1x1, num3x3Reduce, num3x3,
18         num5x5Reduce, num5x5, num1x1Proj):
19         # первая ветка модуля Inception состоит из 1x1
20         # извилины
21         conv_1x1 = MxGoogLeNet.conv_module(данные, num1x1, 1, 1)

```

В строке 21 мы создаем первую ветвь модуля Inception, которая применяет в общей сложности `num1x1` 1×1 извилины. Вторая ветвь в модуле Inception представляет собой набор сверток 1×1 затем 3×3 свертки:

8.1 Понимание Google

```

23         # вторая ветка модуля Inception это набор 1x1
24         # свертки, за которыми следуют свертки 3x3
25         conv_r3x3 = MxGoogLeNet.conv_module(данные, num3x3Reduce, 1, 1)
26         conv_3x3 = MxGoogLeNet.conv_module(conv_r3x3, num3x3, 3, 3,
27             блокнот = (1, 1))

```

Точно такой же процесс применяется для третьей ветки, только на этот раз мы изучаем 5×5 сверток:

```

29         # третья ветка модуля Inception это набор 1x1
30         # свертки, за которыми следуют свертки 5x5
31         conv_r5x5 = MxGoogLeNet.conv_module(данные, num5x5Reduce, 1, 1)
32         conv_5x5 = MxGoogLeNet.conv_module(conv_r5x5, num5x5, 5, 5,
33             блокнот=(2, 2))

```

Последняя ветвь в нашем начальном модуле — это проекция пула, которая представляет собой просто максимальный пул. за которым следует слой CONV 1×1 :

```

35         # последняя ветвь начального модуля - POOL +
36         # набор проекционных слоев
37         pool = mx.sym.Pooling(data=data, pool_type="max", pad=(1, 1),
38             ядро = (3, 3), шаг = (1, 1))
39         conv_proj = MxGoogLeNet.conv_module(pool, num1x1Proj, 1, 1)

```

Выход этих четырех каналов (т. е. сверток) затем объединяется по измерение канала, формирующее выход модуля Inception:

```

41         # объединить фильтры по измерению канала
42         concat = mx.sym.Concat(*[conv_1x1, conv_3x3, conv_5x5,
43             conv_proj])
44
45         # вернуть блок
46         вернуть concat

```

Причина, по которой мы можем выполнить эту конкатенацию, заключается в том, что мы уделяем особое внимание добавлению каждой из сверток 3×3 и 5×5 так, чтобы размеры выходного объема были одинаковыми. Если формы выходных объемов для каждой ветви не совпадали, тогда мы не смогли бы выполнить конкатенацию.

Учитывая conv_module и inception_module, теперь мы готовы создать сборку . ответственный метод или использование этих строительных блоков для построения архитектуры GoogLeNet:

```

48     @статический метод
49     деф- сборка (классы):
50         # ввод данных
51         данные = mx.sym.Variable("данные")
52
53         # Блок №1: КОНВ => ПУЛ => КОНВ => ПУЛ
54         conv1_1 = MxGoogLeNet.conv_module(данные, 64, 7, 7,
55             pad=(3, 3), шаг=(2, 2))
56         pool1 = mx.sym.Pooling(data=conv1_1, pool_type="max",

```

```

57         pad=(1, 1), kernel=(3, 3), step=(2, 2))
58     conv1_2 = MxGoogLeNet.conv_module(pool1, 64, 1, 1)
59     conv1_3 = MxGoogLeNet.conv_module(conv1_2, 192, 3, 3,
60             блокнот = (1, 1))
61     pool2 = mx.sym.Pooling(data=conv1_3, pool_type="max",
62             pad=(1, 1), kernel=(3, 3), step=(2, 2))

```

Следуя рис. 8.2 выше, мы начинаем с применения CONV => POOL => CONV => POOL к уменьшить пространственные входные размеры с 224× 224 пикселей до 28× 28 пикселей.

 Имейте в виду, что мы просто применяем пакетную нормализацию и активацию после каждого CONV. Эти уровни были исключены из описания кода (1) для краткости и (2) потому что мы уже обсудили, как они применяются внутри функции conv_module.

Затем мы применяем два начальных модуля (3a и 3b), а затем POOL:

```

64     # Блок №3: (INCEP * 2) => ПУЛ
65     in3a = MxGoogLeNet.inception_module(pool2, 64, 96, 128, 16,
66             32, 32)
67     in3b = MxGoogLeNet.inception_module(in3a, 128, 128, 192, 32,
68             96, 64)
69     pool3 = mx.sym.Pooling(data=in3b, pool_type="max",
70             pad=(1, 1), kernel=(3, 3), step=(2, 2))

```

Точные параметры для каждой из ветвей можно найти в таблице 8.2 выше. Сегеди и др. определили эти параметры путем собственных экспериментов, и рекомендуется оставьте количество фильтров для каждой ветви как есть. Я предлагаю более подробное обсуждение предположения, которые мы делаем относительно количества фильтров для каждого слоя CONV в начальном модуле в главе 12 комплекта для практикующих.

Чтобы изучить более глубокие и богатые функции, которые являются более разборчивыми, мы затем применяем пять начальных модули (4a-4e), за которыми следует еще один POOL:

```

72     # Блок №4: (INCEP * 5) => ПУЛ
73     in4a = MxGoogLeNet.inception_module(pool3, 192, 96, 208, 16,
74             48, 64)
75     in4b = MxGoogLeNet.inception_module(in4a, 160, 112, 224, 24,
76             64, 64)
77     in4c = MxGoogLeNet.inception_module(in4b, 128, 128, 256, 24,
78             64, 64)
79     in4d = MxGoogLeNet.inception_module(in4c, 112, 144, 288, 32,
80             64, 64)
81     in4e = MxGoogLeNet.inception_module(in4d, 256, 160, 320, 32,
82             128, 128)
83     pool4 = mx.sym.Pooling(data=in4e, pool_type="max",
84             pad=(1, 1), kernel=(3, 3), step=(2, 2))

```

Обратите внимание, чем глубже мы погружаемся в архитектуру и чем меньше становится выходной объем, тем мы изучаем больше фильтров — это распространенный шаблон при создании сверточных нейронных сетей.

Используются еще два начальных модуля (5a и 5b), за которыми следует средний пул и отсев:

```

86      # Блок №5: (INCEP * 2) => POOL => DROPOUT
87      in5a = MxGoogLeNet.inception_module(pool4, 256, 160, 320, 32,
88          128, 128)
89      in5b = MxGoogLeNet.inception_module(in5a, 384, 192, 384, 48,
90          128, 128)
91      pool5 = mx.sym.Pooling(data=in5b, pool_type="avg",
92          ядро = (7, 7), шаг = (1, 1))
93      do = mx.sym.Dropout (данные = pool5, p = 0,4)

```

Изучив строки 89 и 90, вы увидите, что всего $384+384+128+128 = 1024$ фильтра . выводятся в результате операции конкатенации во время начального модуля 5b. Таким образом, это в результате получается размер выходного тома $7 \times 7 \times 1024$. Чтобы уменьшить потребность в полностью подключенных слоев, мы можем усреднять пространственные размеры 7×7 до $1 \times 1 \times 1024$, обычная техника используется в GoogLeNet, ResNet и SqueezeNet. Это позволяет сократить общее количество параметров в нашей сети существенно.

Наконец, мы определяем один плотный слой для нашего общего количества классов и применяем softmax. классификатор:

```

95      # классификатор softmax
96      сгладить = mx.sym.Flatten (данные = сделать)
97      fc1 = mx.sym.FullyConnected (данные = сглаживание, num_hidden = классы)
98      модель = mx.sym.SoftmaxOutput (данные = fc1, имя = "softmax")
99
100     # вернуть архитектуру сети
101     модель возврата

```

В следующем разделе мы рассмотрим скрипт train_googlenet.py , используемый для обучения наших реализаций GoogLeNet.

8.1.4 Обучение GoogLeNet

Теперь, когда мы внедрили GoogLeNet в mxnet, мы можем перейти к его обучению на ImageNet. набор данных с нуля. Прежде чем мы начнем, давайте взглянем на структуру каталогов нашего проекта:

```

--- mx_imagenet_googlenet
| |--- конфигурация
| | |--- __init__.py
| | |--- imagenet_googlenet_config.py
| |--- вывод/
| | |--- test_googlenet.py
| |--- train_googlenet.py

```

Как и в главах AlexNet и VGGNet, структура проекта идентична. train_googlenet.py _ script будет отвечать за обучение реальной сети. Затем мы можем использовать test_googlenet.py сценарий для оценки производительности определенной эпохи GoogLeNet на тестовом наборе ImageNet.



Файл test_googlenet.py фактически идентичен как test_alexnet.py , так и test_vggnet.py.
– Я просто скопировал и включил файл в этот каталог, если вы захотите изменить его для себя.

собственные нужды.

Самое главное, у нас есть файл `imagenet_googlenet_config.py`, который управляет нашей конфигурацией для обучения GoogLeNet. Я скопировал этот файл прямо из `imagenet_alexnet_config.py`, переименовав его в `imagenet_googlenet_config.py`. Пути к файлам конфигурации ImageNet идентичны нашим предыдущим экспериментам; единственным исключением является конфигурация `BATCH_SIZE` и `NUM_DEVICES`:

```
1 # определить размер пакета и количество устройств, используемых для обучения
2 BATCH_SIZE = 128
3 ЧИСЛО_УСТРОЙСТВ = 3
```

Здесь я указываю, что за один раз по сети должны передаваться пакеты из 128 изображений — это удобно для моего графического процессора Titan X с 12 ГБ памяти. Если у вас меньше памяти на вашем графическом процессоре, просто уменьшите размер пакета (в степени двойки), пока пакет не будет соответствовать размеру сети. Затем я установил `NUM_DEVICES` = 3, указав, что хочу использовать три графических процессора для обучения GoogLeNet. Используя три графических процессора, я мог проходить эпоху примерно каждые 1,7 часа или 14 эпох за один день. Если бы вы использовали графический процессор, вы могли бы ожидать от шести до восьми эпох в день, что делает полностью возможным обучение GoogLeNet менее чем за неделю.

Теперь, когда наша конфигурация обновлена, пришло время реализовать `train_googlenet.py`. Я бы предложил скопировать либо `train_alexnet.py`, либо `train_vggnet.py` из предыдущих глав, так как нам нужно будет лишь внести небольшие изменения в `train_googlenet.py`. После копирования файла откройте его, и мы просмотрим содержимое:

```
1 # импортируем необходимые пакеты 2 из
config import imagenet_googlenet_config as config 3 из
pyimagesearch.nn.mxconv import MxGoogLeNet 4 import mxnet as mx 5
import argparse 6 import logging 7 import json 8 import os
```

В строке 2 мы импортируем наш файл `imagenet_googlenet_config`, чтобы иметь доступ к переменным внутри конфигурации. Затем мы импортируем нашу реализацию `MxGoogLeNet` из раздела 8.1.3.

Далее, давайте проанализируем наши аргументы командной строки и создадим файл журнала, чтобы мы могли сохранить процесс обучения (что позволяет нам просматривать журнал и даже отображать потери/точность обучения):

```
10 # построить разбор аргумента и разобрать аргументы 11 ap =
argparse.ArgumentParser() 12 ap.add_argument("-c", "-checkpoints",
required=True,
13         help="путь к выходному каталогу контрольных точек")
14 ap.add_argument("-p", "--prefix", required=True,
15         help="имя префикса модели") 16
ap.add_argument("-s", "--start-epoch", type=int, default=0,
17         help="эпоха для возобновления обучения") 18
args = vars(ap.parse_args())
19
20 # установить уровень логирования и выходной
файл 21 logging.basicConfig(level=logging.DEBUG,
22         filename="training_{}.log".format(args["start_epoch"]), filemode="w")
23
```

```

24
25 # загрузите средства RGB для тренировочного набора, затем определите пакет
26 # размер
27 означает = json.loads(open(config.DATASET_MEAN).read())
28 batchSize = config.BATCH_SIZE * config.NUM_DEVICES

```

Эти аргументы командной строки идентичны нашим главам AlexNet и VGGNet. Сначала мы укажите переключатель `-checkpoints`, путь к каталогу для сериализации каждого из весов модели после каждой эпохи. Префикс – служит названием модели – в этом случае вы, вероятно, захотите указать `-значение префикса googlenet` при вводе скрипта в ваш терминал. В случае, если мы перезапускаем обучение с определенной эпохи, мы можем предоставить `--start-epoch`.

Строки 21-23 создают файл журнала на основе `--start-epoch`, чтобы мы могли сохранить результаты обучения. а затем постройте график потерь/точности. Чтобы выполнить нормализацию среднего, мы загружаем наши средние значения RGB. с диска в строке 27.

Затем мы получаем размер партии для оптимизатора на основе (1) размера партии, указанного в файл конфигурации и (2) количество устройств, которые мы будем использовать для обучения сети. Если мы используем несколько устройств (GPU/CPU/машины/и т. д.), то нам нужно масштабировать шаг обновления градиента на этот размер партии.

Далее создадим итератор обучающих данных:

```

30 # построить итератор обучающего изображения
31 trainIter = mx.io.ImageRecordIter(
32     path_imgrec=config.TRAIN_MX_REC,
33     data_shape=(3, 224, 224),
34     batch_size = размер партии,
35     rand_crop = Верно,
36     rand_mirror = Верно,
37     повернуть=15,
38     max_shear_ratio=0,1,
39     mean_r = означает ["R"],
40     mean_g = означает ["G"],
41     mean_b = означает ["B"],
42     preprocess_threads=config.NUM_DEVICES * 2)

```

А также итератор проверки:

```

44 # создать итератор проверочного изображения
45 valIter = mx.io.ImageRecordIter(
46     path_imgrec=config.VAL_MX_REC,
47     data_shape=(3, 224, 224),
48     batch_size = размер партии,
49     mean_r = означает ["R"],
50     mean_g = означает ["G"],
51     mean_b = означает ["B"])

```

Как и в пакете Practitioner Bundle, мы будем использовать оптимизатор Adam для обучения GoogLeNet:

```

53 # инициализировать оптимизатор
54 opt = mx.optimizer.Adam (learning_rate = 1e-3, wd = 0,0002,
55             rescale_grad=1.0 / размер партии)

```

Как вы увидите в Разделе 8.3, я первоначально использовал SGD в качестве оптимизатора, но обнаружил, что он возвращает результаты ниже среднего. Дальнейшая точность была достигнута с помощью Адама. Я также включил небольшое количество веса распада (wd=0,0002) в соответствии с рекомендациями Szegedy et al.

Далее создадим checkpointsPath, путь к каталогу, где находится GoogLeNet. веса будут сериализованы после каждой эпохи:

```

57 # построить путь контрольных точек, инициализировать аргумент модели и
58 # вспомогательные параметры
59 checkpointsPath = os.path.sep.join([args["checkpoints"],
60         аргументы["предикс"]])
61 argParams = Нет
62 дополнительных параметра = нет

```

Мы будем использовать метод ctrl + c для обучения GoogLeNet на ImageNet, поэтому нам нужен метод для (1) обучения GoogLeNet с нуля и (2) перезапуска обучения с определенной эпохи:

```

64 # если не указана начальная эпоха конкретной модели, то
65 # инициализируем сеть
66 , если args["start_epoch"] <= 0:
67     # построить архитектуру LeNet
68     print("[INFO] строим сеть...")
69     модель = MxGoogLeNet.build(config.NUM_CLASSES)
70
71 # в противном случае указана конкретная контрольная точка
72 еще:
73     # загружаем чекпойнт с диска
74     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
75     модель = mx.model.FeedForward.load(путь к контрольным точкам,
76             аргументы["начала_эпохи"])
77
78     # обновить модель и параметры
79     argParams = модель.arg_params
80     auxParams = модель.aux_params
81     модель = модель.символ

```

Строки 66-69 обрабатывают создание экземпляра архитектуры MxGoogLeNet . В противном случае строки 71-81 предполагают, что мы загружаем контрольную точку с диска и перезапускаем обучение с определенной эпохи. (вероятно, после обновления гиперпараметров, таких как скорость обучения).

Наконец, мы можем скомпилировать нашу модель:

```

83 # компилируем модель
84 модель = mx.model.FeedForward(
85     ctx=[mx.gpu(0), mx.gpu(1), mx.gpu(2)],
86     символ = модель,
87     инициализатор=mx.initializer.Xavier(),
88     arg_params = параметры аргумента,
89     aux_params = вспомогательные параметры,
90     оптимизатор=выбор,
91     число_эпох = 90,
92     begin_epoch=args["start_epoch"])

```

Здесь вы можете видеть, что я обучаю GoogLeNet с использованием трех графических процессоров — вам следует изменить строку 85 в зависимости от количества устройств, доступных на вашем компьютере. Я также инициализирую свои веса, используя метод Xavier/Glorot, тот же метод, который был предложен в оригинальной статье GoogLeNet . Мы позволим нашей сети обучаться максимум в течение 90 эпох, но, как будет показано в Разделе 8.3, нам не понадобится столько эпох.

Я рекомендую установить для num_epochs значение, превышающее то, что, по вашему мнению, вам понадобится. Один из худших пожирателей времени — это наблюдение за экспериментом, отход ко сну и пробуждение на следующее утро только для того, чтобы обнаружить, что обучение остановилось раньше и могло бы быть продолжено — вы всегда можете вернуться к предыдущей эпохе, если произойдет переобучение или потеря валидации. /точность падает.

Далее давайте создадим наши обратные вызовы и метрики оценки:

```
94 # инициализировать обратные вызовы и метрики оценки
95 batchEndCBs = [mx.callback.Speedometer(batchSize, 250)] 96
epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)] 97 metrics =
[mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5) , mx.metric.CrossEntropy()]
98
```

И, наконец, мы можем обучить нашу сеть:

```
100 # обучить сеть
101 print("[ИНФО] тренировочная сеть...")
102 model.fit(
103     X=trainIter,
104     eval_data=valIter,
105     eval_metric=метрики,
106     batch_end_callback=batchEndCB,
107     epoch_end_callback=epochEndCB)
```

К этому моменту просмотр файлов train_*.py для ImageNet должен показаться излишним. Я специально создал эти файлы так, чтобы они ощущались как «каркас». Всякий раз, когда вы хотите изучить новую архитектуру в ImageNet, скопируйте один из этих каталогов, обновите файл конфигурации, измените любой импорт модели, настройте оптимизатор и схему инициализации, а затем начните обучение — глубокое обучение может быть сложным, но наш код должен не быть.

8.2 Оценка GoogLeNet

Чтобы оценить GoogLeNet на тестовом наборе ImageNet, мы будем использовать сценарий test_googlenet.py , упомянутый в структуре нашего проекта выше. Этот файл идентичен как test_alexnet.py , так и test_vggnet.py , поэтому мы пропустим просмотр файла, чтобы избежать избыточности. Я включил test_googlenet.py в структуру каталогов и загрузок, связанных с этой книгой, для полноты картины. Подробный обзор этого файла см. в главе 6 на сайте AlexNet.

8.3 Эксперименты GoogLeNet

Результаты ILSVRC 2014 показывают, что GoogLeNet лишь немного опередил VGGNet за первое место [26]; однако многим исследователям глубокого обучения (включая меня) было трудно воспроизвести эти точные результаты [24] — результаты должны быть как минимум на одном уровне с VGG. Вероятно, во время обучения отсутствует параметр, которого нет у меня и других в Szegedy et al. бумага.

Несмотря на это, по-прежнему интересно просмотреть эксперименты и мыслительный процесс, который он использует, чтобы взять GoogLeNet и добиться приемлемой точности (т. е. лучше, чем у AlexNet, но не так хорошо, как у VGG).

8.3.1 GoogLeNet: эксперимент №1

В моем первом эксперименте с GoogLeNet я использовал оптимизатор SGD (в отличие от оптимизатора Adam, подробно описанного в разделе 8.1.4 выше) с начальной скоростью обучения $1e - 2$, базовой скоростью обучения, рекомендованной авторами, а также импульсом 0,9 и уменьшение веса L2 0,0002. Насколько я мог судить, эти параметры оптимизатора были точной копией того, что Szegedy et al. сообщить в своей газете. Затем я начал тренироваться, используя следующую команду:

```
$ python train_googlenet.py --checkpoints контрольные точки --prefix googlenet
```

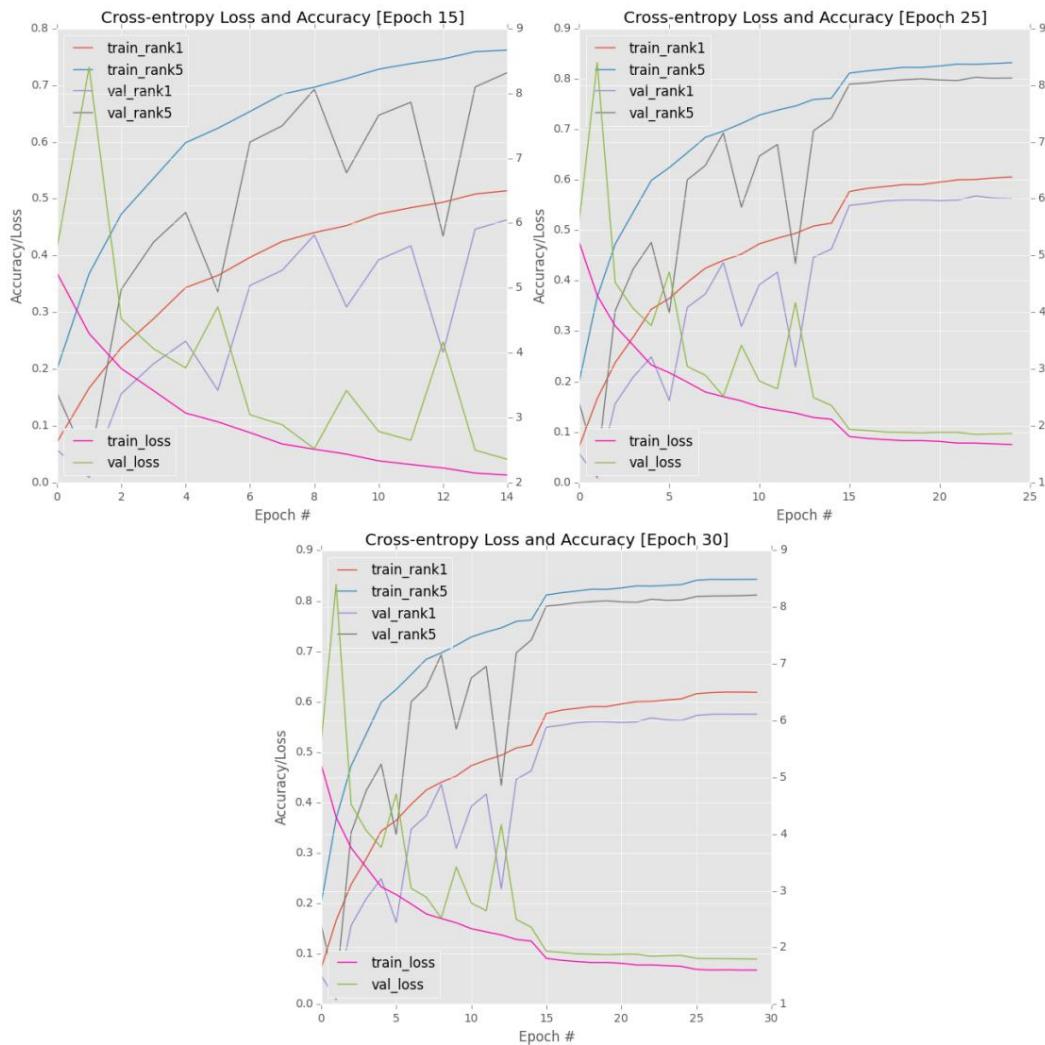


Рисунок 8.3: Верхний левый: первые 15 эпох обучения GoogLeNet в ImageNet очень изменчивы. Вверху справа: снижение скорости обучения на порядок стабилизирует волатильность; однако обучение быстро застаетается. Внизу: снижение скорости обучения до $1e - 4$ не улучшает результаты.

Как показывают мои первые 15 эпох, обучение чрезвычайно изменчиво при изучении проверочного набора (рис. 8.3, вверху слева). Наблюдается резкое падение точности проверки наряду со значительным увеличением потерь при проверке. Пытаясь бороться с волатильностью, я прекратил обучение после 15-й эпохи и снизил скорость обучения до $1e-3$, а затем возобновил обучение:

```
$ python train_googlenet.py --checkpoints контрольные точки --prefix googlenet \
--начало эпохи 15
```

К сожалению, это обновление привело к полной стагнации производительности сети после небольшого повышения точности (рис. 8.3, вверху справа). Чтобы убедиться, что стагнация действительно имела место, я прекратил обучение после 25-й эпохи, снизил скорость обучения до $1e-4$, а затем возобновил обучение еще на пять эпох:

```
$ python train_googlenet.py --checkpoints контрольные точки --prefix googlenet \
--начало эпохи 25
```

Как показывает мой график, есть небольшой скачок в точности, поскольку SGD может перемещаться в область с меньшими потерями, но в целом обучение стабилизировалось (рис. 8.3, внизу). После эпохи 30 я прекратил обучение и присмотрелся к результатам. Быстрый просмотр журналов показал, что я получил 57,75% точности ранга 1 и 81,15% ранга 5 на проверочном наборе. Хотя это неплохое начало, оно далеко от того, что я ожидал: где-то между точностью AlexNet и VGGNet.

8.3.2 GoogLeNet: Эксперимент №2

Учитывая крайнюю изменчивость скорости обучения $1e-2$, я решил полностью перезапустить обучение, на этот раз используя базовую скорость обучения $1e-3$, чтобы упростить процесс обучения. Я использовал ту же сетевую архитектуру, импульс и регуляризацию L2, что и в предыдущем эксперименте. Такой подход привел к устойчивому обучению; однако процесс был мучительно медленным (рис. 8.4, вверху слева).

Когда к 70-й эпохе обучение начало резко замедляться, я прекратил тренироваться и снизил скорость обучения до $1e-4$, затем возобновление обучения:

```
$ python train_googlenet.py --checkpoints контрольные точки --prefix googlenet \
--начало эпохи 70
```

Мы видим небольшой скачок в точности и снижение потерь, как и ожидалось, но обучение снова не происходит (рис. 8.4, вверху справа). Я позволил GoogLeNet продолжить работу до эпохи 80, затем снова прекратил обучение и снизил скорость обучения на порядок до $1e-5$:

```
$ python train_googlenet.py --checkpoints контрольные точки --prefix googlenet \
--начало эпохи 80
```

Результатом, как и следовало ожидать, является стагнация (рис. 8.4, внизу). После завершения 85-й эпохи я вообще перестал тренироваться. К сожалению, этот эксперимент не удался — я достиг только 53,67 % точности проверки ранга 1 и 77,85% ранга 5, что хуже, чем в моем первом эксперименте.

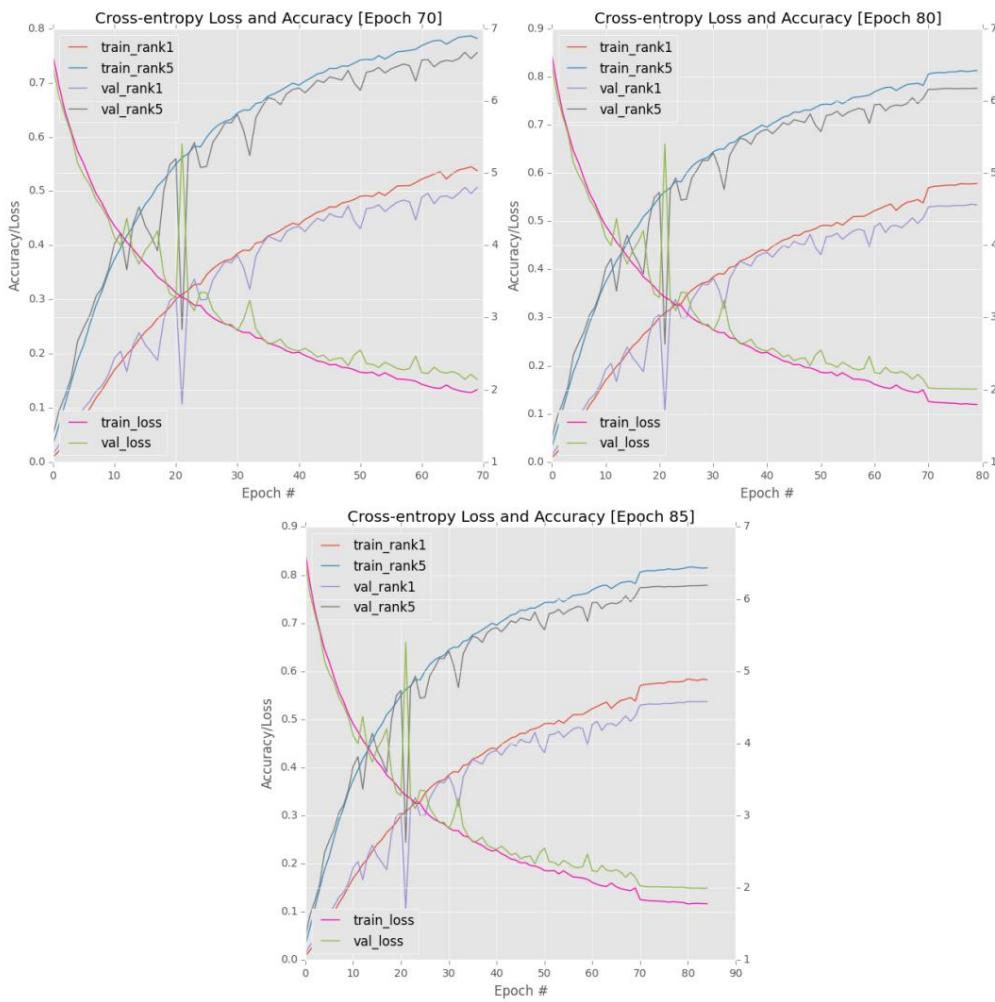


Рисунок 8.4: Верхний левый: первые 70 эпох демонстрируют чрезвычайно медленное, но стабильное обучение. Вверху справа: после стагнации обучения я снизил скорость обучения с $1e-3$ до $1e-4$ с помощью оптимизатора SGD . Однако после первоначального удара обучения быстро останавливается. Внизу: дальнейшее снижение скорости обучения не улучшает результаты.

8.3.3 GoogLeNet: эксперимент №3

Учитывая мой опыт работы с GoogLeNet и Tiny ImageNet в пакете Practitioner Bundle, я решил заменить оптимизатор SGD для Адама с начальной (по умолчанию) скоростью обучения $1e-3$. Используя этот подход, я затем следовал графику снижения скорости обучения, показанному в таблице 8.1.

График точности и потерь показан на рис. 8.5. По сравнению с другими экспериментами мы видим, что GoogLeNet + Adam быстро достиг $> 50\%$ точности проверки (менее 10 эпох по сравнению с 15 эпохами, которые потребовались в эксперименте № 1, и это только из-за изменения скорости обучения). Вы можете видеть, что мое снижение скорости обучения с $1e-3$ до $1e-4$ в эпоху 30 привело к скачку точности. К сожалению, на 45-й эпохе не было такого большого скачка, когда я переключился на $1e-5$ и позволил продолжить обучение еще на пять эпох.

После 50-й эпохи я полностью прекратил тренировки. Здесь GoogLeNet достиг $63,45\%$ 1-го ранга и $84,90\%$ точность 5-го ранга на проверочном наборе, что намного лучше, чем в предыдущих двух экспериментах.

Затем я решил запустить эту модель на тестовом наборе:

Скорость обучения эпохи	
1	30 1e 3
31	45 1d 4
46	50 1d 5

Таблица 8.1: График скорости обучения, используемый при обучении GoogLeNet на ImageNet в эксперименте № 3.

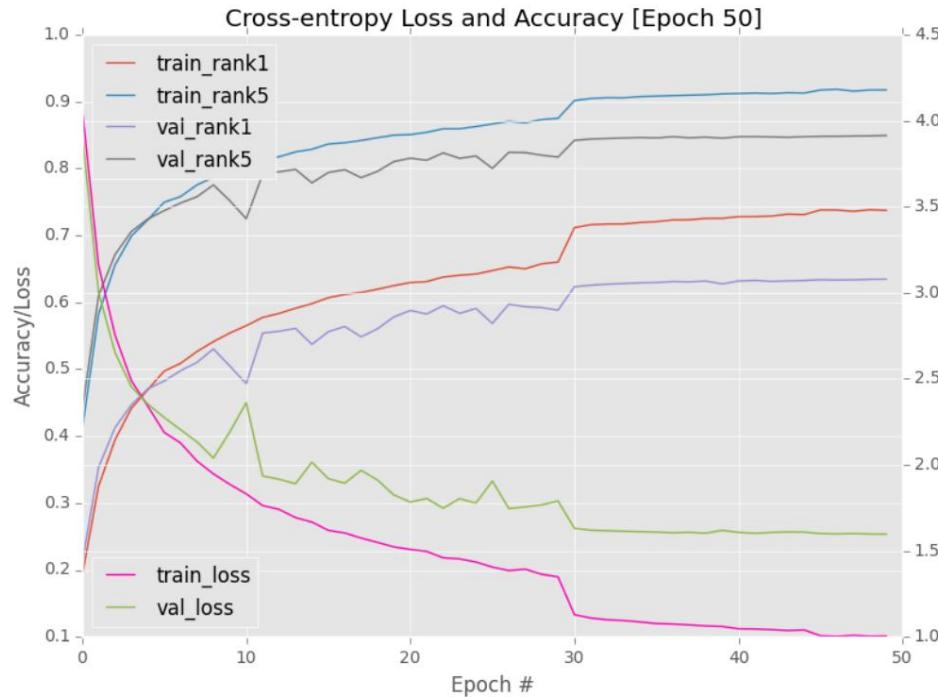


Рисунок 8.5: Использование оптимизатора Adam при обучении GoogLeNet дало наилучшие результаты; однако мы не можем воспроизвести оригинальную работу Szegedy et al. Наши результаты согласуются с независимыми экспериментами [24], которые пытались воспроизвести первоначальную производительность, но немного не дотянули.

```
$ python test_googlenet.py --checkpoints контрольные точки --prefix googlenet \
--epoch 50
[INFO] загрузка модели...
[INFO] прогнозирование на основе тестовых данных...
[ИНФО] ранг-1: 65,87%
[ИНФО] ранг-5: 86,48%
```

На тестовом наборе мы получили точность 65,87% ранга 1 и 86,48% ранга 5. Этот результат, безусловно, лучше, чем у AlexNet, но не идет ни в какое сравнение с 71,42% ранга 1 и 90,03% ранга 5 VGG.

8.4 Резюме

В этой главе мы изучили архитектуру GoogLeNet, предложенную Szegedy et al. в своей статье 2014 года *Going Deeper with Convolutions* [23]. Затем мы обучили GoogLeNet с нуля на наборе данных ImageNet, пытаясь воспроизвести их первоначальные результаты. Однако вместо того, чтобы копировать

Сегеди и др. результаты, мы вместо этого проверили результаты vlfeat [24], где наша сеть получает 65% точности ранга-1. Основываясь на наших результатах, а также на тех, о которых сообщает vlfeat, кажется вероятным, что существуют другие дополнительные параметры и/или процедуры обучения, о которых мы не знаем, необходимые для получения точности на уровне VGG. К сожалению, без дальнейших разъяснений от Szegedy et al. трудно точно определить, что это за дополнительные параметры.

Мы могли бы продолжить изучение этой архитектуры и внести дальнейшие оптимизации, в том числе:

1. Использование упорядочения слоев CONV => RELU => BN, а не CONV => BN => RELU.

2. Замена ReLU на ELU.

3. Используйте MSRA/He et al. инициализация в сочетании с PreLU.

Однако гораздо интереснее изучить две оставшиеся сетевые архитектуры, которые мы будем обучать на ImageNet — ResNet и SqueezeNet. Архитектура ResNet впечатляет тем, что в ней представлен остаточный модуль, способный получать самые современные точности, более высокие, чем в любой (в настоящее время) опубликованной статье на ImageNet. Затем у нас есть SqueezeNet, который достигает точности уровня AlexNet, но делает это с в 50 раз меньшими параметрами и размером модели 4,9 МБ.

9. Обучение ResNet на ImageNet

В этой главе мы будем внедрять и обучать основополагающую архитектуру ResNet с нуля. ResNet чрезвычайно важен в истории глубокого обучения, поскольку он представил концепцию остаточных модулей и сопоставления идентификаторов. Эти концепции позволили нам обучать сети, которые имеют > 200 слоев в ImageNet и > 1000 слоев в CIFAR-10 — глубины, которые ранее считались невозможными для успешного отслеживания сети.

Мы подробно рассмотрели архитектуру ResNet в главе 12 пакета Practitioner Bundle; тем не менее, эта глава по-прежнему будет кратко рассматривать текущее воплощение остаточного модуля для полноты картины. Оттуда мы будем реализовывать ResNet, используя Python и библиотеку mxnet.

Наконец, мы проведем ряд экспериментов по обучению ResNet с нуля ImageNet с конечной целью повторить работу He et al. [21, 22].

9.1 Понимание ResNet

Краеугольным камнем ResNet является остаточный модуль, впервые представленный He et al. в своей статье 2015 года «Глубокое остаточное обучение для распознавания изображений» [21]. Остаточный модуль состоит из двух ветвей. Первый — это просто ярлык, который соединяет вход с дополнением второй ветви, серией сверток и активаций (рис. 9.1, слева).

Однако в той же статье было обнаружено, что остаточные модули узких мест работают лучше, особенно при обучении более глубоких сетей. Узким местом является простое расширение остаточного модуля. У нас остался наш модуль быстрого доступа, только теперь изменилась вторая ветка к нашей микроархитектуре. Вместо того, чтобы применять только две свертки, мы теперь применяем три свертки (рис. 9.1, середина).

Первая CONV состоит из фильтров 1×1 , вторая из фильтров 3×3 и третья из фильтров 1×1 . Кроме того, количество фильтров, изученных первыми двумя CONV, составляет $1/4$ числа, изученного последним CONV — именно здесь возникает «узкое место».

Наконец, в обновленной публикации 2016 года «Отображение идентичности в глубоких остаточных сетях» [22] He et al. экспериментировал с упорядочением слоев свертки, активации и пакетной нормализации в остаточном модуле. Они обнаружили, что путем применения предварительной активации (т. е. размещения RELU и BN перед CONV) можно получить более высокую точность (рис. 9.1, справа).

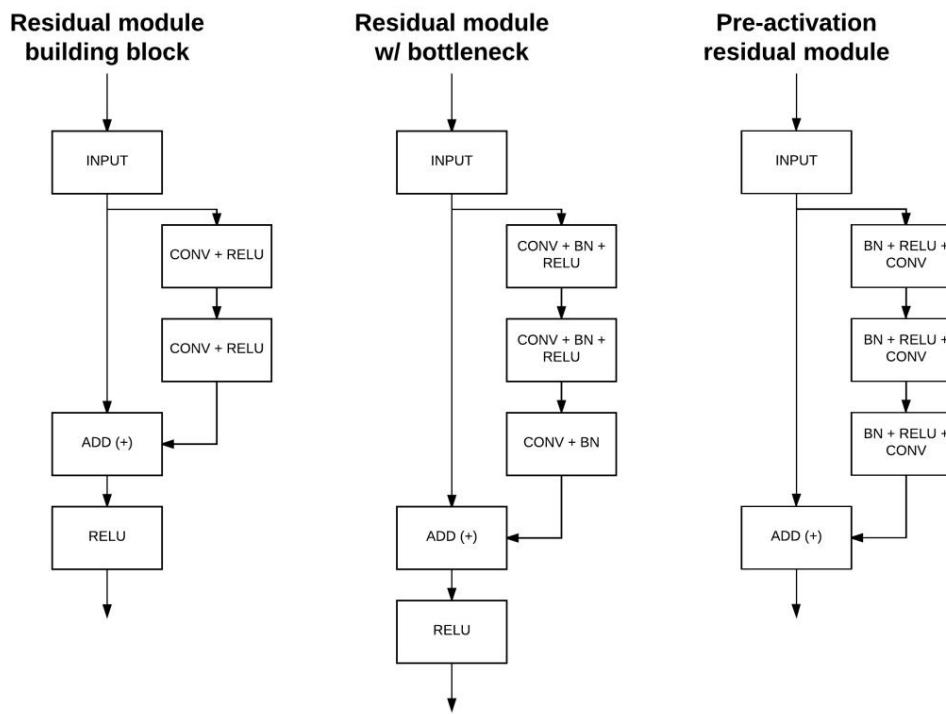


Рисунок 9.1: Слева: исходный строительный блок остаточного модуля. В центре: остаточный модуль с узким местом. Узкое место добавляет дополнительный слой CONV . Этот модуль также помогает уменьшить размерность пространственных объемов. Справа: обновленный модуль предварительной активации, который меняет порядок RELU, CONV и BN.

Мы будем использовать версию остаточного модуля с узким местом + предварительная активация при реализации ResNet в этой главе; однако в разделе 9.5.1 я приведу результаты обучения ResNet без предварительной активации, чтобы эмпирически продемонстрировать, как предварительная активация может повысить точность при применении к сложному набору данных ImageNet. Для получения более подробной информации об остаточном модуле, узких местах и предварительных активациях, пожалуйста, обратитесь к Главе 13 пакета Practitioner Bundle, где эти концепции обсуждаются более подробно.

9.2 Внедрение ResNet

Версия ResNet, которую мы будем внедрять в этой главе, — это ResNet50 от He et al. Публикация 2015 г. (рис. 9.2). Мы называем это ResNet50, потому что в сетевой архитектуре имеется пятьдесят весовых уровней (т. е. CONV или FC). Учитывая, что каждый остаточный модуль содержит три слоя CONV , мы можем вычислить общее количество слоев CONV внутри сети с помощью:

$$(3 \times 3) + (4 \times 3) + (6 \times 3) + (3 \times 3) = 48 \quad (9.1)$$

Наконец, мы добавляем один слой CONV перед остаточными модулями, за которым следует слой FC до классификатор softmax, и мы получаем в общей сложности 50 слоев — отсюда и название ResNet50.

Следуя рисунку 9.2, наш первый слой CONV применит свертку 7×7 с шагом 2×2 .

Затем мы будем использовать слой максимального пула (единственный слой максимального пула во всей сети) размером 3×3 и 2×2 . Это позволит нам быстро уменьшить пространственные размеры входного тома с 224×224 до 56×56 .

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2.x	56×56	$\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$
conv3.x	28×28	$\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 8$
conv4.x	14×14	$\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 6$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 6$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 23$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 36$
conv5.x	7×7	$\left[\begin{array}{l} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$

Рисунок 9.2: Таблица 1 архитектур ResNet для ImageNet, используемых He et al. в их публикации 2015 года [21]. Остаточные модули показаны в скобках и расположены друг над другом. Количество остаточных модулей, уложенных друг на друга, можно определить по числу ×N рядом со скобками.

Оттуда мы начнем укладывать оставшиеся модули друг на друга. Для начала мы сложим три остаточных модуля (с узким местом) для изучения $K = 256$ фильтров (первые два слоя CONV изучат 64 фильтра соответственно, а последний CONV в узком месте изучит 256 фильтров). Затем будет происходить понижение частоты дискретизации через промежуточный слой CONV, уменьшая пространственные размеры до 28×28 .

Затем четыре остаточных модуля будут наложены друг на друга, каждый из которых будет отвечать за обучение $K = 512$ фильтров. Опять же, промежуточный остаточный модуль будет использоваться для уменьшения размера тома с 28×28 до 14×14 .

Мы снова будем складывать еще остаточные модули (на этот раз шесть из них), где $K = 1024$. Окончательная остаточная субдискретизация выполняется для уменьшения пространственных размеров до 7×7 , после каждого мы складываем три остаточных модуля, чтобы изучить фильтры $K = 2048$. Затем применяется средний пул 7×7 (чтобы устранить необходимость в плотных, полностью связанных слоях), за которым следует классификатор softmax. Интересно отметить, что в ResNet не применяется отсев.

Я лично обучал эту версию ResNet50 с нуля на ImageNet с использованием восьми графических процессоров. Каждая эпоха занимала примерно 1,7 часа, что позволило мне пройти чуть более 14 эпох за один день.



Авторам ResNet понадобилось более двух недель, чтобы обучить самые глубокие варианты архитектуры

с использованием 8 графических процессоров.

Положительным здесь является то, что для успешного обучения ResNet потребовалось всего 40 эпох (менее трех дней). Однако вполне вероятно, что не у всех из нас есть доступ к восьми графическим процессорам, не говоря уже о четырех графических процессорах. Если у вас есть доступ к четырем-восьми графическим процессорам, вам следует обучать варианты ResNet с 50 и 101 уровнем с нуля.

Однако, если у вас есть только два-три графических процессора, я бы рекомендовал тренировать более поверхностные варианты ResNet, уменьшив количество этапов в сети. Если у вас только один графический процессор, рассмотрите возможность обучения ResNet18 или ResNet 34 (рис. 9.2). Хотя этот вариант ResNet не будет иметь такой высокой точности, как его более глубокие братья и сестры, он все же даст вам возможность попрактиковаться в этой основополагающей архитектуре. Опять же, имейте в виду, что ResNet и VGGNet — две наиболее ресурсоемкие сети, описанные в этой книге. Все остальные сети, включая AlexNet, GoogLeNet и SqueezeNet, можно обучать с меньшим количеством графических процессоров.

Теперь, когда мы рассмотрели архитектуру ResNet, давайте приступим к реализации. Создать новый файл с именем mxresnet.py внутри подмодуля mxconv файла pyimagesearch.nn.conv:

```

--- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- нн
| | |--- __init__.py
| | |--- конв.
| | |--- mxconv
| | | |--- __init__.py
| | | |--- mxalexnet.py
| | | |--- mxgooglenet.py
| | | |--- mxresnet.py
| | | |--- mxvggnet.py
| |--- предварительная обработка
| |--- утилиты

```

Здесь будет жить наша реализация архитектуры ResNet. Откройте mxresnet.py и приступим к работе:

```

1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3
4 класс MxResNet:
5     # использует модуль «узкое место» с предварительной активацией (He et al. 2016)
6     @статический метод
7     определение остаточного_модуля (данные, K, шаг, красный = False, bnEps = 2e-5,
8         bnMom=0,9):

```

В строках 7 и 8 мы определяем функцию остаточного_модуля , которая очень похожа на нашу реализация из главы 13 комплекта для практиков. Фактически наша реализация ResNet близко следует за He et al. [27] и Wei Wu [28] – будут использованы предложенные ими параметры внутри этой реализации. Остаточный_модуль принимает следующие параметры:

- **данные**: входной слой для остаточного модуля.
- **K**: количество фильтров, которым будет обучен конечный слой CONV . Первые два слоя CONV в Узким местом будет изучение K/4 слоев.
- **шаг**: шаг свертки.
- **красный**: логическое значение, указывающее, следует ли нам применять дополнительный остаточный модуль к уменьшить пространственные размеры объема (данное понижение частоты дискретизации выполняется между этапами).
- **bnEps**: Эпсилон-значение пакетной нормализации, используемое для предотвращения ошибок деления на ноль.
- **bnMom**: Импульс нормализации пакета, который служит средний» сохраняется.

Продолжим определение остаточного_модуля:

```

9     # ярлык модуля ResNet должен быть
10    # инициализируется как входные (идентификационные) данные
11    ярлык = данные
12
13    # первый блок модуля ResNet - это 1x1 CONV
14    bn1 = mx.sym.BatchNorm(data=данные, fix_gamma=False,
15                           eps=bnEps, моментум=bnMom)
16    act1 = mx.sym.Activation(data=bn1, act_type="relu")
17    conv1 = mx.sym.Convolution (данные = act1, pad = (0, 0),

```

```
18     ядро = (1, 1), шаг = (1, 1), num_filter = int (K * 0,25),
19     no_bias=Истина)
```

В строке 11 мы инициализируем ветвь быстрого доступа как входные данные для модуля. Этот ярлык позже будет добавлен к выходу нашей узкой ветки.

Оттуда строки 14-19 определяют шаблон BN => RELU => CONV первого блока 1×1 в узкое место. Мы называем это «предактивационной» версией ResNet, поскольку мы применяем пакетную нормализацию и активация перед сверткой. Мы также исключаем смещение из свертки (no_bias=True), так как смещения включены в нормализацию партии [29].

Перейдем ко второй CONV в узком месте, которая отвечает за обучение фильтров 3×3 :

```
21     # второй блок модуля ResNet - это  $3 \times 3$  CONV
22     bn2 = mx.sym.BatchNorm(data=conv1, fix_gamma=False,
23         eps=bnEps, моментум=bnMom)
24     act2 = mx.sym.Activation(data=bn2, act_type="relu")
25     conv2 = mx.sym.Convolution (data = act2, pad = (1, 1),
26         ядро = (3, 3), шаг = шаг, num_filter = int (K * 0,25),
27         no_bias=Истина)
```

Последний блок CONV в остаточном модуле узкого места состоит из K фильтров 1×1 :

```
29     # третий блок модуля ResNet - еще один набор  $1 \times 1$ 
30     # CONV.
31     bn3 = mx.sym.BatchNorm(data=conv2, fix_gamma=False,
32         eps=bnEps, моментум=bnMom)
33     act3 = mx.sym.Activation(data=bn3, act_type="relu")
34     conv3 = mx.sym.Convolution (данные = act3, pad = (0, 0),
35         ядро = (1, 1), шаг = (1, 1), num_filter = K, no_bias = True)
```

В случае, если мы должны уменьшить пространственные размеры объема, например, когда мы находимся в между этапами (красный = True) мы применяем последний слой CONV с шагом > 1 :

```
37     # если мы хотим уменьшить пространственный размер, применяем слой CONV
38     # к ярлыку
39     если красный:
40         ярлык = mx.sym.Convolution (данные = act1, pad = (0, 0),
41             ядро = (1, 1), шаг = шаг, num_filter = K,
42             no_bias=Истина)
43
44     # складываем вместе ярлык и конечный CONV
45     добавить = conv3 + ярлык
46
47     # вернуть дополнение в качестве вывода модуля ResNet
48     вернуть добавить
```

Эта проверка позволяет нам применить идеи Springenberg et al. [25] где мы используем свертки с большими шагами (а не с максимальным объединением), чтобы уменьшить размер тома. Обратите внимание, как в этом случае мы применяем CONV к выходу act1, первой активации в сети. Причина, по которой мы выполнить CONV в акте 1, а не в необработанном ярлыке, потому что (1) данные не были партия нормализована и (2) входные данные являются выходными данными операции сложения. На практике это

процесс имеет тенденцию повышать точность нашей сети. Наконец, мы заканчиваем остаточный модуль, добавляя ярлык для вывода узкого места (строка 45) и возврата вывода в строке 48.

Перейдем к методу сборки:

```

50     @статический метод
51     def build(классы, этапы, фильтры, bnEps=2e-5, bnMom=0.9):
52         # ввод данных
53         данные = mx.sym.Variable("данные")

```

Метод сборки требует трех параметров, за которыми следуют два необязательных. Первый аргумент это классы, общее количество меток классов, которые мы хотим, чтобы наша сеть узнала. Параметр этапов _ представляет собой список целых чисел, указывающих количество остаточных модулей, которые будут уложены поверх каждого Другие. Список фильтров также представляет собой список целых чисел, только этот список содержит количество фильтров в каждом Уровень CONV будет учиться на основе того, к какой стадии относится остаточный модуль. Наконец, мы можем поставить необязательные значения для эпсилон и импульса пакетной нормализации — мы оставим их на своих местах. значения по умолчанию в соответствии с рекомендацией He et al. и Вэй Ву.

Наш следующий блок кода обрабатывает создание серии BN => CONV => RELU => BN => RELU => Уровни POOL , первый набор слоев в сети перед размещением остаточных модулей поверх друг с другом:

```

55     # Блок №1: BN => CONV => ACT => POOL, затем инициализируем
56     # "тело" сети
57     bn1_1 = mx.sym.BatchNorm(данные=данные, fix_gamma=Истина,
58                             eps=bnEps, моментум=bnMom)
59     conv1_1 = mx.sym.Convolution (данные = bn1_1, pad = (3, 3),
60                                 ядро = (7, 7), шаг = (2, 2), num_filter = фильтры [0],
61                                 no_bias=Истина)
62     bn1_2 = mx.sym.BatchNorm (данные = conv1_1, fix_gamma = False,
63                             eps=bnEps, моментум=bnMom)
64     act1_2 = mx.sym.Activation(data=bn1_2, act_type="relu")
65     pool1 = mx.sym.Pooling(data=act1_2, pool_type="max",
66                            pad=(1, 1), kernel=(3, 3), step=(2, 2))
67     тело = пул1

```

Строки 57 и 58 сначала применяют BN к входным данным. Нормализация партии рекомендуется Он и др. и служит еще одной формой нормализации (помимо среднего вычитания). Оттуда мы узнать общее количество фильтров filter[0] , каждый из которых имеет размер 7×7 . Мы используем шаг 2×2 , чтобы уменьшить пространственные размеры нашего входного объема. Затем к выходным данным применяется пакетная нормализация. CONV (строки 62 и 63) с последующей активацией ReLU (строка 64).

Первый и единственный слой максимального пула затем используется в строках 65 и 66. Цель этого слой должен быстро уменьшить пространственные размеры от 112×112 (после первого слоя CONV) до 56×56 . Отсюда мы можем начать укладывать оставшиеся модули друг на друга (где «ResNet» получает название).

Чтобы сложить оставшиеся модули друг на друга, нам нужно начать цикл по количеству этапы:

```

69     # цикл по количеству стадий
70     для i в диапазоне (0, len (этапы)):
71         # инициализируем шаг, затем применяем остаточный модуль
72         # используется для уменьшения пространственного размера входного объема

```

```

73         шаг = (1, 1), если я == 0 иначе (2, 2)
74         body = MxResNet.residual_module(тело, фильтры[i + 1],
75                                         шаг, красный = True, bnEps = bnEps, bnMom = bnMom)
76
77         # цикл по количеству слоев в сцене
78         для j в диапазоне (0, этапы [i] - 1):
79             # применить модуль ResNet
80             body = MxResNet.residual_module(тело, фильтры[i + 1],
81                                         (1, 1), bnEps=bnEps, bnMom=bnMom)

```

Наш первый остаточный_модуль (строки 74 и 75) предназначен для уменьшения пространственных размеров наш входной объем на 50% (когда шаг = (2, 2)). Однако, поскольку это наш первый этап остаточного модуля, мы вместо этого установим шаг = (1, 1), чтобы гарантировать, что пространственное измерение не уменьшается. Оттуда строки 78-81 обрабатывают остаточные модули.

Имейте в виду, что этапы — это просто список целых чисел. Когда мы реализуем наш train_resnet.py script этапы будут установлены на (3, 4, 6, 3), что означает, что первый этап в ResNet будет состоять из трех остаточных модулей, уложенных друг на друга. Как только эти три остаточных модуля были сложены, мы возвращаемся к строкам 73-75, где мы уменьшаем пространственные измерения. Следующее исполнение из строк 78-81 будут складываться четыре остаточных модуля. Затем мы снова уменьшаем пространственные размеры на Строки 73-75. Этот процесс повторяется при укладке шести остаточных модулей и трех остаточных модулей, соответственно. К счастью, используя циклы for в строке 70 и строке 78, мы можем уменьшить объем кода, который нам на самом деле нужно написать (в отличие от предыдущих архитектур, где каждый уровень был «жестко запрограммированный»).

Также стоит отметить, что такое расположение списка фильтров делает его очень простым для нас . для управления глубиной данной реализации ResNet. Если вы хотите, чтобы ваша версия ResNet была глубже, просто увеличивайте количество остаточных модулей, уложенных друг на друга на каждом этапе.

В случае ResNet152 мы могли бы установить stage=(3, 8, 36, 3), чтобы создать очень глубокую сеть.

Если вы предпочитаете, чтобы ваш ResNet был более поверхностным (в случае, если вы хотите обучить его с помощью небольшое количество графических процессоров), вы уменьшите количество остаточных модулей, укладывающихся в стек на каждом этапе. Для ResNet18 мы можем установить stage =(2, 2, 2), чтобы создать неглубокую, но более быструю для обучения сеть.

После того, как мы закончили складывать оставшиеся модули, мы можем применить пакетную нормализацию, активацию, а затем средний пул:

```

83         # применяем BN => ACT => POOL
84         bn2_1 = mx.sym.BatchNorm (данные = тело, fix_gamma = False,
85                               eps=bnEps, моментум=bnMom)
86         act2_1 = mx.sym.Activation(data=bn2_1, act_type="relu")
87         pool2 = mx.sym.Pooling(data=act2_1, pool_type="avg",
88                               global_pool = Истина, ядро = (7, 7))

```

Имейте в виду, что после нашего окончательного набора остаточных модулей наш выходной объем имеет пространственный размеры $7 \times 7 \times$ классов. Следовательно, если мы применим усреднение по региону 7×7 , мы останется объем вывода размером $1 \times 1 \times$ классов — уменьшение объема вывода в таких способ также называется глобальным пулированием усреднения.

Последний шаг — просто построить один слой FC с нашим количеством классов и применить классификатор softmax:

```

90         # классификатор softmax
91         сгладить = mx.sym.Flatten (данные = pool2)
92         fc1 = mx.sym.FullyConnected (данные = сглаживание, num_hidden = классы)

```

```

93     модель = mx.sym.SoftmaxOutput (данные = fc1, имя = "softmax")
94
95     # вернуть архитектуру сети
96     модель возврата

```

9.3 Обучение ResNet

Теперь, когда мы внедрили ResNet, мы можем обучить его на наборе данных ImageNet. Но сначала давайте определить нашу структуру каталогов:

```

-- mx_imagenet_resnet
| |--- конфигурация
| | |--- __init__.py
| | |--- imagenet_resnet_config.py
| |--- вывод/
| |--- test_resnet.py
| |--- train_resnet.py

```

Я скопировал эту структуру каталогов из нашей реализации AlexNet в главе 6 и просто переименовал все вхождения «alexnet» в «resnet». Скрипт train_resnet.py будет отвечать для обучения нашей сети. Затем мы можем использовать test_resnet.py для оценки производительности ResNet. на испытательном наборе ImageNet.

Наконец, файл imagenet_resnet_config.py содержит наши конфигурации для предстоящих экспериментов. Этот файл идентичен imagenet_alexnet_config.py, только я обновил BATCH_SIZE и NUM_DEVICES:

```

53 # определить размер пакета и количество устройств, используемых для обучения
54 ПАКЕТ_РАЗМЕР = 32
55 ЧИСЛО_УСТРОЙСТВ = 8

```

Учитывая глубину ResNet50, мне нужно было уменьшить BATCH_SIZE до 32 изображений на моем Titan X. 12 ГБ графического процессора. На вашем собственном графическом процессоре вам может потребоваться уменьшить это число до 16 или 8, если графический процессор работает недостаточно памяти. Я также установил для NUM_DEVICES значение восемь, чтобы быстрее обучать сеть.

Опять же, я бы не рекомендовал обучать ResNet50 на машине без 4-8 графических процессоров. Если вы хотите для обучения ResNet на меньшем количестве графических процессоров рассмотрите возможность использования варианта с 18 или 34 уровнями, так как это будет намного быстрее. тренировать. Помимо VGGNet, это единственная сетевая архитектура, без которой я не рекомендую тренироваться. несколько графических процессоров — и даже с учетом сказанного, ResNet10 все еще можно обучать с одним графическим процессором, если вы терпелив (и вы все равно превзойдете AlexNet из главы 6).

Теперь, когда наш файл конфигурации обновлен, давайте также обновим файл train_resnet.py. сценарий. Как и во всех предыдущих главах ImageNet, сценарии train_*.py предназначены для использования в качестве framework, требуя от нас вносить как можно меньше изменений. Я быстро просмотрю этот файл, так как мы уже неоднократно рассматривались в этой книге. Для более подробного ознакомления со сценариями обучения, обратитесь к главе 6. Теперь откройте train_resnet.py, и мы приступим к работе:

1 # импортируем необходимые пакеты
2 из конфига импортировать imagenet_resnet_config как конфиг
3 из pyimagesearch.nn.mxconv импортировать MxResNet
4 импортировать mxnet как mx
5 импортировать синтаксический анализ
6 журнал импорта

```
7 импортировать json
8 импорт ОС
```

Здесь мы просто импортируем необходимые пакеты Python. Обратите внимание, как мы импортируем наш Конфигурация ResNet на линии 2 вместе с нашей реализацией MxResNet на линии 3.

Затем мы можем проанализировать наши аргументы командной строки и динамически установить наш файл журнала на основе --начало эпохи:

```
10 # построить разбор аргумента и разобрать аргументы
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-c", "-checkpoints", required=True,
13                  help="путь к выходному каталогу контрольных точек")
14 ap.add_argument("-p", "-prefix", required=True,
15                  help="имя префикса модели")
16 ap.add_argument("-s", "-start-epoch", type=int, default=0,
17                  help="эпоха для возобновления обучения")
18 аргументов = переменные (ap.parse_args())
19
20 # установить уровень логирования и выходной файл
21 logging.basicConfig(level=logging.DEBUG,
22                     имя_файла="training_{}.log".format(args["start_epoch"]),
23                     режим файла = "w")
```

Переключатель --checkpoints должен указывать на выходной каталог, в котором мы хотим сохранить сериализованные веса для ResNet после каждой эпохи. --prefix управляет именем архитектура, которая в данном случае будет resnet. Наконец, ключ --start-epoch используется для указать, с какой эпохи возобновить обучение.

Оттуда мы можем загрузить наши средства RGB из тренировочного набора и вычислить размер партии. на основе BATCH_SIZE и NUM_DEVICES в нашем файле конфигурации:

```
25 # загрузите средства RGB для тренировочного набора, затем определите пакет
26 # размер
27 означает = json.loads(open(config.DATASET_MEAN).read())
28 batchSize = config.BATCH_SIZE * config.NUM_DEVICES
```

Давайте продолжим и создадим наш итератор обучающих данных:

```
30 # построить итератор обучающего изображения
31 trainIter = mx.io.ImageRecordIter(
32     path_imgrec=config.TRAIN_MX_REC,
33     data_shape=(3, 224, 224),
34     batch_size = размер партии,
35     rand_crop = Верно,
36     rand_mirror = Верно,
37     повернуть=15,
38     max_shear_ratio=0,1,
39     mean_r = означает ["R"],
40     mean_g = означает ["G"],
41     mean_b = означает ["B"],
42     preprocess_threads=config.NUM_DEVICES * 2)
```

Вместе с итератором проверки:

```
44 # создать итератор проверочного изображения
45 valIter = mx.io.ImageRecordIter(
46     path_imgrec=config.VAL_MX_REC,
47     data_shape=(3, 224, 224),
48     batch_size = размер партии,
49     mean_r = означает ["R"],
50     mean_g = означает ["G"],
51     mean_b = означает ["B"])
```

Мы будем использовать оптимизатор SGD для обучения ResNet:

```
53 # инициализировать оптимизатор
54 opt = mx.optimizer.SGD(learning_rate=1e-1, импульс=0,9, wd=0,0001,
55     rescale_grad=1.0 / размер партии)
```

Мы начнем с начальной скорости обучения $1e^{-1}$, а затем снизим ее на порядок, когда плато потери/точности, или мы начинаем беспокоиться о переоснащении. Мы также обеспечим импульс 0,9 и снижение веса L2 0,0001, в соответствии с рекомендацией He et al.

Теперь, когда наш оптимизатор инициализирован, мы можем создать checkpointsPath, каталог где мы будем хранить сериализованные веса после каждой эпохи:

```
57 # построить путь контрольных точек, инициализировать аргумент модели и
58 # вспомогательные параметры
59 checkpointsPath = os.path.sep.join([args["checkpoints"],
60     аргументы["предикс"]])
61 argParams = Нет
62 дополнительных параметра = нет
```

В случае, если мы обучаем ResNet с нуля, нам сначала нужно создать экземпляр нашей модели:

```
64 # если не указана начальная эпоха конкретной модели, то
65 # инициализируем сеть
66 , если args["start_epoch"] <= 0:
67     # построить архитектуру LeNet
68     print("[INFO] строим сеть...")
69     модель = MxResNet.build(config.NUM_CLASSES, (3, 4, 6, 3),
70     (64, 256, 512, 1024, 2048))
```

Здесь мы будем обучать ResNet50, используя список этапов (3, 4, 6, 3) и список фильтров . из (64, 256, 512, 1024, 2048). Первый слой CONV в ResNet (перед любым остаточным модулем) выучит $K = 64$ фильтра. Затем первый набор из трех остаточных модулей выучит $K = 256$ фильтров. количество изученных фильтров увеличится до $K = 512$ для четырех остаточных модулей, сложенных вместе. В на третьем этапе будут сложены шесть остаточных модулей, каждый из которых отвечает за обучение $K = 1024$ фильтров. И, наконец, на четвертом этапе будут складываться три остаточных модуля, которые будут обучать $K = 2048$ фильтров.



Опять же, мы называем эту архитектуру «ResNet50», так как есть $1 + (3 \times 3) + (4 \times 3) + (6 \times 3) + (3 \times 3) + 1 = 50$ обучаемых слоев в сети.

Если вместо этого мы перезапускаем обучение с определенной эпохи (предположительно, после корректировки скорость обучения нашего оптимизатора), мы можем просто загрузить сериализованные веса с диска:

```

72 # в противном случае указана конкретная контрольная точка
73 еще:
74     # загружаем чекпойнт с диска
75     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
76     модель = mx.model.FeedForward.load(путь к контрольным точкам,
77                                         аргументы["начало_эпохи"])
78
79     # обновить модель и параметры
80     argParams = модель.arg_params
81     auxParams = модель.aux_params
82     модель = модель.символ

```

Теперь мы готовы скомпилировать нашу модель:

```

84 # компилируем модель
85 модель = mx.model.FeedForward(
86     ctx=[mx.gpu(i) для i в диапазоне (0, config.NUM_DEVICES)],
87     символ = модель,
88     инициализатор=mx.initializer.MSRAPrelu(),
89     arg_params = параметры аргумента,
90     aux_params = вспомогательные параметры,
91     оптимизатор=выбор,
92     число_эпох = 100,
93     begin_epoch=args["start_epoch"])

```

Я буду использовать восемь графических процессоров для обучения ResNet (строка 86), но вы захотите настроить этот список ctx на основе **количество ЦП/ГП на вашем компьютере** (а также обязательно обновите NUM_DEVICES переменная в `imagenet_resnet_config.py`). Поскольку мы обучаем очень глубокую нейронную сеть, мы будем использовать MSRA/He et al. инициализация (строка 88). Мы установим `num_epochs` для обучения на большее число (100), хотя на самом деле мы, скорее всего, не будем тренироваться так долго.

Оттуда мы можем построить наш набор обратных вызовов для мониторинга производительности сети и сериализации. контрольные точки на диск:

```

95 # инициализировать обратные вызовы и метрики оценки
96 batchEndCBs = [mx.callback.Speedometer(batchSize, 250)]
97 epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)]
98 метрик = [mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5),
99             mx.metric.CrossEntropy()]

```

Последний блок кода обрабатывает фактическое обучение сети:

```

101 # обучаем сеть
102 print("[INFO] обучающая сеть...")
103 модель.подходит(
104     X=поеzd,
105     eval_data=значение,
106     eval_metric = показатели,
107     batch_end_callback=batchEndCB,
108     epoch_end_callback=epochEndCBs)

```

9.4 Оценка ResNet

Для оценки ResNet мы будем использовать сценарий `test_resnet.py`, упомянутый в структуре нашего проекта выше. Этот сценарий идентичен всем остальным сценариям `test_*.py`, используемым в экспериментах ResNet. Поскольку эти скрипты одинаковы, я не буду рассматривать программу Python построчно. Если вам нужен подробный обзор этого оценочного сценария, обратитесь к Главе 6.

9.5 Эксперименты ResNet

В этом разделе я поделюсь тремя экспериментами, которые я провел при обучении ResNet на наборе данных ImageNet. Как мы узнаем, ResNet может быть немного сложно обучить, так как из-за количества параметров в сети он может переобучиться. Однако с положительной стороны (и как мы узнаем) обучение ResNet потребует меньше эпох, чем предыдущие эксперименты.

9.5.1 ResNet: эксперимент №1 В первых

двух экспериментах с ResNet я хотел продемонстрировать, как: 1. Использование остаточного модуля без предварительной активации приводит к неоптимальной производительности. 2. Меньшая начальная скорость обучения может снизить точность сети. Поэтому в этом первом эксперименте я реализовал остаточный модуль без предварительных активаций. Затем я использовал оптимизатор SGD с начальной скоростью обучения $1e^{-2}$. Я оставил все остальные параметры идентичными реализации, подробно описанной в разделе 9.2 выше. Обучение запускалось по следующей команде:

```
$ python train_resnet.py --checkpoints контрольные точки --prefix resnet
```

Первые 20 периодов обучения можно увидеть на рис. 9.3 (вверху слева). Этот график демонстрирует, что потери при обучении падают быстрее, чем потери при проверке. Точность обучения также растет быстрее, чем проверка. Я прекратил обучение после эпохи 20, снизил скорость обучения до $1e^{-3}$, а затем возобновил обучение:

```
$ python train_resnet.py --checkpoints контрольные точки --prefix resnet \ --start-epoch 20
```

Обучение продолжалось еще 10 эпох (рис. 9.3, вверху справа). Сначала мы видим большой скачок в точности; однако наше обучение быстро останавливается. Чтобы проверить, я прекратил обучение на 30-й эпохе, скорректировал скорость обучения до $1e^{-4}$, а затем возобновил обучение:

```
$ python train_resnet.py --checkpoints контрольные точки --prefix resnet \ --start-epoch 30
```

Я позволил продолжать обучение только в течение трех эпох, чтобы подтвердить свои выводы (рис. 9.3, внизу). Разумеется, обучение фактически застопорилось. Однако, глядя на точность ранга 1 и ранга 5, я обнаружил, что мы достигли 60,91% и 83,19%, уже превзойдя AlexNet.

9.5.2 ResNet: эксперимент №2 В моем

втором эксперименте использовалась большая базовая скорость обучения $1e-1$ (а не $1e-2$ из первого эксперимента). Никаких других изменений, кроме скорости обучения, в эксперимент внесено не было. Затем я применил тот же график скорости обучения, что и в эксперименте № 1 (таблица 9.1).

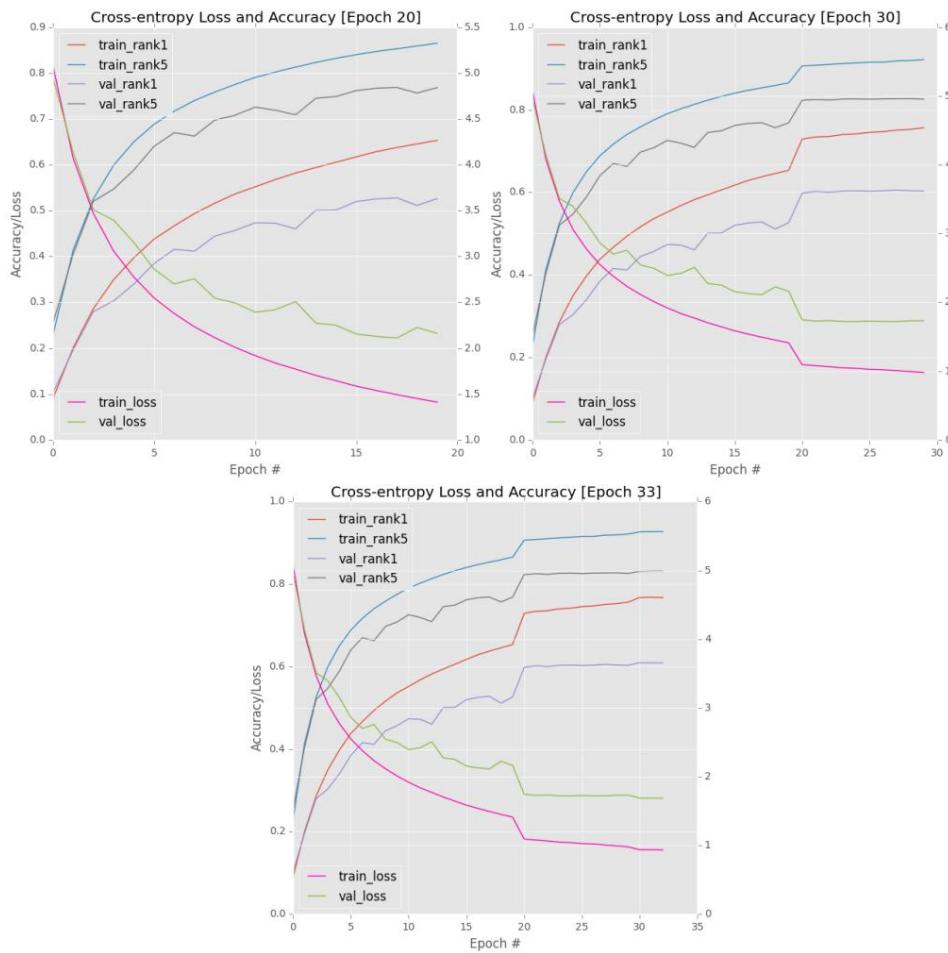


Рисунок 9.3: Вверху слева: первые 20 эпох обучения ResNet (без предварительной активации) в ImageNet.

Вверху справа: снижение обучения с $1e-2$ до $1e-3$. Это позволяет повысить точность, но обучение быстро стагнирует. Внизу: дальнейшее снижение скорости обучения лишь незначительно повышает точность.

Результаты процесса обучения можно увидеть на рис. 9.4. Опять же, точность обучения превосходит точность проверки, но мы можем быстрее получить более высокую точность. Кроме того, снижение скорости обучения с $1e-1$ до $1e-2$ в эпоху 20 дает нам огромный скачок в точности проверки на 13%. Меньший скачок в точности получается во время падения с $1e-2$ до $1e-3$ в эпоху 25; однако этот небольшой скачок сопровождается резким увеличением точности обучения/снижением потерь при обучении, что означает, что мы рискуем слишком точно моделировать данные обучения.

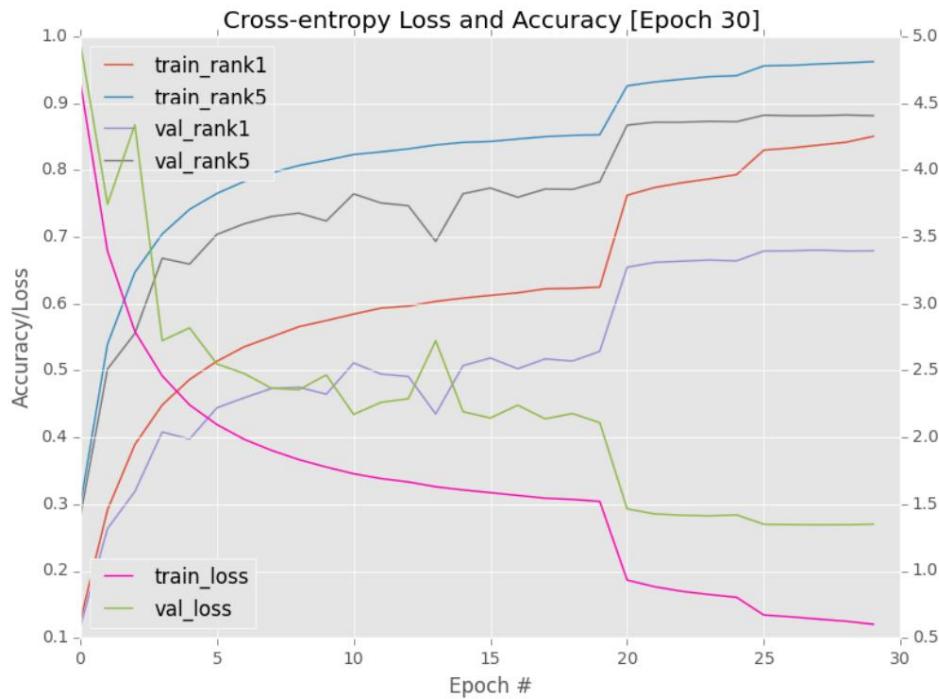
После завершения эпохи 30 я оценил производительность и обнаружил, что эта версия ResNet обеспечивает точность проверки 67,91% ранга 1 и 88,13% ранга 5, что заметно выше, чем в нашем первом эксперименте. Основываясь на этих двух экспериментах, мы, несомненно, можем сделать вывод, что $1e-1$ является лучшей начальной скоростью обучения.

9.5.3 ResNet: эксперимент №3

В моем последнем эксперименте с ResNet я заменил исходный остаточный модуль на остаточный модуль перед активацией (тот, который мы внедрили в разделе 9.2 выше). Затем я обучил ResNet с помощью оптимизатора SGD с базовой скоростью обучения $1e-1$, импульсом 0,9 и уменьшением веса L2.

Скорость обучения эпохи			
1	20	1e	2
21	25	1д	3
26	30	1д	4

Таблица 9.1: График скорости обучения, используемый при обучении ResNet на ImageNet в эксперименте № 2.

Рисунок 9.4: В эксперименте № 2 мы начинаем обучение ResNet с начальной скоростью обучения $\alpha = 1e - 1$. Это позволяет нам повысить точность по сравнению с Экспериментом №1.

0,0001. Я начал обучение, используя следующую команду:

```
$ python train_resnet.py --checkpoints контрольные точки --prefix resnet
```

В эпоху 30 мы видим, что точность обучения опережает точность проверки, но не в неразумной степени (рис. 9.5, вверху слева). Соотношение между разрывами в каждую эпоху вполне устойчиво. Однако, когда потери при валидации начали стагнировать (и даже немного увеличиваться), я прекратил обучение, снизил скорость обучения с $1e-1$ до $1e-2$ и возобновил обучение с эпохи 30:

```
$ python train_resnet.py --checkpoints контрольные точки --prefix resnet \ --start-epoch 30
```

Я позволил продолжить обучение, но, как показывают мои результаты, мы наблюдаем классические признаки сильного переобучения (рис. 9.5, вверху справа). Обратите внимание, как наши потери при проверке увеличиваются, точность проверки снижается, в то время как точность обучения и показатели потерь улучшаются. Так как

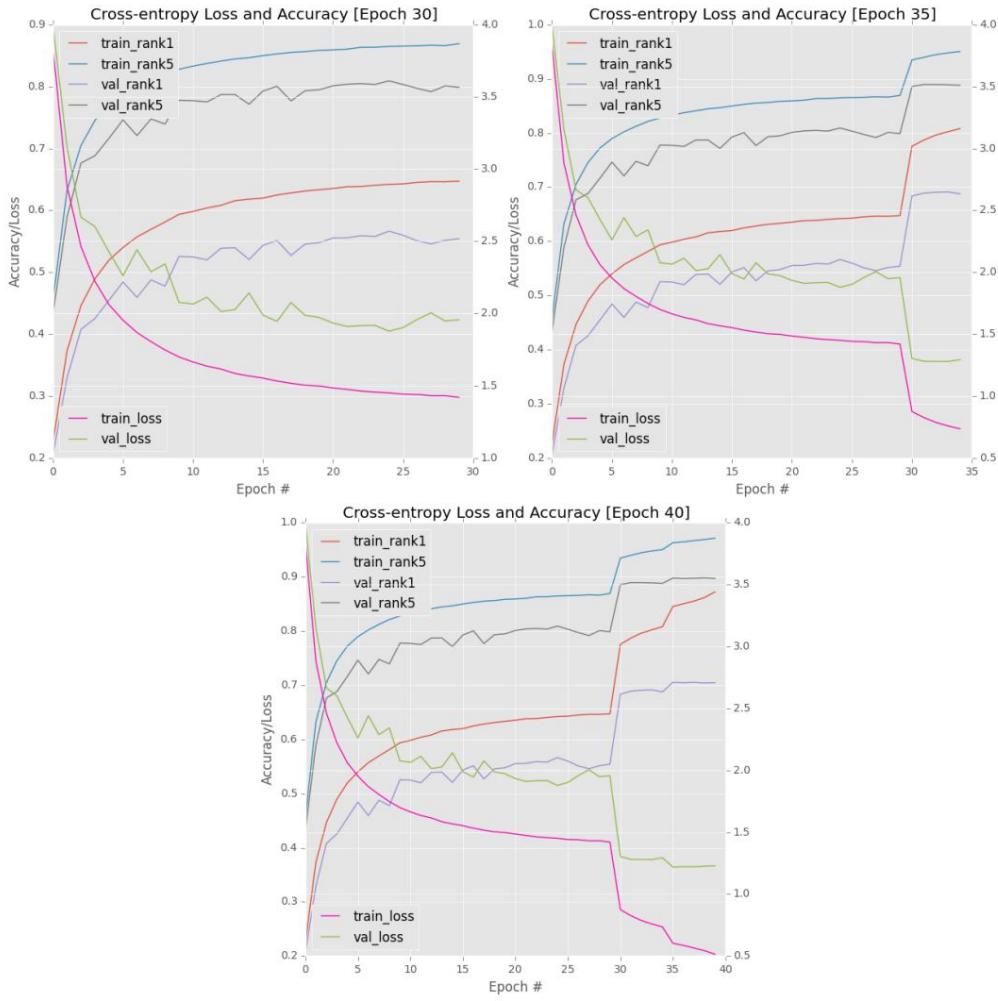


Рисунок 9.5: Верхний левый: Обучение ResNet на ImageNet с использованием остаточного модуля узкого места с предварительными активациями. Вверху справа: снижение скорости обучения до $\alpha = 1e-2$ позволяет резко повысить точность; однако нам нужно быть осторожными с переоснащением. Внизу: мы можем тренироваться только еще 5 эпох, так как переобучение становится проблематичным; тем не менее, мы по-прежнему получаем наилучшую точность на ImageNet.

Из-за этого переобучения я был вынужден прекратить обучение, вернуться к эпохе 35, изменить скорость обучения с $1e-2$ до $1e-3$ и тренироваться еще пять эпох:

```
$ python train_resnet.py --checkpoints контрольные точки --prefix resnet \ --start-epoch 35
```

Я не хотел обучать более пяти эпох, потому что, как показывают мои графики, потери при обучении уменьшаются значительно быстрее, чем потери при проверке, а точность обучения увеличивается значительно быстрее, чем точность проверки (рис. 9.5, внизу). Учитывая, что обучение больше не могло продолжаться, я остановился после эпохи 40 и проверил метрики валидации — 70,47% ранга 1 и 89,72% ранга 5 соответственно. Затем я перешел к оценке ResNet на тестовом наборе ImageNet.

```
$ python test_resnet.py --checkpoints контрольные точки --prefix resnet \
--epoch 40
[INFO] загрузка модели...
[INFO] прогнозирование на основе тестовых данных...
[ИНФО] ранг-1: 73,00%
[ИНФО] ранг-5: 91,08%
```

Здесь мы можем увидеть наши лучшие результаты: 73,01% 1-го ранга и 91,08% 5-го ранга на тестовый набор. Это, безусловно, наша лучшая производительность в наборе данных ImageNet.

Однако, согласно независимым экспериментам, проведенным vlfeat [24], мы должны получить примерно 75,40% точности ранга 1 и 92,30% ранга 5. Мы, вероятно, можем приписать наши немного более низкие результаты точности переобучению в нашей сети. Будущие эксперименты должны быть более агрессивными с регуляризацией, включая снижение веса, увеличение данных и даже попытки применить отсев. В любом случае, мы подошли очень близко к воспроизведению оригинальной работы He et al. Мы также смогли подтвердить, что версия узкого места + предварительная активация остаточного модуля действительно приводит к получению более высокой точности, чем исходный остаточный модуль.

9.6 Резюме

В этой главе мы обсудили архитектуру ResNet, включая исходный остаточный модуль, модуль узкого места и модуль узкого места + предварительной активации. Затем мы обучили ResNet50 с нуля на наборе данных ImageNet. В целом, мы смогли получить точность ранга 1 73,01% и ранга 5 91,08%, что выше, чем в любом из наших предыдущих экспериментов с набором данных ImageNet.

Хотя ResNet — чрезвычайно интересная архитектура для изучения, я не рекомендую обучать эту сеть с нуля, если у вас нет на это времени и/или финансовых ресурсов. ResNet50 требует много времени для обучения, поэтому мне пришлось использовать восемь графических процессоров для обучения сети (чтобы собрать результаты и опубликовать эту книгу вовремя). На самом деле вы также можете обучить ResNet50 с использованием четырех графических процессоров, но вам придется проявить немного больше терпения. С менее чем четырьмя графическими процессорами я бы не стал обучать ResNet50 и рекомендовал бы вам обучать более мелкие варианты, такие как ResNet10 и ResNet18. Хотя ваши результаты будут не такими хорошими, как у ResNet50, вы все равно сможете извлечь уроки из этого процесса.

Имейте в виду, что, как специалист по глубокому обучению, результаты — это еще не все — вы должны подвергать себя как можно большему количеству новых проектов и экспериментов. Глубокое обучение — это отчасти наука, отчасти искусство. Вы изучаете «искусство» обучения сети глубокого обучения, проводя сотни экспериментов. Не расстраивайтесь, если ваши результаты не совпадают с моими или другими современными публикациями после первого испытания — все это часть процесса.

Стать экспертом по глубокому обучению не получится за одну ночь. Моя цель в этой серии глав на ImageNet заключалась в том, чтобы познакомить вас с множеством различных экспериментов и показать вам, как я систематически использовал результаты каждого эксперимента, чтобы многократно улучшать свою предыдущую работу. Ведите собственный журнал экспериментов, отмечая, что сработало (и не сработало), и вы сможете применить этот процесс самостоятельно.

10. Обучение SqueezeNet на ImageNet

В нашей последней главе, посвященной обучению глубоких нейронных сетей в программе ImageNet Large Scale Visual Recognition Challenge (ILSVRC), мы обсудим, возможно, мою любимую архитектуру глубокого обучения на сегодняшний день — SqueezeNet, представленную Яндой и соавт. в своей статье 2016 года SqueezeNet: точность уровня AlexNet с в 50 раз меньшим количеством параметров и размером модели <0,5 МБ [30].

Уже одно название этой статьи должно вас заинтересовать. Большинство предыдущих сетевых архитектур, которые мы обсуждали в этой книге, имеют размер модели от 100 МБ (ResNet) до 553 МБ (VGGNet). AlexNet также находится в середине этого диапазона размеров с модельным весом 249 МБ. Ближе всего к «крошечному» размеру модели мы подошли к модели GoogLeNet с размером 28 МБ, но можем ли мы уменьшить ее, сохранив при этом современную точность?

В работе Iandola et al. показывает, что да, мы абсолютно можем уменьшить размер модели, применив новое использование сверток 1×1 и 3×3 , а также отсутствие полно связанных слоев. Конечным результатом является модель весом 4,9 МБ, которая может быть дополнительно уменьшена до 0,5 МБ за счет сжатия модели, также называемого сокращением веса и «разрежением модели» (обнуление 50% наименьших значений веса по слоям). В этой главе мы сосредоточимся на исходной реализации SqueezeNet. Концепция сжатия модели, включая квантование, выходит за рамки этой книги, но ее можно найти в соответствующей научной публикации [30].

10.1 Понимание SqueezeNet

В этом разделе мы будем:

1. Просмотрите модуль Fire, критически важную микроархитектуру, отвечающую за сокращение модели . параметры и поддержание высокого уровня точности.
2. Просмотрите всю архитектуру SqueezeNet (т. е. макроархитектуру).
3. Реализовать SqueezeNet вручную с помощью mxnet.

Давайте продолжим и начнем с обсуждения модуля Fire.

10.1.1 Пожарный модуль

Работа Яндолы и др. служил двум целям. Первая заключалась в том, чтобы предоставить возможность для дополнительных исследований в области разработки сверточных нейронных сетей, которые можно обучать с резким сокращением времени.

количество параметров (при сохранении высокого уровня точности). Вторым вкладом стал сам модуль Fire, фактическая реализация этой цели.

Модуль Fire очень умен в том, как он работает, поскольку он опирается на фазы расширения и сокращения, состоящие только из сверток 1×1 и 3×3 — визуализацию модуля Fire можно увидеть на рис. 10.1.

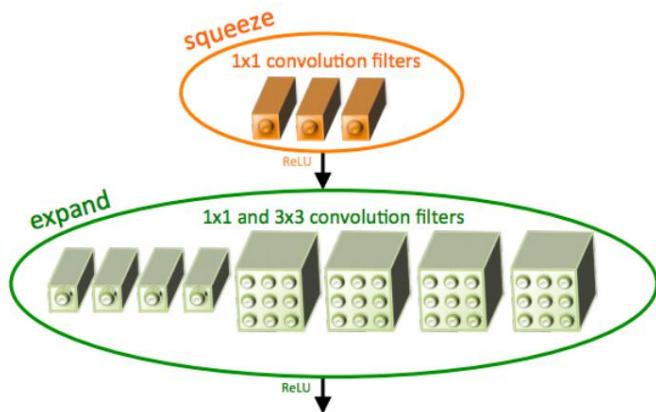


Рисунок 10.1: Модуль Fire в SqueezeNet. Сначала изучается набор фильтров 1×1 для уменьшения размерности объема. Затем изучаются комбинации фильтров 3×3 и 1×1 , которые объединяются по размеру канала для расширения выходного объема.

Фаза сжатия модуля Fire изучает набор фильтров 1×1 , за которым следует функция активации ReLU. Количество изученных фильтров сжатия всегда меньше размера тома, вводимого в сжатие; следовательно, процесс сжатия можно рассматривать как форму уменьшения размерности.

Во-вторых, используя фильтры 1×1 , мы изучаем только локальные особенности. Обычно мы используем большее ядро 3×3 или даже 5×5 для изучения функций, фиксирующих пространственную информацию о пикселях, расположенных близко друг к другу во входном объеме. Это не относится к фильтрам 1×1 — вместо того, чтобы пытаться изучить пространственные отношения между соседними пикселями, мы строго заинтересованы в отношениях, которые этот пиксель имеет между своими каналами.

Наконец, несмотря на то, что мы уменьшаем размерность во время сжатия (поскольку количество фильтров, изученных на этапе сжатия, всегда меньше, чем введенные в него фильтры), у нас фактически есть шанс добавить больше нелинейности, поскольку активация ReLU применяется после каждой свертки 1×1 .

После того, как мы применяем сжатие, вывод подается в расширение. На этапе расширения мы изучаем комбинацию сверток 1×1 и 3×3 (опять же, имея в виду, что результат сжатия — это вход для расширения). Свертки 3×3 здесь особенно полезны, поскольку они позволяют нам получать пространственную информацию из исходных фильтров 1×1 .

В архитектуре SqueezeNet мы изучаем фильтры расширения $N, 1 \times 1$, за которыми следуют фильтры $N, 3 \times 3$ — в Iandola et al. бумаги, N в 4 раза больше, чем количество скимающих фильтров, поэтому мы называем это этапом «расширения» микроархитектуры. Сначала мы «скжимаем» размерность ввода, затем «расширяем» его. Выходные данные расширений 1×1 и 3×3 объединяются по измерению фильтра, выступая в качестве конечного вывода модуля. Свертки 3×3 дополняются нулями по одному пикселю в каждом направлении, чтобы обеспечить возможность суммирования выходных данных.

Модуль Fire может поддерживать небольшое количество общих параметров за счет уменьшения размерности на этапе сжатия. Например, предположим, что ввод в модуль Fire был $55 \times 55 \times 96$. Затем мы могли бы применить сжатие с 16 фильтрами, уменьшив размер тома до $55 \times 55 \times 16$.

Затем расширение может увеличить количество изученных фильтров с 16 до 128 (64 фильтра для сверток 1×1 и 64 фильтра для сверток 3×3). Кроме того, с помощью множества фильтров 1×1

вместо фильтров 3×3 требуется меньше параметров. Этот процесс позволяет нам уменьшить размер пространственного тома по всей сети, сохраняя при этом относительно низкое количество изученных фильтров (по сравнению с другими сетевыми архитектурами, такими как AlexNet, VGGNet и т. д.). Мы обсудим второй метод сохранения больших параметров (без полносвязных слоев) в следующем разделе.

10.1.2 Архитектура SqueezeNet Всю

архитектуру SqueezeNet, включая размер выходного тома, можно найти на рис. 10.2.

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s _{1x1} (#1x1 squeeze)	e _{1x1} (#1x1 expand)	e _{3x3} (#3x3 expand)
input image	224x224x3					
conv1	111x111x96	7x7/2 (x96)	1			
maxpool1	55x55x96	3x3/2	0			
fire2	55x55x128		2	16	64	64
fire3	55x55x128		2	16	64	64
fire4	55x55x256		2	32	128	128
maxpool4	27x27x256	3x3/2	0			
fire5	27x27x256		2	32	128	128
fire6	27x27x384		2	48	192	192
fire7	27x27x384		2	48	192	192
fire8	27x27x512		2	64	256	256
maxpool8	13x12x512	3x3/2	0			
fire9	13x13x512		2	64	256	256
conv10	13x13x1000	1x1/1 (x1000)	1			
avgpool10	1x1x1000	13x13/1	0			

Рисунок 10.2: Архитектура SqueezeNet, воспроизведенная из таблицы 1 Iandola et al. [30]. Учитывая входные данные, мы применяем слой CONV с шагом 2×2 , за которым следует максимальное объединение, чтобы быстро уменьшить размер тома. Затем ряд пожарных модулей укладываются друг на друга. Столбец $s1 \times 1$ указывает количество сжатых фильтров, изученных на первом уровне CONV пожарного модуля. $e1 \times 1$ и $e3 \times 3$ указывают количество фильтров, изученных во втором наборе слоев CONV, соответственно.

Предполагается, что активации R ReLU применяются после каждой свертки и удаляются из этой таблицы, поскольку их использование подразумевается (и для экономии места, включая эту таблицу в книгу).

Интересно отметить, что в оригинале Iandola et al. публикации они сообщили, что входное изображение имеет пространственные размеры $224 \times 224 \times 3$; однако, если мы применим следующий расчет, используемый в Starter Bundle (Глава 11):

$$((224 - 7+2(0))/4) + 1 = 55,25 \quad (10.1)$$

Тогда мы знаем, что свертка 7×7 с шагом 2×2 никак не может подойти. Подобно путанице с AlexNet в главе 6, я предполагаю, что это опечатка в оригинальной работе. Использование входных данных размером $227 \times 227 \times 3$ пикселей позволяет разбивать свертки и обеспечивает точный размер выходного объема для всех других слоев, как подробно описано в оригинальном Iandola et al. публикация. Поэтому в этой книге мы будем предполагать, что входное изображение имеет размер $227 \times 227 \times 3$.

После подачи входного изображения в сеть изучаются 96 фильтров размером 7×7 каждый. Шаг 2×2 используется для уменьшения пространственных размеров с 227×227 до 111×111 . Для дальнейшего уменьшить пространственные размеры, максимальный пулинг применяется с размером пула 3×3 и шагом 2×2 , выходной объем $55 \times 55 \times 96$.

Далее применяем три огневых модуля. Первые два пожарных модуля используют 16 фильтров сжатия, уменьшение размера входного тома с $55 \times 55 \times 96$ до $55 \times 55 \times 16$. Затем применяется расширение 64 фильтра 1×1 и 64 фильтра 3×3 — эти 128 фильтров объединены по размеру канала где выходной пространственный размер $55 \times 55 \times 128$. Третий пожарный модуль увеличивает количество 1×1 сжимаем фильтры до 32, а количество 1×1 и 3×3 расширяем до 64 соответственно. До операции максимального объединения размер нашего пространственного измерения составляет $55 \times 55 \times 256$. Максимальный пул слой с размером пула 3×3 и шагом 2×2 уменьшает наши пространственные размеры до $27 \times 27 \times 256$.

Оттуда последовательно применяем четыре модели огня. В первом огневом модуле используется 32, 1×1 фильтры сжатия и 128 фильтров для расширения 1×1 и 3×3 соответственно.

Вторые два огневых модуля увеличивают количество выжимок 1×1 до 48, а также увеличивают 1×1 и 3×3 расширяются до 192, что приводит к размеру выходного тома $27 \times 257 \times 384$. последний пожарный модуль выполняет еще одно увеличение сжатия, на этот раз до 64 фильтров 1×1 . Количество Расширения 1×1 и 3×3 также увеличиваются до 256 соответственно. После объединения этих расширенных фильтров, размер нашего тома теперь $27 \times 27 \times 512$. Затем применяется еще одна операция максимального объединения, чтобы уменьшить наши пространственные размеры до $13 \times 13 \times 512$.

Применяется последний огневой модуль, увеличивающий сжимающие фильтры 1×1 до 64. Расширяющие фильтры затем увеличилось до 256 для 1×1 и 3×3 соответственно. Объединение по каналу размер, мы получаем размер выходного тома $13 \times 13 \times 512$.

Снова применяется последний модуль пожарной безопасности, идентичный пожарному 8 ранее в таблице. Отсев тогда применяется сразу после fire9 с вероятностью 50% — это сделано для уменьшения переобучения. окончательная свертка (conv10) состоит из фильтров 1×1 с N общими фильтрами, где N должно равняться общее количество меток классов — в случае ImageNet N = 1000.

Мы можем выполнить глобальное усреднение по всему объему 13×13 , чтобы уменьшить $13 \times 13 \times 1000$ до объема $1 \times 1 \times 1000$. Мы берем эти активации, а затем пропускаем их через softmax, чтобы получить наши окончательные выходные вероятности для каждой из меток класса N.

Как видите, в этой сетевой архитектуре не применяются полно связанные слои. Похожий для GoogLeNet и ResNet, использование глобального среднего пула может помочь уменьшить (а часто и устраниТЬ) полностью потребность в слоях FC. Удаление слоев FC также имеет дополнительное преимущество, заключающееся в уменьшении количества параметров, требуемых сетью, существенно. Имейте в виду, что все узлы в полно связанный слой плотно связан (т. е. каждый узел в текущем слое кодируется каждым другим узлом в последующем слое). Однако сверточные слои по определению разрежены и, следовательно, требуют меньше памяти. В любое время мы можем заменить набор полно связанных слоев сверткой и среднего пула мы можем значительно сократить количество параметров, требуемых нашей сетью.

10.1.3 Реализация SqueezeNet

Теперь, когда мы рассмотрели архитектуру SqueezeNet, давайте приступим к ее реализации. Создать новый файл с именем mxsqueezenet.py внутри подмодуля mxconv файла pyimagesearch.nn.conv:

```
-- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- нн
| | |--- __init__.py
| | |--- конв.
| | |--- mxconv
| | |--- __init__.py
```

```

    | | |--- mxalexnet.py
    | | |--- mxgooglenet.py
    | | |--- mxresnet.py
    | | |--- mxsqueezezenet.py
    | | |--- mxvggnet.py
    |--- предварительная обработка
    |--- утилиты

```

В этом файле будет жить наша реализация сети SqueezeNet. Откройте mxsqueezezenet.py и мы приступим к работе:

```

1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3
4 класс MxSqueezeNet:
    @статический метод
6     деф сжатие (ввод, numFilter):
7         # первая часть модуля FIRE состоит из числа 1x1
8         # фильтр сжимает входные данные с последующей активацией
9         сжатие_1x1 = mx.sym.Convolution (данные = ввод, ядро = (1, 1),
10             шаг=(1, 1), num_filter=numFilter)
11         act_1x1 = mx.sym.LeakyReLU(данные=squeeze_1x1,
12             act_type="элу")
13
14         # вернуть активацию для сжатия
15     вернуть акт_1x1

```

Строка 4 определяет наш класс MxSqueezezenet — все методы, отвечающие за построение и создание внутри этого класса будут существовать архитектура SqueezeNet. Затем в строке 6 определяется метод сжатия . который является частью модуля Fire, обсуждавшегося в предыдущем разделе. Функция сжатия требует вход (т. е. предыдущий слой) вместе с numFilter , число 1×1 фильтрует CONV слой должен учиться (строки 9 и 10).

Как только мы вычислим свертки 1×1 , мы передаем значения через функцию активации на Строки 11 и 12. В оригинале Iandola et al. публикации они использовали стандартную активацию ReLU функцию, но здесь мы собираемся использовать ELU. Как и предыдущие эксперименты с разными архитектурами показали, что замена ReLU на ELU может повысить точность классификации. Я оправдаю это требование в Разделе 10.4.4 ниже. Наконец, активации 1×1 возвращаются в вызывающую функцию. на линии 15.

Теперь, когда функция сжатия определена, давайте также создадим функцию огня:

```

17     @статический метод
18     защита огня (ввод, numSqueezeFilter , numExpandFilter):
19         # построить сжатие 1x1, за которым следует расширение 1x1
20         сжатие_1x1 = MxSqueezeNet.squeeze (ввод, numSqueezeFilter)
21         expand_1x1 = mx.sym.Свертка (данные = сжатие_1x1,
22             ядро = (1, 1), шаг = (1, 1), num_filter = numExpandFilter)
23         relu_expand_1x1 = mx.sym.LeakyReLU(данные=expand_1x1,
24             act_type="элу")

```

Метод огня требует трех параметров:

1. input: Входной слой для текущего модуля Fire.

2. numSqueezeFilter: количество фильтров сжатия для изучения.
3. numExpandFilter: количество расширяемых фильтров для изучения. Согласно Iandola et al. реализации, мы будем использовать одинаковое количество фильтров расширения как для 1×1 , так и для 3×3 свертки соответственно.

Строка 20 вызывает функцию сжатия, уменьшая наши входные пространственные размеры. Основанный на вне сжатия 1×1 мы можем вычислить расширение 1×1 в строках 21 и 22. Здесь укажите что вход в слой CONV должен быть выходом сжатия 1×1 , и что мы хотим узнать numExpandFilter, каждый размером 1×1 . ELU применяется к выходу свертки на Строки 23 и 24.

Мы выполняем аналогичную операцию для расширения 3×3 ниже:

```

26      # построить расширение 3x3
27      expand_3x3 = mx.sym.Convolution(данные=squeeze_1x1, pad=(1, 1),
28          ядро = (3, 3), шаг = (1, 1), num_filter = numExpandFilter)
29      relu_expand_3x3 = mx.sym.LeakyReLU(данные=expand_3x3,
30          act_type="элу")
31
32      # вывод модуля FIRE представляет собой конкатенацию
33      # активация для  $1 \times 1$  и  $3 \times 3$  расширяется по каналу
34      # измерение
35      вывод = mx.sym.Concat (relu_expand_1x1, relu_expand_3x3,
36          тусклый=1)
37
38      # вернуть вывод модуля FIRE
39      возвратный вывод

```

На этот раз мы изучаем numExpandFilter, каждый из которых имеет размер 3×3 , используя в качестве параметра slot_1x1 . вход. Мы применяем нулевое заполнение 1×1 , чтобы гарантировать, что пространственные размеры не уменьшаются, поэтому мы можем конкатенация вдоль измерения канала позже в этой функции. Затем ELU также применяется к выводу расширения_3x3. Когда у нас есть активация для расширений 1×1 и 3×3 , мы можем соединить их по измерению канала в строках 35 и 36. Это объединение возвращается к вызывающему методу в строке 39.

Чтобы лучше визуализировать пожарный модуль, обратитесь к рисунку 10.1 выше. Вот ты можно увидеть, что модуль Fire берет входной слой и применяет сжатие, состоящее из нескольких ядра 1×1 . Это сжатие затем разветвляется на две вилки. В одном форке мы вычисляем расширение 1×1 , в то время как вторая вилка вычисляет расширение 3×3 . Две вилки снова встречаются и соединяются вдоль измерения канала, чтобы служить конечным выходом модуля канала.

Давайте продолжим обсуждение реализации SqueezeNet, так как мы достигли всего важный метод сборки:

```

41      @статический метод
42      деф- сборка (классы):
43          # ввод данных
44          данные = mx.sym.Variable("данные")
45
46          # Блок №1: CONV => RELU => POOL
47          conv_1 = mx.sym.Convolution (данные = данные, ядро = (7, 7),
48              шаг=(2, 2), num_filter=96)
49          relu_1 = mx.sym.LeakyReLU(data=conv_1, act_type="elu")
50          pool_1 = mx.sym.Pooling (данные = relu_1, ядро = (3, 3),
51              шаг=(2, 2), pool_type="max")

```

Метод сборки принимает единственный параметр — общее количество классов , которые мы хотим изучить. В случае с ImageNet мы установим class=1000.

Затем в строке 44 создается экземпляр нашей переменной данных , входного пакета для нашей сети. Учитывая данные, мы определяем первый слой свертки в SqueezeNet. Этот слой изучает 96 фильтров, каждый из которых 7x7 с шагом 2x2 . Активация ELU применяется (строка 49) до того, как мы уменьшим пространственное размеры через максимальное объединение (строки 50 и 51).

Затем идут наши три огневых модуля, за которыми следует еще одна операция максимального объединения:

```

53     # Блок №2-4: (FIRE * 3) => POOL
54     fire_2 = MxSqueezeNet.fire(pool_1, numSqueezeFilter=16,
55         numExpandFilter=64)
56     fire_3 = MxSqueezeNet.fire(fire_2, numSqueezeFilter=16,
57         numExpandFilter=64)
58     fire_4 = MxSqueezeNet.fire(fire_3, numSqueezeFilter=32,
59         numExpandFilter=128)
60     pool_4 = mx.sym.Pooling(данные=fire_4, ядро=(3, 3),
61         шаг=(2, 2), pool_type="max")

```

Следуя рисунку 10.2 выше, мы применяем наши четыре огневых модуля, а затем еще один. максимальный пул:

```

63     # Блок №5-8: (FIRE * 4) => POOL
64     fire_5 = MxSqueezeNet.fire(pool_4, numSqueezeFilter=32,
65         numExpandFilter=128)
66     fire_6 = MxSqueezeNet.fire(fire_5, numSqueezeFilter=48,
67         numExpandFilter=192)
68     fire_7 = MxSqueezeNet.fire(fire_6, numSqueezeFilter=48,
69         numExpandFilter=192)
70     fire_8 = MxSqueezeNet.fire(fire_7, numSqueezeFilter=64,
71         numExpandFilter=256)
72     pool_8 = mx.sym.Pooling (данные = fire_8, ядро = (3, 3),
73         шаг=(2, 2), pool_type="max")

```

На данный момент мы находимся в самой глубокой части сети — нам нужно применить еще один пожарный модуль, отсев для уменьшения переоснащения, свертка для настройки размера тома до 13 x 13 x классов, и, наконец, глобальное усреднение для усреднения пространственных измерений 13 x 13 до 1 x 1 x классы:

```

75     # Блок #9-10: FIRE => DROPOUT => CONV => RELU => POOL
76     fire_9 = MxSqueezeNet.fire(pool_8, numSqueezeFilter=64,
77         numExpandFilter=256)
78     do_9 = mx.sym.Dropout (данные = fire_9, p = 0,5)
79     conv_10 = mx.sym.Convolution (данные = do_9, num_filter = классы,
80         ядро = (1, 1), шаг = (1, 1))
81     relu_10 = mx.sym.LeakyReLU(data=conv_10, act_type="elu")
82     pool_10 = mx.sym.Pooling (данные = relu_10, ядро = (13, 13),
83         тип пула = "среднее")

```

Наш последний блок кода обрабатывает выходные данные средней операции объединения и создает вывод softmax:

```

85         # классификатор softmax
86         сгладить = mx.sym.Flatten (данные = pool_10)
87         модель = mx.sym.SoftmaxOutput(data=flatten, name="softmax")
88
89         # вернуть архитектуру сети
90         модель возврата

```

Как видите, наша реализация соответствует сетевой архитектуре, показанной на рис. 10.2. над. Вопрос в том, можем ли мы обучить его на ImageNet? И насколько это будет сложно? Отвечать эти вопросы, давайте перейдем к следующему разделу.

10.2 Обучение SqueezeNet

Теперь, когда мы внедрили SqueezeNet, мы можем обучить его на наборе данных ImageNet. Но сначала давайте определить структуру проекта:

```

--- mx_imagenet_squeezezenet
| |--- конфигурация
| | |--- __init__.py
| | |--- imagenet_squeezezenet_config.py
| |--- вывод/
| |--- test_squeezezenet.py
| |--- train_squeezezenet.py

```

Структура проекта практически идентична всем другим проектам ImageNet, рассмотренным ранее в этой книге. Сценарий `train_squeezezenet.py` будет отвечать за обучение реальной сети. Ну тогда используйте `test_squeezezenet.py` для оценки обученной модели SqueezeNet в ImageNet. Наконец, Файл `imagenet_squeezezenet_config.py` содержит наши конфигурации для эксперимента.

При определении этой структуры проекта я скопировал весь каталог `mx_imagenet_alexnet`. и переименовал файлы, чтобы сказать сжатие сети вместо `alexnet`. Пути конфигурации ImageNet идентичны всем предыдущим экспериментам, поэтому давайте кратко рассмотрим `BATCH_SIZE` и Конфигурация `NUM_DEVICES`:

```

53 # определить размер пакета и количество устройств, используемых для обучения
54 ПАКЕТ_РАЗМЕР = 128
55 ЧИСЛО_УСТРОЙСТВ = 3

```

Здесь я указываю, что следует использовать `BATCH_SIZE`, равный 128, что удобно для моих 12 ГБ. Графический процессор Титан X. Если у вас GPU с меньшим объемом памяти, просто уменьшите размер пакета, чтобы сеть подходит для вашего графического процессора. Затем я использовал три графических процессора для обучения, так как торопился собрать результаты. для этой главы. Эту сеть также можно легко обучить с помощью одного или двух графических процессоров с небольшим объемом ресурсов. терпения.

Теперь, когда наш файл конфигурации обновлен, давайте также обновим `train_squeezezenet.py`. Только что как и во всех предыдущих главах ImageNet, скрипты `train_*`.py предназначены для использования в качестве основы, требуя от нас вносить как можно меньше изменений; поэтому я быстро просмотрю этот файл, выделю изменения и отложу тщательный обзор обучающих сценариев до главы 6. Продолжайте и откройте `train_squeezezenet.py`, и мы можем начать:

```

1 # импортируем необходимые пакеты
2 из конфига импортировать imagenet_squeezeconfig как конфиг
3 из pyimagesearch.nn.mxconv импортировать MxSqueezeNet
4 импортировать mxnet как mx
5 импортировать синтаксический анализ
6 журнал импорта
7 импортировать json
8 импорт OS

```

Строки 2-8 импортируют необходимые пакеты Python. Обратите внимание, как мы импортируем SqueezeNet файл конфигурации (строка 2) вместе с классом MxSqueezeNet (наша реализация SqueezeNet архитектура) на линии 3.

Давайте проанализируем наши аргументы командной строки и создадим наш файл журнала, чтобы мы могли регистрировать обучение. процесс к нему:

```

10 # построить разбор аргумента и разобрать аргументы
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-c", "--checkpoints", required=True,
13                 help="путь к выходному каталогу контрольных точек")
14 ap.add_argument("-p", "--prefix", required=True,
15                 help="имя префикса модели")
16 ap.add_argument("-s", "--start-epoch", type=int, default=0,
17                 help="эпоха для возобновления обучения")
18 аргументов = переменные (ap.parse_args())
19
20 # установить уровень логирования и выходной файл
21 logging.basicConfig(level=logging.DEBUG,
22                     имя_файла="training_{}.log".format(args["start_epoch"]),
23                     режим файла = "w")
24
25 # загрузите средства RGB для тренировочного набора, затем определите пакет
26 # размер
27 означает = json.loads(open(config.DATASET_MEAN).read())
28 batchSize = config.BATCH_SIZE * config.NUM_DEVICES

```

Это те же аргументы командной строки, что и в наших предыдущих экспериментах. Нам нужно укажите каталог --checkpoints для сериализации весов модели после каждой эпохи, --prefix (т. е. имя) модели и, необязательно, --start-epoch для возобновления обучения. Строки 21 и 22 динамически создавать файл журнала на основе начальной эпохи. Затем мы загружаем наши средства RGB в строке 27 , чтобы мы могли применить нормализацию среднего вычитания. BatchSize выводится на линии 28 на основе общего количества устройств, используемых для обучения SqueezeNet.

Далее создадим итератор обучающих данных:

```

30 # построить итератор обучающего изображения
31 trainIter = mx.io.ImageRecordIter(
32     path_imgrec=config.TRAIN_MX_REC,
33     data_shape=(3, 227, 227),
34     batch_size = размер партии,
35     rand_crop = Верно,
36     rand_mirror = Верно,
37     повернуть=15,
38     max_shear_ratio=0.1,

```

```

39     mean_r = означает ["R"],
40     mean_g = означает ["G"],
41     mean_b = означает ["B"],
42 preprocess_threads=config.NUM_DEVICES * 2)

```

А также итератор данных проверки:

```

44 # создать итератор проверочного изображения
45 valIter = mx.io.ImageRecordIter(
46     path_imgrec=config.VAL_MX_REC,
47     data_shape=(3, 227, 227),
48     batch_size = размер партии,
49     mean_r = означает ["R"],
50     mean_g = означает ["G"],
51     mean_b = означает ["B"])

```

SGD снова будет использоваться для обучения сети:

```

53 # инициализировать оптимизатор
54 opt = mx.optimizer.SGD(learning_rate=1e-2, импульс=0,9, wd=0,0002,
55     rescale_grad=1.0 / размер партии)

```

Яндола и др. рекомендовал скорость обучения 4×10^{-2} в своей оригинальной публикации; Однако, Я обнаружил, что эта скорость обучения слишком велика. Обучение было чрезвычайно изменчивым, и это сделало сеть трудно конвергентна; таким образом, во всех моих экспериментах с «хорошой» точностью я использовал начальная скорость обучения 1×10^{-2} . Импульсный термин 0,9 и день веса L2 0,0002 - это те, рекомендованный Iandola et al.

Теперь, когда наш оптимизатор инициализирован, мы можем создать checkpointsPath, каталог где мы будем хранить сериализованные веса после каждой эпохи:

```

57 # построить путь контрольных точек, инициализировать аргумент модели и
58 # вспомогательные параметры
59 checkpointsPath = os.path.sep.join([args["checkpoints"],
60     аргументы["предикс"]])
61 argParams = Нет
62 дополнительных параметра = нет

```

Далее мы можем справиться, если мы (1) обучаем SqueezeNet с самой первой эпохи или (2) перезапускаем обучение определенной эпохи:

```

64 # если не указана начальная эпоха конкретной модели, то
65 # инициализируем сеть
66 , если args["start_epoch"] <= 0:
67     # построить архитектуру LeNet
68     print("[INFO] строим сеть...")
69     модель = MxSqueezeNet.build(config.NUM_CLASSES)
70
71 # в противном случае указана конкретная контрольная точка
72 еще:
73     # загружаем чекпойнт с диска

```

```

74     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
75     модель = mx.model.FeedForward.load(путь к контрольным точкам,
76                                         аргументы["начало_эпохи"])
77
78     # обновить модель и параметры
79     argParams = модель.arg_params
80     auxParams = модель.aux_params
81     модель = модель.символ

```

Строки 66-69 обрабатывают, если мы обучаем SqueezeNet без предварительной контрольной точки. Если это действительно в этом случае мы создаем экземпляр MxSqueezeNet в строке 69 , используя предоставленное количество найденных меток классов. в нашем конфигурационном файле (1000 для ImageNet). В противном случае строки 71-81 предполагают, что мы загружаем контрольную точку с диска и перезапуск обучения с определенной эпохи.

Наконец, мы можем скомпилировать нашу модель:

```

83 # компилируем модель
84 модель = mx.model.FeedForward(
85     ctx=[mx.gpu(0), mx.gpu(1), mx.gpu(2)],
86     символ = модель,
87     инициализатор=mx.initializer.Xavier(),
88     arg_params = параметры аргумента,
89     aux_params = вспомогательные параметры,
90     оптимизатор=выбор,
91     число_эпох = 90,
92     begin_epoch=args["start_epoch"])

```

Здесь вы можете видеть, что я тренирую SqueezeNet с тремя графическими процессорами. SqueezeNet также может быть обучается на одном графическом процессоре (хотя, конечно, это займет больше времени). Не стесняйтесь ускоряться или замедляться сократить время обучения, выделяя больше/меньше графических процессоров для процесса обучения. Мы инициализируем взвешивает слои в сети с помощью инициализации Xavier (строка 87) и позволяет сети обучаться не более 90 эпох. Как покажут наши эксперименты, мы будем применять раннюю остановку в эпоха 80, чтобы уменьшить переоснащение.

В следующем блоке кода мы определяем наши обратные вызовы и метрики оценки:

```

94 # инициализировать обратные вызовы и метрики оценки
95 batchEndCBs = [mx.callback.Speedometer(batchSize, 250)]
96 epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)]
97 метрик = [mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5),
98            mx.metric.CrossEntropy()]

```

И, наконец, мы можем обучить нашу сеть:

```

100 # обучить сеть
101 print("[INFO] обучающая сеть...")
102 модель.подходит(
103     X=поезд,
104     eval_data=значение,
105     eval_metric = показатели,
106     batch_end_callback=batchEndCB,
107     epoch_end_callback=epochEndCBs)

```

Опять же, как и во всех предыдущих главах этой книги, посвященных ImageNet, наш `train_squeezezenet.py` скрипт почти идентичен всем другим сценариям `train_*.py` - единственная реальная разница заключается в:

1. Файл импорта конфигурации.
2. Файл импорта модели.
3. Реализация модели.
4. Обновления оптимизатора SGD.
5. Любые весовые инициализации.

Используя этот шаблон, вы можете легко создать собственный сценарий обучения за считанные минуты, при условии, что вы уже закодировали свою сетевую архитектуру.

10.3 Оценка SqueezeNet

Чтобы оценить SqueezeNet, мы будем использовать скрипт `test_squeezezenet.py`, упомянутый в структуре нашего проекта выше. Опять же, как и во всех других экспериментах ImageNet в книге, этот сценарий идентичен `test_alexnet.py` и всем другим сценариям `test_*.py`, используемым для оценки данной сетевой архитектуры. Поскольку эти скрипты идентичны, я не буду рассматривать здесь `test_squeezezenet.py`.

Пожалуйста, обратитесь к Главе 6 на сайте `test_alexnet.py` для подробного ознакомления с кодом.

Кроме того, вы можете использовать часть этой книги, посвященную загрузкам, для изучения проекта и просмотра содержимого файла `test_squeezezenet.py`. Опять же, содержимое этих файлов идентично, поскольку они являются частью нашей структуры для обучения и оценки CNN, обученных в ImageNet. Я просто предоставил здесь отдельный скрипт `test_squeezezenet.py` на тот случай, если вы захотите выполнить свои собственные модификации.

10.4 Эксперименты SqueezeNet

В 2016 году Яндоля и соавт. заявил, что SqueezeNet может получить точность уровня AlexNet с в 50 раз меньшим количеством параметров. Я бы определил точность AlexNet в диапазоне 55-58% точности ранга 1. Таким образом, если наша версия SqueezeNet сможет попасть в этот диапазон точности ранга 1, я назову наше воспроизведение эксперимента успешным.

Как и в предыдущих главах, я объясню свои первоначальные базовые эксперименты, расскажу, что сработало (а что нет) и как я изменил какие-либо параметры перед следующим экспериментом. Всего мне потребовалось четыре эксперимента, чтобы воспроизвести результаты Iandola et al. Вы можете найти результаты моих экспериментов ниже — используйте эти результаты, чтобы помочь вам при проведении собственных экспериментов.

Важен не только конечный результат, важен научный метод разработки эксперимента, его проведения, сбора результатов и настройки параметров. Чтобы стать настоящим экспертом по глубокому обучению, вам нужно изучить путь, ведущий к оптимальной CNN, а не только конечный результат.

Чтение тематических исследований, подобных этим, поможет вам развить этот навык.

10.4.1 SqueezeNet: эксперимент №1

В своем первом эксперименте со SqueezeNet я попытался воспроизвести работу Яндолы и др. используя их точную архитектуру и параметры оптимизатора. Вместо использования ELU, как сообщалось в разделе реализации выше, я использовал ReLU. Я также использовал начальную скорость обучения $4e-2$, как рекомендовано в исходной статье. Чтобы начать процесс обучения, я выполнил следующую команду:

```
$ python train_squeezezenet.py --checkpoints чекпоинты --prefix squeezezenet
```

Я внимательно следил за своим тренировочным процессом, но заметил значительную нестабильность в тренировочном процессе, особенно в эпохи 15 и в эпоху начала 20-х годов (рис. 10.3, вверху слева). В последнем случае точность резко упала до 0,9%, но снова смогла восстановиться. После эпохи 25 я решил снизить скорость обучения с $4e-2$ до $4e-3$, чтобы посмотреть, поможет ли более низкая скорость обучения стабилизировать сеть:

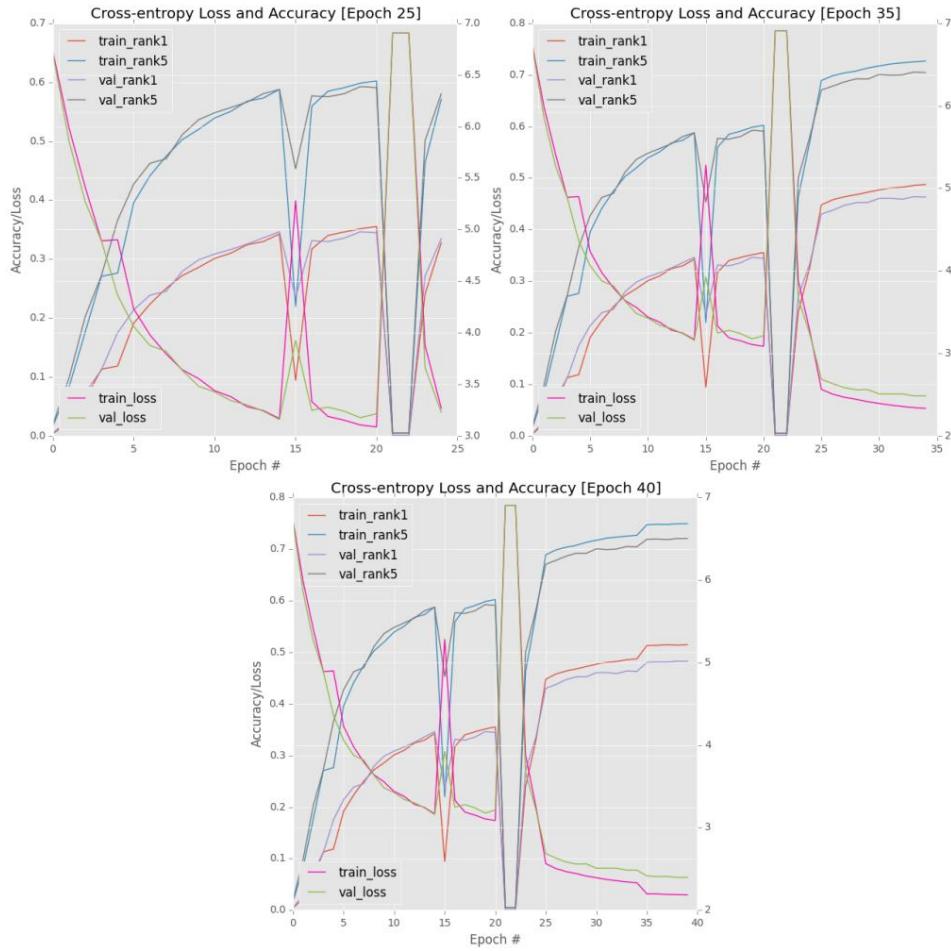


Рисунок 10.3: Верхний левый: Обучение SqueezeNet на ImageNet с начальной скоростью обучения $4e-2$ (предложено Яндой и др. [30]) демонстрирует значительную волатильность. Вверху справа: снижение скорости обучения с $4e-2$ до $4e-3$ помогает снизить волатильность; однако обучение начинает застаиваться. Внизу: изменение a до $4e-4$ приводит к полной стагнации.

```
$ python train_squeeze.py --checkpoints контрольные точки --префикс squeeze \n
--начало эпохи 25
```

Снижение скорости обучения помогло стабилизировать кривую обучения; однако потери при проверке начали снижаться гораздо медленнее (рис. 10.3, вверху справа). В эпоху 35 я прекратил обучение и изменил скорость обучения с $4e-3$ до $4e-4$, в основном просто для того, чтобы подтвердить свою интуицию, что это изменение а полностью застопорит обучение:

```
$ python train_squeeze.py --checkpoints контрольные точки --префикс squeeze \n
--начало эпохи 35
```

В данном случае моя интуиция была верна — снижение скорости обучения привело к полной стагнации результатов (рис. 10.3, внизу). В эпоху 40 я прекратил эксперимент. Учитывая как волатильность, так и стагнацию, было ясно, что начальную скорость обучения необходимо скорректировать (и, вероятно, несколько других гиперпараметров).

Однако этот эксперимент был не напрасным. Самое главное, я смог получить базовую точность. После эпохи 40 SqueezeNet достигал точности 48,93% ранга 1 и 72,07% ранга 5 на проверочном наборе. Это далеко от точности уровня AlexNet, поэтому нам явно есть над чем поработать.

10.4.2 SqueezeNet: эксперимент №2

Учитывая, что обучение SqueezeNet со скоростью обучения $4e-2$ было настолько нестабильным, я решил уменьшить скорость обучения до более стандартного $1e-2$. Я также сохранил активации ReLU и не заменил их на ELU (это будет позже). Я начал тренировать SqueezeNet с помощью следующей команды:

```
$ python train_squeeze.py --checkpoints чекпоинты --prefix squeeze
```

Затем я позволил SqueezeNet обучаться в течение 50 эпох со скоростью обучения $1e-2$ (рис. 10.4, вверху слева).

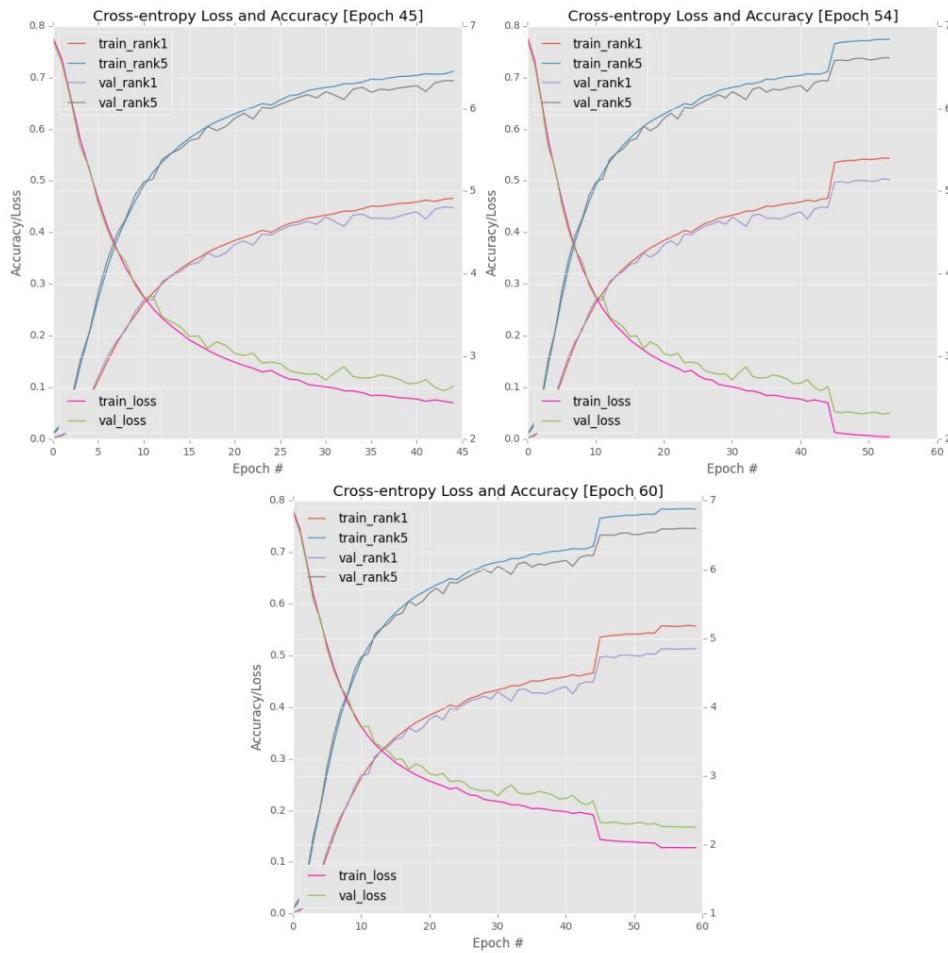


Рисунок 10.4: Верхний левый: Обучение SqueezeNet с начальным значением $\alpha = 1e-2$ стабильно, но медленно. Вверху справа: снижение скорости обучения до $1e-3$ приводит к увеличению точности/снижению потерь; однако обучение быстро застывает. Внизу: обучение полностью застопорилось с $\alpha = 1e-4$.

Из этого сюжета можно сделать два ключевых вывода:

1. Обучение проходит гораздо более плавно и менее изменчиво. SqueezeNet не выпадает случайным образом до 0% точности классификации.
2. Точно так же кривые обучения и проверки точно имитируют друг друга почти до эпохи 30, где мы видим расхождение.

Однако после эпохи 50 расхождение продолжается, и наблюдается застой в потерях/точности проверки. Из-за этого застоя я решил выйти из эксперимента `ctrl + c` и перезапустить обучение с эпохи 45 с более низкой скоростью обучения `1e - 3`:

```
$ python train_squeeze.py --checkpoints контрольные точки --prefix squeeze \n
--начало эпохи 45
```

Как видите, потери немедленно падают, а точность возрастает, поскольку оптимизатор попадает в более низкие области потерь в ландшафте потерь (рис. 10.4, вверху справа). Однако это увеличение точности быстро стагнирует, поэтому я снова выхожу из эксперимента, нажав `Ctrl + C`, и перезапускаю обучение в эпоху 55 со скоростью обучения `1e - 4`:

```
$ python train_squeeze.py --checkpoints контрольные точки --prefix squeeze \n
--начало эпохи 55
```

Обучение продолжается еще пять эпох, прежде чем я официально останавливаю эксперимент, так как больше никаких улучшений не будет (рис. 10.4, внизу). Глядя на мои данные проверки, я обнаружил, что SqueezeNet теперь получает 51,30% точности ранга 1 и 74,60% точности ранга 5 в наборе проверки, что является заметным улучшением по сравнению с предыдущим экспериментом; однако мы все еще не на уровне AlexNet точность.

10.4.3 SqueezeNet: эксперимент №3

Мой третий эксперимент — отличный пример неудачного эксперимента, который хорошо звучит в вашей голове, но когда вы его применяете на практике, он с треском проваливается. Учитывая, что микроархитектуры, такие как GoogLeNet и ResNet, выигрывают от наличия слоев пакетной нормализации, я решил обновить архитектуру SqueezeNet, чтобы включить пакетную нормализацию после каждого ReLU, пример которой можно увидеть ниже:

```
46      # Блок №1: CONV => RELU => POOL
47      conv_1 = mx.sym.Convolution(данные = данные, ядро = (7, 7), шаг
48          = (2, 2), num_filter = 96)
49      relu_1 = mx.sym.Activation(data=conv_1, act_type="relu") bn_1 =
50          mx.sym.BatchNorm(data=relu_1) pool_1 = mx.sym.Pooling(data=bn_1,
51          kernel=(3, 3), шаг=(2, 2), pool_type="max")
```

Затем я снова начал тренироваться:

```
$ python train_squeeze.py --checkpoints чекпоинты --prefix squeeze
```

Потребовалось менее 20 эпох, чтобы понять, что этот эксперимент не увенчается успехом (рис. 10.5). В то время как SqueezeNet начиналась с более высокой начальной точностью в более ранние эпохи (около 10 % ранга 1 и 20% ранга 5), эти преимущества были быстро потеряны. Сеть медленно обучалась

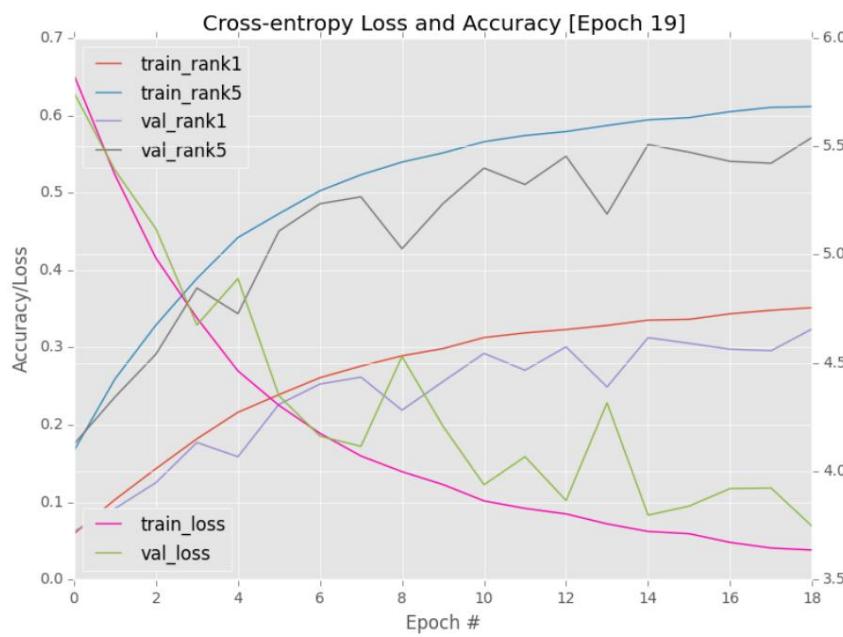


Рисунок 10.5: Добавление слоев пакетной нормализации в SqueezeNet было полным провалом, и я быстро убил эксперимент.

и, кроме того, данные проверки сообщают о волатильности. Основываясь на эксперименте № 2, я ожидал, что иметь > 40% точности ранга 1 к тому времени, когда я достиг эпохи 20 — как показано на рисунке 10.5 выше, этого результата не должно было случиться.

Я прервал этот эксперимент и отметил в своем лабораторном журнале, что слои пакетной нормализации не помогают архитектуре SqueezeNet, что меня очень удивило. Затем я удалил всю партию слоев нормализации из SqueezeNet и перешел к моему следующему эксперименту.

10.4.4 SqueezeNet: Эксперимент №4

На данный момент мой лучший результат был получен в эксперименте № 2 (51,30% ранга 1 и 74,60% ранга 5). но я не смог пробиться к точности уровня AlexNet с точностью 55-58%. В попытке Достигнув этого момента, я решил выполнить свой старый трюк с заменой ReLU на ELU:

```

46      # Блок №1: CONV => RELU => POOL
47      conv_1 = mx.sym.Convolution(данные = данные, ядро = (7, 7),
48                                  шаг=(2, 2), num_filter=96)
49      relu_1 = mx.sym.LeakyReLU(data=conv_1, act_type="elu")
50      pool_1 = mx.sym.Pooling(данные = relu_1, ядро = (3, 3),
51                               шаг=(2, 2), pool_type="max")

```

Затем обучение было запущено с помощью следующей команды:

```
$ python train_squeeze.py --checkpoints чекпоинты --prefix squeeze
```

Поначалу меня немного беспокоила кривая обучения — замена ReLU на ELU. ожидается чуть более 40% точности ранга 1 к эпохе 20; тем не менее, я позволяю тренировкам продолжаться

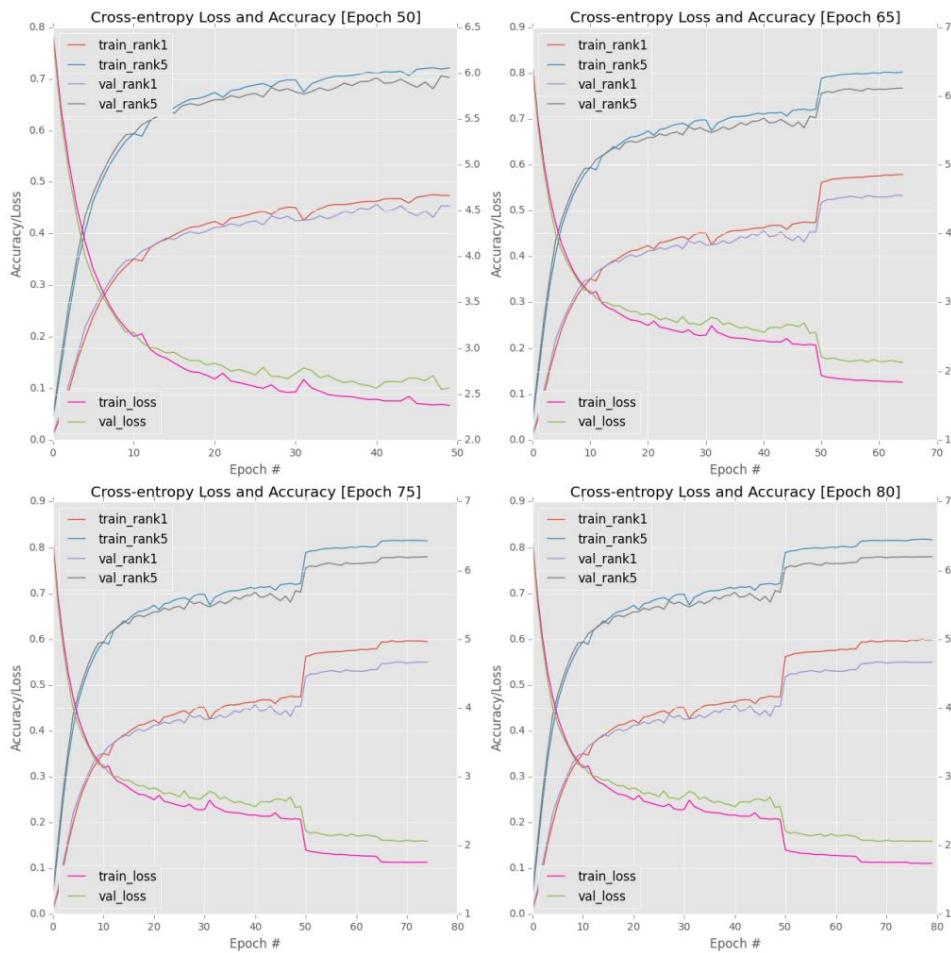


Рисунок 10.6: Вверху слева: обучение SqueezeNet на ImageNet с ELU вместо ReLU при $\alpha = 1e - 2$. Вверху справа: снижение α до $1e - 3$ дает нам хороший прирост точности. Внизу слева: еще один небольшой импульс достигается при $\alpha = 1e - 4$. Внизу справа: обучение по существу застопорилось, как только мы достигли $\alpha = 1e - 5$.

обучение было достаточно стабильным. Это оказалось хорошим решением, поскольку оно позволило SqueezeNet обучаться дольше без необходимости обновлять скорость обучения (рис. 10.6, вверху слева).

К тому времени, когда я достиг эпохи 50, я начал замечать стагнацию в точности/прогрыше проверки (и даже несколько сопутствующих провалов), поэтому я прекратил обучение, снизил скорость обучения с $1e-2$ до $1e-3$, а затем продолжил обучение . :

```
$ python train_squeeze.py --checkpoints контрольные точки --префикс squeeze \n
--начало эпохи 50
```

Как мы видим, скачок в точности валидации (и снижение потерь при валидации) весьма драматичен, более выражен, чем во втором эксперименте — один только этот скачок позволил мне превысить отметку ранга 1 в 51 % (рис. 10.6, вверху справа). В этот момент я был уверен, что использование ELU было хорошим выбором. Я позволил обучению продолжаться в течение 15 эпох, после чего снова остановил обучение, уменьшил скорость обучения с $1e-3$ до $1e-4$ и возобновил обучение:

```
$ python train_squeeze.py --checkpoints контрольные точки --префикс squeeze \n
--начало эпохи 65
```

Опять же, мы видим еще один скачок в точности проверки и снижение потерь при проверке, хотя и менее выраженное, чем в предыдущий раз (рис. 10.6, внизу слева). SqueezeNet достигал точности классификации 54% после эпохи 75, но наблюдался определенный застой. Я решил попробовать скорость обучения $1e-5$ для пяти эпох, просто чтобы посмотреть, имеет ли это значение, но на данный момент скорость обучения была слишком низкой, чтобы оптимизатор мог найти более низкие области потерь в ландшафте потерь:

```
$ python train_squeeze.py --checkpoints контрольные точки --префикс squeeze \n
--начало эпохи 75
```

В конце эпохи 80 я полностью прекратил обучение и оценил SqueezeNet (рис. 10.6, внизу справа). На проверочном наборе эта итерация SqueezeNet достигла точности 54,49% ранга 1 и 77,98% ранга 5, что очень близко к уровням AlexNet. Учитывая этот обнадеживающий результат, я переключился на тестовый набор и оценил:

```
$ python test_squeeze.py --checkpoints контрольные точки --префикс squeeze \n
--epoch 80
[INFO] загрузка модели...
[INFO] прогнозирование на основе тестовых данных...
[ИНФО] ранг-1: 57,01%
[ИНФО] ранг-5: 79,45%
```

Как показывают результаты, SqueezeNet удалось получить 57,01% ранга 1 и 79,45% ранга 5. Точность находится в пределах результатов уровня AlexNet.

Благодаря этому эксперименту мне удалось успешно воспроизвести результаты Яндолы и др., хотя я должен признать, что большая часть этого успеха зависела от замены их первоначальной скорости обучения $4e-2$ на $1e-2$. Проведя небольшое онлайн-исследование в отношении опыта других практиков глубокого обучения при обучении SqueezeNet с нуля, большинство других сообщают, что $4e-2$ было слишком большим, иногда им приходилось снижать скорость обучения до $1e-3$, чтобы получить какую-либо тягу.

Подобные эксперименты показывают, насколько сложным может быть повторение результатов опубликованной работы. В зависимости от используемой вами библиотеки глубокого обучения и версий CUDA/cuDNN, работающих в вашей системе, вы можете получить разные результаты, даже если ваша реализация идентична реализации авторов. Кроме того, некоторые библиотеки глубокого обучения реализуют слои по-разному, поэтому могут быть базовые параметры, о которых вы не знаете.

Прекрасным примером этого является уровень активации

ELU: • В Keras активация ELU использует значение α по умолчанию, равное 1,0. • Но в mxnet значение ELU α по умолчанию равно 0,25.

Это тонко, но небольшие вариации, подобные этим, действительно могут складываться. Вместо того, чтобы испытывать недовольство и разочарование из-за того, что ваша реализация данной сети не достигает точных результатов, о которых сообщил автор, сначала проведите небольшое исследование. Посмотрите, сможете ли вы определить, какие библиотеки и какие предположения они сделали при обучении своей сети. Оттуда начните экспериментировать. Отмечайте, что работает, а что нет, и обязательно записывайте результаты в журнал! После завершения эксперимента просмотрите результаты еще раз и изучите, что можно изменить для дальнейшего повышения точности. В большинстве случаев это будет ваша скорость обучения, регуляризация, инициализация веса и функции активации.

10.5 Резюме

В этой главе мы узнали, как обучить архитектуру SqueezeNet с нуля на наборе данных ImageNet. Кроме того, мы смогли успешно воспроизвести результаты Iandola et al., успешно получив точность на уровне AlexNet — 57,01% ранга 1 и 79,45% ранга 5 на тестовом наборе соответственно. Чтобы достичь этого результата, нам нужно было уменьшить скорость обучения $4e^{-2}$, первоначально предложенную Iandola et al. и используйте меньшую скорость обучения $1e^{-2}$. Эта меньшая скорость обучения помогла снизить волатильность модели и, в конечном итоге, получить более высокую классификацию.

Во-вторых, замена ReLU на ELU позволила нам дольше обучать сеть с меньшими задержками. В конечном итоге именно обмен функциями активации позволил нам получить результаты уровня AlexNet.

Как я упоминал в предыдущих главах, всегда начинайте со стандартной функции активации ReLU, чтобы (1) убедиться, что вашу модель можно правильно обучить и (2) получить базовый уровень. Затем вернитесь и настройте другие гиперпараметры сети, включая планирование скорости обучения, регуляризацию/уменьшение веса, инициализацию веса и даже нормализацию партии. После того, как вы максимально оптимизировали свою сеть, замените ReLU на ELU, чтобы (обычно) получить небольшое повышение точности классификации. Единственное исключение из этого правила — если вы тренируете очень глубокую CNN и требуете использования MSRA/He et al. инициализация — в этом случае, как только вы перейдете к инициализации MSRA, вам также следует подумать о замене стандартных ReLU на PReLU.

В этой главе мы завершаем обсуждение обучения современных сверточных нейронных сетей работе со сложным набором данных ImageNet. Обязательно попрактикуйтесь и поэкспериментируйте с этими сетями как можно больше. Создавайте собственные эксперименты и проводите эксперименты, не описанные в этой книге. Изучите различные гиперпараметры и их влияние на обучение данной сетевой архитектуры.

В этой книге я объединил свои эксперименты с ImageNet, чтобы их было легко читать и усваивать. В некоторых случаях мне нужно было провести более 20 экспериментов (ярким примером является AlexNet), прежде чем я получил наилучшие результаты. Затем я выбрал наиболее важные эксперименты, чтобы включить их в эту книгу, чтобы продемонстрировать правильные (и неправильные) повороты, которые я сделал.

То же самое будет верно и для вас. Не расстраивайтесь, если многие из ваших экспериментов не дают сразу же отличных результатов, даже если вы чувствуете, что применяете ту же архитектуру и параметры, что и первоначальные авторы — эти пробы и ошибки являются частью процесса.

Эксперименты по глубокому обучению повторяются. Разработайте гипотезу. Проведите эксперимент. Оценивать результаты. Настройте параметры. И проведите еще один эксперимент.

Имейте в виду, что ведущие мировые эксперты по глубокому обучению проводят от десятков до сотен экспериментов в каждом проекте — то же самое будет верно и для вас, когда вы развиваете свою способность обучать глубокие нейронные сети на сложных наборах данных. Этот навык находится на стыке «науки» и «искусства», и единственный способ получить этот столь необходимый навык — это практика.

В оставшихся главах пакета ImageNet мы рассмотрим различные тематические исследования, включая распознавание эмоций/выражений лица, прогнозирование и исправление ориентации изображения, классификацию марки и модели транспортного средства, а также прогнозирование возраста и пола человека. Человек в видео. Эти тематические исследования помогут вам еще больше улучшить свои навыки в качестве специалиста по глубокому обучению.

11. Практический пример: распознавание эмоций

В этой главе мы собираемся решить задачу Kaggle по распознаванию выражений лица [31] и претендовать на пятерку лучших в таблице лидеров. Для выполнения этой задачи мы собираемся обучить VGG-подобную сеть с нуля на обучающих данных, принимая во внимание, что наша сеть должна быть достаточно маленькой и достаточно быстрой, чтобы работать в режиме реального времени на нашем ЦП. Как только наша сеть будет обучена (и мы зайдем свое место в пятерке лучших в таблице лидеров), мы напишем некоторый код для загрузки сериализованной модели с диска и применим ее к видеопотоку для обнаружения эмоций в режиме реального времени.

Однако, прежде чем мы начнем, важно понять, что наши эмоции, как люди, находятся в постоянном текучем состоянии. Мы никогда не бываем на 100% счастливы или на 100% грустны. Вместо этого наши эмоции смешиваются. Испытывая «сюрприз», мы также можем испытывать «счастье» (например, сюрприз на вечеринке по случаю дня рождения) или «испуг» (если сюрприз не является желанным). И даже во время «испуганной» эмоции мы также можем чувствовать намеки на «гнев».

При изучении распознавания эмоций важно не сосредотачиваться на одной метке класса (как мы иногда делаем в других задачах классификации). Вместо этого нам гораздо выгоднее посмотреть на вероятность каждой эмоции и охарактеризовать распределение. Как мы увидим позже в этой главе, изучение распределения вероятностей эмоций дает нам более точную меру предсказания эмоций, чем просто выбор одной эмоции с наибольшей вероятностью.

11.1 Задача распознавания выражений лица Kaggle

Набор обучающих данных Kaggle Emotion and Facial Expression Recognition состоит из 28 709 изображений, каждое из которых представляет собой изображение в градациях серого 48×48 (рис. 11.1). Лица были автоматически выровнены таким образом, чтобы они были примерно одинакового размера на каждом изображении. Учитывая эти изображения, наша цель состоит в том, чтобы разделить эмоции, выраженные на каждом лице, на семь различных классов: злость, отвращение, страх, радость, грусть, удивление и нейтральность.

11.1.1 Набор данных FER13

Набор данных, использованный в конкурсе Kaggle, был агрегирован Goodfellow et al. в своей статье 2013 года «Проблемы репрезентативного обучения: отчет о трех конкурсах по машинному обучению» [32].



Рисунок 11.1: Образец выражений лица в Kaggle: Facial Expression Recognition Challenge. Мы обучим CNN распознавать и идентифицировать каждую из этих эмоций. Эта CNN также сможет работать в режиме реального времени на вашем процессоре, что позволит вам распознавать эмоции в видеопотоках.

Этот набор данных выражений лица называется набором данных FER13, его можно найти на официальной странице соревнований Kaggle и загрузить файл fer2013.tar.gz: <http://pyimg.co/a2soy>. Архив набора данных .tar.gz весит 92 МБ, поэтому перед загрузкой убедитесь, что у вас хорошее подключение к Интернету. После загрузки набора данных вы найдете файл с именем fer2013.csv с тремя столбцами:

- эмоция: метка класса.
- пиксели: сглаженный список $48 \times 48 = 2304$ пикселя в градациях серого, представляющий само лицо.
- использование: предназначено ли изображение для обучения, частного тестирования (проверки) или общедоступного тестирования (тестирование).

Теперь наша цель — взять этот файл .csv и преобразовать его в формат HDF5, чтобы нам было проще обучать на его основе сверточную нейронную сеть.

После того, как я распаковал файл fer2013.tar.gz, я установил следующую структуру каталогов для проекта:

```
--- fer2013
    |--- fer2013 |---
    |   |--- hdf5 |--- вывод
```

В каталог fer2013 я включил содержимое разархивированного файла fer2013.tar.gz (то есть сам набор данных). Каталог hdf5 будет содержать сплиты для обучения, тестирования и проверки. Наконец, мы сохраним все журналы/графики обучения в выходной каталог.

11.1.2 Создание набора данных FER13

Давайте продолжим и настроим нашу структуру каталогов для этого проекта распознавания эмоций:

```

--- эмоция_распознавание
| |--- конфигурация
| | |--- __init__.py
| | |--- эмоция_config.py
| |--- build_dataset.py
| |--- эмоция_detector.py
| |--- haarcascade_frontface_default.xml
| |--- test_recognizer.py
| |--- train_recognizer.py

```

Как и в предыдущих экспериментах, мы создадим каталог с именем config и превратим его в файл Python. подмодуль, поместив в него поле __init__.py . Внутри конфига мы создадим файл с именем эмоция_config.py — в этом файле мы будем хранить любые переменные конфигурации, в том числе пути к входному набору данных, выходным файлам HDF5 и размерам пакетов.

build_dataset.py будет отвечать за прием файла набора данных fer2013.csv и вывод набора файлов HDF5; по одному для каждого из разделов обучения, проверки и тестирования, соответственно. Мы обучим нашу CNN распознавать различные эмоции, используя train_recognizer.py.

Точно так же сценарий test_recognizer.py будет использоваться для оценки производительности CNN.

Как только мы будем довольны точностью нашей модели, мы перейдем к реализации Скрипта эмоций_detector.py для:

1. Обнаружение лиц в режиме реального времени (как в главе о детекторе улыбки в Starter Bundle).
2. Примените нашу CNN, чтобы распознать наиболее доминирующую эмоцию и отобразить распределение вероятности. для каждой эмоции.

Самое главное, эта CNN сможет запускать и определять выражения лиц в режиме реального времени на нашем ПРОЦЕССОР. Давайте продолжим и просмотрим файл эмоция_config.py сейчас:

```

1 # импортируем необходимые пакеты
2 из пути импорта ОС
3
4 # определить базовый путь к набору данных эмоций
5 BASE_PATH = "/raid/наборы данных/fer2013"
6
7 # используйте базовый путь для определения пути к входному файлу эмоций
8 INPUT_PATH = path.sep.join([BASE_PATH, "fer2013/fer2013.csv"])

```

В строке 5 мы определяем BASE_PATH для нашего корневого каталога проекта. Это место, где входной набор данных FER13 будет жить вместе с выходными файлами набора данных HDF5, журналами и графиками. Стока 8 затем определяет INPUT_PATH для самого файла fer2013.csv.

Давайте также определим количество классов в наборе данных FER13:

```

10 # определить количество классов (установите 6, если вы игнорируете
11 # класс "отвращение")
12 # NUM_CLASSES = 7
13 NUM_CLASSES = 6

```

Всего в FER13 семь классов: злость, отвращение, страх, радость, грусть, удивление и нейтральный. Тем не менее, существует сильный дисбаланс классов с классом «отвращение», так как он имеет только 113 изображений. образцы (остальные имеют более 1000 изображений на класс). Проведя некоторое исследование, я наткнулся на проект Mememoji [33], который предлагает объединить «отвращение» и «гнев» в один класс

(поскольку эмоции визуально похожи), тем самым превращая FER13 в задачу 6 класса. В этой главе мы исследуем распознавание эмоций, используя как шесть, так и семь классов; однако, когда придет время развернуть и использовать нашу CNN для классификации эмоций, мы уменьшим количество эмоций до шести, чтобы улучшить результаты.

Поскольку мы будем преобразовывать файл fer2013.csv в серию наборов данных HDF5 для обучения, проверки и тестирования нам нужно определить пути к этим выходным файлам HDF5:

```
15 # определить путь к выходным файлам обучения, проверки и тестирования 16 #
файлы HDF5 17 TRAIN_HDF5 = path.sep.join([BASE_PATH, "hdf5/train.hdf5"])
18 VAL_HDF5 = path.sep.join([BASE_PATH, "hdf5/val.hdf5"])
19 TEST_HDF5 = path.sep.join([BASE_PATH, "hdf5/test.hdf5"])
```

Наконец, мы инициализируем размер пакета при обучении нашей CNN вместе с выходным каталогом, где будут храниться любые журналы или графики:

```
21 # определить размер партии
22 ПАКЕТ_РАЗМЕР = 128
23
24 # определяем путь, где будут храниться выходные журналы 25
OUTPUT_PATH = path.sep.join([BASE_PATH, "output"])
```

Теперь, когда наш файл конфигурации создан, мы можем создать набор данных:

```
1 # импортировать необходимые пакеты 2
из конфига импортировать motion_config как конфиг 3 из
pyimagesearch.io import HDF5DatasetWriter 4 импортировать
numpy как np
5
6 # открыть входной файл для чтения (пропустив заголовок), затем 7 #
инициализировать список данных и меток для обучения, 8 # проверки и тестовых
наборов 9 print("[INFO] loading input data...") 10 f = open(config.INPUT_PATH) 11
f.__next__() # f.next() для Python 2.7 12 (trainImages, trainLabels) = ([], [])
13 (valImages,
valLabels) = ([], [])
14 (testImages, testLabels) = ([], [])
```

В строке 2 мы импортируем наш только что созданный файл motion_config . Затем строка 3 импортирует наш HDF5DatasetWriter , который мы много раз использовали в этой книге для преобразования входного набора изображений в набор данных HDF5.

Строка 10 открывает указатель на входной файл fer2013.csv . Вызывая метод .next указателя файла, мы можем перейти к следующей строке, что позволит нам пропустить заголовок CSV-файла. Строки 12-14 затем инициализируют списки изображений и меток для обучающего, проверочного и тестового наборов соответственно.

Теперь мы готовы начать строить наши разбиения данных:

```
16 # цикл по строкам во входном файле 17 для строки
в f:
18         # извлечь метку, изображение и использование из строки
```

```

19     (метка, изображение, использование) = row.strip().split(",")
20     метка = интервал(метка)
21
22     # если мы игнорируем класс "отвращение", всего будет 6
23     # метки классов вместо 7
24     если config.NUM_CLASSES == 6:
25         # объединить классы "гнев" и "отвращение"
26         если метка == 1:
27             метка = 0
28
29         # если метка имеет значение больше нуля, вычесть единицу из
30         # сделать все метки последовательными (не обязательно, но помогает
31         # при интерпретации результатов)
32         если метка > 0:
33             метка -= 1

```

В строке 17 мы начинаем перебирать каждую строку входного файла. Стока 19 занимает строку и разбивает его на 3 кортежа: метка изображения, необработанные интенсивности пикселей изображения, а также использование (т. е. обучение, тестирование или проверка). По умолчанию мы предполагаем, что рассматриваем FER13 как задача классификации 7 классов; однако, в случае, если мы хотим слить гнев и отвращение вместе (строка 24), нам нужно изменить метку отвращения с 1 на 0 (строки 26 и 27). Хорошо также вычтите значение 1 из каждой метки в строках 32 и 33, чтобы гарантировать, что каждая метка класса последовательный — это вычитание не требуется, но помогает при интерпретации наших результатов.

На данный момент наше изображение представляет собой просто строку целых чисел. Нам нужно взять эту строку, разбить ее на список, преобразовать его в беззнаковый 8-битный целочисленный тип данных и преобразовать его в изображение в градациях серого 48×48:

```

35     # преобразовать сглаженный список пикселей в 48x48 (оттенки серого)
36     # изображение
37     изображение = np.array(image.split(" "), dtype="uint8")
38     изображение = изображение.изменить((48, 48))

```

Имейте в виду, что каждый столбец изображения представляет собой список из 2304 целых чисел. Эти 2304 целых числа представляют квадратное изображение 48×48. Мы можем выполнить это преобразование в строке 38.

Последним шагом в анализе файла fer2013.csv является простая проверка использования и назначение изображения и метки в соответствующие списки обучения, проверки или тестирования:

```

40     # проверяем, проверяется ли мы тренировочное изображение
41     если использование == "Обучение":
42         trainImages.append(изображение)
43         trainLabels.append(метка)
44
45     # проверяем, является ли это проверочным изображением
46     использование elif == "PrivateTest":
47         valImages.append(изображение)
48         valLabels.append(метка)
49
50     # в противном случае это должно быть тестовое изображение
51     еще:
52         testImages.append(изображение)
53         testLabels.append(метка)

```

Отсюда наш код создания набора данных начинает становиться более знакомым:

```

55 # составить список, объединяющий обучение, проверку и тестирование
56 # изображений вместе с соответствующими им метками и выводом HDF5
57 # файлов
58 наборов данных =
59     (trainImages, trainLabels, config.TRAIN_HDF5),
60     (valImages, valLabels, config.VAL_HDF5),
61     (testImages, testLabels, config.TEST_HDF5)]

```

Здесь мы просто инициализируем список наборов данных . Каждая запись в списке представляет собой тройку необработанных изображения, метки и выходной путь HDF5.

Последний шаг — перебрать каждый из наборов для обучения, проверки и тестирования:

```

63 # цикл по кортежам набора данных
64 для (изображения, метки, outputPath) в наборах данных:
65     # создаем средство записи HDF5
66     print("[INFO] здание {}".format(outputPath))
67     писатель = HDF5DatasetWriter((len(images), 48, 48), outputPath)
68
69     # перебираем изображение и добавляем его в набор данных
70     для (изображение, метка) в zip (изображения, метки):
71         Writer.add([изображение], [метка])
72
73     # закрыть модуль записи HDF5
74     писатель.close()
75
76 # закрыть входной файл
77 ф.закрыть()

```

Строка 67 создает экземпляр HDF5DatasetWriter для набора данных. Затем мы перебираем пары изображения и метки, записывая их на диск в формате HDF5 (строки 70 и 71).

Чтобы создать набор данных FER2013 для распознавания эмоций, просто выполните следующую команду:

```

$ Python build_dataset.py
[INFO] загрузка входных данных...
[INFO] здание /raid/datasets/fer2013/hdf5/train.hdf5...
[INFO] сборка /raid/datasets/fer2013/hdf5/val.hdf5...
[INFO] сборка /raid/datasets/fer2013/hdf5/test.hdf5...

```

После завершения выполнения команды вы можете убедиться, что файлы HDF5 созданы. Изучив содержимое каталога, в котором вы указали, что ваши файлы HDF5 должны храниться внутри `emoция_config.py`:

```

$ ls -l fer2013/hdf5
всего 646268
-rw-rw-r-- 1 адриан адриан 66183304 29 августа 08:25 test.hdf5
-rw-rw-r-- 1 адриан адриан 529396104 29 авг 08:25 train.hdf5
-rw-rw-r-- 1 адриан адриан 66183304 29 августа 08:25 val.hdf5

```

Обратите внимание, что у меня есть три файла: `train.hdf5`, `val.hdf5` и `test.hdf5` — это файлы, которые я буду использовать при обучении и оценке моей сети на наборе данных FER2013.

Тип слоя	Выходной размер	Размер фильтра/шаг
ВХОДНОЕ ИЗОБРАЖЕНИЕ	48×48×1	
КОНВ.	48×48×32 3×3, K = 32	
КОНВ.	48×48×32 3×3, K = 32	
БАССЕЙН	24×24×32 2×2	
КОНВ.	24×24×64 3×3, K = 64	
КОНВ.	24×24×64 3×3, K = 64	
БАССЕЙН	12×12×64 2×2	
КОНВ.	12×12×128 3×3, K = 128	
КОНВ.	12×12×128 3×3, K = 128	
БАССЕЙН	6×6×128 2×2	
ФК	64	
ФК	64	
ФК	6	
СОФТМАКС	6	

Таблица 11.1: Сводная таблица архитектуры EmotionVGGnet. Размеры выходного тома включены для каждого слоя, а также размер сверточного фильтра/размер пула, если это необходимо.

11.2 Реализация сети, подобной VGG

Сеть, которую мы собираемся внедрить для распознавания различных эмоций и выражений лица, вдохновлено семейством сетей VGG:

1. Слои CONV в сети будут только 3×3.

2. Мы будем удваивать количество фильтров, изученных каждым слоем CONV, чем глубже мы погружаемся в сеть.

Чтобы помочь в обучении сети, мы применим некоторые априорные знания, полученные в ходе эксперимента с VGG и ImageNet в главе 9:

1. Мы должны инициализировать наши слои CONV и FC, используя MSRA/He et al. метод - сделать так позволит нашей сети учиться быстрее.

2. Поскольку было показано, что ELU и PReLU повышают точность классификации во всех наших экспериментах, давайте просто начнем с ELU вместо ReLU.

Я включил краткую информацию о сети под названием EmotionVGGNet в Таблицу 11.1. После каждого CONV, мы применим активацию с последующей пакетной нормализацией — эти слои были намеренно исключены из таблицы для экономии места. Давайте продолжим и реализуем класс EmotionVGGNet. Теперь. Добавьте новый файл с именем emotionvggnet.py в подмодуль nn.conv pyimagesearch:

```

--- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- ио
| |--- нн
| | |--- __init__.py
| | |--- конв.
| | | |--- __init__.py
...
| | | |--- эмоцияvggnet.py
...
| | |--- предварительная обработка
| | |--- утилиты

```

Оттуда откройте эмоциюvggnet.py и начните с импорта наших необходимых пакетов Python:

```
1 # импортируем необходимые пакеты
2 из импорта keras.models Последовательный
3 из keras.layers.normalization import BatchNormalization
4 из keras.layers.convolutional импорт Conv2D
5 из keras.layers.convolutional импортировать MaxPooling2D
6 из keras.layers.advanced_activations импортировать ELU
7 из keras.layers.core импорт Активация
8 из keras.layers.core import Flatten
9 из keras.layers.core import Dropout
10 из keras.layers.core импорт Плотный
11 из keras импортировать бэкэнд как K
```

Все эти импорты являются стандартными при построении сверточных нейронных сетей с помощью Keras, поэтому мы можем пропустить их подробное обсуждение (не говоря уже о том, что вы видели эти импорты много раз ранее в этой книге).

Оттуда мы можем определить метод сборки EmotionVGGNet:

```
13 класс EmotionVGGNet:
14     @статический метод
15     def build(ширина, высота, глубина, классы):
16         # инициализируем модель вместе с входной формой, чтобы она была
17         # "каналы последние" и само измерение каналов
18         модель = Последовательный()
19         inputShape = (высота, ширина, глубина)
20         ЧанДим = -1
21
22         # если мы используем "сначала каналы", обновите форму ввода
23         # и размер каналов
24         если K.image_data_format() == "channels_first":
25             inputShape = (глубина, высота, ширина)
26             ЧанДим = 1
```

Строка 18 инициализирует объект модели , к которому мы будем добавлять слои. Поскольку семейство VGG сетей последовательно применяет слои один за другим, мы можем использовать класс Sequential здесь. Мы также предполагаем порядок «каналы последними» (строки 19 и 20), но если вместо этого мы используем «Сначала каналы», мы также можем обновить входную форму и ось измерения канала (линии 24-26).

Каждый блок в блоке свертки в EmotionVGGNet будет состоять из (CONV => RELU => BN)

* 2 => наборы слоев POOL. Теперь определим первый блок:

```
28     # Блок №1: первый CONV => RELU => CONV => RELU => POOL
29     # набор слоев
30     model.add(Conv2D(32, (3, 3), padding="то же самое",
31                     kernel_initializer="he_normal", input_shape=inputShape))
32     модель.добавить(ЭЛУ())
33     model.add(Пакетная нормализация (ось = chanDim))
34     model.add(Conv2D(32, (3, 3), kernel_initializer="he_normal",
35                     отступ = «то же самое»))
36     модель.добавить(ЭЛУ())
37     model.add(Пакетная нормализация (ось = chanDim))
```

```
38         model.add(MaxPooling2D(pool_size=(2, 2)))
39         model.add(Dropout(0.25))
```

Первый слой CONV выучит 32 фильтра 3×3 . Затем мы применим активацию ELU, а затем путем пакетной нормализации. Точно так же второй слой CONV применяет тот же шаблон, обучая 32 фильтры 3×3 с последующим ELU и нормализацией партии. Затем мы применяем максимальный пул, а затем слой отсева с вероятностью 25%.

Второй блок в EmotionVGGNet идентичен первому, только теперь мы удваиваем количество фильтров в слоях CONV до 64, а не 32:

```
41         # Блок №2: второй CONV => RELU => CONV => RELU => POOL
42         # набор слоев
43         model.add(Conv2D(64, (3, 3), kernel_initializer="he_normal",
44                         отступ = <то же самое>))
45         модель.добавить(ЭЛУ())
46         model.add(Пакетная нормализация (ось = chanDim))
47         model.add(Conv2D(64, (3, 3), kernel_initializer="he_normal",
48                         отступ = <то же самое>))
49         модель.добавить(ЭЛУ())
50         model.add(Пакетная нормализация (ось = chanDim))
51         model.add(MaxPooling2D(pool_size=(2, 2)))
52         model.add(Dropout(0.25))
```

Переходя к третьему блоку, мы снова применяем ту же схему, теперь увеличивая количество фильтров от 64 до 128 — чем глубже мы погружаемся в CNN, тем больше фильтров мы изучаем:

```
54         # Блок №3: третий CONV => RELU => CONV => RELU => POOL
55         # набор слоев
56         model.add(Conv2D(128, (3, 3), kernel_initializer="he_normal",
57                         отступ = <то же самое>))
58         модель.добавить(ЭЛУ())
59         model.add(Пакетная нормализация (ось = chanDim))
60         model.add(Conv2D(128, (3, 3), kernel_initializer="he_normal",
61                         отступ = <то же самое>))
62         модель.добавить(ЭЛУ())
63         model.add(Пакетная нормализация (ось = chanDim))
64         model.add(MaxPooling2D(pool_size=(2, 2)))
65         model.add(Dropout(0.25))
```

Далее нам нужно построить наш первый полносвязный набор слоев:

```
67         # Блок №4: первый набор слоев FC => RELU
68         model.add(Dense(64, kernel_initializer="he_normal"))
69         модель.добавить(ЭЛУ())
70         model.add(Пакетная нормализация())
71         model.add(Dropout(0.5))
```

Здесь мы изучаем 64 скрытых узла, после чего применяем функцию активации ELU и пакетную обработку. нормализация. Мы применим второй слой FC таким же образом:

```
74 # Блок №6: второй набор слоев FC => RELU  
75 model.add(Dense(64, kernel_initializer="he_normal")) model.add(ELU())  
76 model.add(BatchNormalization()) model.add(Dropout( 0,5))  
77  
78
```

Наконец, мы применим слой FC с предоставленным количеством классов вместе с классификатором softmax, чтобы получить наши вероятности меток выходного класса:

```
80     # Блок №7: классификатор softmax  
81     model.add(Dense(classes, kernel_initializer="he_normal"))  
82     model.add(Activation("softmax"))  
83  
84     # вернуть построенную сетевую архитектуру  
85     модель возврата
```

Теперь, когда мы реализовали наш класс EmotionVGGNet, давайте продолжим и обучим его.

11.3 Обучение нашего распознавателя выражения лица

В этом разделе мы рассмотрим, как обучить EmotionVGGNet с нуля на наборе данных FER2013. Мы начнем с обзора сценария `train_recognizer.py`, используемого для обучения FER2013, а затем рассмотрим ряд экспериментов, которые я провел для максимальной точности классификации. Для начала создайте новый файл с именем `train_recognizer.py` и вставьте следующий код:

```
1 # установить бэкэнд matplotlib, чтобы цифры можно было сохранять в фоновом режиме 2
import matplotlib 3 matplotlib.use("Agg")  
4  
5 # импортируем необходимые пакеты 6
из config _____ 12 из
keras.preprocessing.image import ImageDataGenerator 13 из keras.optimizers import
Adam 14 из keras.models import load_model 15 import keras.backend as K 16 import
argparse 17 import os
```

Строки 2 и 3 настраивают наш бэкэнд matplotlib, чтобы мы могли сохранять графики на диск. Оттуда строки 6-17 импортируют остальные наши пакеты Python. Мы использовали все эти пакеты раньше, но я обращаю ваше внимание на строку 11 , где импортируется недавно реализованный пакет EmotionVGGNet .

Далее, давайте проанализируем наши аргументы командной строки:

19 # построить разбор аргумента и разобрать аргументы 20 ap = argparse.ArgumentParser()

```

21 ap.add_argument("-c", "-checkpoints", required=True,
22     help="путь к выходному каталогу контрольных точек")
23 ap.add_argument("-m", "--model", type=str, help="путь к
24     *конкретной* контрольной точке модели для загрузки") 25
ap.add_argument("-s", "--start-epoch", тип=целое, по умолчанию=0,
26     help="эпоха, в которой нужно возобновить
обучение") 27 args = vars(ap.parse_args())

```

Эти аргументы командной строки указывают на настройку эксперимента в стиле Ctrl + С. Нам понадобится каталог -- checkpoints для хранения весов EmotionVGGNet по мере обучения сети. Если мы загружаем определенную эпоху с диска и (предположительно) перезапускаем обучение, мы можем указать путь --model к конкретной контрольной точке и предоставить --start-epoch связанной контрольной точки.

Отсюда мы можем создавать экземпляры наших объектов дополнения данных:

```

29 # создание обучающего и тестового генераторов изображений для данных 30
# увеличение, затем инициализация препроцессора изображений 31 trainAug =
ImageDataGenerator(rotation_range=10, zoom_range=0.1,
32     horizontal_flip = True, масштабирование = 1 / 255.0, fill_mode = "ближайший")
33 valAug = ImageDataGenerator (масштабирование = 1 / 255.0) 34 iap =
ImageToArrayPreprocessor ()

```

Как и ожидалось, мы будем применять увеличение данных к обучающему набору, чтобы уменьшить переоснащение и повысить точность классификации нашей модели (строки 31 и 32). Но то, с чем вы можете быть незнакомы, — это применение дополнения данных к набору проверки (строка 33). Зачем нам нужно применять дополнение данных к набору проверки? Разве этот набор не должен оставаться статичным и неизменным?

Ответ кроется в атрибуте масштабирования (который также является частью аугментера обучающих данных). Вспомните раздел 11.1.2, где мы преобразовали файл fer2013.csv в набор данных HDF5. Мы сохранили эти изображения как необработанные, ненормализованные изображения RGB, что означает, что значения пикселей могут существовать в диапазоне [0,255]. Однако обычной практикой является либо (1) выполнение средней нормализации, либо (2) уменьшение интенсивности пикселей до более ограниченного изменения. Кчастью для нас, класс ImageDataGenerator , предоставляемый Keras, может автоматически выполнять это масштабирование за нас. Нам просто нужно обеспечить значение масштабирования 1/255 — каждое изображение будет кратно этому коэффициенту, таким образом уменьшая пиксели до [0,1].

Давайте также инициализируем наши обучающие и проверочные объекты HDF5DatasetGenerator:

```

36 # инициализировать генераторы наборов данных для обучения и проверки
37 trainGen = HDF5DatasetGenerator(config.TRAIN_HDF5, config.BATCH_SIZE, aug=trainAug,
38     preprocessors=[iap], класс=config.NUM_CLASSES)
39 valGen = HDF5DatasetGenerator(config.VAL_HDF5, config.BATCH_SIZE, aug=valAug,
40     препроцессоры=[iap], классы=config.NUM_CLASSES)

```

Путь к входным файлам HDF5, размер пакета и количество классов — все это исходит из нашего Emotion_config, что упрощает изменение этих параметров.

В случае, если не указана конкретная контрольная точка модели, мы будем считать, что обучаем нашу модель. модель самой первой эпохи:

```

42 # если нет конкретной контрольной точки модели, то инициализируем 43 # сеть и
компилируем модель

```

```

44 , если args["model"] равно None:
45     print("[INFO] компилируемая модель...")
46     модель = EmotionVGGNet.build(ширина = 48, высота = 48, глубина = 1,
47         классы=config.NUM_CLASSES)
48     опт = Адам (lr = 1e-3)
49     model.compile(потеря="categorical_crossentropy", оптимизатор=опт,
50         метрики=["точность"])

```

В противном случае мы загружаем определенную контрольную точку с диска, обновляем скорость обучения и возобновление обучения:

```

52 # иначе загрузить чекпоинт с диска
53 еще:
54     print("[INFO] загрузка {}".format(args["model"]))
55     модель = load_model (аргументы ["модель"])
56
57     # обновить скорость обучения
58     print("[INFO] старая скорость обучения: {}".format(
59         K.get_value(model.optimizer.lr)))
60     K.set_value(model.optimizer.lr, 1e-3)
61     print("[INFO] новая скорость обучения: {}".format(
62         K.get_value(model.optimizer.lr)))

```

Прежде чем мы сможем начать процесс обучения, мы должны составить список обратных вызовов, используемых для сериализовать контрольные точки эпохи на диск и записать точность/потери на диск с течением времени:

```

64 # построить набор коллбэков
65 figPath = os.path.sep.join([config.OUTPUT_PATH,
66     "vggnet_emotion.png"])
67 jsonPath = os.path.sep.join([config.OUTPUT_PATH,
68     "vggnet_emotion.json"])
69 обратных вызовов =
70     EpochCheckpoint(args["контрольные точки"], каждый=5,
71         startAt=args["start_epoch"]),
72     TrainingMonitor (figPath, jsonPath = jsonPath,
73         startAt=args["start_epoch"]))

```

Наш последний блок кода обрабатывает обучение сети:

```

75 # обучаем сеть
76 модель.fit_generator(
77     поездГен.генератор(),
78     steps_per_epoch=trainGen.numImages // config.BATCH_SIZE,
79     validation_data=valGen.generator(),
80     validation_steps=valGen.numImages // config.BATCH_SIZE,
81     эпохи=15,
82     max_queue_size=config.BATCH_SIZE * 2,
83     обратные вызовы = обратные вызовы, подробный = 1)
84
85 # закрыть базы
86 поездГен.закрыть()
87 valGen.close()

```

Скорость обучения эпохи		
1	20	1e 2
21	40	1д 3
41	60	1e 4

Таблица 11.2: График скорости обучения, использованный при обучении EmotionVGGNet на FER2013 в эксперименте №1.

Точное количество эпох, которое нам понадобится для обучения сети, зависит от сети.

производительность и как мы останавливаем обучение, обновляем скорость обучения и возобновляем обучение.

В следующем разделе мы рассмотрим четыре эксперимента, которые я провел для постепенного улучшения производительности классификации EmotionVGGNet для распознавания выражений лица. Используйте это тематическое исследование, чтобы помочь вам в ваших собственных проектах — точно такие же методы, которые я использую для установки базовых показателей, проведения экспериментов, изучения результатов и обновления параметров, можно использовать, когда вы применяете глубокое обучение к проектам, не относящимся к этой книге.

11.3.1 EmotionVGGNet: эксперимент №1

Как я всегда делаю со своим первым экспериментом, я стремлюсь установить базовый уровень, который я могу постепенно улучшать — ваш эксперимент редко будет вашей самой эффективной моделью. Имейте в виду, что получение нейронной сети с высокой классификацией — это итеративный процесс. Вам нужно провести эксперименты, собрать результаты, интерпретировать их, настроить параметры и снова запустить эксперимент. Еще раз повторюсь, это наука, поэтому вам нужно применять научный метод — легких путей нет.

В этом эксперименте я начал с оптимизатора SGD с базовой скоростью обучения $1e - 2$, импульсом 0,9 и примененным ускорением Нестерова. Метод инициализации Xavier/Glorot (по умолчанию) использовался для инициализации весов в слоях CONV и FC . Кроме того, единственным добавлением данных, которое я применил, было горизонтальное отражение — никакие другие дополнения данных (такие как вращение, масштабирование и т. д.) не применялись.

Я начал обучение, используя следующую команду:

```
$ python train_recognizer.py --checkpoints контрольные точки
```

А затем использовал график скорости обучения в таблице 11.2 для обучения остальной части сети: график потери/точности для полных 60 эпох можно увидеть на рисунке 11.2. Интересно отметить, что как только я снизил скорость обучения с $1e - 2$ до $1e - 3$, сеть фактически прекратила обучение. Переход с $1e - 3$ на $1e - 4$ практически незаметен — при таких падениях порядка величины мы ожидаем увидеть, по крайней мере, некоторое повышение точности и соответствующее снижение потерь.

В конце 60-й эпохи я проверил точность проверки и заметил, что сеть достигает 62,91%. Если бы это был тестовый набор, мы бы были в первой десятке лидеров по распознаванию выражений лица Kaggle.

11.3.2 EmotionVGGNet: Эксперимент №2

Учитывая, что SGD привел к застою в обучении при снижении скорости обучения, я решил заменить SGD на Адама, используя базовую скорость обучения $1e - 3$. За исключением настройки оптимизатора, этот эксперимент идентичен первому. Я начал обучение, используя следующую команду:

```
$ python train_recognizer.py --checkpoints контрольные точки
```

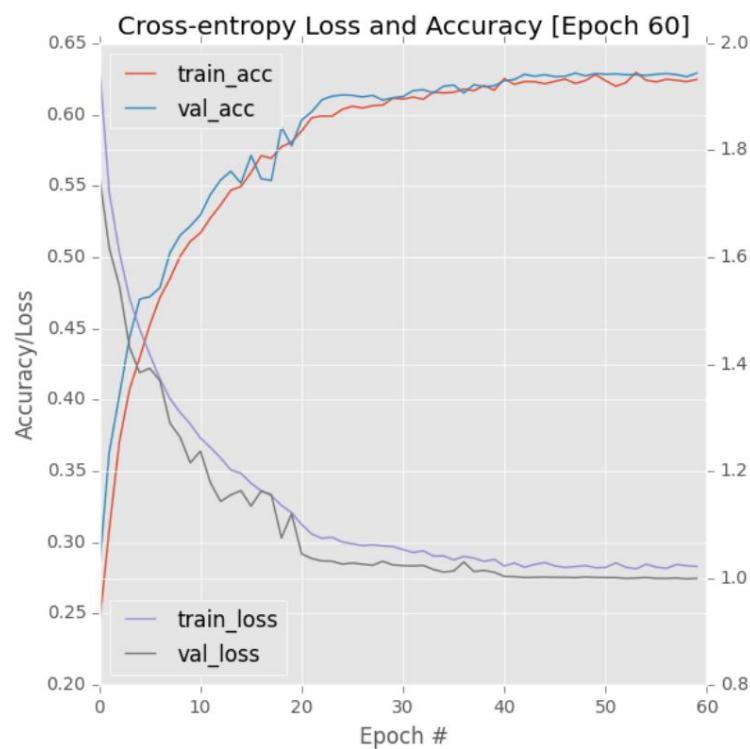


Рисунок 11.2: Учебный график для EmotionVGGNet на FER2013 в эксперименте №1. В прошлые эпохи 40 обучение застопорилось.

В эпоху 30 я начал замечать резкое расхождение между потерями при обучении и потерями при проверке (рис. 11.3, слева), поэтому я прекратил обучение и уменьшил скорость обучения с $1e-3$ до $1e-4$, а затем позволил сети обучиться еще раз. 15 эпох:

```
$ python train_recognizer.py --checkpoints контрольные точки \--  
model контрольные точки/epoch_30.hdf5 --start-epoch 30
```

Однако результат был не очень хорошим (рис. 11.3, справа). Как мы видим, имеет место явная переобучение — потери при обучении продолжают снижаться, в то время как потери при проверке не только остаются на прежнем уровне, но и продолжают расти. При всем этом точность сети по-прежнему составляла 66,34% в конце 45-й эпохи, что определенно лучше, чем у SGD. Если бы я мог найти способ обуздить переоснащение, подход оптимизатора Адама, вероятно, работал бы очень хорошо в этой ситуации.

11.3.3 EmotionVGGNet: Эксперимент №3

Обычным лекарством от переобучения является сбор большего количества обучающих данных, репрезентативных для вашего набора проверки и тестирования. Однако, поскольку набор данных FER2013 предварительно скомпилирован, и мы хотим, чтобы наши результаты соответствовали требованиям конкурса Kaggle, о сборе дополнительных данных не может быть и речи. Вместо этого мы можем применить увеличение данных, чтобы уменьшить переоснащение.

В моем третьем эксперименте я сохранил свой оптимизатор Adam, но также добавил случайный диапазон поворота в 10 градусов вместе с диапазоном масштабирования 0,1 (другие параметры увеличения, предоставляемые Keras, здесь не подходят). С новой схемой увеличения данных я повторил свой второй эксперимент:

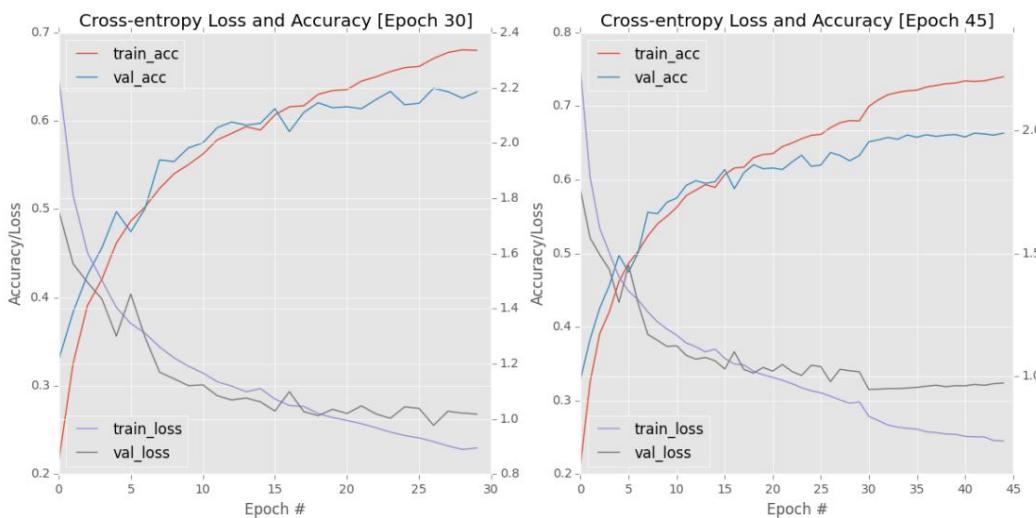


Рисунок 11.3: Слева: первые 30 эпох для скорости обучения $1e^{-3}$ и оптимизатора Adam. Справа: снижение скорости обучения до $a = 1e^{-4}$ вызывает явное переоснащение.

```
$ python train_recognizer.py --checkpoints контрольные точки
```

Как показывает приведенный ниже график, обучение гораздо более стабильно, когда насыщение начинает происходить примерно в эпоху 40 (рис. 11.4, вверху слева). В этот момент я прекратил тренировку, снизил скорость обучения Адама с $1e^{-3}$ до $1e^{-4}$ и возобновил тренировку:

```
$ python train_recognizer.py --checkpoints контрольные точки \--  
model контрольные точки/epoch_40.hdf5 --start-epoch 40
```

Этот процесс привел к характерному снижению потерь/повышению точности, которого мы ожидаем при такой настройке скорости обучения (рис. 11.4, вверху справа). Однако обучение остановилось, поэтому я снова прекратил обучение на 60-й эпохе, снизил скорость обучения с $1e^{-4}$ до $1e^{-5}$ и возобновил обучение еще на 15 эпох:

```
$ python train_recognizer.py --checkpoints контрольные точки \--  
model checkpoints/epoch_60.hdf5 --start-epoch 60
```

Окончательный график сети можно увидеть на рисунке 11.4 (внизу). Как мы видим, мы не подвержены риску переобучения — недостатком является то, что мы не наблюдаем каких-либо значительных улучшений в точности после 45-й эпохи. Все это говорит о том, что, применяя аугментацию данных, мы смогли стабилизировать обучение, уменьшить переоснащение, и позволяют нам достичь точности классификации 67,53% на проверочном наборе.

11.3.4 EmotionVGGNet: Эксперимент №4

В моем последнем эксперименте с FER2013 и EmotionVGGNet я решил внести несколько изменений:

1. Я заменил инициализацию Xavier/Glorot (по умолчанию используется Keras) для MSRA/He et al. инициализация.

Этот обмен связан с тем, что He et al. инициализация, как правило, лучше работает для сетей семейства VGG (как мы видели в предыдущих экспериментах в этой книге).

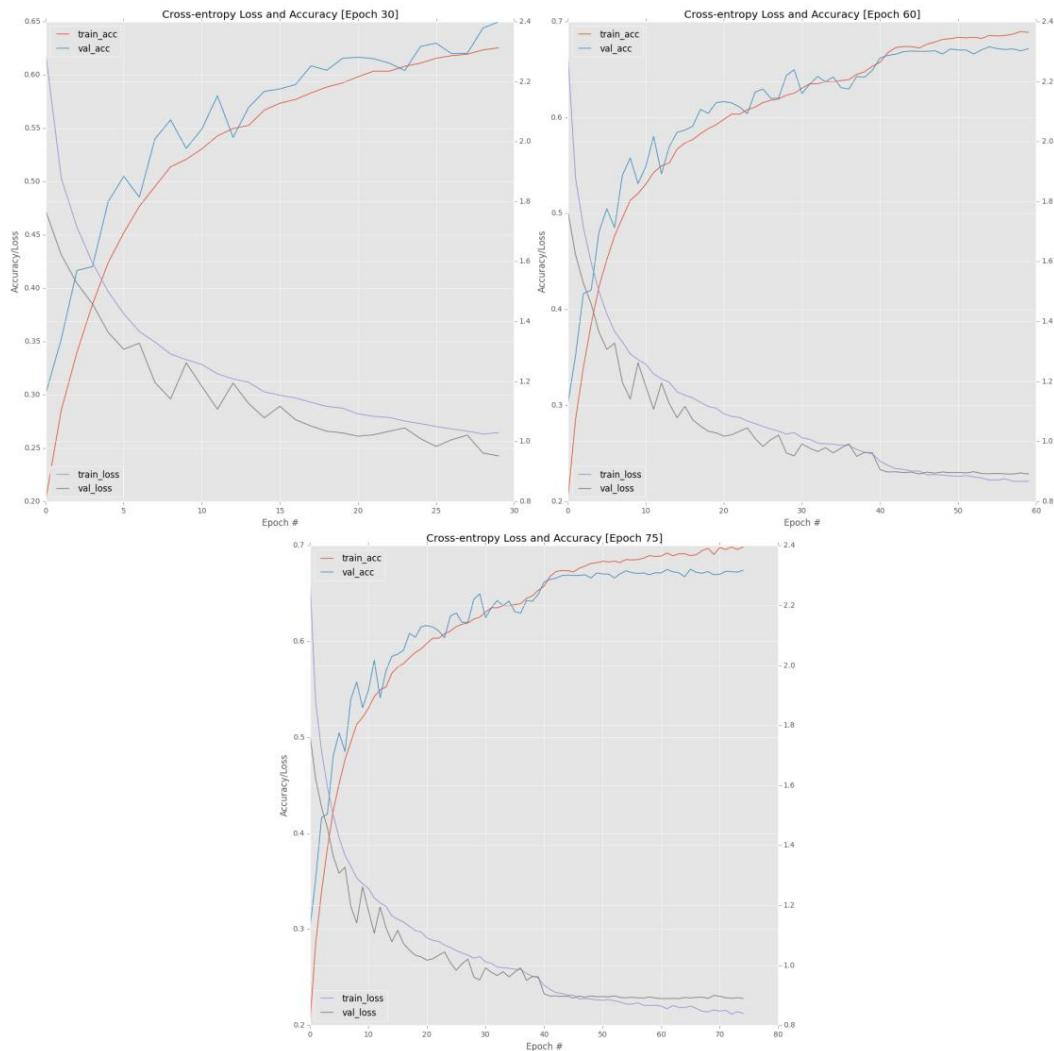


Рисунок 11.4: Верхний левый: первые 30 эпох для Эксперимента №3. Вверху справа: настройка скорости обучения на $1e^{-4}$. Внизу: окончательная корректировка $\alpha = 1e^{-5}$.

2. Я заменил все ReLU на ELU, чтобы еще больше повысить точность.
3. Учитывая дисбаланс классов, вызванный ярлыком «отвращение», я объединил «гнев» и «отвращение» в один ярлык в соответствии с рекомендацией проекта Mememoji [33]. Причина такого слияния меток двояка: (1) чтобы мы могли получить более высокую точность классификации, и (2) наша модель будет лучше обобщать при применении к распознаванию эмоций в реальном времени позже в этой главе.

Чтобы объединить эти два класса, мне нужно было снова запустить `build_dataset.py` с `NUM_CLASSES`, равным шести, а не семи.

Опять же, оптимизатор Адама использовался с базовой скоростью обучения $1e^{-3}$, которая уменьшалась по таблице 11.3.

График потери/точности этого эксперимента можно увидеть на рис. 11.5. Обратите внимание, как я всегда снижаю скорость обучения, как только потеря проверки и точность выходят на плато. Сам сюжет выглядит так же, как и в третьем эксперименте; однако, когда мы проверяем выходные данные 75-й эпохи, мы видим, что EmotionVGGNet достигает точности 68,51% — это самая высокая точность, которую мы когда-либо видели. Отсюда давайте перейдем к оценке нашего распознавателя выражения лица на тестовом наборе.

Скорость обучения эпохи		
1	40	1e 3
40	60	1e 4
61	75	1e 5

Таблица 11.3: График скорости обучения, использованный при обучении EmotionVGGNet на FER2013 в эксперименте №4.

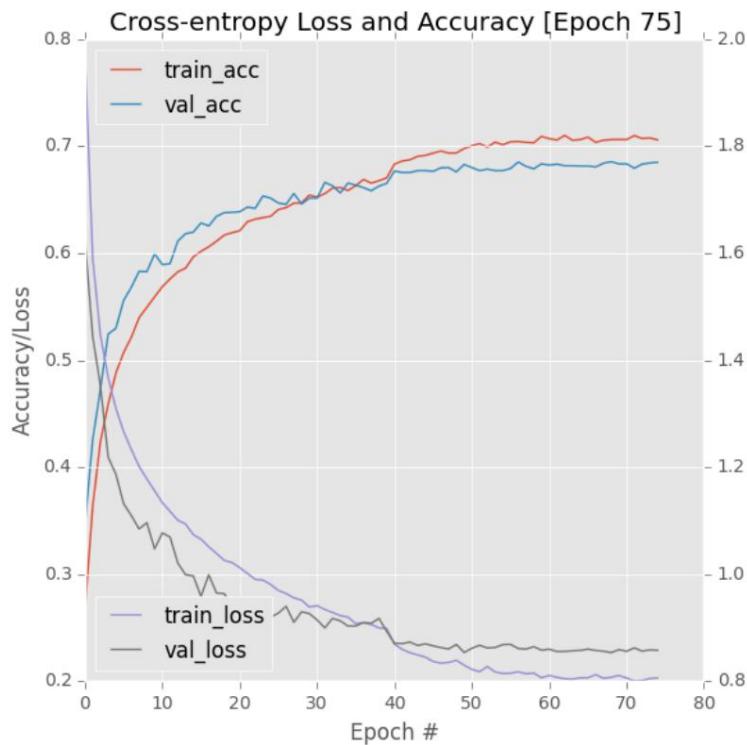


Рисунок 11.5: Наш последний эксперимент по обучению EmotionVGGNet на FER2013. Здесь мы достигаем точности 68,51%, объединяя классы «гнев» и «отвращение» в один ярлык.

11.4 Оценка нашего распознавателя выражения лица

Чтобы оценить EmotionVGGnet на тестовом наборе FER2013, давайте откроем `test_recognizer.py` и вставим следующий код:

```

1 # импортируем необходимые пакеты 2
из конфига импортируем эмоции_config как конфиг 3 из
pyimagesearch.preprocessing импортируем ImageToArrayPreprocessor 4 из
pyimagesearch.io импортируем HDF5DatasetGenerator 5 из keras.preprocessing.image
импортируем ImageDataGenerator 6 из keras.models импортируем load_model 7
импортируем argparse

```

Строки 2-7 импортируют необходимые пакеты Python. Стока 2 импортирует нашу эмоцию_config, поэтому у нас есть доступ к переменным конфигурации нашего проекта. HDF5DatasetGenerator потребуется для

получить доступ к тестовому набору (строка 4). Мы также импортируем `ImageDataGenerator`, чтобы мы могли масштабировать изображения в FER2013 до диапазона [0,1].

Наш скрипт потребует только один аргумент командной строки, `--model`, который является путем к конкретная контрольная точка модели для загрузки:

```
9 # построить разбор аргумента и разобрать аргументы 10 ap =
argparse.ArgumentParser() 11 ap.add_argument("-m", "--model",
type=str,
12           help="путь к контрольной точке модели для
загрузки") 13 args = vars(ap.parse_args())
```

Оттуда мы инициализируем наш класс увеличения данных, чтобы масштабировать изображения в тестовом наборе:

```
15 # инициализируем генератор тестовых данных и препроцессор
изображений 16 testAug = ImageDataGenerator(rescale=1 / 255.0) 17 iap =
ImageToArrayPreprocessor()
18
19 # инициализируем генератор тестовых наборов данных
```

21

Указатель на тестовый набор данных HDF5 затем открывается в строках 20 и 21, гарантируя, что мы применяем как: (1) наше увеличение данных для масштабирования, так и (2) наше изображение в Keras-совместимом массиве. преобразователь.

Наконец, мы можем загрузить контрольную точку нашей модели с диска:

```
23 # загрузить модель с диска
24 print("[INFO] loading {}".format(args["model"])) 25 model =
load_model(args["model"])
```

И оцените его на тестовом наборе:

```
27 # оценить сеть
28 (убыток, акк) = model.evaluate_generator(
29     testGen.generator(),
30     steps=testGen.numImages // config.BATCH_SIZE,
31     max_queue_size=config.BATCH_SIZE * 2) 32 print("[INFO]
точность: {:.2f}".format(acc * 100))
33
34 # закрыть тестовую базу данных 35 testGen.close()
```

Чтобы оценить EmotionVGGNet на FER2013, просто откройте терминал и выполните следующие действия. команда:

```
$ python test_recognizer.py --model checkpoints/epoch_75.hdf5 [INFO]
загрузка контрольных точек/epoch_75.hdf5...
[ИНФО] Точность: 66,96
```

Как показывают мои результаты, мы смогли получить точность 66,96% на тестовом наборе, достаточно для нас, чтобы претендовать на 5-е место в конкурсе распознавания выражений лица Kaggle [31].

 Этот результат классификации 66,96% был получен из 6-классового варианта FER2013, а не из 7-классовой исходной версии в задаче распознавания Kaggle. Тем не менее, мы можем легко переобучить сеть на версии 7-го класса и получить аналогичную точность. Причина, по которой мы заканчиваем сеть с 6 классами, заключается в том, что мы можем получить более значимые результаты применительно к обнаружению эмоций в реальном времени в следующем разделе.

11.5 Обнаружение эмоций в режиме реального времени

Теперь, когда наша CNN обучена и оценена, давайте применим ее для обнаружения эмоций и выражений лица в видеопотоках в режиме реального времени. Откройте новый файл, назовите egomotion_detector.py и вставьте следующий код:

```

1 # импортируем необходимые пакеты 2 из
keras.preprocessing.image import img_to_array 3 из keras.models import
load_model 4 import numpy as np 5 import argparse 6 import imutils 7
import cv2

8
9 # построить разбор аргумента и разобрать аргументы 10 ap =
argparse.ArgumentParser() 11 ap.add_argument("-c", "--cascade", required=True,
import cv2

12     help="путь к расположению каскада лиц")
13 ap.add_argument("-m", "--model", required=True,
14     help="путь к предварительно обученному детектору эмоций CNN")
15 ap.add_argument("-v", "--video", help="путь к (необязательному) видеофайлу")
16
17 аргументов = вары (ap.parse_args())

```

Строки 2-7 импортируют необходимые пакеты Python. Оттуда мы анализируем наши аргументы командной строки. Наш сценарий требует двух переключателей, за которыми следует третий необязательный. Переключатель --cascade — это путь к нашему каскаду Хаара для обнаружения лиц (включен в раздел загрузки этой книги). --model — это наша предварительно обученная CNN, которую мы будем использовать для обнаружения эмоций. Наконец, если вы хотите применить обнаружение эмоций к видеофайлу, а не к видеопотоку, вы можете указать путь к файлу с помощью переключателя --video.

Оттуда давайте загрузим наш каскад обнаружения лиц, обнаружение эмоций CNN, а также инициализируем список меток эмоций, которые может предсказать наша CNN:

```

19 # загрузите каскад детектора лиц, обнаружение эмоций CNN, затем определите 20 # список меток эмоций

21 детектор = cv2.CascadeClassifier (аргументы ["каскад"]) 22 модель = load_model (аргументы ["модель"])

23 ЭМОЦИИ = ["сердитый", "испуганный", "счастливый", "грустный", "удивленный",
24     "нейтральный"]

```

Обратите внимание, что наш список ЭМОЦИИ содержит шесть элементов, подразумевая, что мы используем 6-й класс. версия FER2013 для большей точности.

Наш следующий блок кода создает экземпляр объекта cv2.VideoCapture в зависимости от того, (1) доступ к веб-камере или (2) чтение из видеофайла:

```

26 # если видеотракт не указан, взять ссылку на веб-камеру
27 , если не args.get("видео", False):
28     камера = cv2.VideoCapture(1)
29
30 # иначе загрузить видео
31 еще:
32     камера = cv2.VideoCapture (аргументы ["видео"])

```

Теперь мы готовы начать перебирать кадры с видео указателя:

```

34 # продолжаем зацикливаться
35 , пока верно:
36     # захват текущего кадра
37     (снято, кадр) = камера.read()
38
39     # если мы просматриваем видео и мы не захватили
40     # кадр, значит мы дошли до конца видео
41     если args.get("video") и не захвачен:
42         переменна

```

Строка 37 считывает следующий кадр из видеопотока. Если кадр не был захвачен (т.е. установлен на False) и мы читаем кадры из видеопотока, мы дошли до конца файла, значит должен выйти из цикла (строки 41 и 42).

В противном случае пришло время предварительно обработать кадр, изменив его размер до ширины 300 пикселей и преобразование его в оттенки серого:

```

44     # изменить размер кадра и преобразовать его в оттенки серого
45     кадр = imutils.resize (кадр, ширина = 300)
46     серый = cv2.cvtColor (кадр, cv2.COLOR_BGR2GRAY)
47
48     # инициализируем холст для визуализации, затем клонируем
49     # рамка, чтобы мы могли рисовать на ней
50     холст = np.zeros((220, 300, 3), dtype="uint8")
51     frameClone = кадр.копировать()
52
53     # обнаруживать лица во входном кадре, затем клонировать кадр так, чтобы
54     # мы можем рисовать на нем
55     rect = детектор.detectMultiScale (серый, масштабный коэффициент = 1,1,
56                                         minNeighbors=5, minSize=(30, 30),
57                                         флаги = cv2.CASCADE_SCALE_IMAGE)

```

Мы инициализируем пустой холст NumPy (строка 50) шириной 300 пикселей и высотой 200px. Мы будем использовать холст для рисования распределения вероятностей, предсказанного нашей CNN, что позволит нам визуализировать диапазон и смесь эмоций.

Строки 55-57 затем обнаруживают лица в кадре, используя предварительно обученный каскад Хаара OpenCV. Если вы хотите узнать больше об обнаружении объектов с помощью каскадов Хаара, см.

Практический Python и OpenCV [34] (<http://pyimg.co/ppao>) и курс PyImageSearch Gurus [35] (<http://pyimg.co/gurus>).

В следующем блоке кода мы подготавливаем ROI лица для классификации через CNN:

```

59     # убедитесь, что хотя бы одно лицо было найдено, прежде чем продолжить
60     если длина (прямоугольники) > 0:
61         # определить наибольшую площадь лица
62         прямоугольник = отсортировано (прямоугольники, реверс = истина,
63             Ключ = лямбда x: (x [2] - x [0]) * (x [3] - x [1])) [0]
64         (fX, fY, fW, fH) = прямоугольник
65
66         # извлечь область интереса лица из изображения, а затем выполнить предварительную обработку
67         # это для сети
68         roi = серый[fY:fY + fH, fX:fX + fW]
69         roi = cv2.resize(roi, (48, 48))
70         roi = roi.astype ("с плавающей запятой") / 255,0
71         roi = img_to_array(roi)
72         roi = np.expand_dims (roi, ось = 0)

```

Строка 60 гарантирует, что в кадре было обнаружено хотя бы одно лицо. При условии, что было хотя бы обнаружено одно лицо, мы сортируем прямоугольник списка ограничивающих рамок в соответствии с размером ограничивающей рамки, с крупными лицами в начале списка (строки 62 и 64).

Мы, безусловно, могли бы применить обнаружение эмоций и распознавание выражений лица к каждому лицу в рамке; однако смысл этого скрипта в том, чтобы продемонстрировать, как мы можем (1) обнаруживать наиболее доминирующее выражение лица и (2) график распределения эмоций. Построение этого распределения для каждого лица в кадре будет отвлекать, и нам будет труднее его визуализировать. Таким образом, я оставлю это как упражнение для читателя, чтобы перебрать каждый из прямоугольников по отдельности - для простоты, мы просто будем использовать самую большую область лица в нашем примере.

Строка 68 извлекает область лица из изображения в градациях серого с помощью нарезки массива NumPy. Затем мы предварительно обрабатываем область интереса, изменяя ее размер до фиксированных 48×48 пикселей (размер ввода, требуемый EmotionVGGNet). Оттуда мы преобразуем `roi` в тип данных с плавающей запятой, масштабируем его до диапазона $[0,1]$ и конвертируем его в Keras-совместимый массив (строки 69-72).

Теперь, когда область интереса предварительно обработана, мы можем передать ее через нашу модель, чтобы получить класс вероятности:

```

74     # делаем прогноз ROI, затем ищем класс
75     # Метка
76     pres = model.predict(roi)[0]
77     label = ЭМОЦИИ[preds.argmax()]
78
79     # перебираем метки + вероятности и рисуем их
80     for (i, (эмоция, проблема)) в enumerate(zip(ЭМОЦИИ, преды)):
81         # построить текст метки
82         text = "{}: {:.2f}%".format(эмоция, проблема * 100)
83
84         # рисуем метку + полосу вероятности на холсте
85         ш = целое (вероятность * 300)
86         cv2.rectangle (холст, (5, (i * 35) + 5),
87             (ш, (i * 35) + 35), (0, 0, 255), -1)
88         cv2.putText (холст, текст, (10, (i * 35) + 23),
89             cv2.FONT_HERSHEY_SIMPLEX, 0,45,
90             (255, 255, 255), 2)

```

Строка 76 вызывает метод прогнозирования модели, который возвращает прогнозируемую метку класса. вероятности. Таким образом, метка является меткой с наибольшей ассоциированной вероятностью (строка 77).

Однако, поскольку выражение лица человека часто представляет собой смесь эмоций, это гораздо важнее. Интересно изучить распределение вероятностей меток. Чтобы выполнить это определение, мы перебираем метки и связанные с ними вероятности в строке 80. Строки 86 и 87 рисуют гистограмму, где ширина каждой полосы пропорциональна прогнозируемой вероятности метки класса. Строки 88-90 затем рисуем название этикетки на холсте.

 Если вам интересно понять, как эти функции рисования OpenCV работают в более подробности см. в публикации «Практический Python и OpenCV» [34] (<http://pyimg.co/prao>).

Наш следующий блок кода обрабатывает отрисовку метки с наибольшей вероятностью на нашем экране как а также рисование связанного прямоугольника, окружающего лицо, мы предсказали эмоции для:

```

92     # рисуем метку на рамке
93     cv2.putText (frameClone, метка, (fx, fy - 10),
94                 cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
95     cv2.rectangle (frameClone, (fx, fy), (fx + fW, fy + fH),
96                   (0, 0, 255), 2)

```

Последний блок кода в этом примере просто отображает выходные изображения на нашем экране, чтобы мы могли визуализировать результаты в режиме реального времени:

```

98     # показать наши классификации + вероятности
99     cv2.imshow ("Лицо", фреймКлон)
100    cv2.imshow ("Вероятности", холст)

101
102    # если нажата клавиша 'q', останавливаем цикл
103    если cv2.waitKey(1) & 0xFF == ord("q"):
104        переменна
105
106 # очистить камеру и закрыть все открытые окна
107 камера.релиз()
108 cv2.destroyAllWindows()

```

Если вы хотите применить детектор эмоций к видеопотоку с веб-камеры, откройте свой терминал и выполните следующую команду:

```
$ pythonmotion_detector.py --cascade haarcascade_frontalface_default.xml \
--модели контрольных точек/epoch_75.hdf5
```

В противном случае, если вместо этого вы применяете предсказание выражения лица к видеофайлу, обновите Команду для использования переключателя --video для указания на ваш видеофайл, находящийся на диске:

```
$ pythonmotion_detector.py --cascade haarcascade_frontalface_default.xml \
--model checkpoints/epoch_75.hdf5 --видео путь/к/вашему/video.mp4
```

В разделе загрузки глав, связанных с этой книгой, я предоставил пример видео, которое вы можно использовать, чтобы убедиться, что ваш предсказатель эмоций работает правильно. Например, в следующем в кадре мое лицо явно счастливо (рис. 11.6, вверху слева). В то время как в правом верхнем углу мое выражение лица изменился на сердитый. Однако обратите внимание, что вероятность грусти также выше, что означает, что

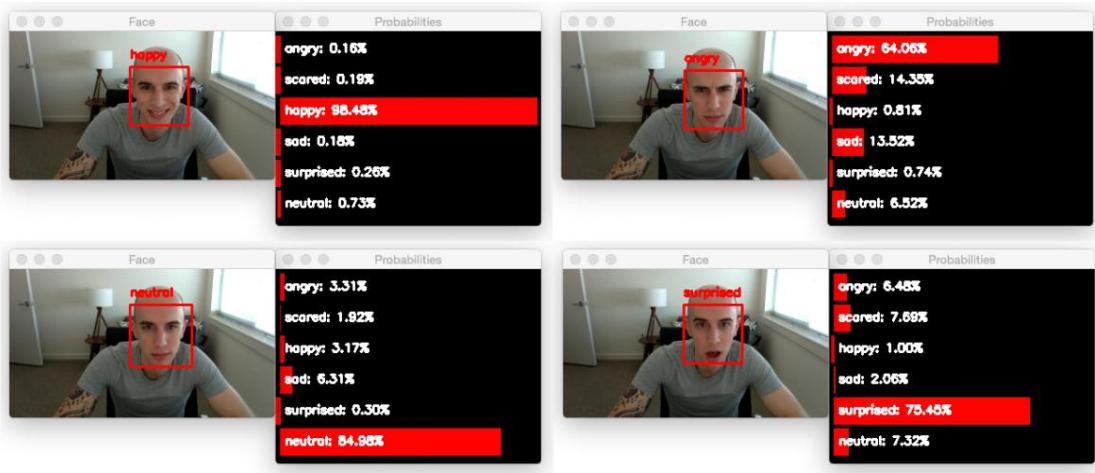


Рисунок 11.6: Примеры распознавания выражения лица с использованием нашей сверточной нейронной сети. В каждом случае мы можем правильно распознать эмоцию. Кроме того, обратите внимание, что распределение вероятностей представляет выражение лица как смесь эмоций.

есть смесь эмоций. Через несколько секунд мое лицо возвращается в спокойное нейтральное положение (внизу слева). Наконец, в правом нижнем углу я явно удивлен.

Важно отметить, что эта модель не обучалась ни на каких примерах моих изображений — использовался только набор данных FER2013. Учитывая, что наша модель способна правильно предсказывать мою мимику, мы можем сказать, что наша EmotionVGGNet выражает очень хорошую способность к обобщению.

11.6 Резюме

В этой главе мы узнали, как реализовать свёрточную нейронную сеть, способную предсказывать эмоции и выражения лица. Для выполнения этой задачи мы обучили VGG-подобную CNN с именем EmotionVGGNet. Эта сеть состояла из двух слоев CONV, расположенных друг над другом, с удвоением количества фильтров в каждом блоке. Было важно, чтобы наша CNN была:

1. Достаточно глубоко для получения высокой точности.
2. Но не настолько глубоко, чтобы нельзя было запустить в реальном времени на ЦП.

Затем мы обучили нашу CNN набору данных FER2013, что является частью задачи Kaggle по распознаванию эмоций и выражений лица. В целом, мы смогли получить точность 66,96%, достаточную для того, чтобы занять 5-е место в таблице лидеров. Дальнейшей точности, вероятно, можно добиться, если более агрессивно увеличивать наши данные, углублять сеть, увеличивать количество слоев и добавлять регуляризацию.

Наконец, мы закончили эту главу, создав скрипт Python, который может (1) обнаруживать лица в видеопотоке и (2) применять нашу предварительно обученную СНС для распознавания доминирующего выражения лица в режиме реального времени. Кроме того, мы также включили распределение вероятности для каждой эмоции, что позволило нам легче интерпретировать результаты нашей сети.

Опять же, имейте в виду, что наши эмоции подвижны и постоянно меняются. Кроме того, наши эмоции часто являются смесью друг друга — мы редко, если вообще когда-либо, на 100% счастливы или на 100% опечалены — вместо этого мы всегда являемся смесью чувств. Из-за этого факта важно изучить распределение вероятностей, возвращаемое EmotionVGGNet, когда вы пытаетесь пометить выражение лица данного человека.

12. Практический пример: коррекция ориентации изображения

В этом тематическом исследовании мы узнаем, как применять трансферное обучение (в частности, извлечение признаков) для автоматического определения и исправления ориентации изображения. Причина, по которой мы используем извлечение признаков из предварительно обученной сверточной нейронной сети, двояка:

1. Продемонстрировать, что фильтры, изученные CNN, обученной на ImageNet, не инвариантны к вращению во всем 360-градусном спектре (в противном случае эти функции нельзя было бы использовать для различия поворота изображения).
2. Перенос обучения с помощью извлечения признаков обеспечивает высочайшую точность при прогнозировании изображения . ориентация.

Как я упоминал в главе 11 Starter Bundle, распространено заблуждение, что отдельные фильтры, изученные CNN, инвариантны к вращению, а сами фильтры — нет. Вместо этого CNN может изучить набор фильтров, которые активируются, когда они видят конкретный объект при заданном повороте. Следовательно, сами фильтры не инвариантны к вращению, но CNN может получить некоторый уровень инвариантности к вращению благодаря количеству изученных фильтров, причем в идеале некоторые из этих фильтров активируются при различных вращениях. Как мы увидим в этой главе, CNN не полностью инвариантны к вращению, иначе мы не смогли бы определить ориентацию изображения строго по признакам, извлеченным на этапе тонкой настройки.

12.1 Набор данных CVPR для помещений

Набор данных, который мы будем использовать для этого тематического исследования, представляет собой набор данных Indoor Scene Recognition (также называемый Indoor CVPR) [36], выпущенный MIT. Эта база данных содержит 67 категорий помещений/сцен , включая дома, офисы, общественные места, магазины и многое другое. Образец этого набора данных можно увидеть на рис. 12.1.

Причина, по которой мы используем этот набор данных, заключается в том, что человеку очень легко определить, неправильно ли ориентировано изображение естественной сцены — наша цель — воспроизвести этот уровень производительности с помощью сверточной нейронной сети. Однако все изображения в Indoor CVPR ориентированы правильно; поэтому нам нужно создать собственный набор данных из Indoor CVPR с помеченными изображениями при различных поворотах.

Чтобы загрузить набор данных Indoor CVPR, используйте эту ссылку:



Рисунок 12.1: Пример классов и изображений из набора данных Indoor Scene Recognition [36].

<http://pyimg.co/x772x> И

форма там нажмите на ссылку «Скачать». Весь архив .tar весит 2,4 ГБ, поэтому планируйте загрузку соответствующим образом. После загрузки файла разархивируйте его. После разархивирования вы найдете каталог с именем `Images`, который содержит несколько подкаталогов, каждый из которых содержит определенную метку класса в наборе данных:

```
$ cd Изображения
$ ls -l | head -n 10
total 0
drwxr-xr-x@ 610 adrianrosebrock staff 20740
```

```
16 марта 2009 г. airport_inside drwxr-xr-x@ 142 adrianrosebrock staff 4828 16 марта 2009 г. xr-x@ 407 персонал adrianrosebrock
13838 16 марта 2009 пекарня drwxr-xr-x@ 606 персонал adrianrosebrock 20604 16 марта 2009 бар
```

```
drwxr-xr-x@ 199 adrianrosebrock staff 6766 16 марта 2009 ванная комната
drwxr-xr-x@ 664 adrianrosebrock staff 22576 16 марта 2009 спальня
drwxr-xr-x@ 382 adrianrosebrock staff 12988 16 марта 2009 книжный магазин
drwxr-xr-x@ 215 adrianrosebrock staff 7310 16 марта 2009 боулинг
```

Чтобы упорядочить набор данных Indoor CVPR, я создал новый каталог с именем `door_cvpr`, переместил каталог изображений внутрь (изменив прописную букву «I» на строчную «i») и создал два новых подкаталиога — `hdf5` и `rotated_images`. Моя структура каталогов для этого проекта выглядит следующим образом:

```
--- крытый_cvpr | |---
hdf5 | |изображения
| |---
повернутые_изображения
```

Каталог `hdf5` будет хранить функции, извлеченные из наших входных изображений с использованием предварительно обученной сверточной нейронной сети. Чтобы сгенерировать наши обучающие данные, мы создадим собственный скрипт Python, который генерирует случайно вращающиеся изображения. Эти повернутые изображения будут храниться в файле `rotated_images`. Функции, извлеченные из этих изображений, будут храниться в наборах данных HDF5.

12.1.1 Создание набора данных Прежде

чем мы начнем, давайте взглянем на структуру нашего проекта:

```
--|-- ориентация_изображения | |---
creat_dataset.py
```

```

|--- extract_features.py |---
door_cvpr/ |--- models/ |---
orient_images.py |---
train_model.py

```

Мы будем использовать скрипт `create_dataset.py` для создания обучающих и тестовых наборов для нашего входного набора данных. Оттуда `extract_features.py` будет использоваться для создания файла HDF5 для разделения набора данных. Учитывая извлеченные функции, мы можем использовать `train_model.py` для обучения классификатора логистической регрессии распознавать ориентацию изображения и сохранять полученную модель в каталоге моделей . Наконец, `orient_images.py` можно применять для тестирования входных изображений на ориентацию.

Так же, как я сделал с набором данных ImageNet, я создал символическую ссылку `hadoop_cvpr`, чтобы упростить ввод команд; однако вы можете пропустить этот шаг, если хотите, и вместо этого указать полный путь к входным/выходным каталогам при выполнении скриптов Python.

Давайте продолжим и узнаем, как мы можем создать собственный пользовательский набор данных ориентации изображения из существующего набора данных. Убедитесь, что вы загрузили файл `.tar` из набора данных Indoor Scene Recognition, и оттуда откройте новый файл, назовите его `create_dataset.py`, и мы заставим его работать:

```

1 # импортируем необходимые пакеты
2 из imutils import paths 3 import numpy as
np 4 import progressbar 5 import argparse 6
import imutils 7 import random 8 import cv2
9 import os

10
11 # построить аргумент parse и проанализировать аргументы 12 ap =
argparse.ArgumentParser() 13 ap.add_argument("-d", "--dataset",
required=True, help="путь к входному каталогу изображений") 15
14     ap.add_argument("-o", "--output", required=True,
16             help="путь к выходному каталогу повернутых изображений")
17 args = vars(ap.parse_args())

```

Строки 2–9 импортируют необходимые пакеты Python, все из которых мы рассмотрели ранее в этой книге. Мы снова будем использовать дополнительный пакет `progressbar` для отображения обновлений в нашем терминале по мере выполнения нашего процесса создания набора данных. Если вы не хотите использовать индикатор выполнения, просто закомментируйте несколько строк в этом скрипте, которые ссылаются на него.

Строки 12–17 затем анализируют наши аргументы командной строки. Здесь нам нужны только два аргумента командной строки: `--dataset`, путь к каталогу, содержащему наши входные изображения (набор данных Indoor CVPR), а также `--output` путь к каталогу, в котором будут храниться наши помеченные повернутые изображения.

Затем давайте случайным образом выберем 10 000 изображений из нашего каталога `--dataset` и инициализируем наш индикатор выполнения:

¹⁹ # берем пути к входным изображениям (ограничившись 10 000 ²⁰ # изображений) и перемешиваем их, чтобы упростить создание обучающего и тестового ²¹ # сплита

```

22 imagePaths = list(paths.list_images(аргументы["набор данных"])[:10000]
23 random.shuffle (изображения)
24
25 # инициализируем словарь, чтобы отслеживать номер каждого угла
26 # выбрано до сих пор, затем инициализируйте индикатор выполнения
27 углов = {}
28 widgets = ["Building Dataset: ", progressbar.Percentage(), " ",
29             progressbar.Bar(), " ", progressbar.ETA()]
30 pbar = progressbar.ProgressBar (maxval = len (imagePaths),
31                                 виджеты=виджеты).start()

```

Учитывая наши образцы imagePath, мы теперь готовы создать наш набор данных повернутого изображения:

```

33 # цикл по путям изображений
34 для (i, imagePath) в перечислении (imagePaths):
35     # определяем угол поворота и загружаем изображение
36     угол = np.random.choice([0, 90, 180, 270])
37     изображение = cv2.imread (путь к изображению)
38
39     # если изображение имеет значение None (это означает, что возникла проблема с загрузкой
40     # образ с диска, просто пропустите)
41     если изображение отсутствует :
42         Продолжать

```

В строке 34 мы начинаем перебирать каждое отдельно взятое изображение. Стока 36, затем случайным образом выбирает угол поворота, на который мы собираемся повернуть изображение — либо на 0 градусов (без изменений), 90 градусов, 180 градусов (по вертикали) или 270 градусов. Стока 37 загружает наш образ с диска.

Я обнаружил, что некоторые изображения в наборе данных Indoor CVPR не загружаются должным образом с диска из-за проблемы с кодировкой JPEG; таким образом, строки 41 и 42 обеспечивают правильную загрузку изображения, а если нет, мы просто пропускаем изображение.

Следующий блок кода будет обрабатывать поворот нашего изображения на случайно выбранный угол , а затем запись образа на диск:

```

44     # повернуть изображение на выбранный угол, затем построить
45     # путь к базовому выходному каталогу
46     изображение = imutils.rotate_bound (изображение, угол)
47     base = os.path.sep.join([args["выход"], str(угол)])
48
49     # если базовый путь еще не существует, создайте его
50     если нет os.path.exists (база):
51         os.makedirs (база)
52
53     # извлечь расширение файла изображения, затем создать полный путь
54     # в выходной файл
55     ext = imagePath[imagePath.rfind("."):]
56     outputPath = [base, "image_{}".format(
57         угол(углы.получить(угол, 0).zfill(5), доб))]
58     outputPath = os.path.sep.join(outputPath)
59
60     # сохранить изображение
61     cv2.imwrite (выходной путь, изображение)

```

Строка 46 поворачивает наше изображение на угол, чтобы все изображение оставалось в поле зрения (<http://pyimg.co/7xnpk6>). Затем мы снова определяем базовый выходной путь для изображения на основе угла. Это означает, что наш выходной набор данных будет иметь следующую структуру каталогов:

```
/output/{angle_name}/{image_filename}.jpg
```

Строки 50 и 51 проверяют, существует ли структура каталогов `/output/{angle_name}`, и если нет, создают требуемый подкаталог. Учитывая базовый выходной путь, мы можем получить путь к фактическому изображению в строках 55-58, извлекая: 1. Исходное расширение файла изображения.

2. Количество изображений, выбранных в данный момент с заданным углом.

Наконец, строка 61 записывает наше повернутое изображение на диск в подкаталог с правильным углом, гарантируя, что мы сможем легко определить метку изображения, просто изучив путь к файлу. Например, мы знаем, что путь к выходному изображению `/rotated_images/180/image_00000.jpg` был повернут на 180 градусов, потому что он находится в подкаталоге 180.

Для того, чтобы отслеживать количество изображений на ракурс, мы обновляем учет ракурсов словарь в следующем блоке кода:

```
63 # обновить счетчик угла c =
64 angles.get(angle, 0) angulars[angle] = c +
65 1 pbar.update(i)
66
```

Наконец, мы отображаем некоторую статистику по количеству изображений на угол в нашем наборе данных:

```
68 # завершаем индикатор выполнения 69 pbar.finish()

70
71 # цикл по углам и отображение счетчиков для каждого из них 72 для
угла в sorted(angles.keys()): print("[INFO] angle={}: {}".format(angle, angulars[angle]))
73
```

Чтобы создать наш набор данных повернутого изображения, откройте терминал и выполните следующую команду:

```
$ python create_dataset.py --datasetdoor_cvpr/images \ -outputdoor_cvpr/
rotated_images
Набор данных здания: 100% [########################################] | Время: 0:01:19
[ИНФОРМАЦИЯ] угол=0: 2487
[ИНФОРМАЦИЯ] угол = 90: 2480
[ИНФО] угол = 180: 2525
[ИНФО] угол = 270: 2483
```

Как видно из вывода, весь процесс создания набора данных занял 1 мин 19 с — 2500 изображений на угол поворота, плюс-минус несколько изображений из-за процесса случайной выборки. Теперь, когда мы создали наш набор данных, мы можем перейти к применению трансферного обучения посредством извлечения признаков — эти признаки затем будут использоваться в классификаторе логистической регрессии для прогнозирования (и исправления) ориентации входного изображения.

12.2 Извлечение признаков

Чтобы извлечь функции из нашего набора данных, мы будем использовать сетевую архитектуру VGG16, предварительно обученную на наборе данных ImageNet. Этот сценарий идентичен тому, который мы реализовали в главе 3 пакета «Практик». Причина, по которой мы можем повторно использовать один и тот же код, заключается в следующем:

1. Модульность инструментов, которые мы создавали на протяжении всей книги, гарантируя возможность повторного использования.

сценарии при условии, что наши входные наборы данных следуют определенной структуре каталогов.

2. Убедитесь, что наши изображения соответствуют шаблону каталога /dataset_name/class_label/example
_изображение.jpg

Для полноты картины я рассмотрю Extract_features.py ниже, но для получения более подробной информации, пожалуйста, обратитесь к Главе 3 пакета Practitioner Bundle:

```
1 # импортируем необходимые пакеты 2
из keras.applications import VGG16 3 из
keras.applications import imagenet_utils 4 из
keras.preprocessing.image import img_to_array 5 из
keras.preprocessing.image import load_img 6 из sklearn.preprocessing
import LabelEncoder 7 из pyimagesearch.io импортировать
HDF5DatasetWriter 8 из путей импорта imutils 9 импортировать
пимпу как np 10 импортировать индикатор выполнения 11
импортировать argparse 12 импортировать случайные 13
импортировать ОС
```

Строки 2-13 импортируют необходимые пакеты Python. Стока 2 импортирует сетевую архитектуру VGG16, которую мы будем рассматривать как экстрактор признаков. Функции, извлеченные CNN, будут записаны в набор данных HDF5 с помощью нашего HDF5DatasetWriter.

Далее, давайте проанализируем наши аргументы командной строки:

```
15 # построить разбор аргумента и разобрать аргументы 16 ap =
argparse.ArgumentParser() 17 ap.add_argument("-d", "--dataset",
required=True,
18     help="путь к входному набору
данных") 19 ap.add_argument("-o", "--output", required=True,
20     help="путь к выходному файлу HDF5")
21 ap.add_argument("-b", "--batch-size", type=int, default=32,
22     help="размер пакета изображений для передачи по сети") 23
ap.add_argument("-s", "--buffer-size", type=int, default=1000, help="размер буфера
извлечения объектов") 25 аргументов = переменные (ap.parse_args())
```

Здесь нам нужно указать входной путь к нашему --dataset повернутых изображений на диске вместе с --output путем к нашему файлу HDF5. Учитывая путь к нашему --dataset, мы можем получить пути к отдельным изображениям:

```
27 # сохранить размер партии в удобной переменной
28 бит = аргументы["размер_пакета"]
29
30 # возьмите список изображений, которые мы будем описывать, затем случайным образом 31 # перетасуйте их, чтобы упростить
обучение и тестирование сплитов с помощью
```

```

32 # нарезка массива во время обучения
33 print("[INFO] загрузка изображений...")
34 imagePaths = список (пути.list_images (аргументы ["набор данных"]))
35 random.shuffle (изображения)

```

Затем следует кодирование меток путем извлечения угла ориентации из пути изображения:

```

37 # извлечь метки классов из путей к изображениям, затем закодировать
38 # ярлыки
39 метка = [p.split(os.path.sep)[-2] для p в imagePaths]
40 le = LabelEncoder()
41 метка = le.fit_transform(метки)

```

Следующий блок кода обрабатывает загрузку нашей предварительно обученной сети VGG16 с диска, обеспечивая остаются слои FC (что позволяет нам выполнять извлечение признаков):

```

43 # загрузить сеть VGG16
44 print("[INFO] загрузка сети...")
45 модель = VGG16(веса="imagenet", include_top=False)
46
47 # инициализировать средство записи набора данных HDF5, затем сохранить метку класса
48 # имен в наборе данных
49 набор данных = HDF5DatasetWriter((len(imagePaths), 512 * 7 * 7),
50         args["output"], dataKey="features", bufSize=args["buffer_size"])
51 набор данных.storeClassLabels(le.classes_)

```

Мы также инициализируем наш HDF5DatasetWriter для записи извлеченных функций на диск. И я инициализируем индикатор выполнения, чтобы мы могли отслеживать процесс извлечения признаков:

```

53 # инициализировать индикатор выполнения
54 widgets = ["Извлечение признаков: ", progressbar.Percentage(), " ",
55             progressbar.Bar(), " ", progressbar.ETA()]
56 pbar = progressbar.ProgressBar (maxval = len (imagePaths),
57         виджеты=виджеты).start()

```

Теперь мы готовы применить трансферное обучение с помощью извлечения признаков:

```

59 # цикл по изображениям в патчах
60 для i в пр.arange(0, len(imagePaths), bs):
61     # извлечь пакет изображений и меток, затем инициализировать
62     # список актуальных изображений, которые будут передаваться по сети
63     # для извлечения признаков
64     batchPaths = imagePaths[i:i + bs]
65     batchLabels = метки[i:i + bs]
66     пакетные изображения = []
67
68     # цикл по изображениям и меткам в текущем пакете
69     для (j, imagePath) в перечислении (batchPaths):
70         # загружаем входное изображение с помощью вспомогательной утилиты Keras
71         # при изменении размера изображения до 224x224 пикселей
72         изображение = load_img (imagePath, target_size = (224, 224))

```

```

73     изображение = img_to_array(изображение)
74
75     # предварительно обработать изображение путем (1) увеличения размеров и
76     # (2) вычитание средней интенсивности пикселя RGB из
77     # Набор данных ImageNet
78     изображение = np.expand_dims(изображение, ось = 0)
79     изображение = imagenet_utils.preprocess_input(изображение)
80
81     # добавляем изображение в пакет
82     batchImages.append(изображение)

```

Строка 60 начинает цикл по всем изображениям в нашем списке `imagePaths`. Для каждого из изображения, мы загружаем их с диска, предварительно обрабатываем и сохраняем в списке `batchImages`. Затем пакетные изображения передаются через предварительно обученную сеть VGG16, что дает нам наши функции, которые записываются в набор данных HDF5:

```

84     # передавать изображения по сети и использовать выходные данные как
85     # наши актуальные возможности
86
87     пакетные изображения = np.vstack(пакетные изображения)
88     функции = model.predict(batchImages, batch_size = bs)
89
90     # изменить форму объектов так, чтобы каждое изображение было представлено
91     # слаженный вектор признаков выходных данных 'MaxPooling2D'
92     функции = функции.изменить((особенности.форма[0], 512 * 7 * 7))
93
94     # добавляем функции и метки в наш набор данных HDF5
95     dataset.add(функции, пакетные метки)
96     pbar.update(я)
97 # закрыть набор данных
98 набор данных.close()
99 pbar.finish()

```

Чтобы извлечь функции из нашего набора данных повернутых изображений, просто выполните следующую команду:

```

$ python extract_features.py --dataset indoor_cvpr/rotated_images \
    --outputdoor_cvpr/hdf5/orientation_features.hdf5
[INFO] загрузка изображений...
[INFO] загрузка сети...
Особенности извлечения: 100% [########################################| Время: 0:02:13

```

В зависимости от того, используете ли вы ЦП или ГП, этот процесс может занять несколько минут. несколько часов. После завершения процесса вы найдете файл с именем `ориентация_features.hdf5`. в вашем выходном каталоге:

```

$ ls -l крытый_cvpr/hdf5/
всего 1955192
-rw-rw-r-- 1 адриан адриан 2002108504 21 ноября 2016 г. ориентация_features.hdf5

```

Этот файл содержит функции, извлеченные нашей CNN, поверх которых мы будем обучать Logistic. Регрессионный классификатор. Опять же, для получения дополнительной информации о переносе обучения с помощью извлечения признаков, пожалуйста, см. главу 3 комплекта для практикующих.

12.3 Обучение классификатора коррекции ориентации

Обучение классификатора для прогнозирования ориентации изображения также будет выполняться с использованием скрипта `train_model.py` из пакета Practitioner Bundle — нам просто нужно указать путь к нашему входному набору данных HDF5, и наш скрипт позаботится о настройке гиперпараметров логистической регрессии и записи вывода модели на диск. Опять же, для полноты картины я рассмотрю `train_model.py` ниже, но вы можете обратиться к главе 3 пакета Practitioner Bundle для получения более подробной информации об этом скрипте.

```

1 # импортируем необходимые пакеты 2
из sklearn.linear_model import LogisticRegression 3 из
sklearn.model_selection import GridSearchCV 4 из sklearn.metrics
import classification_report 5 import argparse 6 import pickle 7 import
h5py
8
9 # построить разбор аргумента и разобрать аргументы 10 ap =
argparse.ArgumentParser() 11 ap.add_argument("-d", "--db", required=True,
help="path HDF5 database")
12
13 ap.add_argument("-m", "--model", required=True,
14     help="путь к выходной модели") 15
ap.add_argument("-j", "--jobs", type=int, default=-1,
16     help="количество заданий для запуска при настройке
гиперпараметров") 17 args = vars(ap.parse_args())

```

Строки 2-7 импортируют наши необходимые пакеты Python, а строки 10-17 анализируют наши аргументы командной строки. Здесь нам нужно указать только два аргумента командной строки: `--db`, который является путем к нашему входному набору данных HDF5, и `--model`, путь к нашему выходному сериализованному классификатору логистической регрессии после его обучения.

Затем мы строим разделение обучения и тестирования на основе количества записей в базе данных, используя 75% данных для тестирования и 25% для тестирования:

```

19 # открыть базу данных HDF5 для чтения, затем определить индекс 20 # разделения
обучения и тестирования при условии, что эти данные уже были перемешаны 21 #
до записи на диск 22 db = h5py.File(args["db"], "r") 23 i = int(db["labels"].shape[0] * 0,75)

```

Чтобы настроить параметры нашего классификатора логистической регрессии, мы выполним поиск по сетке ниже:

```

25 # определяем набор параметров, которые мы хотим настроить, затем запускаем
поиск по сетке 26 #, где мы оцениваем нашу модель для каждого значения C 27
print("[INFO] настройки гиперпараметров...") 28 params = {"C" : [0,01, 0,1, 1,0, 10,0,
100,0, 1000,0, 10000,0]} 29 модель = GridSearchCV(LogisticRegression(), params, cv=3,
30     n_jobs=args["jobs"]) 31
модель.fit(db["features"][:i], db["labels"][:i]) 32 print("[INFO] лучшие
гиперпараметры: {}".format(модель.best_params_))

```

На основе лучших гиперпараметров мы оценим модель на данных тестирования:

```

34 # оценить модель
35 print("[INFO] оценка...")
36 pres = model.predict(db["features"][:i:])
37 print(classification_report(db["labels"][:i:], pres,
38     target_names=db["label_names"]))

```

А затем, наконец, записать обученный классификатор на диск:

```

40 # сериализовать модель на диск
41 print("[INFO] сохранение модели...")
42 f = открыть(args["модель"], "wb")
43 f.write(pickle.dumps(model.best_estimator_))
44 ф.закрыть()
45
46 # закрыть базу
47 db.close()

```

Чтобы обучить наш классификатор логистической регрессии функциям, извлеченным из нашей сети VGG16, просто выполните следующую команду:

```

$ python train_model.py --db indoor_cvpr/hdf5/orientation_features.hdf5 \
    --model модели/ориентация.cpickle
[INFO] настройка гиперпараметров...
[INFO] лучшие гиперпараметры: {'C': 0,01}
[INFO] оценка...

      точность      вспомнить поддержку f1-score

      0        0,95        0,93        0,94        646
      180       0,92        0,94        0,93        612
      270       0,92        0,92        0,92        635
      90        0,93        0,94        0,93        601

среднее / общее        0,93        0,93        0,93        2494

```

Как только `train_model.py` завершит выполнение, вы заметите, что точность нашего классификатора составляет 93 %, что означает, что в 93 % случаев наша модель может правильно предсказать ориентацию изображения. Учитывая, что мы можем предсказать ориентацию изображения, мы можем исправить ее следующим образом.

Опять же, здесь стоит упомянуть, что мы не смогли бы получить такой высокий прогноз точность, если VGG16 действительно инвариантен к вращению. Вместо этого (коллективные) фильтры внутри VGG16 способен распознавать объекты в различных поворотах — отдельные фильтры сами по себе не вращаются инвариант. Если бы эти фильтры действительно были инвариантны к вращению, то мы не смогли бы определить ориентацию изображения из-за того, что извлеченные функции будут почти идентичными независимо от ориентации изображения.

Это важный урок, который следует усвоить при изучении сверточных нейронных сетей. И рассчитывайте развернуть их в реальных ситуациях. Если вы намерены распознавать объекты под любую ориентацию, вы должны убедиться, что ваши тренировочные данные в достаточной мере отражают это, или использовать извлечение признаков + метод машинного обучения, который естественным образом обрабатывает ротацию.

12.4 Исправление ориентации

Чтобы продемонстрировать, что наш классификатор извлечения признаков VGG16 + логистическая регрессия может в достаточной степени классифицировать изображения, давайте напишем скрипт Python, который применяет конвейер к входным изображениям. Откройте новый файл, назовите его `orient_images.py` и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2
из keras.applications import VGG16 3 из
keras.applications import imagenet_utils 4 из
keras.preprocessing.image import img_to_array 5 из
keras.preprocessing.image import load_img 6 из imutils import paths
7 import numpy as np 8 import argparse 9 импорт pickle 10 импорт
imutils 11 импорт h5py 12 импорт cv2
```

Начнем с импорта необходимых пакетов Python. Обратите внимание, что нам снова понадобится VGG16 для извлечения признаков. Мы также загрузим некоторые вспомогательные утилиты, используемые для облегчения загрузки нашего входного изображения с диска и подготовки его к извлечению признаков с помощью VGG16.

Далее, давайте проанализируем наши аргументы командной строки:

```
14 # построить разбор аргумента и разобрать аргументы 15 ap =
argparse.ArgumentParser() 16 ap.add_argument("-d", "--db", required=True,
help="path HDF5 database")
17
18 ap.add_argument("-i", "--dataset", required=True, help="путь к
19     набору входных изображений") 20 ap.add_argument("-
m", "--model", required=True , help="путь к обученной модели
21     ориентации") 22 args = vars(ap.parse_args())
```

Наш сценарий требует три аргумента командной строки, каждый из которых подробно описан ниже:

- `--db`:

Здесь мы указываем путь к нашему набору данных HDF5 извлеченных объектов, находящихся на диске.

Нам нужен только этот набор данных, чтобы мы могли извлечь имена меток (т. е. углы) из файла HDF5, иначе этот переключатель не понадобился бы.

• `--dataset`: это путь к нашему набору данных повернутых изображений, находящихся на диске.

• `--model`: здесь мы можем указать путь к нашему классификатору логистической регрессии, используемому для предсказать ориентацию изображений.

Давайте продолжим и загрузим имена меток из нашего набора данных HDF5, а затем создадим образец

из десяти изображений из нашего набора входных данных, которые мы будем использовать для визуализации нашей коррекции ориентации:

```
24 # загрузить имена меток (т.е. углы) из набора данных HDF5 25 db =
h5py.File(args["db"]) 26 labelNames = [int(angle) for angle in db["label_names"][:]]
27 db.close()
```

28

29 # получаем пути к тестовым изображениям и случайным образом сэмплируем их 30 print("[INFO] сэмплировать изображения...")

```
31 imagePaths = список (пути.list_images (аргументы ["набор данных"]))
32 imagePaths = np.random.choice(imagePaths, size=(10,), replace=False)
```

Мы также загрузим с диска сетевую архитектуру VGG16 и классификатор логистической регрессии. также:

```
34 # загружаем сеть VGG16
35 print("[INFO] загрузка сети...")
36 vgg = VGG16(weights="imagenet", include_top=False)
37
38 # загрузить модель ориентации
39 print("[INFO] загрузка модели...")
40 модель = pickle.loads(open(args["model"], "rb").read())
```

Ядро конвейера коррекции ориентации можно найти ниже:

```
42 # цикл по путям изображений
43 для imagePath в imagePaths:
44     # загрузить изображение через OpenCV, чтобы мы могли манипулировать им после
        # классификация
45     orig = cv2.imread (путь к изображению)
46
47     # загрузите входное изображение с помощью вспомогательной утилиты Keras, пока
48     # обеспечение изменения размера изображения до 224x224 пикселей
49     изображение = load_img (imagePath, target_size = (224, 224))
50     изображение = img_to_array (изображение)
51
52     # предварительно обработать изображение путем (1) увеличения размеров и (2)
53     # вычитание средней интенсивности пикселей RGB из ImageNet
54     # набор данных
55     изображение = np.expand_dims (изображение, ось = 0)
56     изображение = imagenet_utils.preprocess_input(изображение)
57
58     # пропустить изображение через сеть, чтобы получить вектор признаков
59     функции = vgg.predict(изображение)
60     функции = функции.изменить((особенности.форма[0], 512 * 7 * 7))
```

В строке 43 мы перебираем каждый из отдельных путей выборки изображения. Для каждого из этих изображений мы загружаем оригинал с диска через OpenCV (строка 46), чтобы мы могли исправить его ориентацию позже в сценарий. Строки 50 и 51 загружают образ с диска с помощью вспомогательной утилиты Keras, гарантируя, что каналы изображения правильно упорядочены.

Затем мы продолжаем предварительную обработку изображения для классификации в строках 56 и 67. Стока 60 выполняет прямой проход, пропуская изображение через vgg и получая выходные признаки. Затем эти функции сглаживаются из массива $512 \times 7 \times 7$ в сглаженный список из 9 975 элементов с плавающей запятой. Выравнивание вывода слоя POOL является требованием перед передачей изображений через классификатор логистической регрессии, который мы делаем ниже:

```
63     # теперь, когда у нас есть функции CNN, передайте их через наш
64     # классификатор для получения прогнозов ориентации
65     угол = модель.predict(функции)
66     угол = меткиИмена[угол[0]]
```

Учитывая предсказанный угол из входных объектов, теперь мы можем исправить ориентацию изображение и вывести на экран как исходное, так и исправленное изображение:

```

68     # теперь, когда у нас есть предсказанная ориентация изображения, мы можем
69     # исправьте это
70     повернуто = imutils.rotate_bound(оригинал, 360 - угол)
71
72     # отображать исходное и исправленное изображения
73     cv2.imshow("Оригинал", оригинал)
74     cv2.imshow("«Исправлено» , повернуто")
75     cv2.waitKey(0)

```

Чтобы увидеть наш скрипт коррекции ориентации в действии, выполните следующую команду:

```
$ python orient_images.py --db indoor_cvpr/hdf5/orientation_features.hdf5 \
--dataset internal_cvpr/rotated_images --модели моделей/ориентация.cpickle
```



Рисунок 12.2: Исправление ориентации изображения с помощью признаков, извлеченных из предварительно обученной CNN.

Результаты нашего алгоритма можно увидеть на рис. 12.2. Слева у нас есть примеры двух оригинальные, необработанные изображения. Затем справа у нас есть наши изображения после применения ориентации коррекция. В каждом из этих случаев мы смогли предсказать угол поворота исходного изображения. а затем исправьте это в выходном изображении.

Подобные алгоритмы и конвейеры глубокого обучения можно использовать для обработки больших наборов данных, возможно, отсканированных изображений, где ориентация входных изображений неизвестна. Применение такого конвейера значительно сократит количество времени, затрачиваемого человеком на ручное исправление ошибок. ориентации перед сохранением изображения в электронной базе данных.

12.5 Резюме

В этом тематическом исследовании мы научились предсказывать ориентацию изображений без априорных знаний. Для выполнения этой задачи мы создали размеченный набор данных, в котором изображения были случайным образом повернуты с помощью

{0,90,180,360} градусов. Затем мы применили трансферное обучение с помощью извлечения признаков и сетевой архитектуры VGG16, чтобы извлечь признаки из последнего уровня POOL в сети. Эти функции были переданы в классификатор логистической регрессии, что позволило нам правильно предсказать ориентацию изображения с точностью 93%. Сочетание сети VGG16 и нашей обученной модели логистической регрессии позволило нам построить конвейер, который может автоматически определять и исправлять ориентацию изображения.

Наконец, еще раз стоит упомянуть, что этот результат был бы невозможен, если бы наша СНС была действительно инвариантной к вращению. CNN являются надежными инструментами классификации изображений и могут правильно классифицировать изображения в различных ориентациях; однако отдельные фильтры внутри CNN не инвариантны к вращению.

Кроме того, CNN могут классифицировать только те объекты, на которых они были обучены. Конечно, здесь тоже есть некоторая степень обобщения; однако, если бы эти фильтры действительно были инвариантны к вращению, функции, которые мы извлекли из наших входных изображений, были бы почти идентичны друг другу. Если бы функции были почти идентичными, то классификатор логистической регрессии не смог бы различать ориентацию наших изображений. Это важный урок, который следует учитывать при разработке собственных приложений для глубокого обучения. Если вы ожидаете, что входные объекты будут существовать во многих ориентациях, убедитесь, что ваши обучающие данные отражают это требование.

13. Практический пример: идентификация автомобиля

Вы когда-нибудь смотрели фильм «Особое мнение»? В фильме есть сцена, где главный герой (Том Круз) идет по переполненному торговому центру. Комбинация датчиков камеры, установленных рядом с каждым ЖК-дисплеем, наряду со специализированными алгоритмами компьютерного зрения, способна (1) распознавать, кто он, и (2) отображать целевую рекламу на основе его конкретного профиля.

Еще в 2002 г., когда был опубликован отчет меньшинства, такая надежная таргетированная реклама звучала как научная фантастика. Но правда в том, что мы не так уж далеки от этого. Представьте, что вы едете на машине по шоссе, заполненному рекламными щитами — только вместо плакатов, расклеенных на большой стене, это будут массивные ЖК-экраны. Затем, в зависимости от того, какой тип автомобиля вы водите, вы увидите другую рекламу. Интересы человека, управляющего новым BMW 5-й серии, могут получить рекламу уроков игры в гольф, в то время как водителя минивэна более поздней модели может привлечь реклама о том, как сэкономить деньги для колледжа их детей. Сомнительно, отражают ли эти объявления интересы отдельного водителя или нет, но в целом реклама работает следующим образом:

1. Проанализируйте демографические данные.

2. Определите их интересы.
3. Показывать рекламу, отражающую эти интересы.

В этой главе мы узнаем, как создать компонент компьютерного зрения для интеллектуальной системы рекламных щитов. В частности, мы узнаем, как настроить предварительно обученную CNN с библиотекой mxnet для распознавания более 164 марок и моделей автомобилей (используя очень мало обучающих данных) с точностью более 96,54%.

13.1 Набор данных Stanford Cars

Стэнфордский набор данных об автомобилях представляет собой набор из 16 185 изображений 196 автомобилей (рис. 13.1), составленный Krause et al. в их публикации 2013 г. «3D Object Representation for Fine-Grained Classification» [37].

Вы можете скачать архив набора данных здесь: <http://ruimg.co/9s9mx>

После загрузки используйте следующую команду, чтобы распаковать набор данных:



Рисунок 13.1: Набор данных Stanford Cars состоит из 16 185 изображений 196 марок автомобилей и модельные классы.

```
$ tar -xvf car_ims.tar.gz
```

Мы обсудим этот набор данных более подробно в Разделе 13.1.1. Цель оригинального Krause et др. paper состоит в том, чтобы правильно предсказать марку, модель и год выпуска данного автомобиля. Однако, учитывая, что Существуют:

1. Экстремальный дисбаланс классов в наборе данных, когда некоторые марки и модели автомобилей представлены значительно преувеличенно (например, у Audi и BMW более 1000 точек данных, в то время как у Tesla всего 77 примеров).

2. Очень мало данных даже для больших классов.

Я решил удалить метку года и вместо этого классифицировать изображения строго по их марке и модель. Даже при такой категоризации бывают случаи, когда у нас есть < 100 изображений. на класс, что очень усложняет тонкую настройку глубокой CNN на этом наборе данных. Однако с правильный подход к тонкой настройке, мы по-прежнему сможем получить > 95% точности классификации.

После удаления метки года у нас осталось 164 марки и модельных классов автомобилей, которые можно было бы использовать. распознавать. Нашей целью будет тонкая настройка VGG16 для идентификации каждого из 164 классов.

13.1.1 Создание набора данных Stanford Cars

Чтобы точно настроить VGG16 в наборе данных Stanford Cars, нам сначала нужно сжать изображения в эффективно упакованные файлы записей, как в наших главах по обучению ImageNet. Но прежде чем мы получим до сих пор давайте взглянем на структуру каталогов для нашего проекта:

```
--- car_classification
| --- конфигурация
| | --- __init__.py
| | --- car_config.py
| | --- build_dataset.py
| | --- Fine_tune_cars.py
| | --- test_cars.py
| | --- vgg16
| | | --- vgg16-0000.params
| | | --- vgg16-symbol.json
| | --- vis_classification.py
```

У нас снова будет модуль конфигурации , в котором мы будем хранить car_config.py — этот файл будет

содержит все необходимые конфигурации для создания набора данных Stanford Cars и тонкой настройки VGG16. Кроме того, потребуется значительно меньше переменных конфигурации.

`build_dataset.py`, как следует из названия, будет вводить наши пути к изображениям и метки классов, а затем создавать файл `.lst` для каждого из разделов обучения, проверки и тестирования соответственно. Получив эти файлы `.lst`, мы можем применить двоичный файл `mxnet im2rec` для создания базы данных записей.

Сценарий `Fine_tune_cars.py` будет отвечать за тонкую настройку VGG16 в нашем наборе данных. После того, как мы успешно настроим VGG16, мы захотим оценить производительность на тестовом наборе — это именно то, что выполнит `test_cars.py`.

Конечно, оценка сети на предварительно созданном наборе данных не дает вам никакого представления о том, как применить CNN, обученную с помощью `mxnet`, к одномуциальному изображению. Чтобы узнать, как дальше классифицировать одно изображение (и продемонстрировать, как вы можете создать интеллектуальный рекламный щит), мы создадим `vis_classification.py`. Этот скрипт загрузит входное изображение с диска, предварительно обработает его, пропустит через наш точно настроенный VGG16 и отобразит выходные прогнозы.

Файл конфигурации Stanford Cars Прежде чем

мы сможем создать наши файлы `.lst` и `.rec`, нам сначала нужно создать наш файл `car_config.py`. Откройте `car_config.py` прямо сейчас, и мы просмотрим его содержимое:

```

1 # импортируем необходимые пакеты 2 из
2 # импорта ОС
3
4 # определяем базовый путь к набору данных cars
5 BASE_PATH = "/raid/наборы данных/автомобили"
6
7 # на основе базового пути получить путь к изображениям и путь к метафайлу 8 IMAGES_PATH =
8 IMAGES_PATH =
9 LABELS_PATH = path.sep.join([BASE_PATH, "complete_dataset.csv"])

```

В строке 5 я определяю `BASE_PATH` для своего набора данных Stanford Cars. Внутри `BASE_PATH` вы найдете мой неархивированный каталог `cars_ims` вместе с моими контрольными точками, списками, выходными данными и каталогами `rec`, которые используются для хранения контрольных точек модели, файлов списка наборов данных, выходных данных модели, необработанных изображений и наборов данных, закодированных `im2rec`:

```

$ ls
car_ims car_ims.tgz checkpoints complete_dataset.csv списки выходных записей

```

Эта конфигурация — то, как я лично предпочитаю определять свою структуру каталогов для данного набора данных, но вы должны настроить свою среду так, как вам наиболее удобно — просто имейте в виду, что вам нужно будет обновить `BASE_PATH` и любые другие пути, чтобы они соответствовали вашим системам.

 Файл `complete_dataset.csv` — это пользовательский CSV-файл, включенный в загрузку Deep Learning for Computer Vision with Python. Вы можете найти файл `complete_dataset.csv` в каталоге кода, связанном с этой главой. Пожалуйста, либо (1) скопируйте `complete_dataset.csv` в свой набор данных `cars`, как указано в структуре каталогов выше, либо (2) соответствующим образом обновите `LABELS_PATH`.

Строка 8 использует `BASE_PATH` для получения `IMAGES_PATH`, где находятся входные изображения для набора данных Stanford Cars. Как видите, имеется 16 185 файлов JPEG, каждый из которых назван последовательно:

```
$ ls -l car_ims/*.jpg | туалет -л 16185
$ ls -l car_ims/*.jpg | head -n 5 -rw-r--r-- 1
adrian adrian 466876 28 февраля 2015 car_ims/000001.jpg -rw-r--r-- 1 adrian adrian
25139 28 февраля 2015 car_ims/000002.jpg -rw-r--r-- 1 adrian adrian 19451 28 февраля
2015 car_ims/000003.jpg -rw-r--r-- 1 adrian adrian 16089 28 февраля 2015 car_ims/
000004.jpg -rw-r--r-- 1 adrian adrian 1863 28 февраля 2015 car_ims/000005.jpg
```

Затем в строке 9 определяется путь к нашему файлу `complete_dataset.csv`. Этот файл на самом деле не является частью набора данных, созданного Krause et al. Вместо этого оригинальный DevKit, предоставленный Krause et al. включены метафайлы MATLAB, которые было трудно проанализировать, поскольку все три марки, модели и названия транспортных средств были сохранены в виде одной строки. Чтобы упростить работу с набором данных Stanford Cars, я создал файл `complete_dataset.csv`, в котором перечислены имя файла изображения, марка автомобиля, модель, тип (седан, купе и т. д.) и год выпуска, разделенные удобной запятой. файл. Используя этот файл, мы сможем более легко построить наш набор данных.

Далее мы определим путь к нашим выходным файлам обучения, проверки и тестирования `.lst`:

11 # определить путь к выходным спискам обучения, проверки и тестирования 12 #

```
13 MX_OUTPUT = BASE_PATH 14
TRAIN_MX_LIST = path.sep.join([MX_OUTPUT, "lists/train.lst"])
15 VAL_MX_LIST = path.sep.join([MX_OUTPUT, "lists/val.lst"])
16 TEST_MX_LIST = path.sep.join([MX_OUTPUT, "lists/test.lst"])
```

А также пути к нашим файлам `.rec` для каждого из разбиений данных:

18 # определить путь к выходным данным обучения, проверки и тестирования 19 #
записи изображений 20 TRAIN_MX_REC = path.sep.join([MX_OUTPUT, "rec/train.rec"])

```
21 VAL_MX_REC = path.sep.join([MX_OUTPUT, "rec/val.rec"])
22 TEST_MX_REC = path.sep.join([MX_OUTPUT, "rec/test.rec"])
```

Обратите внимание, как я использую переменную `path.sep`, предоставляемую Python, чтобы сделать эти конфигурации настолько (в разумных пределах) переносимыми, насколько это возможно. Нам нужно будет закодировать удобочитаемую марку автомобиля + названия моделей в виде целых чисел во время обучения, для чего потребуется кодировщик меток:

24 # определяем путь к кодировщику меток 25
LABEL_ENCODER_PATH = path.sep.join([BASE_PATH, "output/le.cpickle"])

Этот кодировщик также позволит нам выполнить обратное преобразование и получить удобочитаемые имена из меток целочисленного класса.

Чтобы выполнить нормализацию среднего, нам нужно определить средние значения RGB:

27 # определить средние значения RGB из набора данных
ImageNet 28 R_MEAN = 123,68 29 G_MEAN = 116,779 30 B_MEAN =
103,939

Имейте в виду, что это средние значения RGB из оригинальной статьи Симоняна и Зиссермана [17]. Поскольку мы занимаемся точной настройкой VGG, мы должны использовать средние значения RGB, которые они получили из набора данных ImageNet — для нас не имеет никакого смысла вычислять средние значения RGB для набора данных об автомобилях, а затем использовать их для точной настройки VGG. Нам нужны исходные средства RGB, на которых был обучен VGG16 , чтобы успешно настроить сеть для распознавания марок и моделей автомобилей.

Далее нам нужно определить общее количество классов, а также количество проверок и

тестовые изображения, которые мы будем использовать:

```
32 # определить процент проверочных и тестовых изображений относительно 33 #
количество обучающих изображений 34 NUM_CLASSES = 164 35 NUM_VAL_IMAGES = 0,15
36 NUM_TEST_IMAGES = 0,15
```

Наш CNN должен уметь распознавать 164 различных марки и модели транспортных средств (строка 34). Затем мы возьмем 30% исходного набора данных и используем 15% для проверки и 15% для тестирования, оставив 70% для обучения.

Наш последний блок кода обрабатывает настройку размера пакета и количества устройств для обучения:

```
38 # определить размер партии
39 BATCH_SIZE = 32
40 NUM_DEVICES = 1
```

Мы будем использовать мини-пакет из 32 изображений вместе с одним графическим процессором для тонкой настройки VGG на наборе данных автомобилей. Как мы узнаем, тонкая настройка VGG на наборе данных Stanford Cars выполняется очень быстро из-за небольшого набора данных — мы сможем завершить одну эпоху примерно за 10 минут с помощью всего одного графического процессора.

Создание файлов

списка Теперь, когда наш файл конфигурации создан, мы можем перейти к сценарию build_dataset.py . Как и в наших экспериментах с ImageNet, этот сценарий будет отвечать за создание файлов .lst для обучения, проверки и тестирования . Откройте build_dataset.py и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2 из
config import car_config as config 3 из sklearn.preprocessing
import LabelEncoder 4 из sklearn.model_selection import
train_test_split 5 import progressbar 6 import pickle 7 import os
```

⁸
9 # прочитать содержимое файла этикеток, затем инициализировать список из 10 # путей
к изображениям и меток 11 print("[INFO] загрузка путей и меток к изображениям...") 12 rows
= open(config.LABELS_PATH).read() 13 строк = rows.strip().split("\n")[1:] 14 trainPaths = [] 15
trainLabels = []

Строки 2-7 импортируют необходимые пакеты Python. Мы обязательно импортируем наш модуль car_config (с псевдонимом config) , чтобы мы могли получить доступ к нашим путям к файлам и дополнительным конфигурациям.

Функция `train_test_split` будет использоваться для создания разделов проверки и тестирования соответственно.

Затем мы читаем содержимое файла `LABELS_PATH` (т. е. `complete_dataset.csv`) при инициализации двух списков: один для наших путей к обучающим изображениям, а другой — для наших обучающих меток.

Теперь, когда мы загрузили содержимое нашего файла меток, давайте перейдем к каждой строке по отдельности:

17 # петля по рядам 18 для

ряда в рядах:

```
19      # распаковать строку, затем обновить список путей и меток к изображениям
20      # (filename, make) = row.split(",")[:2] (filename, make, model) = row.split(",")[:3]
21      имя_файла = имя_файла[имя_файла.rfind("/") + 1:]
22      trainPaths.append(os.sep.join([config.IMAGES_PATH, имя_файла]))
23      trainLabels.append("{}:{}".format(марка, модель))
24
```

Для каждой строки мы разбиваем строку, разделенную запятыми, на три значения: имя файла, марка автомобиля и модель автомобиля. На данный момент имя файла представлено как `car_ims/000090.jpg`, поэтому мы анализируем путь и удаляем подкаталог, оставляя имя файла как `000090.jpg` (строка 22). Мы обновляем список `trainPaths`, указав путь к нашему входному тренировочному изображению, объединив наш `IMAGES_PATH` с именем файла. Затем создается метка путем объединения марки и модели, после чего она добавляется в список `trainLabels`.

Теперь, когда мы проанализировали весь файл `LABELS_PATH`, нам нужно вычислить общее количество файлов проверки и тестирования, которые нам понадобятся, на основе процентов `NUM_VAL_IMAGES` и `NUM_TEST_IMAGES`:

26 # теперь, когда у нас есть общее количество изображений в наборе данных,
которые 27 # можно использовать для обучения, вычислите количество изображений,
которые 28 # следует использовать для проверки и тестирования 29 numVal =
int(len(trainPaths) * config.NUM_VAL_IMAGES) 30 numTest = int(len(trainPaths) *
config.NUM_TEST_IMAGES)

Мы также потратим время на то, чтобы преобразовать метки наших строк в целые числа:

32 # метки нашего класса представлены в виде строк, поэтому нам нужно кодировать
33 # их 34 print("[INFO] метки кодирования...") 35 le = LabelEncoder().fit(trainLabels) 36
trainLabels = le.transform(trainLabels)

Как упоминалось ранее, у нас нет предустановленного набора для проверки и тестирования, поэтому нам нужно создать их из обучающей выборки. Следующий блок кода создает каждое из этих разделений:

38 # выполнить выборку из обучающего набора для построения валидации 39 # набор

```
40 print("[INFO] создание проверочных данных...") 41 split =
train_test_split(trainPaths, trainLabels, test_size=numVal,
42                 stratify=trainLabels) 43
(trainPaths, valPaths, trainLabels, valLabels) = разделить
44
45 # выполнить стратифицированную выборку из обучающей выборки для построения
46 # тестовой выборки
```

```

47 print("[INFO] создание тестовых данных...")
48 split = train_test_split(trainPaths, trainLabels, test_size=numTest,
49     stratify=trainLabels)
50 (trainPaths, testPaths, trainLabels, testLabels) = разделить

```

Далее давайте создадим наш список наборов данных , где каждая запись представляет собой 4-кортеж, содержащий тип разделения, пути к изображениям, соответствующие метки классов и путь к выходному файлу .lst:

```

52 # составить список, объединяющий обучение, проверку и тестирование
53 # пути к изображениям вместе с соответствующими метками и списком вывода
54 # файлов
55 наборов данных = [
56     ("поезд", trainPaths, trainLabels, config.TRAIN_MX_LIST),
57     ("val", valPaths, valLabels, config.VAL_MX_LIST),
58     ("тест", testPaths, testLabels, config.TEST_MX_LIST)]

```

Чтобы создать каждый файл .lst, нам нужно перебрать каждую запись в наборах данных:

```

60 # цикл по кортежам набора данных
61 для (dType, paths, labels, outputPath) в наборах данных:
62     # открываем выходной файл для записи
63     print("[INFO]здание {}".format(outputPath))
64     f = открыть (выходной путь, "w")
65
66     # инициализируем индикатор выполнения
67     widgets = ["Список построек: ", progressbar.Percentage(), " ",
68                progressbar.Bar(), " ", progressbar.ETA()]
69     pbar = progressbar.ProgressBar (maxval = len (пути),
70                                    виджеты=виджеты).start()

```

Строка 64 открывает указатель файла на текущий outputPath. Затем мы можем перебрать каждый из отдельные пути к изображениям и метки, записывая их в выходной файл .lst, как мы это делаем:

```

72     # перебираем каждое отдельное изображение + метки
73     для (i, (путь, метка)) в перечислении (zip (пути, метки)):
74         # записываем индекс изображения, метку и выходной путь в файл
75         row = "\t".join([str(i), str(label), path])
76         f.write("{}\n.формат(строка)")
77         pbar.update(я)
78
79     # закрываем выходной файл
80     pbar.finish()
81     е.закрыть()

```

Наш последний блок кода обрабатывает сохранение сериализованного кодировщика меток на диск, чтобы мы могли повторно использовать его позже. В этой главе:

```

83 # записать кодировщик меток в файл
84 print("[INFO]сериализующий кодировщик этикеток...")
85 f = открыть (config.LABEL_ENCODER_PATH, "wb")
86 f.write(pickle.dumps(le))
87 ф.закрыть()

```

Чтобы создать файлы .lst для обучения, проверки и тестирования, выполните следующую команду:

```
$ python build_dataset.py [INFO]
загрузка путей и меток изображений...
[INFO] метки кодировки...
[INFO] создание проверочных данных...
[INFO] построение тестовых данных...
[ИНФО] здание /raid/datasets/cars/lists/train.lst...
Список построек: 100% ##### | Время: 0:00:00
[ИНФО] здание /raid/datasets/cars/lists/val.lst...
Список построек: 100% ##### | Время: 0:00:00
[ИНФО] здание /raid/datasets/cars/lists/test.lst...
Список построек: 100% ##### | Время: 0:00:00
[INFO] сериализация кодировщика этикеток...
```

После того, как команда закончит работу, вы должны убедиться, что у вас действительно есть файл train.lst, файлы test.lst и val.lst в вашей системе:

```
$ wc -l списки/*.lst
2427 списков/test.lst
11331 списков/train.lst
2427 списков/val.lst
16185 всего
```

Создание базы данных записей Имея

файлы .lst для каждого из разделений, теперь нам нужно сгенерировать файлы записей mxnet. Ниже вы можете увидеть команду для создания файла train.rec , который представляет собой тот же путь к файлу, что и в конфигурации TRAIN_MX_REC:

```
$ ~/mxnet/bin/im2rec /raid/datasets/cars/lists/train.lst "" \
/raid/datasets/cars/rec/train.rec resize=256 encoding='.jpg' \ качество=100
...
[12:23:13] tools/im2rec.cc:298: Всего: обработано 11331 изображения, прошло 124,967 с.
```

Вы можете сгенерировать набор данных test.rec с помощью этой команды:

```
$ ~/mxnet/bin/im2rec /raid/datasets/cars/lists/test.lst "" \
/raid/datasets/cars/rec/test.rec resize=256 encoding='.jpg' \ качество=100
...
[12:25:18] tools/im2rec.cc:298: Всего: обработано 2427 изображений, прошло 33,5529 с.
```

И файл val.rec с этой командой:

```
$ ~/mxnet/bin/im2rec /raid/datasets/cars/lists/val.lst "" \
/raid/datasets/cars/rec/val.rec resize=256 encoding='.jpg' \ качество=100
...
[12:24:05] tools/im2rec.cc:298: Всего: обработано 2427 изображений, прошло 23,9026 с.
```

В качестве проверки работоспособности перечислите содержимое каталога, в котором вы сохранили базы данных записей:

```
$ ls -l rec total
1188668 -rw-rw-
r-- 1 adrian adrian 182749528 26 августа 12:25 test.rec -rw-rw-r-- 1 adrian
adrian 851585528 26 августа 12:23 train.rec -rw- rw-r-- 1 адриан адриан
182850704 26 августа 12:24 val.rec
```

Как вы можете видеть из моего вывода, моя учебная база данных составляет примерно 850 МБ, в то время как наборы данных проверки и тестирования составляют 181 МБ каждый, что значительно меньше, чем набор данных ImageNet , с которым мы работали ранее.

13.2 Тонкая настройка VGG в наборе данных Stanford Cars

Теперь мы, наконец, готовы к тонкой настройке VGG16 в наборе данных Stanford Cars. Но прежде чем мы сможем написать код для fine_tune_cars.py, нам сначала нужно загрузить предварительно обученные веса для VGG16.

Вы можете скачать веса по этой ссылке: <http://data.dmlc.ml/models/imagenet/vgg/>

Обязательно

загрузите файлы vgg16-0000.params и vgg16-symbol.json . Опять же, имейте в виду, что вес для VGG16 составляет 550 МБ, поэтому наберитесь терпения, пока файл загружается. Файл .params — это фактические веса, а файл .json — архитектура. Мы научимся загружать эти два файла, а затем настраивать VGG16 в файле fine_tune_cars.py. Теперь откройте файл, и мы приступим к работе:

```
1 # импортируем необходимые пакеты 2
из config import car_config as config 3 import mxnet as
mx 4 import argparse 5 import logging 6 import os

7
8 # построить разбор аргумента и разобрать аргументы 9 ap =
argparse.ArgumentParser() 10 ap.add_argument("-v", "--vgg", required=True,
11 help="путь к предварительно обученной сети VGGNet для тонкой настройки")
12 ap.add_argument("-c", "--checkpoints", required=True,
13 help="путь к выходному каталогу контрольных точек")
14 ap.add_argument("-p", "--prefix", required=True,
15 help="имя префикса модели") 16
ap.add_argument("-s", "--start-epoch", type=int, default=0,
17 help="эпоха для возобновления обучения")
18 args = vars(ap.parse_args())
```

Строки 2-6 импортируют наши пакеты Python — здесь мы импортируем наш car_config , чтобы иметь доступ к нашим конфигурациям. Затем мы анализируем три обязательных аргумента командной строки, за которыми следует четвертый необязательный: • --vgg: это путь к предварительно обученным весам VGG, которые мы будем настраивать. • --checkpoints: в процессе тонкой настройки мы будем сериализовать веса после каждого

эпохи. Этот переключатель определяет, где будут храниться сериализованные файлы веса.

• --prefix: Как и во всех других примерах mxnet, нам нужно указать имя для нашей сети.

• `--start-epoch`: если мы решим остановить и перезапустить обучение во время тонкой настройки, мы можем указать с какой эпохи мы хотим возобновить обучение. Этот аргумент командной строки является необязательным , если вы выполняете тонкую настройку с нулевой эпохи.

Далее мы установим наш файл журнала, а также определим размер пакета на основе конфигурации. файл и количество устройств, которые мы будем использовать для обучения:

```
20 # установить уровень логирования и выходной файл
21 logging.basicConfig(level=logging.DEBUG,
22     имя_файла="training_{}.log".format(args["start_epoch"]),
23     режим файла = "w")
24
25 # определить партию
26 batchSize = config.BATCH_SIZE * config.NUM_DEVICES
```

Чтобы получить доступ к нашим обучающим данным, нам нужно создать экземпляр `ImageRecordIter`:

```
28 # построить итератор обучающего изображения
29 trainIter = mx.io.ImageRecordIter(
30     path_imgrec=config.TRAIN_MX_REC,
31     data_shape=(3, 224, 224),
32     batch_size = размер партии,
33     rand_crop = Верно,
34     rand_mirror = Верно,
35     повернуть=15,
36     max_shear_ratio=0,1,
37     среднее_r=config.R_MEAN,
38     mean_g=config.G_MEAN,
39     среднее_b=config.B_MEAN,
40     preprocess_threads=config.NUM_DEVICES * 2)
```

Сетевая архитектура VGG16 предполагает входные изображения размером 224×224 пикселя с 3 каналами; однако, как вы помните из раздела 13.1.1 выше, мы создали файлы .gес с изображениями, которые составляют 256 пикселей по самому короткому измерению. Почему мы создали эти файлы? Ответ лежит в атрибуте `rand_crop` (строка 33). Этот атрибут указывает, что мы хотим случайным образом обрезать области 224×224 из входного изображения 256×256 . Это может помочь нам улучшить классификацию точность. Мы также поставим средства RGB из оригинального VGG16, обученного Симоняном. и Зиссермана, поскольку мы занимаемся тонкой настройкой, а не тренируемся с нуля.

Давайте также создадим итератор проверочного изображения:

```
42 # создать итератор проверочного изображения
43 valIter = mx.io.ImageRecordIter(
44     path_imgrec=config.VAL_MX_REC,
45     data_shape=(3, 224, 224),
46     batch_size = размер партии,
47     среднее_r=config.R_MEAN,
48     mean_g=config.G_MEAN,
49     mean_b=config.B_MEAN)
```

Для тонкой настройки VGG16 мы будем использовать оптимизатор SGD:

```

51 # инициализируем оптимизатор и обучающие контексты
52 opt = mx.optimizer.SGD(learning_rate=1e-4, импульс=0,9, wd=0,0005,
53             rescale_grad=1.0 / размер партии)
54 ctx = [mx.gpu(3)]

```

Сначала мы будем использовать более низкую скорость обучения $1e^{-4}$ — будущие эксперименты помогут нам определить, какова оптимальная начальная скорость обучения. Так же будем тренироваться с импульсом 0.9 и L2 срок регуляризации веса 0,0005. Один графический процессор будет использоваться для тонкой настройки VGG в Стенфорде. Набор данных автомобилей в виде эпох займет всего 11 секунд.

Далее мы построим путь к нашему выходному checkpointsPath, а также инициализируем аргумент параметры, вспомогательные параметры, а также допустимы ли «отсутствующие» параметры в сеть:

```

56 # построить путь контрольных точек, инициализировать аргумент модели и
57 # вспомогательные параметры, и должны ли неинициализированные параметры
58 # разрешено
59 checkpointsPath = os.path.sep.join([args["checkpoints"],
60                                     аргументы["префикс"]])
61 argParams = Нет
62 дополнительных параметра = нет
63 разрешитьMissing = Ложь

```

В этом контексте «отсутствующие параметры» — это параметры, которые еще не были инициализированы в сеть. Обычно мы не допускаем неинициализированные параметры; однако напомним, что тонкая настройка требует от нас отрезать головку сети и заменить ее новой, неинициализированной, полностью подключенной головкой. Поэтому, если мы тренируемся с нулевой эпохи, мы допустим отсутствующие параметры.

Говоря об обучении с нулевой эпохи, давайте посмотрим, как этот процесс выполняется:

```

65 # если не указана начальная эпоха конкретной модели, то мы
66 # нужно построить архитектуру сети
67 , если args["start_epoch"] <= 0:
68     # загрузить предварительно обученную модель VGG16
69     print("[INFO] загрузка предварительно обученной модели...")
70     (символ, argParams, auxParams) = mx.model.load_checkpoint(
71         аргументы["vgg"], 0)
72     разрешитьMissing = Истина

```

Строка 67 проверяет, начинаем ли мы первоначальный процесс тонкой настройки. При условии, что мы Есть, строки 70 и 71 загружают предварительно обученные веса `-vgg` из файла. Мы также обновляем `allowMissing` быть правдой, так как мы собираемся заменить главу сети.

Замена главы сети не так проста, как Keras (поскольку это требует некоторых исследовательская работа в именах слоев), но это все еще относительно простой процесс:

```

74     # захватить слои из предварительно обученной модели, затем найти
75     # выпадающий слой *до* последнего слоя FC (т.е. слой
76     # который содержит количество меток класса)
77     # ПОДСКАЗКА: вы можете найти такие имена слоев:
78     # для слоя в слоях:
79     #     печать (слой.имя)

```

```
80     # затем добавьте строку '_output' к имени слоя layers =
81     symbol.get_internals() net = layers["drop7_output"]
82
```

Строка 81 захватывает все слои внутри сети VGG16, которую мы только что загрузили с диска. Затем нам нужно определить, где находится окончательный выходной слой, а затем отрезать остальную часть головки FC . Этот процесс требует небольшой исследовательской работы, поскольку нам нужно знать имя конечного выходного слоя, который нас интересует (в данном случае, последний слой исключения), прежде чем мы применим слой FC с 1000 узлами для меток ImageNet). Чтобы найти этот выпадающий слой, я рекомендую загрузить VGG16 в отдельную оболочку Python, захватить слои и распечатать имена:

```
$ python
>>> импортировать mxnet
как mx >>> (symbol, argParams, auxParams) = mx.model.load_checkpoint("vgg16", 0) >>> layers =
symbol.get_internals() >>> для слоя в слоях: печать (слой.имя)

...
...

данные
conv1_1_weight
conv1_1_bias
conv1_1 relu1_1
conv1_2_weight
conv1_2_bias
conv1_2 relu1_2
pool1

...
fc7_weight
fc7_bias
ФК7
relu7
drop7
fc8_weight
fc8_bias
ФК8
метка_пробы
```

Здесь мы видим, что drop7 является последним отсеваемым слоем перед слоем FC с 1000 узлами . Чтобы получить вывод этого класса, просто добавьте вывод в конец строки drop7 : drop7_output.

Создание этой строки служит ключом к нашему словарю слов в строке 82, что позволяет нам сохранять все слои вплоть до последнего слоя FC.

Наш следующий блок кода прикрепляет новую головку FC с NUM_CLASSES (164) к телу VGG16, за которым следует классификатор softmax:

```
84     # создать новый слой FC, используя желаемое количество выходных
85     # меток классов, за которыми следует вывод softmax net =
86     mx.sym.FullyConnected(data=net, num_hidden=config.NUM_CLASSES,
87             name="fc8") net = mx.sym .SoftmaxOutput (данные = сеть, ИМЯ =
88             "softmax")
```

```

89
90      # создать новый набор сетевых аргументов, удалив все предыдущие
91      # аргументы, относящиеся к FC8 (это позволит нам обучить
92      # последний слой)
93      argParams = dict({k:argParams[k] для k в argParams
94      если "fc8" не в k})

```

Строки 93 и 94 удаляют все записи параметров для fc8, слоя FC , который мы только что удалили хирургическим путем . из сети. Проблема в том, что argParams не содержит никакой информации о нашем

новый заголовок FC, именно поэтому ранее в коде мы установили для параметра allowMissing значение True.

В случае, если мы перезапускаем нашу тонкую настройку с определенной эпохи, нам просто нужно загрузить соответствующие веса:

```

96 # в противном случае указана конкретная контрольная точка
97 еще:
98     # загружаем чекпойнт с диска
99     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
100    (net, argParams, auxParams) = mx.model.load_checkpoint(
101        контрольные точкиПуть, аргументы["начало_эпохи"])

```

Давайте также инициализируем наш стандартный набор обратных вызовов и метрик:

```

103 # инициализировать обратные вызовы и метрики оценки
104 batchEndCBs = [mx.callback.Спидометр (batchSize, 50)]
105 epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)]
106 метрик = [mx.metric.Accuracy(), mx.metric.TopKAccuracy(top_k=5),
107             mx.metric.CrossEntropy()]

```

Наконец, мы можем настроить нашу сеть:

```

109 # построить модель и обучить ее
110 print("[INFO] обучающая сеть...")
111 модель = mx.mod.Module (символ = сеть, контекст = ctx)
112 модель.подходит(
113     поезд,
114     eval_data=значение,
115     число_эпох = 65,
116     begin_epoch=args["start_epoch"],
117     инициализатор=mx.initializer.Xavier(),
118     arg_params = параметры аргумента,
119     aux_params = вспомогательные параметры,
120     оптимизатор=выбор,
121     allow_missing = разрешить отсутствие,
122     eval_metric = показатели,
123     batch_end_callback=batchEndCB,
124     epoch_end_callback=epochEndCBs)

```

Вызов метода fit в нашем примере тонкой настройки немного более подробный, чем предыдущий. Примеры ImageNet, потому что большинство наших предыдущих параметров (например, arg_params, aux_params, и т. д.) может быть передан через класс FeedForward . Однако, поскольку мы сейчас полагаемся либо на (1),

выполняя сетевую операцию или (2) загружая определенную эпоху, нам нужно переместить все эти параметры в метод подгонки.

Кроме того, обратите внимание, как мы передаем параметр `allow_missing` в `mxnet`, позволяя библиотеке понять, что мы пытаемся выполнить точную настройку. Мы установим максимальное значение `num_epoch` равным 65 — это число может увеличиваться или уменьшаться в зависимости от того, как проходит процесс обучения.

13.2.1 Тонкая настройка VGG: Эксперимент №1

В моем первом эксперименте по тонкой настройке архитектуры VGG16 в наборе данных Stanford Cars я решил использовать оптимизатор SGD с начальной скоростью обучения $1e^{-4}$. Технически это большая скорость обучения для задачи тонкой настройки, но я решил попробовать, чтобы получить базовую точность. Также были предоставлены импульсный член 0.9 и уменьшение веса $L2$ 0.0005 . Затем я начал тренироваться, используя следующую команду:

```
$ python fine_tune_cars.py --vgg vgg16/vgg16 --checkpoints контрольно-пропускные пункты \n
--prefix vggnet
```

В течение первых 10 эпох мы видим, что наша тонкая настройка выглядит достаточно хорошо, с точностью почти 70% для ранга 1 и более 90% точности для ранга 5 (рис. 13.2, слева). Однако в прошлые эпохи 15-30 мы начинаем видеть переобучение VGG тренировочным данным.

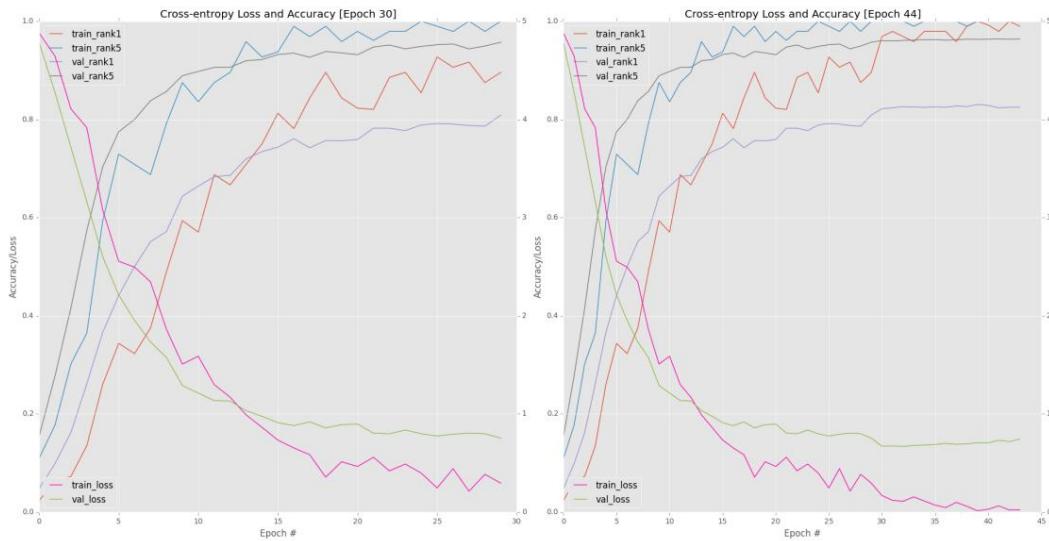


Рисунок 13.2: Слева: тонкая настройка VGG16 на наборе данных cars. Первые десять эпох выглядят хорошо, но после 15 эпохи мы начинаем видеть переоснащение. Справа: снижение α до $1e^{-5}$ насыщает обучение.

Я решил прекратить обучение на этом этапе и снизить скорость обучения с $1e^{-4}$ до $1e^{-5}$, а затем перезапустить тренировочную команду:

```
$ python fine_tune_cars.py --vgg vgg16/vgg16 --checkpoints контрольно-пропускные пункты \n
--prefix vggnet --start-epoch 30
```

Затем я позволил возобновить обучение еще на 15 эпох до эпохи 45 (рис. 13.2, справа). Как мы видим, как потери при обучении, так и точность полностью насыщены — точность ранга 1 для обучающих данных на самом деле выше, чем точность ранга 5 для проверочного набора. Кроме того, изучение

На нашем графике кросс-энтропийных потерь мы видим, что потери при проверке начинают увеличиваться после 35-й эпохи, что является верным признаком переобучения.

Все это говорит о том, что этот первоначальный эксперимент получил точность ранга 1 82,48% и ранга 5 96,38%. Проблема в том, что наша сеть слишком перегружена, и нам нужно искать другие возможности.

13.2.2 Тонкая настройка VGG: Эксперимент №2

Из-за переобучения, вызванного базовой скоростью обучения $1e-4$, я решил снизить начальную скорость обучения до $1e-5$. SGD снова использовался в качестве моего оптимизатора с теми же параметрами затухания импульса и веса, что и в первом эксперименте. Обучение запускалось с помощью следующей команды:

```
$ python fine_tune_cars.py --vgg vgg16/vgg16 --checkpoints контрольно-пропускные пункты \n
--prefix vggnet
```

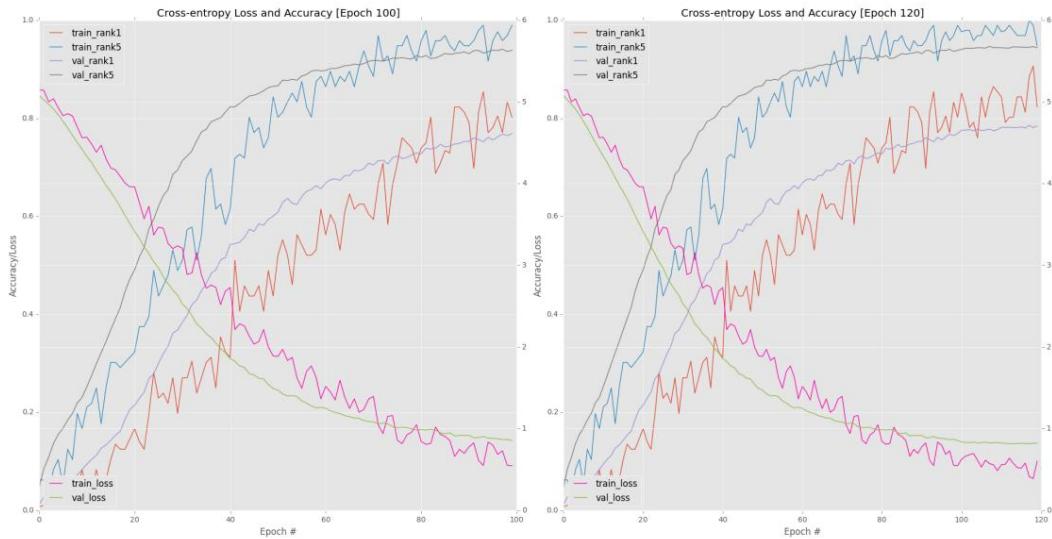


Рисунок 13.3: Слева: обучение намного стабильнее при скорости обучения $1e-5$. Справа: однако точность намного ниже, чем в нашем предыдущем эксперименте, даже при уменьшении α до $1e-6$. Начальная скорость обучения в этом случае слишком низкая.

Изучая графики первых 80 эпох, мы видим, что обучение гораздо более стабильно (рис . 13.3, слева). Эффекты переобучения были уменьшены. Проблема в том, что обучение сети занимает много времени — также сомнительно, будет ли наша точность проверки такой же высокой, как в предыдущем эксперименте. В эпоху 100 я прекратил обучение, снизил скорость обучения с $1e-5$ до $1e-6$, затем возобновил обучение:

```
$ python fine_tune_cars.py --vgg vgg16/vgg16 --checkpoints контрольно-пропускные пункты \n
--prefix vggnet --start-epoch 100
```

Я позволил сети продолжить обучение еще от 20 до 120 эпох, прежде чем остановил эксперимент (рис. 13.3, справа). Проблема здесь в том, что точность значительно упала — до 78,41% ранга-1 и 94,47% ранга-5. Базовая скорость обучения $1e-5$ была слишком низкой и, следовательно, ухудшила точность нашей классификации.

13.2.3 Тонкая настройка VGG: Эксперимент №3

В моем последнем эксперименте по тонкой настройке VGG16 я решил разделить разницу между скоростью обучения $1e-4$ и $1e-5$ и начать обучение со скоростью обучения $5e-4$. Были сохранены те же условия оптимизатора SGD, импульса и регуляризации, что и в первых двух экспериментах. Я еще раз начал процесс тонкой настройки, используя следующую команду:

```
$ python fine_tune_cars.py --vgg vgg16/vgg16 --checkpoints контрольно-пропускные пункты \
--prefix vggnets
```

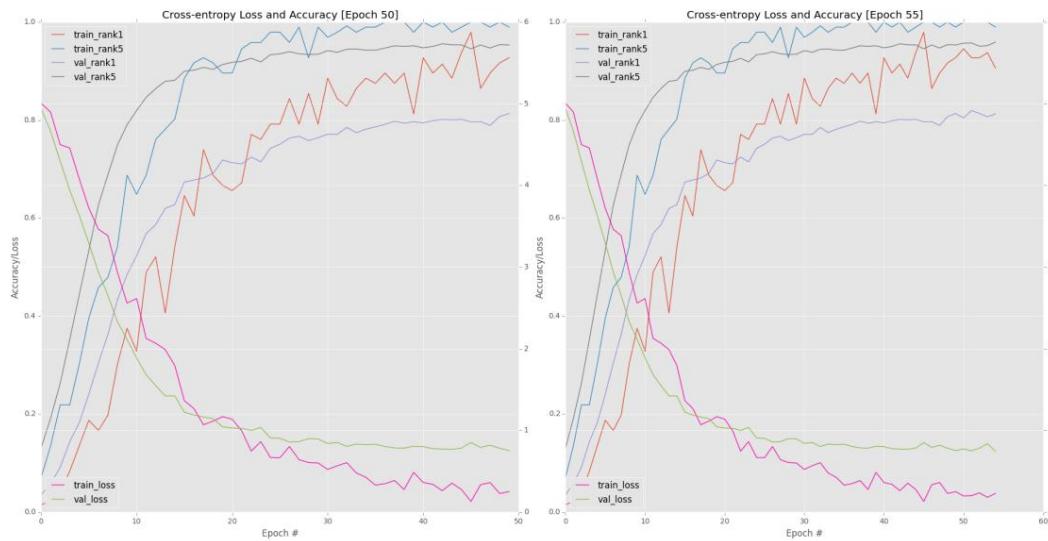


Рисунок 13.4: Слева: первые 50 эпох с использованием скорости обучения $5e-4$. Справа: снижение скорости обучения до $5e-5$ и обучение еще на 5 эпох. Мы получаем немного более низкую точность, чем эксперимент № 1; тем не менее, мы гораздо меньше перетренированы.

Первые 50 эпох представлены на рис. 13.4 (слева). Здесь мы можем видеть, что потери/точность обучения идут в ногу с проверкой и в конечном итоге обгоняют потери/точность проверки. Однако здесь важно отметить, что потеря/точность обучения не достигает насыщения, как это было в наших первых экспериментах.

Я позволил сети продолжать обучение до эпохи 50, где я прекратил обучение, уменьшил скорость обучения с $5e-4$ до $5e-5$ и обучение еще на пять эпох:

```
$ python fine_tune_cars.py --vgg vgg16/vgg16 --checkpoints контрольные точки \
--prefix vggnets --start-epoch 50
```

Окончательный график потери/точности можно увидеть на рис. 13.4 (справа). Хотя, безусловно, существует разрыв между началом формирования потерь при обучении и проверке, потери при проверке не увеличиваются. Глядя на результат последней эпохи, наша сеть достигла точности 81,25% ранга 1 и 95,88% ранга 5 на проверочном наборе. Эти точности, безусловно, лучше, чем во втором эксперименте. Я бы также сказал, что эти точности более желательны, чем в первом эксперименте, поскольку мы не переобучили нашу сеть и не рискуем способностью модели обобщать. В этот момент я почувствовал себя комфортно, полностью остановив эксперимент и перейдя к этапу оценки.

Имейте в виду, что при точной настройке (крупных) сетей, таких как VGG16, на небольших наборах данных (таких как Stanford Cars) переобучение неизбежно. Даже при применении аугментации данных в сети просто слишком много параметров и слишком мало обучающих примеров. Таким образом, чрезвычайно важно правильно определить начальную скорость обучения — это даже более важно, чем при обучении сети с нуля. Не торопитесь при тонкой настройке сетей и изучите различные начальные скорости обучения. Этот процесс даст вам наилучшие шансы на получение максимальной точности во время тонкой настройки.

13.3 Оценка нашего классификатора транспортных средств

Теперь, когда мы успешно настроили VGG16 в наборе данных Stanford Cars, давайте перейдем к оценке производительности нашей сети на тестовом наборе. Чтобы выполнить этот процесс, откройте файл test_cars.py и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2
из config import car_config as config 3 из
pyimagesearch.utils.ranked import rank5_accuracy 4 import mxnet as mx
5 import argparse 6 import pickle 7 import os
```

Строки 1-7 импортируют необходимые пакеты Python. Нам понадобится наш car_config (псевдоним config) , чтобы мы могли получить доступ к конкретной конфигурации и переменным нашего набора данных автомобиля (строка 2). Мы импортируем функцию rank5_accuracy в строку 3 , чтобы мы могли вычислить точность ранга 1 и ранг 5 на тестовом наборе соответственно. Затем библиотека mxnet импортируется в строку 4 , чтобы мы могли получить доступ к привязкам mxnet Python.

Далее, давайте проанализируем наши аргументы командной строки:

```
9 # построить разбор аргумента и разобрать аргументы 10 ap =
argparse.ArgumentParser() 11 ap.add_argument("-c", "--checkpoints",
required=True,
12     help="путь к каталогу выходных контрольных точек")
13 ap.add_argument("-p", "--prefix", required=True,
14     help="имя префикса модели") 15
ap.add_argument("-e", "--epoch", type=int, required=True, help="epoch # для
16     загрузки") 17 args = vars(ap.parse_args())
```

Как и в наших экспериментах по оценке ImageNet, наши аргументы командной строки почти идентичны. Для начала нам нужен --checkpoints, который представляет собой путь к каталогу, в котором весовые коэффициенты VGG16 были сериализованы в процессе тонкой настройки. Префикс --prefix управляет именем сети, в данном случае это будет vggnet. Наконец, --epoch — это целое число, которое управляет весовой эпохой VGG16, которую мы собираемся загрузить для оценки.

Оттуда мы можем загрузить наш кодировщик меток, а также инициализировать ImageRecordIter для тестового набора:

```
19 # загрузить энкодер этикеток
20 le = pickle.loads(open(config.LABEL_ENCODER_PATH, "rb").read())
21
```

```

22 # создать итератор проверочного изображения
23 testIter = mx.io.ImageRecordIter(
24     path_imgrec=config.TEST_MX_REC,
25     data_shape=(3, 224, 224),
26     batch_size=config.BATCH_SIZE,
27     среднее_r=config.R_MEAN,
28     mean_g=config.G_MEAN,
29     mean_b=config.B_MEAN)

```

Следующим шагом будет загрузка нашей предварительно обученной модели с диска:

```

31 # загружаем нашу предварительно обученную модель
32 print("[INFO] загрузка предварительно обученной модели...")
33 checkpointsPath = os.path.sep.join([args["checkpoints"],
34                                     аргументы["предфикс"]])
35 (символ, argParams, auxParams) = mx.model.load_checkpoint(
36     контрольные точкиПуть, аргументы["эпоха"])
37
38 # построить модель
39 модель = mx.mod.Module (символ = символ, контекст = [mx.gpu (0)])
40 модель.bind(data_shapes=testIter.provide_data,
41               label_shapes=testIter.provide_label)
42 модель.set_params(argParams, auxParams)

```

Строки 33 и 34 создают базовый путь к сериализованным весам, используя оба параметра --checkpoints и --prefix переключатель. Чтобы контролировать, какая именно эпоха загружается, мы передаем checkpointsPath и --epoch в функцию load_checkpoint (строки 35 и 36).

В строке 39 создается класс модуля mxnet . Этот класс принимает два параметра: символ модели загружается с диска вместе с контекстом, который представляет собой список устройств, которые мы будем использовать для оценки нашего network в наборе данных testIter . Здесь я указываю, что нужен только один GPU через mx.gpu(0); однако вам следует обновить этот параметр, указав количество устройств в вашей системе.

Вызов метода привязки модели в строках 40 и 41 требуется для того , чтобы модель понимать, что testIter отвечает за предоставление формы (т.е. размеров) как данные и метки. В нашем случае все входные изображения будут 224× 224 с тремя каналами. Наконец, мы установили параметры аргументов и вспомогательные параметры сериализованной сети в строке 42.

На этом этапе мы можем начать оценивать сеть на тестовом наборе:

```

44 # инициализируем список прогнозов и целей
45 print("[INFO] оценка модели...")
46 прогнозов = []
47 целей = []
48
49 # цикл по пакетам прогнозов
50 для (preds, _, batch) в модель.iter_predict(testIter):
51     # преобразовать пакет прогнозов и меток в NumPy
52     # массивы
53     преды = преды[0].asnumpy()
54     метки = пакет.метка[0].asnumpy().astype("целое число")
55
56     # обновить списки прогнозов и целей соответственно
57     предсказания.расширить(предыдущие)
58     target.extend(метки)

```

59

```
60 # применить нарезку массива к целевым объектам, поскольку mxnet вернет 61 #
следующий полный размер пакета, а не *фактическое* количество меток
62 цели = цели[:len(прогнозы)]
```

Строки 46 и 47 инициализируют наш список прогнозов и истинных целей. В строке 50 мы используем функцию `iter_predict`, которая позволяет нам перебирать `testIter` в пакетном режиме, получая как прогнозы (`pres`), так и метки истинности (`batch`).

В строке 53 мы получаем прогнозы из сети и преобразовываем их в массив NumPy. Этот массив имеет форму ($N, 164$), где N — количество точек данных в пакете, а 164 — общее количество меток классов. Стока 54 извлекает метку истинного класса из `testIter`, снова преобразовывая ее в массив NumPy. Имея как предварительные, так и наземные метки, мы теперь можем обновлять наши списки прогнозов и целей соответственно.

Строка 62 гарантирует, что списки целей и прогнозов имеют одинаковую длину. Эта строка является обязательной, так как функция `iter_predict` будет возвращать пакет только в размере степеней двойки из соображений эффективности (или это также может быть небольшой ошибкой в функции). Таким образом, почти всегда список целей длиннее списка прогнозов. Мы можем легко исправить несоответствие, применив нарезку массива.

Последний шаг — взять наши прогнозы и цели из тестового набора и вычислить наши точность ранга 1 и ранга 5:

```
64 # вычислить точность ранга 1 и ранга 5 65 (rank1, rank5)
= rank5_accuracy(прогнозы, цели) 66 print("[INFO] rank-1: {:.2f}%".format(rank1
* 100)) 67 print("[INFO] rank-5: {:.2f}%".format(rank5 * 100))
```

Чтобы оценить нашу точно настроенную сеть VGG16 в наборе данных о марке и модели автомобиля, просто выполните следующая команда:

```
$ python test_cars.py --checkpoints контрольные точки --prefix vggnet \
--epoch 55
[INFO] загрузка предварительно обученной модели...
[INFO] оценка модели...
[ИНФО] ранг-1: 84,22%
[ИНФО] ранг-5: 96,54%
```

Как показывают результаты, мы можем получить точность ранга 1 84,22% и ранга 5 96,54% на тестовом наборе. Этот уровень точности особенно впечатляет, учитывая, что у нас (1) очень ограниченный объем обучающих данных для каждой комбинации марки и модели автомобиля и (2) необходимо предсказать довольно большое количество классов (164) с учетом этого небольшого объема обучения. данные.

Дополнительную точность можно получить, собрав больше обучающих данных для каждой марки и модели автомобиля. Учитывая эти дополнительные данные, мы могли бы либо обучить пользовательскую CNN с нуля, либо продолжить тонкую настройку. Поскольку мы уже очень хорошо работаем с тонкой настройкой, я бы сосредоточил усилия на сборе данных и проведении дополнительных экспериментов по тонкой настройке для повышения точности.

13.4 Визуализация результатов классификации транспортных средств

В нашем предыдущем эксперименте мы узнали, как использовать функцию `iter_predict` для циклического просмотра точек данных в `ImageRecordIter` и создания прогнозов. Но что, если мы хотим зациклиться

необработанные изображения и делать прогнозы на их основе? Что нам делать тогда? К счастью для нас, процесс подготовки изображения для классификации с помощью mxnet мало чем отличается от методов, которые мы использовали для Keras.

В этом разделе мы узнаем, как загружать отдельные изображения с диска, предварительно обрабатывать их и передавать через mxnet, чтобы получить наши первые 5 прогнозов для данного транспортного средства.

Чтобы увидеть, как это делается, откройте файл vis_classification.py и вставьте следующий код:

```

1 # из-за ошибки mxnet seg-fault необходимо поместить импорт OpenCV в начало
файла 2 # import cv2

4
5 # импортируем необходимые пакеты 6
из config import car_config as config 7 из
pyimagesearch.preprocessing import ImageToArrayPreprocessor 8 из
pyimagesearch.preprocessing import AspectAwarePreprocessor 9 из
pyimagesearch.preprocessing import MeanPreprocessor 10 import numpy as np 11
import mxnet as mx 12 import argparse 13 import pickle 14 импорта imutils 15 импорта
ОС

```

В строке 3 мы импортируем наши привязки cv2 в библиотеку OpenCV. На двух из трех машин, на которых я установил OpenCV и mxnet, я получал сообщение об ошибке сегментации всякий раз, когда пытался импортировать mxnet перед OpenCV. Из-за этой ошибки я рекомендую поместить ваш импорт cv2 в начало вашего файла. Тем не менее, это может быть не тот случай, когда ваша система seg-faults, поэтому не стесняйтесь проверить это на своей собственной машине.

Оттуда строки 6-15 импортируют остальные наши пакеты Python. Обратите внимание на строки 7-9, где мы импортируем наши препроцессоры изображений, изначально разработанные для Keras, но их можно легко повторно использовать и при работе с mxnet.

Далее, давайте проанализируем наши аргументы командной строки:

```

17 # построить аргумент parse и разобрать аргументы 18 ap =
argparse.ArgumentParser() 19 ap.add_argument("-c", "--checkpoints",
required=True, help="путь к каталогу контрольной точки") 21
20     ap .add_argument("-p", "--prefix", required=True,
21
22         help="имя префикса модели") 23
ap.add_argument("-e", "--epoch", type=int, required=True, help="epoch # для
24         загрузки") 25 ap.add_argument("-s ", "--sample-size", type=int,
default=10, help="epoch # to load") 27 args = vars(ap.parse_args())
26

```

Нам потребуются три аргумента командной строки для запуска vis_classification.py: • --checkpoints: путь к нашему каталогу контрольных точек VGG16 при точной настройке. • --prefix: Имя нашей сети, которую мы точно настроили. • --epoch: номер эпохи, которую мы будем загружать с диска.

Наконец, можно указать необязательный переключатель --sample-size, чтобы указать количество изображений, которые мы хотим выбрать из нашего набора данных для классификации.

Говоря о выборке наших изображений, давайте продолжим и сделаем это сейчас:

```
29 # загрузить кодировщик меток, а затем файл тестового набора данных, 30 # затем
сэмплировать тестовый набор 31 le = pickle.loads(open(config.LABEL_ENCODER_PATH,
"rb").read()) 32 rows = open(config.TEST_MX_LIST).read().strip().split("\n") 33 строки =
np.random.choice(строки, размер=args["sample_size"])
```

Строка 31 загружает наш сериализованный LabelEncoder , чтобы мы могли преобразовать метки целочисленного класса, созданные mxnet, и преобразовать их в удобочитаемые метки (т. е. названия производителя и модели). Строки 32 и 33 загружают содержимое нашего файла test.lst и выбирают из него строки --sample-size .

Напомним, что файл test.lst содержит пути к нашим тестовым образам. Поэтому, чтобы визуализировать наши прогнозы, нам просто нужно выбрать строки из этого файла.

Затем мы можем загрузить нашу сериализованную сеть VGG16 с диска:

```
35 # загружаем нашу предварительно
обученную модель 36 print("[INFO] загружаем предварительно
обученную модель...")
37 checkpointsPath = os.path.sep.join([args["checkpoints"],
38         args["prefix"]]) 39
model = mx.model.FeedForward.load(checkpointsPath,
40         аргументы["эпоха"])
```

А также скомпилировать модель:

```
42 # компилируем модель 43
model = mx.model.FeedForward( ctx=[mx.gpu(0)],
44     symbol=model.symbol,
45     arg_params=model.arg_params,
46     aux_params=model.aux_params)
47
```

Оттуда давайте инициализируем наши препроцессоры изображений:

```
49 # инициализируем препроцессоры изображений
50 sp = AspectAwarePreprocessor(width=224, height=224) 51 mp =
MeanPreprocessor(config.R_MEAN, config.G_MEAN, config.B_MEAN) 52 iap =
ImageToArrayPreprocessor(dataFormat="channels_first")
```

В строке 50 создается экземпляр нашего препроцессора AspectAwarePreprocessor , который изменит размер изображения до 224×224 пикселей. MeanPreprocessor будет выполнять вычитание среднего, используя средние значения RGB из статьи Симоняна и Зиссермана [17] — средние значения не вычисляются из нашего обучающего набора, поскольку мы выполняем точную настройку, и должны использовать средние значения, вычисленные по набору данных ImageNet.

Наконец, мы инициализируем наш препроцессор ImageToArrayPreprocessor в строке 52. Первоначально этот класс использовался для преобразования необработанных изображений в массивы, совместимые с Keras, в зависимости от того, использовали ли мы «каналы в последнюю очередь» или «каналы в первую очередь» в нашем файле конфигурации keras.json . Однако, поскольку mxnet всегда представляет изображения в каналах в порядке очереди, нам нужно предоставить классу параметр dataFormat="channels_first" , чтобы убедиться, что наши каналы упорядочиваются правильно.

Пришло время просмотреть наши примеры изображений, классифицировать их и отобразить результаты на нашем экране:

```

54 # цикл по тестовым изображениям
55 для строки в строке :
56     # получить метку целевого класса и путь к изображению из строки
57     (цель, путь к изображению) = row.split("\t")[1:]
58     цель = интервал (цель)
59
60     # загрузить изображение с диска и предварительно обработать его, изменив размер
61     # изображение и применение препроцессоров
62     изображение = cv2.imread (путь к изображению)
63     ориг = изображение.копия()
64     orig = imutils.resize (original, ширина = мин (500, orig.shape [1]))
65     изображение = iap.preprocess(mp.preprocess(sp.preprocess(изображение)))
66     изображение = pr.expand_dims (изображение, ось = 0)

```

Строки 57 и 58 извлекают наземную цель и imagePath из входной строки, а затем путем преобразования цели в целое число. Мы тогда:

1. Загрузите наше входное изображение (строка 62).
2. Клонируйте его, чтобы мы могли нарисовать на нем визуализацию метки выходного класса (строка 63).
3. Начните этап предварительной обработки, изменив размер изображения до максимальной ширины 500 пикселей. (строка 64).
4. Применяя все три наших препроцессора изображений (строка 65).
5. Расширьте размеры массива, чтобы изображение можно было передать по сети (Line 66).

Классифицировать изображение с помощью предварительно обученной сети mxnet так же просто, как вызвать функцию прогнозирования . метод модели:

```

68     # классифицируем изображение и получаем индексы топ-5 предсказаний
69     предс = модель.прогноз(изображение)[0]
70     idxs = np.argsort(preds)[::-1][-5]
71
72     # показать истинную метку класса
73     print("[ИНФОРМАЦИЯ] фактическое={}".format(le.inverse_transform(target)))
74
75     # отформатировать и отобразить верхнюю метку прогнозируемого класса
76     метка = le.inverse_transform (idxs [0])
77     метка = метка.заменить (" ":" ")
78     label = "{}: {:.2f}%".format(label, preds[idxs[0]] * 100)
79     cv2.putText (оригинал, метка, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
80                 0.6, (0, 255, 0), 2)

```

Строка 69 возвращает наши прогнозы для каждой из 164 меток классов. Затем мы сортируем индексы эти метки в соответствии с их вероятностью (от наибольшего к наименьшему), сохраняя 5 лучших прогнозов (строка 70). Стока 73 отображает удобочитаемое имя класса метки истинности, а Строки 76-80 нарисуйте предсказанную метку № 1 для нашего изображения, включая вероятность предсказания.

Наш последний блок кода обрабатывает вывод первых 5 прогнозов на наш терминал и отображение нашего выходное изображение:

```

82     # перебираем прогнозы и отображаем их
83     for (i, prob) в zip(idxs, preds):
84         print("т[INFO] предсказано={}, вероятность={:.2f}%".format(
85             le.inverse_transform(i), pres[i] * 100))

```

```

86
87     # показать
88     изображение
89     cv2.imshow("Image", orig) cv2.waitKey(0)

```

Чтобы увидеть vis_classification.py в действии, просто откройте терминал и выполните следующую команду:

```
$ python vis_classification.py --checkpoints контрольные точки --prefix vggnet \
--эпоха 55
```



Рисунок 13.5: Наша точно настроенная сеть VGG16 может правильно распознавать марку и модель автомобиля с точностью более 84 % для ранга 1 и 95 % для ранга 5.

На рис. 13.5 показаны образцы правильно классифицированных марок и моделей автомобилей. Опять же, весьма примечательно, что мы можем получить такую высокую точность классификации, используя такие небольшие обучающие данные с помощью тонкой настройки. Это еще один пример силы и полезности глубокого обучения применительно к классификации изображений.

13.5 Резюме

В этой главе мы узнали, как точно настроить архитектуру VGG16 (предварительно обученную на ImageNet) для правильного распознавания 164 марок и классов моделей автомобилей с точностью более 84,22 % для ранга 1 и 96,54 % для ранга 5. Для выполнения этой задачи мы вырезали последний полно связанный слой в VGG16 (уровень FC, который выводит общее количество классов, равное 1000 в случае ImageNet) и заменили его нашим собственным полно связанным слоем из 164 классов., за которым следует классификатор softmax.

Была проведена серия экспериментов для определения оптимальной скорости обучения при тонкой настройке VGG16 с использованием оптимизатора SGD. После точной настройки и оценки нашей сети мы написали простой скрипт Python с именем vis_classification.py, который поможет нам визуализировать классификацию марки и модели автомобиля для заданного входного изображения. Этот же сценарий можно модифицировать, чтобы создавать собственные интеллектуальные, узконаправленные рекламные щиты, строго основанные на типе автомобиля, которым управляет данный человек.

14. Практический пример: прогнозирование возраста и пола

В этом последнем тематическом исследовании по глубокому обучению для компьютерного зрения с помощью Python мы узнаем, как создать систему компьютерного зрения, способную распознавать возраст и пол человека на фотографии. Для выполнения этой задачи мы будем использовать набор данных Adience, созданный Леви и Хасснером и использованный в их публикации 2015 года «Классификация по возрасту и полу с использованием сверточных нейронных сетей» [38].

Это тематическое исследование будет более сложным, чем предыдущие, и нам потребуется обучить две модели: одну для распознавания возраста, а другую для идентификации пола. Кроме того, нам также придется полагаться на более совершенные алгоритмы компьютерного зрения, такие как ориентиры лица (<http://pyimg.co/xkgwd>) и выравнивание лица (<http://pyimg.co/tnbzf>) чтобы помочь нам предварительно обработать наши изображения перед классификацией.

Мы начнем эту главу с обсуждения набора данных Adience, а затем рассмотрим структуру нашего проекта. Оттуда мы создадим MxAgeGenderNet, реализацию сети, предложенную Levi et al. Учитывая нашу реализацию, мы обучим два отдельных экземпляра MxAgeGenderNet, один для распознавания возраста, а другой для распознавания пола. Когда у нас будут эти две сети, мы применим их к изображениям вне набора данных Adience и оценим производительность.

Из-за длины этой главы мы рассмотрим наиболее важные аспекты тематического исследования, включая обзор кода, используемого для обучения и оценки сети. Дальнейшие подробности, особенно в отношении служебных функций и вспомогательного класса, которые мы определим позже в этой главе, можно найти в дополнительных материалах, сопровождающих эту книгу.

14.1 Этика гендерной идентификации в машинном обучении

Прежде чем мы зайдем слишком далеко в этой главе, я хочу поднять тему этики и гендерной идентификации.

Хотя использование компьютерного зрения и глубокого обучения для распознавания возраста и пола человека на фотографии является интересной задачей классификации, она имеет моральные последствия. Тот факт, что кто-то визуально выглядит, одевается или выглядит определенным образом, не означает, что он идентифицирует себя с этим (или любым) полом.

Прежде чем даже подумать о создании программы, которая пытается распознать пол человека, уделите время изучению гендерных вопросов и гендерного равенства.

Такие приложения, как распознавание пола, лучше всего использовать для «демографических исследований», таких как мониторинг количества людей, которые останавливаются и осматривают различные киоски в универмаге. Эти типы опросов могут помочь вам оптимизировать макет магазина и, в свою очередь, увеличить продажи. Но даже в этом случае вам нужно проявлять крайнюю осторожность, чтобы не допустить маргинализации небольших групп людей. Имейте в виду закон больших чисел, поскольку отдельные точки данных будут усредняться в совокупности.

Вы никогда не должны использовать распознавание пола для настройки пользовательского интерфейса (например, замены местоимений, относящихся к полу), на основе (1) фотографий отдельных пользователей, использующих ваше программное обеспечение, и (2) результатов вашей модели. Мало того, что модели машинного обучения далеки от совершенства, вы также несете моральную ответственность за то, чтобы не учитывать пол пользователя.

Классификация по возрасту и полу — очень интересная задача для изучения с точки зрения компьютерного зрения — это не только сложная задача, но и отличный пример мелкозернистой задачи классификации, где тонкие различия могут иметь значение в правильной или неправильной классификации.

Но в этом и заключается проблема — люди не являются точками данных, а пол не является бинарным. Существует очень мало этических приложений распознавания пола с использованием искусственного интеллекта. Программное обеспечение, которое пытается преобразовать пол в бинарную классификацию, только еще больше приковывает нас к устаревшим представлениям о том, что такое пол. Поэтому я бы посоветовал вам не использовать распознавание пола в своих приложениях, если это вообще возможно. Если вам необходимо выполнить распознавание пола, убедитесь, что вы берете на себя ответственность и убедитесь, что вы не создаете приложения, которые пытаются подчинить других гендерным стереотипам.

Используйте эту главу, чтобы помочь вам изучить компьютерное зрение и глубокое обучение, но используйте свои знания во благо. Пожалуйста, относитесь с уважением к своим собратьям и изучайте гендерные вопросы и гендерное равенство.

Я надеюсь, что когда-нибудь я смогу удалить эту главу из книги «Глубокое обучение компьютерному зрению с помощью Python» не потому, что люди используют ее в гнусных, вопиющих целях, а потому, что гендерная идентификация стала настолько устаревшей, что никто не удосуживается думать о гендере, пусть в одиночку попытается идентифицировать его.

14.2 Набор данных Adience

Набор данных Adience [39] состоит из 26 580 изображений лиц. Всего в этот набор данных было включено 2284 субъекта (человека). Затем набор данных был разделен на два гендерных класса (мужской и женский) вместе с восемью возрастными группами; в частности: 0–2, 4–6, 8–13, 15–20, 25–32, 38–43, 48–53 и 60+. Цель набора данных Adience — правильно предсказать возраст и пол данного человека на фотографии. Образец набора данных можно увидеть на рис. 14.1.

Набор данных несбалансирован: для людей в возрастной группе 25–32 года представлено больше выборок, чем для любой другой возрастной группы, в два раза (4897). Наиболее мало представлены выборки в возрастных диапазонах 48–53 и 60+ (825 и 869 соответственно). Количество мужских выборок (8 192) немного меньше, чем количество женских точек данных (9 411), но все же находится в разумных пределах распределения по классам.

При обучении модели на наборе данных Adience у нас есть два варианта: 1. Обучить один классификатор, который отвечает за прогнозирование как пола, так и возраста, всего 16 возможных классов.

2. Обучить два классификатора, один отвечает за прогнозирование пола (2 класса) и отдельную модель для распознавания возраста (8 классов).

Оба метода являются жизнеспособными вариантами; тем не менее, мы собираемся использовать подход Леви и др. и обучить две отдельные CNN — это не только позволит нам воспроизвести их результаты (и даже превзойти их заявленную точность), но и позволит нашему классификатору быть более надежным.

Черты лица, которым научились различать женщину 60+, вероятно, сильно отличаются от черт лица, которым научились узнавать мужчину, которому тоже 60+; поэтому имеет смысл разделить возраст и пол на два отдельных классификатора.

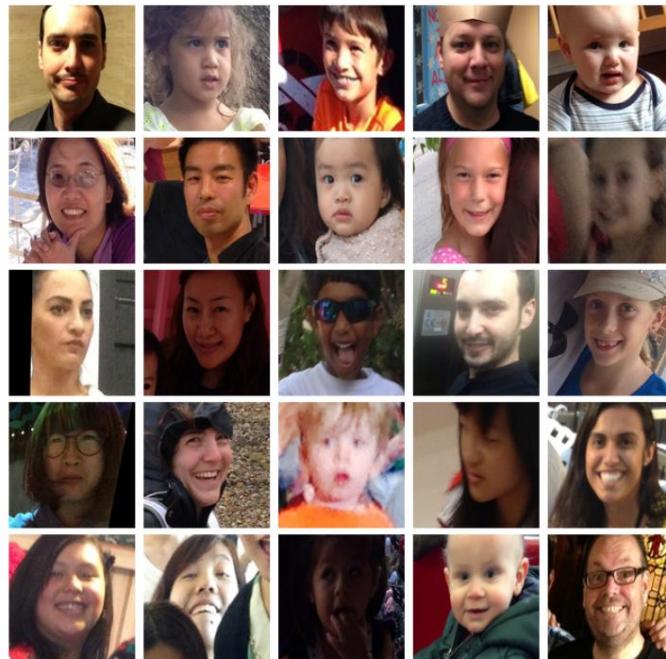


Рисунок 14.1: Пример набора данных Adience для распознавания возраста и пола. Возрастные диапазоны охватывают диапазон от 0 до 60+.

14.2.1 Создание набора данных Adience

Вы можете загрузить набор данных Adience по следующей ссылке: <http://www.open.ac.il/home/hassner/Adience/data.html>

Просто перейдите в раздел «Загрузить» и введите свое имя и адрес электронной почты (эта информация используется только для отправки вам периодических электронных писем, если набор данных когда-либо обновлялся). Оттуда вы сможете войти на FTP-сайт:

<http://www.cslab.open.ac.il/download/> Вы можете загрузить файлalign.tar.gz вместе со всеми файлами fold_frontal_*_data.txt, где звездочка представляет цифры 0-4 (рис. 14.2).

После загрузки набора данных вы можете разархивировать его с помощью следующей команды:

```
$ tar -xvf выровнен.tar.gz
```

При организации набора данных на моей машине я предпочитаю использовать следующую структуру каталогов:

```
--- аудитория
|--- выровненный
|--- складки |---
|--- списки |---
|--- запись
```

Выровненный каталог содержит ряд подкаталогов, которые содержат примеры лиц, которые мы будем использовать для обучения наших возрастных и гендерных CNN. Строго из организационных соображений я также решил хранить все файлы fold .txt (которые Леви и др. первоначально использовали для перекрестной проверки) в каталоге с именем folds. Затем я создал каталог lists для хранения сгенерированных файлов .lst вместе с каталогом rec для баз данных записей ttxnet.

LICENSE.txt	22-Nov-2016 20:35	1.8K
aligned.tar.gz	18-Jun-2014 16:51	2.6G
faces.tar.gz	18-Jun-2014 15:04	1.2G
fold_0_data.txt	15-Dec-2014 09:57	355K
fold_1_data.txt	15-Dec-2014 09:57	297K
fold_2_data.txt	15-Dec-2014 09:57	310K
fold_3_data.txt	15-Dec-2014 09:57	279K
fold_4_data.txt	15-Dec-2014 09:57	307K
fold_frontal_0_data.txt	15-Dec-2014 09:57	253K
fold_frontal_1_data.txt	15-Dec-2014 09:57	242K
fold_frontal_2_data.txt	15-Dec-2014 09:57	190K
fold_frontal_3_data.txt	15-Dec-2014 09:57	202K
fold_frontal_4_data.txt	15-Dec-2014 09:57	192K

Рисунок 14.2: Убедитесь, что вы загрузили файл aligned.tar.gz вместе со всеми файлами fold_frontal_*_data.txt.

В главах, посвященных ImageNet и маркам и моделям транспортных средств, мы вычислили ранг 1 и точность 5 ранга; однако в контексте Adience точность 5-го ранга не имеет интуитивного смысла. Для начала вычисление точности ранга 5 для двухклассовой (гендерной) задачи всегда будет (тривиально) 100%. Поэтому, сообщая о точности гендерной модели, мы просто сообщаем, правильная классификация или нет.

Для возрастной модели мы сообщим, что Levi et al. называют «однократной» точностью. Разовая точность измеряет, соответствует ли метка истинного класса предсказанной метке класса или существует ли метка истинного класса в двух соседних бинах. Например, предположим, что наша возрастная модель предсказала возраст кронштейн 15–20; однако метка достоверности составляет 25–32 — согласно разовой оценке метрика, это значение по-прежнему верно, поскольку 15–20 существует в наборе {15–20, 25–32, 38–48}.

Обратите внимание, что однократная точность — это не то же самое, что точность ранга 2. Например, если наша модель было представлено изображение, на котором изображен человек в возрасте 48–53 лет (т. е. ярлык достоверности) и наша модель предсказала:

- 4–6 с вероятностью 63,7%
- 48–53 с вероятностью 36,3%

Тогда мы будем считать это неправильной классификацией, поскольку первая предсказанная метка (4–6) не смежный с ячейкой 48–53. Если бы вместо этого наша модель предсказывала:

- 38–43 с вероятностью 63,7%
- 48–53, вероятно, 36,3%

Тогда это был бы правильный однократный прогноз, поскольку ячейка 38–43 непосредственно примыкает к 48–53 бин. Мы узнаем, как внедрить пользовательские метрики оценки, такие как одноразовая точность, далее в этой главе.

Файл конфигурации Adience

Как и в наших экспериментах с ImageNet, нам нужно сначала определить структуру каталогов нашего проекта:

```
|--- age_gender
|   |--- конфигурация
|   |   |--- __init__.py
|   |   |--- age_gender_config.py
|   |   |--- age_gender_deploy.py
```

```

|--- контрольно-пропускные пункты
| |--- возраст/
| |--- пол/
|--- build_dataset.py
|--- test_accuracy.py
|--- test_prediction.py
|--- train.py
|--- vis_classification.py

```

Внутри подмодуля конфигурации мы создадим два отдельных файла конфигурации. Один файл age_gender_config.py будет использоваться, когда мы обучаем наши CNN. Второй файл, age_gender_deploy.py будет использоваться после завершения обучения, и мы хотим применить наши CNN к изображениям за пределами набора данных Adience.

Каталог контрольных точек будет хранить все контрольные точки модели во время обучения. Подкаталог age будет содержать все контрольные точки, относящиеся к возрастному CNN, в то время как подкаталог хранить все контрольные точки, связанные с гендером CNN.

Сценарий build_dataset.py будет отвечать за создание наших файлов .lst и .rec для как по возрасту, так и по полу. После того, как мы создали наши наборы данных, мы можем использовать train.py для обучать наши CNN. Чтобы оценить производительность наших сетей на тестовом наборе, мы будем использовать test_accuracy.py. Затем мы сможем визуализировать прогнозы из набора данных Adience, используя vis_classification.py. Каждый раз, когда мы хотим классифицировать изображение за пределами Adience, мы используя test_prediction.py.

Давайте продолжим и просмотрим содержимое age_gender_config.py прямо сейчас:

```

1 # импортируем необходимые пакеты
2 из пути импорта ОС
3
4 # определить тип набора данных, который мы обучаем (т. е. либо «возраст», либо
5 # "пол")
6 DATASET_TYPE = "пол"
7
8 # определяем базовые пути к набору данных лиц и выходному пути
9 BASE_PATH = "/raid/datasets/adience"
10 OUTPUT_BASE = "выход"
11 MX_OUTPUT = БАЗОВЫЙ_ПУТЬ
12
13 # на основе базового пути получить путь к изображениям и путь к сгибам
14 IMAGES_PATH = path.sep.join([BASE_PATH, "выровнено"])
15 LABELS_PATH = path.sep.join([BASE_PATH, "folds"])

```

В строке 6 мы определяем DATASET_TYPE. Это значение может быть либо полом , либо возрастом. В зависимости в DATASET_TYPE мы изменим различные пути вывода и настройки позже в этой конфигурации . файл.

Затем мы определяем BASE_PATH для набора данных об аудитории в строке 9 , используя структуру каталогов I. подробно описано выше — вам нужно будет изменить BASE_PATH , чтобы указать, где находится ваш набор данных Adience. живет на диске. Стока 14 использует BASE_PATH для создания пути к нашему выровненному каталогу, который содержит входные изображения. Стока 15 также использует BASE_PATH для создания пути к каталогу folds .

где мы храним наши файлы .txt, содержащие метки классов.

Далее давайте определим некоторую информацию о наших сплитах обучения, тестирования и проверки:

```

17 # определить процент валидации и тестирования изображений относительно
18 # к количеству обучающих образов
19 NUM_VAL_IMAGES = 0,15
20 NUM_TEST_IMAGES = 0,15

```

Мы будем использовать 70% наших данных для обучения. Остальные 30% будут разделены поровну, 15% для валидации и 15% на тестирование. Мы также будем обучать нашу сеть, используя размер пакета 128 и два графических процессора:

```

22 # определить размер партии
23 ПАКЕТ_РАЗМЕР = 128
24 NUM_DEVICES = 2

```

Я использовал два графических процессора в этом эксперименте, чтобы иметь возможность быстро собирать результаты (порядка 17 второй эпохи). Вы также можете легко обучать свои модели с помощью одного графического процессора.

Наш следующий блок кода обрабатывает, если мы обучаем CNN на наборе данных «возраст»:

```

26 # проверяем, работаем ли мы с «возрастной» частью
27 # набор данных
28 , если DATASET_TYPE == "возраст":
29     # определить количество меток для набора данных "возраст" вместе с
30     # путем к кодировщику меток
31     NUM_CLASSES = 8
32     LABEL_ENCODER_PATH = path.sep.join([OUTPUT_BASE,
33                                         "age_le.cpickle"])
34
35     # определить путь к выходным данным для обучения, проверки и тестирования
36     # списки
37     TRAIN_MX_LIST = path.sep.join([MX_OUTPUT, "lists/age_train.lst"])
38     VAL_MX_LIST = path.sep.join([MX_OUTPUT, "lists/age_val.lst"])
39     TEST_MX_LIST = path.sep.join([MX_OUTPUT, "lists/age_test.lst"])
40
41     # определить путь к выходным данным для обучения, проверки и тестирования
42     # записи изображений
43     TRAIN_MX_REC = path.sep.join([MX_OUTPUT, "rec/age_train.rec"])
44     VAL_MX_REC = path.sep.join([MX_OUTPUT, "rec/age_val.rec"])
45     TEST_MX_REC = path.sep.join([MX_OUTPUT, "rec/age_test.rec"])
46
47     # получаем путь к файлу среднего пикселя
48     DATASET_MEAN = path.sep.join([OUTPUT_BASE,
49                                   "age_adience_mean.json"])

```

Строка 31 определяет NUM_CLASSES для набора возрастных данных, что составляет восемь общих возрастных групп. Хорошо также явно определите LABEL_ENCODER_PATH, в котором будет храниться сериализованный LabelEncoder объект.

Строки 37-39 определяют пути к файлам .lst, а строки 43-45 делают то же самое, только для .rec файлы. Мы также хотим убедиться, что сохраняют значения RGB для тренировочного набора, определив DATASET_MEAN (строки 48 и 49).

В случае, если вместо этого мы обучаем CNN на гендерных данных, мы инициализируем тот же набор переменных, только с разными путями к изображениям:

```

51 # в противном случае проверяем, не выполняем ли мы "гендер"
52 # классификация

```

```

53 elif DATASET_TYPE == "пол":
54     # определить количество меток для набора данных "gender", а также
55     # с путем к энкодеру метки
56     NUM_CLASSES = 2
57     LABEL_ENCODER_PATH = path.sep.join([OUTPUT_BASE,
58                                         "gender_le.cpickle"])
59
60     # определить путь к выходным данным для обучения, проверки и тестирования
61     # списки
62     TRAIN_MX_LIST = path.sep.join([MX_OUTPUT,
63                                    "списки/gender_train.lst"])
64     VAL_MX_LIST = path.sep.join([MX_OUTPUT,
65                                "списки/gender_val.lst"])
66     TEST_MX_LIST = path.sep.join([MX_OUTPUT,
67                                   "списки/gender_test.lst"])
68
69     # определить путь к выходным данным для обучения, проверки и тестирования
70     # записи изображений
71     TRAIN_MX_REC = path.sep.join([MX_OUTPUT, "rec/gender_train.rec"])
72     VAL_MX_REC = path.sep.join([MX_OUTPUT, "rec/gender_val.rec"])
73     TEST_MX_REC = path.sep.join([MX_OUTPUT, "rec/gender_test.rec"])
74
75     # получаем путь к файлу среднего пикселя
76     DATASET_MEAN = path.sep.join([OUTPUT_BASE,
77                                   "gender_adience_mean.json"])

```

Обратите внимание, что теперь переменная NUM_CLASSES была изменена на две (либо «мужской», либо «женский»), с последующей заменой всех вхождений «возраст» на «пол» в путях к файлам. Теперь, когда наш файл конфигурации создан, давайте перейдем к созданию вспомогательного класса, используемого для облегчения сборки набора данных и преобразование его в формат, который мы можем использовать с mxnet.

Вспомогательный класс Adience

Чтобы работать с набором данных Adience, нам сначала нужно определить набор служебных функций, аналогично тому, что мы сделали с набором данных ImageNet. Эта функция будет отвечать за помочь мы создаем наши метки классов, пути к изображениям и помогает в точности вычислений. Назовем этот файл agegenderhelper.py и включите его в подмодуль pyimagesearch.utils:

```

| --- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- ИО
| |--- нн
| |--- предварительная обработка
| |--- утилиты
| | |--- __init__.py
| | |--- agegenderhelper.py
| | |--- captchahelper.py
| | |--- imangenethelper.py
| | |--- Ranked.py

```

Когда будете готовы, откройте agegenderhelper.py, и мы приступим к работе:

```

1 # импортируем необходимые пакеты
2 импортировать numpy как np
3 импортировать глобус
4 импорт cv2
5 импорт ОС
6
7 класс AgeGenderHelper:
8     def __init__(я, конфигурация):
9         # сохранить объект конфигурации и построить используемые возрастные бины
10        # для создания меток классов
11        self.config = конфигурация
12        self.ageBins = self.buildAgeBins()

```

Строки 2-5 импортируют наши необходимые пакеты Python, а строка 8 определяет конструктор для нашего класса AgeGenderHelper. Конструктор требует один аргумент, config, который, как предполагается, быть модулем age_gender_config. Мы сохраняем этот объект конфигурации в строке 11 и строим набор ageBins в строке 12.

Мы определяем функцию buildAgeBins ниже:

```

14     деф buildAgeBins (я):
15         # инициализируем список возрастных бинов на основе Adience
16         # набор данных
17         ageBins = [(0, 2), (4, 6), (8, 13), (15, 20), (25, 32),
18             (38, 43), (48, 53), (60, нп.инф)]
19
20         # вернуть возрастные бины
21         вернуть ageBins

```

Строки 17 и 18 создают список из двух кортежей, представляющих нижнюю границу возрастной группы. и верхняя граница возрастной группы. Как видите, определено восемь возрастных групп, идентичен Леви и др. AgeBins возвращаются вызывающей функции в строке 21.

Учитывая произвольную возрастную группу или гендерное имя, нам нужно закодировать ввод как метку — следующая функция обрабатывает это:

```

23     деф toLabel(я, возраст, пол):
24         # проверяем, должны ли мы определять возрастную метку
25         если self.config.DATASET_TYPE == "возраст":
26             вернуть self.toAgeLabel(возраст)
27
28         # в противном случае предположим, что мы определяем метку пола
29         вернуть self.toGenderLabel(пол)

```

Эта функция принимает как возраст, так и пол. Если мы работаем с набором возрастных данных, то мы вызовите toAgeLabel и верните значение. В противном случае мы будем считать, что работаем с полем набора данных и вместо этого закодируйте метку пола.

Следующим шагом является определение функции toAgeLabel:

```

31     деф toAgeLabel(я, возраст):
32         # инициализируем метку
33         метка = нет
34

```

```

35         # разбить кортеж age на целые числа
36         возраст = возраст.заменить("(", "").заменить(")", "").split(", ")
37         (ageLower, ageUpper) = np.array(возраст, dtype="int")
38
39         # цикл по возрастным корзинам
40         for (нижний, верхний) в self.ageBins:
41             # определить, попадает ли возраст в текущий бин
42             если ageLower >= ниже и ageUpper <= выше:
43                 label = "{}_{}".format(нижний, верхний)
44                 переменя
45
46             # вернуть метку
47             ярлык возврата

```

Здесь мы передаем единственный параметр, строку возраста . В наборе данных Adience возрастные метки представлен в виде строк в виде (0, 2), (4, 6) и т. д. Чтобы определить правильную метку для этой входной строке нам сначала нужно извлечь нижнюю и верхнюю возрастные границы как целые числа (строки 36 и 37).

Затем мы перебираем наши ageBins (строка 40) и определяем, в какую ячейку попадает указанный возраст . (строка 42). Как только мы нашли правильный бункер, мы строим нашу метку в виде строки с нижним и верхние границы разделены знаком подчеркивания. Например, если входной возраст (8, 13) был передан в эту функцию вывод будет 8_13 . Выполнение этого кодирования облегчает нам анализировать метки как при обучении, так и при оценке нашей сети.

Теперь, когда мы можем создавать возрастные метки, нам также нужно создать гендерные метки:

```

49     def toGenderLabel(я, пол):
50         # вернуть 0, если пол мужской, 1, если пол женский
51         вернуть 0 , если пол == "м" , иначе 1

```

Эта функция проста: если введен пол м (мужской), мы возвращаем 0 ; в противном случае, предполагается, что пол женский, поэтому мы возвращаем 1 .

При оценке однократной точности нам нужен эффективный и быстрый метод для определения того, метка прогнозируемого класса равна или примыкает к метке достоверности . Самый простой способ выполнить эта задача состоит в том, чтобы определить словарь, который сопоставляет наземную метку с соответствующими соседними метками . Например, если бы мы знали, что наземная метка для данной точки данных равна 8_13 , мы могли бы использовать это value в качестве ключа к нашему словарю . Тогда значение будет ["4_6" , "8_13" , "15_20"] — набор соседних этикеток . Простая проверка наличия 8_13 в этом списке позволила бы нам быстро оценить разовая точность .

Чтобы определить этот словарь, мы создадим более сложную функцию buildOneOffMappings :

```

53     def buildOneOffMappings (я, файл):
54         # сортируем метки классов по возрастанию (по возрасту)
55         # и инициализируем одноразовые отображения для точности вычислений
56         классы = отсортированные (le.classes_ , ключ = лямбда x:
57             int(x.decode("utf-8").split("_")[0]))
58         одинВыкл = {}

```

Наш метод будет принимать один обязательный аргумент, le , который является экземпляром объекта LabelEncoder , используемый для кодирования меток возрастного класса . Как мы знаем из предыдущего определения этого класса возрастные метки будут кодироваться в виде {нижний}_ {верхний} . Для определения соседнего возраста

границы, нам сначала нужно извлечь имена меток классов из файла , а затем отсортировать имена меток в порядке возрастания по нижней границе (строки 56 и 57). Стока 58 определяет oneOff словарь, который мы будем использовать для быстрой оценки одноразовой точности.

Далее нам нужно перебрать индекс и имя отсортированных меток классов:

```

60      # цикл по индексу и имени (отсортированных) меток классов
61      для (я, имя) в перечислении (классы):
62          # определить индекс *текущего* имени метки класса
63          # в списке *label encoder* (ненумерованном), затем
64          # инициализируем индекс предыдущего и следующего возраста
65          # группы рядом с текущей меткой
66          текущий = np.where (le.classes_ == имя) [0] [0]
67          предыдущая = -1
68          следующий = +1
69
70          # проверяем, должны ли мы вычислять предыдущие соседние
71          # возрастная группа
72          если я > 0:
73              prev = np.where (le.classes_ == классы [i - 1]) [0] [0]
74
75          # проверяем, должны ли мы вычислять следующий соседний
76          # возрастная группа
77          если я < len (классы) - 1:
78              следующий = np.where (le.classes_ == классы [i + 1]) [0] [0]
79
80          # построить кортеж, состоящий из текущего возраста
81          # группа, предыдущая возрастная группа и следующая возрастная группа
82          # скобка
83          oneOff[current] = (текущий, предыдущий, следующий)
84
85          # возвращаем одноразовые сопоставления
86          вернуть одинВыкл.

```

Стока 66 определяет индекс имени текущей метки класса. При условии, что наша возрастная группа превышает диапазон 0–2 , затем мы определяем предыдущую соседнюю скобку (строки 72 и 73). В случае, если текущая возрастная группа меньше диапазона 60+ , мы определяем следующую соседнюю скобку (строки 77 и 78).

Затем словарь oneOff обновляется, используя текущую возрастную группу в качестве ключа с 3-кортежем, состоящим из текущей метки, а также двух соседних меток предыдущего и следующего классов. Затем словарь oneOff возвращается вызывающей функции в строке 86.

Вспомогательные функции, подобные этим, — отличный пример того, почему важно изучать основы программирования на Python, библиотеки NumPy и иметь хотя бы базовое представление о scikit-learn библиотеке. Используя эти методы, мы можем легко построить методы, которые позволят нам оценить сложные метрики эффективно. Если у вас возникли проблемы с пониманием этой функции, возьмите время, чтобы поработать с ним и вставить операторы печати , чтобы вы могли видеть, как данный возрастной класс в файле кодировщик меток может вычислить две соседние возрастные группы.

Наша следующая функция занимается построением путей изображения и соответствующих меток к точкам данных. в наборе данных Adience:

```

88      деф buildPathsAndLabels (я):
89          # инициализируем список путей к изображениям и меток
90          пути = []

```

```

91         метки = []
92
93         # берем пути к файлам folds
94         foldPaths = os.path.sep.join([self.config.LABELS_PATH,
95                                         "*.текст"])
96         foldPaths = glob.glob(foldPaths)

```

В строках 90 и 91 мы инициализируем наши списки путей и меток соответственно. И возраст, и пол данного человека хранится в файлах LABELS_PATH или «свернутых» файлах .txt — мы берем пути ко всем свернутым файлам в строках 94-96.

Затем давайте пройдемся по каждому из foldPaths и проанализируем их по отдельности:

```

98     # защищаем пути скобами
99     для foldPath в foldPaths:
100        # загружаем содержимое файла folds, пропуская
101        # заголовок
102        строки = открыть(foldPath).read()
103        строки = строки.strip().split("\n")[1:]
104
105        # цикл по строкам
106        для строки в строке:
107            # распаковываем необходимые компоненты строки
108            строка = строка.split("\t")
109            (идентификатор пользователя, путь к изображению, идентификатор лица, возраст, пол) = строка [: 5]
110
111        # если возраст или пол недействительны, игнорируйте образец
112        если age[0] != "(" или пол не в ("m", "f"):
113            Продолжить

```

Для всех файлов foldPath мы загружаем содержимое в строки 102 и 103. Затем мы перебираем каждая из строк во входном файле в строке 106. Каждая строка разделена табуляцией, которую мы разбиваем на список строк в строке 108. Затем мы извлекаем пять первых записей из строки:

1. userID: уникальный идентификатор объекта на фотографии.
2. imagePath: путь к входному изображению.
3. faceID: Уникальный идентификатор самого лица.
4. age: метка возраста, закодированная строкой в формате (нижний, верхний).
5. пол: метка пола, представленная одним символом, либо m, либо f.

Строки 112 и 113 гарантируют, что точка данных верна, обеспечивая правильное кодирование возраста . и пол либо мужской, либо женский. Если хотя бы одно из этих условий не выполняется, мы выбрасываем точку данных из-за неоднозначной маркировки. Если этикетки пройдут проверку на работоспособность, мы сможем двигаться дальше. для создания путей к нашим входным изображениям и кодирования метки:

```

115        # создаем путь к входному изображению и строим
116        # метку класса
117        p = "landmark_aligned_face.{}.{}".format(faceID,
118                                         путь к изображению)
119        p = os.path.sep.join([self.config.IMAGES_PATH,
120                             ID пользователя, p])
121        label = self.toLabel(возраст, пол)

```

Строки 117 и 118 начинают строить путь к входному изображению, комбинируя faceID с именем файла. Затем мы заканчиваем построение пути к изображению в строках 119 и 120.

Мы можем проверить, работает ли этот код, проверив пример пути к изображению в наборе данных Adience:

adience/aligned/100003415@N08/landmark_aligned_face.2174.9523333835_c7887c3fde_o.jpg

Обратите внимание, что первым компонентом пути является adience, наш базовый каталог. Следующим подкаталогом является идентификатор пользователя. Внутри подкаталога userID есть несколько изображений, каждое из которых начинается с базового имени файлагround_aligned_face.*. Чтобы завершить построение пути к изображению, мы указываем идентификатор лица, а затем остальную часть имени файла, imagePath. Опять же, вы можете проверить этот код самостоятельно, изучив пути к файлам в наборе данных Adience. Затем в строке 121 метка кодируется с помощью нашей функции toLabel.

Наконец, мы можем обновить наши списки путей и меток:

```

123      # если метка None, то возраст не вписывается # в наши
124      # возрастные рамки, игнорируйте образец , если метка None :
125
126          Продолжать
127
128      # обновить соответствующие пути к изображениям и списки
129      # меток paths.append(p) labels.append(label)
130
131
132      # вернуть кортеж путей к изображениям и меток
133      return (paths, labels)

```

Строка 125 обрабатывает случай, когда наша метка имеет значение « Нет», что происходит, когда возраст не соответствует нашим возрастным группам, вероятно, из-за (1) ошибки при ручной маркировке набора данных или (2) устаревшей возрастной группы, которая не была удалена . из набора данных Adience. В любом случае мы отбрасываем точку данных, поскольку метка неоднозначна. Строки 129 и 130 обновляют наши списки путей и меток соответственно, которые затем возвращаются вызывающему методу в строке 133.

Еще две функции определены в нашем классе AgeGenderHelper с именами visualizeAge и visualizeGender соответственно. Эти функции принимают предсказанное распределение вероятностей для каждой метки и строят простую столбчатую диаграмму, позволяющую визуализировать распределение меток классов точно так же, как мы делали это в главе 11, посвященной распознаванию эмоций и выражений лица. Поскольку это уже длинная глава, я оставлю обзор этих функций в качестве упражнения для читателя — они просто используются для визуализации и не играют никакой роли в создании классификатора глубокого обучения для распознавания возраста и пола.

Создание файлов списка

Наш сценарий build_dataset.py почти идентичен сценарию, который мы использовали при создании набора данных о марке и модели автомобиля в главе 13. Давайте продолжим и (кратко) просмотрим имя файла:

```

1 # импортировать необходимые пакеты
2 из config импортировать age_gender_config as config 3 из
sklearn.preprocessing import LabelEncoder 4 из
sklearn.model_selection import train_test_split 5 из pyimagesearch.utils
import AgeGenderHelper 6 import numpy as np 7 import progressbar 8
import pickle 9 import json

```

```

10 import cv2
11
12 # инициализируем наш вспомогательный класс, затем создаем набор путей к
изображениям 13 # и метки классов
14 print("[INFO] создание путей и меток...")
15 agh =
AgeGenderHelper(config)
16 (trainPaths, trainLabels) =
agh.buildPathsAndLabels()

```

Строки 2-10 импортируют необходимые пакеты Python. Обратите внимание на импорт нашего age_gender_config — этот импорт позволит нам использовать один и тот же скрипт build_dataset.py , независимо от того, строим ли мы набор данных о возрасте или поле. Чтобы упростить создание путей к входным изображениям и соответствующих меток классов на основе DATASET_TYPE , мы также импортируем наш недавно реализованный AgeGenderHelper .

Затем с помощью AgeGenderHelper мы строим пути к входным изображениям и меткам классов в строках 15 и 16.

Теперь, когда у нас есть общее количество изображений в обучающем наборе, давайте выведем количество изображений для проверочного и тестового наборов соответственно:

```

18 # теперь, когда у нас есть общее количество изображений в наборе данных,
которые 19 # можно использовать для обучения, вычислите количество изображений,
которые 20 # следует использовать для проверки и тестирования 21 numVal =
int(len(trainPaths) * config.NUM_VAL_IMAGES) 22 numTest = int(len(trainPaths) *
config.NUM_TEST_IMAGES)
23
24 # наши метки классов представлены в виде строк, поэтому нам нужно 25 # их
закодировать
26 print("[INFO] метки кодирования...")
27 le =
LabelEncoder().fit(trainLabels) 28 trainLabels =
le.transform(trainLabels)

```

Строки 27 и 28 кодируют наши входные метки из строк в целые числа. Далее давайте попробуем numVal проверочные изображения из обучающего набора:

```

30 # выполнить выборку из обучающего набора для построения валидации 31 # набор
32 print("[INFO] создание проверочных данных...")
33 split =
train_test_split(trainPaths, trainLabels, test_size=numVal,
stratify=trainLabels) 35
34
36 (trainPaths, valPaths, trainLabels, valLabels) = разделить

```

Затем то же самое делается для изображений numTest для создания нашего тестового набора:

```

37 # выполнить стратифицированную выборку из обучающей выборки для построения
38 # тестовой выборки 39 print("[INFO] построение тестовых данных...")
40 split =
train_test_split(trainPaths, trainLabels, test_size=numTest,
stratify=trainLabels) 42
41
43 (trainPaths, testPaths, trainLabels, testLabels) = разделить

```

Затем мы создадим наши списки наборов данных , где каждая запись представляет собой 4-кортеж, содержащий разделение набора данных. тип, пути к изображениям, метки классов и выходной файл .lst:

```

44 # составить список, объединяющий обучение, проверку и тестирование
45 # пути к изображениям вместе с соответствующими метками и списком вывода
46 # файлов

47 наборов данных =
48     ("поезд", trainPaths, trainLabels, config.TRAIN_MX_LIST),
49     ("val", valPaths, valLabels, config.VAL_MX_LIST),
50     ("тест", testPaths, testLabels, config.TEST_MX_LIST)
51

52 # инициализируем списки усреднений каналов RGB
53 (R, G, B) = ([], [], [])

```

Мы также инициализируем наши средние значения RGB — эти значения позволят нам выполнить средняя нормализация в процессе обучения.

Далее нам нужно перебрать каждую из записей в списке наборов данных:

```

55 # цикл по кортежам набора данных
56 для (dType, paths, labels, outputPath) в наборах данных:
57     # открываем выходной файл для записи
58     print("[INFO] здание {}".format(outputPath))
59     f = открыть (выходной путь, "w")

60
61     # инициализируем индикатор выполнения
62     widgets = ["Список построек: ", progressbar.Percentage(), " ",
63                 progressbar.Bar(), " ", progressbar.ETA()]
64     pbar = progressbar.ProgressBar (maxval = len (пути),
65                                    виджеты=виджеты).start()

```

Строка 59 открывает указатель файла на текущий файл .lst . Мы также инициализируем виджеты индикатора прогресса в строках 62-65. Индикатор выполнения не является обязательным требованием, но часто полезно иметь информацию о ожидаемом времени прибытия. отображается на нашем экране при построении набора данных.

Наш следующий блок кода обрабатывает каждый из путей к изображению и соответствующие метки в разделение данных:

```

67     # перебираем каждое отдельное изображение + метки
68     для (i, (путь, метка)) в перечислении (zip (пути, метки)):
69         # если мы строим обучающий набор данных, то вычисляем
70         # среднее значение каждого канала в изображении, затем обновите
71         # соответствующие списки
72         если dType == "поезд":
73             изображение = cv2.imread (путь)
74             (b, g, r) = cv2.mean(изображение)[:3]
75             R.добавить(r)
76             G. добавить (g)
77             Б. добавить (б)

78
79         # записываем индекс изображения, метку и выходной путь в файл
80         row = "\t".join([str(i), str(label), path])
81         f.write("{}\n".format(строка))
82         pbar.update(я)

83
84         # закрываем выходной файл
85         pbar.finish()
86         е.закрыть()

```

Если мы изучаем тренировочный сплит, мы загружаем изображение с диска, вычисляем среднее значение RGB и обновляем наши списки средних значений (строки 72–77). В противном случае мы записываем в выходной файл строку в формате mxnet , содержащую уникальное целое число изображения i , метку класса и путь к входному изображению (строки 80 и 81). Стока 86 закрывает указатель файла, и цикл for перезапускается для следующего разделения данных (до тех пор, пока мы не создадим файлы .lst для каждого из разделений данных).

Наш последний блок кода обрабатывает сериализацию средств RGB на диск, а также кодировщик меток:

```
88 # создать словарь средних значений, затем сериализовать значения в файл 89 # JSON 90 print("[INFO]
серIALIZУЮЩИЕ СРЕДСТВА...")
```

```
91 D = {"R": np.mean(R), "G": np.mean(G), "B": np.mean(B)} 92 f =
open(config.DATASET_MEAN, "w") 93 f.write(json.dumps(D)) 94 f.close()
```

```
95
96 # сериализовать кодировщик этикеток
97 print("[INFO] сериализующий кодировщик этикеток...")
98 f = open(config.LABEL_ENCODER_PATH, "wb") 99
f.write(pickle.dumps(le)) 100 f.close()
```

Чтобы создать наши файлы .lst для набора данных «пол», убедитесь, что для параметра DATASET_TYPE установлено значение « пол ». в age_gender_config.py и выполните следующую команду:

```
$ python build_dataset.py [INFO]
создание путей и меток...
[INFO] метки кодировки...
[INFO] создание проверочных данных...
[INFO] построение тестовых данных...
[INFO] здание /raid/datasets/adience/lists/gender_train.lst...
Список построек: 100% ##### | Время: 0:01:01
[ИНФО] здание /raid/datasets/adience/lists/gender_val.lst...
Список построек: 100% ##### | Время: 0:00:00
[ИНФО] здание /raid/datasets/adience/lists/gender_test.lst...
Список построек: 100% ##### | Время: 0:00:00
[INFO] сериализация означает...
[INFO] сериализация кодировщика этикеток...
```

Вы можете проверить правильность создания файлов .lst для определения пола , перечислив содержимое вашего *_MX_LIST переменные:

```
$ wc -l adience/lists/gender_*.lst
1712 adience/lists/gender_test.lst 7991
adience/lists/gender_train.lst 1712 adience/
lists/gender_val.lst
всего 11415
```

Здесь вы можете видеть, что у нас есть в общей сложности 11 415 изображений в нашем наборе данных, из которых 8 000 изображений для обучения и 1 700 изображений для проверки и тестирования.

Чтобы создать файлы .lst для набора данных «возраст», вернитесь к файлу age_gender_config.py и обновите DATASET_TYPE до возраста. Затем еще раз выполните build_dataset.py:

```
$ python build_dataset.py [INFO]
создание путей и меток...
[INFO] метки кодировки...
[INFO] создание проверочных данных...
[INFO] построение тестовых данных...
[INFO] здание /raid/datasets/adience/lists/age_train.lst...
Список построек: 100% ##### | Время: 0:00:52 [INFO]
здания /raid/datasets/adience/lists/age_val.lst...
Список построек: 100% ##### | Время: 0:00:00 [ИНФО]
здания /raid/datasets/adience/lists/age_test.lst...
Список построек: 100% ##### | Время: 0:00:00 [INFO]
сериализация означает...
[INFO] сериализация кодировщика этикеток...
```

Еще раз убедитесь, что ваши файлы .lst были успешно созданы для каждого разделения данных :

```
$ wc -l аудитория/справки/возраст_*.lst
1711 adience/lists/age_test.lst 7986 adience/
lists/age_train.lst 1711 adience/lists/
age_val.lst Всего 11408
```

Создание базы данных записей Имея

наши файлы .lst для наборов данных о возрасте и поле, давайте создадим наши файлы .rec, используя двоичный файл mxnet im2rec . Мы начнем с создания файлов возрастных записей:

```
$ ~/mxnet/bin/im2rec /raid/datasets/adience/lists/age_train.lst "" \
/raid/datasets/adience/rec/age_train.rec resize=256 encoding='.jpg' \ качество=100
...
[07:28:16] tools/im2rec.cc:298: Всего: обработано 7986 изображений, прошло 42,5361 с $ ~/
mxnet/bin/im2rec /raid/datasets/adience/lists/age_val.lst "" \
/raid/datasets/adience/rec/age_val.rec resize=256 encoding='.jpg' \ качество=100
...
[07:28:51] tools/im2rec.cc:298: Всего: обработано 1711 изображений, прошло 10,5159 с $ ~/
mxnet/bin/im2rec /raid/datasets/adience/lists/age_test.lst "" \
/raid/datasets/adience/rec/age_test.rec resize=256 encoding='.jpg' \ качество=100
...
[07:29:20] tools/im2rec.cc:298: Всего: обработано 1711 изображений, прошло 10,7158 с.
```

А затем переключитесь на файлы гендерных записей:

```
$ ~/mxnet/bin/im2rec /raid/datasets/adience/lists/gender_train.lst "" \
/raid/datasets/adience/rec/gender_train.rec resize=256 encoding='.jpg' \ качество=100
...
[07:30:35] tools/im2rec.cc:298: Всего: обработано 7991 изображения, прошло 43,2748 с $ ~/
mxnet/bin/im2rec /raid/datasets/adience/lists/gender_val.lst "" \
/raid/datasets/adience/rec/gender_val.rec resize=256 encoding='.jpg' \ качество=100
```

```
...
[07:31:00] tools/im2rec.cc:298: Всего: обработано 1712 изображений, затрачено 9,81215 с.
$ ~/mxnet/bin/im2rec/raid/datasets/adience/lists/gender_test.lst "" \
    /raid/datasets/adience/rec/gender_test.rec resize=256 encoding='jpg' \
    качество=100
...
[07:31:27] tools/im2rec.cc:298: Всего: обработано 1712 изображений, прошло 9,5392 с.
```

Чтобы убедиться, что ваши файлы записей были созданы, просто проверьте пути к *_MX_REC. переменные:

```
$ ls -l аудитория/запись/
всего 1082888
-rw-rw-r-- 1 адриан адриан 82787512 28 августа 07:29 age_test.rec
-rw-rw-r-- 1 адриан адриан 387603688 28 августа 07:28 age_train.rec
-rw-rw-r-- 1 адриан адриан 83883944 28 августа 07:28 age_val.rec
-rw-rw-r-- 1 адриан адриан 83081304 28 августа 07:31 гендер_тест.rec
-rw-rw-r-- 1 адриан адриан 388022840 28 августа 07:30 gender_train.rec
-rw-rw-r-- 1 адриан адриан 83485564 28 августа 07:31 gender_val.rec
```

Конечно же, все шесть наших файлов там: три для обучения, тестирования и проверки, разделенные на возраст. И три файла для тренировочного тестирования и валидации с разбивкой по полу. Каждый из обучающих файлов составляют примерно 388 МБ, в то время как все файлы тестирования и проверки весят 83 МБ. Хорошо использовать эти файлы записей для обучения нашей CNN, но сначала давайте определим саму сетевую архитектуру.

14.3 Реализация нашей сетевой архитектуры

Сетевая архитектура, используемая Levi et al. похожа на AlexNet (глава 6), только:

1. Более мелкий, без нескольких слоев CONV => RELU, наложенных друг на друга.
2. Меньше узлов в полносвязных слоях.

Мы воспроизведем их точную архитектуру за двумя исключениями. Во-первых, мы будем использовать пакетный нормализация, а не ныне устаревшая нормализация локального ответа (LRN), которая использовалась в более ранних версиях. Архитектуры CNN. Во-вторых, мы добавим небольшое количество отсева после каждого объединяющего слоя. помочь уменьшить переоснащение.

Давайте продолжим и реализуем Levi et al. архитектуры, которую мы будем называть MxAgegenderNet.

Создайте новый файл с именем mxagegendernet.py внутри подмодуля nn.mxconv pyimagesearch:

```
-- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- нн
| |--- __init__.py
| |--- конв.
| |--- mxconv
| | |--- __init__.py
| | |--- mxagegender.py
| | |--- mxalexnet.py
| | |--- mxgooglenet.py
| | |--- mxresnet.py
| | |--- mxsqueezenet.py
| | |--- mxvggnet.py
| |--- предварительная обработка
| |--- утилиты
```

Оттуда откройте файл и вставьте следующий код:

```

1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3
4 класс MxAgeGenderNet:
    @статический метод
    деф- сборка (классы):
        # ввод данных
        данные = mx.sym.Variable("данные")
9
10     # Блок №1: первый набор слоев CONV => RELU => POOL
11     conv1_1 = mx.sym.Convolution (данные = данные, ядро = (7, 7),
12         шаг=(4, 4), num_filter=96)
13     act1_1 = mx.sym.Activation(data=conv1_1, act_type="relu")
14     bn1_1 = mx.sym.BatchNorm (данные = act1_1)
15     pool1 = mx.sym.Pooling(data=bn1_1, pool_type="max",
16         ядро = (3, 3), шаг = (2, 2))
17     do1 = mx.sym.Dropout (данные = pool1, p = 0,25)

```

Строка 6 определяет метод построения нашей сети, как и во всех предыдущих архитектурах, описанных в эта книга. Переменная данных в строке 8 служит входом в нашу сеть.

Оттуда мы определяем серию слоев CONV => RELU => POOL в строках 11-17. Наш первый слой CONV в сети изучит 96 ядер 7×7 с шагом 4×4 , используемых для уменьшения пространственного размеры входных изображений 227×227 . Активация применяется после свертки, за которой следует путем пакетной нормализации в соответствии с рекомендуемым использованием пакетной нормализации (глава 11, Стартовый набор).

Максимальное объединение применяется к строкам 15 и 16 с размером ядра 3×3 и шагом 2×2 для еще раз уменьшить пространственные размеры. Отсев с вероятностью 25% используется, чтобы помочь уменьшить переоснащение.

Наша вторая серия слоев применяет ту же структуру, только на этот раз настраивая слой CONV . чтобы выучить фильтры 256, 5×5 :

```

19     # Блок №2: второй набор слоев CONV => RELU => POOL
20     conv2_1 = mx.sym.Convolution (данные = do1, ядро = (5, 5),
21         pad=(2, 2), num_filter=256)
22     act2_1 = mx.sym.Activation(data=conv2_1, act_type="relu")
23     bn2_1 = mx.sym.BatchNorm (данные = act2_1)
24     pool2 = mx.sym.Pooling(data=bn2_1, pool_type="max",
25         ядро = (3, 3), шаг = (2, 2))
26     do2 = mx.sym.Dropout (данные = pool2, p = 0,25)

```

Окончательный набор слоев CONV => RELU => POOL почти идентичен, только количество фильтров увеличен до 384, а размер фильтра уменьшен до 3×3 :

```

28     # Блок №3: второй набор слоев CONV => RELU => POOL
29     conv2_1 = mx.sym.Convolution (данные = do2, ядро = (3, 3),
30         pad=(1, 1), num_filter=384)
31     act2_1 = mx.sym.Activation(data=conv2_1, act_type="relu")
32     bn2_1 = mx.sym.BatchNorm (данные = act2_1)
33     pool2 = mx.sym.Pooling(data=bn2_1, pool_type="max",

```

```

34     ядро = (3, 3), шаг = (2, 2))
35     do3 = mx.sym.Dropout (данные = pool2, p = 0,25)

```

Далее идет наш первый набор полно связанных слоев:

```

37     # Блок №4: первый набор слоев FC => RELU
38     сгладить = mx.sym.Flatten (данные = do3)
39     fc1 = mx.sym.FullyConnected (данные = сглаживание, num_hidden = 512)
40     act4_1 = mx.sym.Activation(data=fc1, act_type="relu")
41     bn4_1 = mx.sym.BatchNorm (данные = act4_1)
42     do4 = mx.sym.Dropout (данные = bn4_1, p = 0,5)

```

В отличие от архитектуры AlexNet, где изучается 4096 скрытых узлов, мы изучаем только 512 узлов . здесь. Мы также применяем активацию с последующей нормализацией партии в этом наборе слоев.

Затем применяется второй набор полно связанных слоев:

```

44     # Блок №5: второй набор слоев FC => RELU
45     fc2 = mx.sym.FullyConnected (данные = do4, num_hidden = 512)
46     act5_1 = mx.sym.Activation(data=fc2, act_type="relu")
47     bn5_1 = mx.sym.BatchNorm (данные = act5_1)
48     do5 = mx.sym.Dropout (данные = bn5_1, p = 0,5)

```

Наш последний блок кода — это просто уровень FC для изучения желаемого количества классов, а также классификатор softmax:

```

50     # классификатор softmax
51     fc3 = mx.sym.FullyConnected (данные = do5, num_hidden = классы)
52     модель = mx.sym.SoftmaxOutput (данные = fc3, имя = "softmax")
53
54     # вернуть архитектуру сети
55     модель возврата

```

С точки зрения сетевой архитектуры MxAgeGenderNet , возможно, является самой простой сетевой архитектурой , которую мы исследовали в течение всего пакета ImageNet. Однако, поскольку наши результаты будут продемонстрировать, что эта простая последовательная сетевая архитектура способна не только воспроизвести результаты Леви и др., но также улучшить их работу.

14.4 Измерение «разовой» точности

Как упоминалось в разделе 14.2.1 выше, при обучении и оценке MxAgeGenderNet на гендерный набор данных, мы хотим вычислить «однократную точность». В отличие от точности пятого ранга, этот показатель помечает данный прогноз как «правильный», если предсказанная метка является либо (1) меткой достоверности, либо (2) непосредственно рядом с меткой достоверности.

На практике этот показатель оценки имеет практический смысл, и часто очень субъективны и сильно коррелируют с генетикой, внешностью и факторами окружающей среды. За Например, 18-летний мужчина, выкуривающий по блоку сигарет в день на протяжении 20 лет, скорее всего, будет выглядеть намного старше в 38 лет, чем биологически правильный 38-летний мужчина.

Чтобы создать нашу специальную одноразовую метрику, давайте создадим новый подмодуль внутри pyimagesearch. с именем mxcallbacks, и внутри этого нового подмодуля мы поместим новый файл — mxmetrics.py:

```

--- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- mxcallbacks
| | |--- __init__.py
| | |--- mxmetrics.py
| |--- нн
| |--- предварительная обработка
| |--- утилиты

```

Функция `mxmetrics.py` сохранит нашу функцию для вычисления разовой оценочной метрики. К сожалению, определить метрики обратного вызова `mxnet` не так просто, как с `Keras`, поэтому нам нужно уделите особое внимание определению функции. Откройте `mxmetrics.py`, и мы приступим к работе:

```

1 # импортируем необходимые пакеты
2 импортировать mxnet как mx
3 журнал импорта
4
5 def one_off_callback (trainIter, testIter, oneOff, ctx):
6     def _callback(iterNum, sym, arg, aux):
7         # построить модель для символа, чтобы мы могли делать прогнозы
8         # по нашим данным
9         модель = mx.mod.Module (символ = символ, контекст = ctx)
10        model.bind(data_shapes=testIter.provide_data,
11                   label_shapes=testIter.provide_label)
12        model.set_params (аргумент, вспомогательный)
13
14        # вычислить одноразовую метрику как для обучения, так и для тестирования
15        # данные
16        trainMAE = _compute_one_off (модель, trainIter, oneOff)
17        testMAE = _compute_one_off (модель, testIter, oneOff)
18
19        # регистрируем значения
20        logging.info("Эпоха[{}] Train-one-off={:.5f}".format(iterNum,
21                                               поездМАЭ))
22        logging.info("Эпоха[{}] Test-one-off={:.5f}".format(iterNum,
23                                               тестМАЭ))
24
25        # вернуть метод обратного вызова
26        вернуть _callback

```

Строки 2 и 3 импортируют необходимые пакеты Python. Нам понадобится пакет ведения журнала по порядку чтобы записывать одноразовую оценочную метрику в тот же файл журнала, что и наш прогресс обучения.

Оттуда мы определяем фактическую функцию обратного вызова `one_off_callback` в строке 5. Это функция принимает четыре обязательных параметра:

- `trainIter`: объект `ImageRecordIter` для обучающего набора.
- `testIter`: `imageRecordIter` либо для проверочного, либо для тестового набора.
- `oneOff`: словарь одноразовых отображений, созданный классом `AgeGenderHelper`.
- `ctx`: контекст (т. е. `CPU/GPU` и т. д.), который будет использоваться во время оценки.

Чтобы превратить это в настоящую функцию «обратного вызова», `mxnet` требует, чтобы мы определили инкапсулированную функцию Python (т. е. «внутреннюю» функцию) с именем `_callback`.

Инкапсулированные функции должны сначала прочитать это замечательное введение (<http://pyimg.co/fbchc>). Согласно `mxnet` инкапсулированная функция `_callback` должна принимать четыре параметра:

1. Номер итерации (т. е. эпохи) обучаемой сети.
2. Символ сети (sym), который представляет веса модели после текущей эпохи.
3. Параметры аргумента (arg) для сети.
4. Вспомогательные параметры (aux).

На основе этих значений мы можем создать модуль для параметров сети в строках 9-12. Мы затем вызовите `_compute_one_off` (который будет определен в следующем разделе) в строках 16 и 17 для вычисления разовые точности как для обучающего набора, так и для набора проверки/тестирования. Эти разовые точности записаны в наш файл истории тренировок (строки 20-23). Внешняя функция `one_off_callback` возвращает функцию `_callback` для `mxnet`, позволяя передать номер эпохи, символ модели и соответствующие параметры после каждой эпохи (строка 26).

Следующим шагом является определение функции `_compute_one_off`. Как следует из названия, этот метод будет отвечать за принятие экземпляра модели, итератор данных (`dataIter`) и `oneOff` сопоставление словаря, а затем вычисление разовой точности:

28 по определению `_compute_one_off` (модель, dataIter, oneOff):

```

29      # инициализируем общее количество выборок вместе с
30      # количество правильных (максимум разовых) классификаций
31      всего = 0
32      правильно = 0
33
34      # цикл по предсказаниям пакетов
35      для (preds, _, batch) в model.iter_predict(dataIter):
36          # преобразовать пакет прогнозов и меток в NumPy
37          # массивы
38          прогнозы = preds[0].asnumpy().argmax(ось=1)
39          метки = пакет.метка[0].asnumpy().astype("целое число")
40
41          # цикл по предсказанным меткам и меткам истинности
42          # в пакете
43          for (pred, label) в zip(predictions, labels):
44              # если правильная метка в наборе "одноразовая"
45              # предсказания, затем обновить правильный счетчик
46              если метка в oneOff[pred]:
47                  правильно += 1
48
49          # увеличить общее количество выборок
50          всего += 1
51
52          # закончить вычисление одноразовой метрики
53          вернуть правильно / с плавающей запятой (всего)

```

Строка 31 инициализирует общее количество выборок в `dataIter`. Затем строка 32 инициализирует количество правильных однократных прогнозов. Чтобы вычислить количество правильных одноразовых прогнозов, мы должны использовать метод модели `iter_predict` для циклического перебора сэмплов в `dataIter`, точно так же, как мы делали это в главе 13 по определению марки и модели автомобиля (строка 35).

Строка 38 получает выходные прогнозы из нашей модели для данного пакета, а затем находит индекс метки с наибольшей вероятностью. В строке 39 мы извлекаем метки достоверности, поэтому мы можем сравнить их с нашими предсказаниями. Мы перебираем каждый из предсказанных и основанных на истине метки в строке 43. Если метка истинности существует в наборе одноразовых меток для прогноза (пред), то мы увеличиваем количество правильных классификаций.

Затем мы увеличиваем общее количество выборок, проверенных в `dataIter` в строке 50. Наконец, строка 53 возвращает одноразовую метрику как отношение правильных прогнозов к общему количеству

точки данных.

Если этот обратный вызов на первый взгляд кажется немного запутанным, будьте уверены, вы не единственный разработчик, который изо всех сил пытался создать обратные вызовы mxnet (включая меня). Если вы столкнулись с трудностями при работе с этим кодом, я предлагаю рассматривать его как «метрику оценки черного ящика» и вернуться к нему позже, когда вы увидите, как он используется в нашем скрипте train.py.

Я бы также посоветовал вам прочитать документацию mxnet по существующим обратным вызовам (<http://pyimg.co/d2u11>) . а также обзор реальных реализаций этих обратных вызовов (<http://pyimg.co/fos3c>). В частности, обратите внимание на функцию module_checkpoint , которая дает пример использования инкапсулированного метода _callback.

14.5 Обучение предиктора нашего возраста и пола

Теперь, когда у нас есть сгенерированный набор данных записи, а также реализована наша одноразовая метрика оценки, давайте перейдем к train.py, сценарию, отвечающему за обучение как возрастных, так и гендерных CNN. Этот сценарий сможет обрабатывать как возрастные, так и гендерные CNN благодаря нашему файлу age_gender_config.py — на что бы мы ни установили DATASET_TYPE , CNN будет делать ставку. Кроме того, DATASET_TYPE также устанавливает все соответствующие выходные каталоги внутри age_gender_config. Это еще один отличный пример важности использования файлов конфигурации для проектов глубокого обучения.

Наш скрипт train.py будет почти идентичен всем другим нашим проектам в ImageNet Bundle, Итак, давайте кратко рассмотрим файл. Во-первых, мы начнем с нашего импорта:

```
1 # импортируем необходимые пакеты 2
из config import age_gender_config as config 3 из
pyimagesearch.nn.mxconv import MxAgeGenderNet 4 из
pyimagesearch.utils import AgeGenderHelper 5 из
pyimagesearch.mxcallbacks import one_off_callback 6 import mxnet as mx
7 import argparse 8 import logging 9 import pickle 10 импорт json 11
импорт ос
```

Три наиболее важных импорта, на которые следует обратить внимание, — это наша реализация MxAgeGenderNet , AgeGenderHelper и наш one_off_callback (строки 3–5). Оттуда мы можем проанализировать наши аргументы командной строки, а также наш файл журнала:

```
13 # построить разбор аргумента и разобрать аргументы 14 ap =
argparse.ArgumentParser() 15 ap.add_argument("-c", "--checkpoints",
required=True,
16     help="путь к выходному каталогу контрольных точек")
17 ap.add_argument("-p", "--prefix", required=True,
18     help="имя префикса модели") 19
ap.add_argument("-s", "--start-epoch", type=int, default=0,
20     help="эпоха для возобновления обучения")
21 args = vars(ap.parse_args())
22
23 # установить уровень логирования и выходной
файл 24 logging.basicConfig(level=logging.DEBUG,
25     filename="training_{}.log".format(args["start_epoch"]), filemode="w")
26
```

Переключатель `--checkpoints` управляет путем, по которому мы будем хранить сериализованные веса. MxAgeGenderNet после каждой эпохи. Переключатель `--prefix` управляет именем CNN. И наконец, если мы перезапускаем обучение с предыдущей эпохи, мы можем точно указать, какую эпоху через `--начало эпохи`.

Нам также нужно установить размер партии на основе `BATCH_SIZE` и `NUM_DEVICES` в модуль конфигурации:

```
28 # определить пакет и загрузить средние значения пикселей
29 batchSize = config.BATCH_SIZE * config.NUM_DEVICES
30 означает = json.loads(open(config.DATASET_MEAN).read())
```

Строка 30 обрабатывает загрузку наших средних значений RGB для набора данных о возрасте или поле соответственно. Давайте продолжайте и создайте наш итератор обучающих данных:

```
32 # построить итератор обучающего изображения
33 trainIter = mx.io.ImageRecordIter(
34     path_imgrec=config.TRAIN_MX_REC,
35     data_shape=(3, 227, 227),
36     batch_size = размер партии,
37     rand_crop = Верно,
38     rand_mirror = Верно,
39     повернуть = 7,
40     mean_r = означает ["R"],
41     mean_g = означает ["G"],
42     mean_b = означает ["B"],
43     preprocess_threads=config.NUM_DEVICES * 2)
```

Поскольку наши изображения уже предварительно выровнены в наборе данных Adience, мы не хотим применять слишком большое вращение, поэтому поворот на ± 7 градусов кажется здесь уместным. Мы также будем случайным образом обрезать 227×227 областей из входного изображения 256×256 , указав флаг `rand_crop`.

Нам также понадобится соответствующий итератор проверочного изображения:

```
45 # создать итератор проверочного изображения
46 valIter = mx.io.ImageRecordIter(
47     path_imgrec=config.VAL_MX_REC,
48     data_shape=(3, 227, 227),
49     batch_size = размер партии,
50     mean_r = означает ["R"],
51     mean_g = означает ["G"],
52     mean_b = означает ["B"])
```

Леви и др. использовали оптимизатор SGD для обучения своей сети — то же самое сделаем с базой скорость обучения $1e - 4$, момент импульса 0,9 и уменьшение веса L2 0,0005:

```
54 # инициализировать оптимизатор
55 opt = mx.optimizer.SGD(learning_rate=1e-3, импульс=0,9, wd=0,0005,
56     rescale_grad=1.0 / размер партии)
```

Давайте также определим выходной путь к контрольным точкам нашей модели и инициализируем аргумент модели. и вспомогательные параметры соответственно:

```

58 # построить путь контрольных точек, инициализировать аргумент модели и
59 # вспомогательные параметры
60 checkpointsPath = os.path.sep.join([args["checkpoints"],
61                                     аргументы["предикс"]])
62 argParams = Нет
63 вспомогательных параметра = нет

```

Далее мы можем определить, начинаем ли мы обучение с нуля или загружаемся с определенной эпохи:

```

65 # если не указана начальная эпоха конкретной модели, то
66 # инициализируем сеть
67 , если args["start_epoch"] <= 0:
68     # построить архитектуру LeNet
69     print("[INFO] строим сеть...")
70     модель = MxAgeGenderNet.build(config.NUM_CLASSES)
71
72 # в противном случае указана конкретная контрольная точка
73 еще:
74     # загружаем чекпойнт с диска
75     print("[INFO] загрузка эпохи {}".format(args["start_epoch"]))
76     (модель, argParams, auxParams) = mx.model.load_checkpoint(
77         контрольные точкиПуть, аргументы["начала_эпохи"])

```

Независимо от того, тренируемся ли мы с нуля или перезапускаем тренировки с определенной эпохи, следующая
Шаг - скомпилировать модель:

```

79 # компилируем модель
80 модель = mx.model.FeedForward(
81     ctx=[mx.gpu(2), mx.gpu(3)],
82     символ = модель,
83     инициализатор=mx.initializer.Xavier(),
84     arg_params = параметры аргумента,
85     aux_params = вспомогательные параметры,
86     оптимизатор=выбор,
87     Число_эпох = 110,
88     begin_epoch=args["start_epoch"])

```

Опять же, я использую два графических процессора для обучения нашей CNN; однако этот эксперимент можно легко провести с
только один графический процессор. Мы начинаем инициализировать наши обратные вызовы и метрики оценки ниже:

```

90 # инициализировать обратные вызовы и метрики оценки
91 batchEndCBs = [mx.callback.Спидометр (batchSize, 10)]
92 epochEndCBs = [mx.callback.do_checkpoint(checkpointsPath)]
93 метрики = [mx.metric.Accuracy(), mx.metric.CrossEntropy()]

```

Эти обратные вызовы будут применяться независимо от того, тренируемся ли мы на наборе данных age или gen.
набор данных. Однако, если мы конкретно работаем с возрастом, то у нас есть дополнительная работа:

```

95 # проверьте, следует ли использовать обратный вызов одноразовой точности
96 , если config.DATASET_MEAN == "возраст":

```

```

97      # загрузите кодировщик меток, затем создайте одноразовые сопоставления для
98      # точность вычислений
99      le = pickle.loads(open(config.LABEL_ENCODER_PATH, "rb").read())
100     agh = AgeGenderHelper(конфигурация)
101     oneOff = agh.buildOneOffMappings(le)
102     epochEndCBs.append(one_off_callback(trainIter, valIter,
103                               oneOff, mx.gpu(0)))

```

Строка 99 загружает наш age LabelEncoder с диска, а строка 100 создает экземпляр AgeGenderHelper.

Основываясь на конфигурации и файле, мы можем сгенерировать сопоставление словаря oneOff.

Наконец, строки 102 и 103 обновляют список epochEndCBs , добавляя one_off_callback функция. Мы поставляем итератор обучающих данных (trainIter), итератор данных проверки (valIter), сопоставления словаря oneOff и контекст устройства (mx.gpu(0)) в one_off_callback . Эти линии позволяют нам (1) вычислить разовую оценочную метрику как для обучения, так и для набор проверки и (2) также записывать результаты в наш файл истории обучения.

Вы также заметите, что я использую отдельный графический процессор для своего контекста. При работе с mxnet, я обнаружил, что для оптимальной производительности следует использовать отдельный GPU/CPU/устройство для оценки, чем те, которые используются для обучения. Я не совсем уверен, почему это быстрее, но я замечаю, что оценка замедляется, если вы пытаетесь использовать один и тот же контекст для обучения и оценки, используя one_off_callback — об этом несоответствии в скорости оценки следует помнить, когда разработка собственных метрик оценки.

Наш последний шаг — просто обучить сеть:

```

105 # обучить сеть
106 print("[INFO] обучающая сеть...")
107 модель.подходит(
108     X=поезд,
109     eval_data=значение,
110     eval_metric = показатели,
111     batch_end_callback=batchEndCB,
112     epoch_end_callback=epochEndCBs)

```

В следующем разделе мы создадим test_accuracy.py , чтобы мы могли оценить нашу MxAgeGenderNet. веса на соответствующих тестовых наборах. Затем в разделе 14.7 мы применим как train.py , так и test_accuracy.py для проведения экспериментов по предсказанию возраста и пола.

14.6 Оценка прогноза возраста и пола

Точно так же, как train.py можно использовать для обучения сети на наборах данных о возрасте или поле, наш

Скрипт test_accuracy.py сможет оценить точность наборов данных о возрасте и поле.

— для переключения между наборами данных нам просто нужно обновить DATASET_TYPE в age_gender_config.

Давайте теперь рассмотрим test_accuracy.py:

```

1 # поскольку 'AgeGenderHelper' также импортирует OpenCV, нам нужно поместить его
2 # над импортом mxnet, чтобы избежать ошибки сегментации
3 из pyimagesearch.utils import AgeGenderHelper
4
5 # импортируем необходимые пакеты
6 из конфига импортировать age_gender_config как конфиг
7 из pyimagesearch.mxcallbacks.mxmetrics import _compute_one_off
8 импортировать mxnet как mx

```

```
9 import argparse
импорт pickle 11 импорт
json 12 импорт os
```

Строки 3-12 отвечают за наш импорт Python. Поскольку класс AgeGenderHelper импортирует библиотеку cv2 , я поместил ее в начало файла. Как я упоминал ранее в этой книге, на двух из трех машин я настроил mxnet + OpenCV, импортируя cv2 до того, как mxnet вызвал ошибку сегментации. Размещение любого импорта cv2 перед импортом mxnet решило проблему. Маловероятно , что вы столкнетесь с этой проблемой, но если вы это сделаете, просто следуйте рекомендациям , приведенным выше, чтобы решить проблему.

Давайте теперь разберем наши аргументы командной строки:

```
14 # построить разбор аргумента и разобрать аргументы 15 ap =
argparse.ArgumentParser() 16 ap.add_argument("-c", "--checkpoints",
required=True,
17     help="путь к выходному каталогу контрольных
точек") 18 ap.add_argument("-p", "--prefix", required=True,
19     help="имя префикса модели") 20
ap.add_argument("-e", "--epoch", type=int, required=True, help="epoch # для
21     загрузки") 22 args = vars(ap.parse_args())

23
24 # загрузить средства RGB для тренировочного
набора 25 mean = json.loads(open(config.DATASET_MEAN).read())

```

Вы видели все эти аргументы командной строки в предыдущих главах пакета ImageNet. Переключатель --checkpoints управляет базовым каталогом, в котором находятся наши сериализованные веса MxAgeGenderNet. Префикс --это имя нашей сети. И, наконец, --epoch управляет целочисленным значением эпохи, которую мы хотим загрузить с диска. В сочетании друг с другом мы можем загрузить файл определенного веса с диска. Затем в строке 25 загружаются наши средние значения RGB, чтобы мы могли выполнить нормализацию среднего значения.

Доступ к тестовому набору требует создания ImageRecordIter:

```
27 # создать итератор тестового изображения _ _
["G"], mean_b=означает["B"])

29
30
31
32
33
34
```

Оттуда мы можем загрузить конкретную --epoch с диска:

```
36 # загрузить контрольную точку с диска
37 print("[INFO] loading model...") 38
checkpointsPath = os.path.sep.join([args["checkpoints"],
39     args["prefix"]]) 40
model = mx.model.FeedForward.load(checkpointsPath,
```

```

41     аргументы["эпоха"])
42
43 # компилируем модель
44 модель = mx.model.FeedForward(
45     ctx=[mx.gpu(0)],
46     символ=модель.символ,
47     arg_params=model.arg_params,
48     aux_params=model.aux_params)

```

Независимо от того, оцениваем ли мы нашу сеть по набору данных о возрасте или поле, нам нужно вычислить точность ранга 1, которая обрабатывается следующим кодовым блоком:

```

50 # делать прогнозы на данных тестирования
51 print("[INFO] предсказание по тестовым данным '{}' ...".format(
52     config.DATASET_TYPE))
53 метрики = [mx.metric.Accuracy()]
54 acc = model.score(testIter, eval_metric=metrics)
55
56 # отображать точность ранга 1
57 print("[INFO] rank-1: {:.2f}%".format(acc[0] * 100))

```

Однако, если мы работаем с набором возрастных данных, нам также необходимо вычислить разовую точность. также:

```

59 # проверьте, следует ли использовать обратный вызов одноразовой точности
60 , если config.DATASET_TYPE == "возраст":
61     # перекомпилируйте модель, чтобы мы могли вычислить наш собственный одноразовый
62     # показатель оценки
63     аргумент = model.arg_params
64     aux = модель.aux_params
65     модель = mx.mod.Module (символ = модель.символ, контекст = [mx.gpu (1)])
66     модель.bind(data_shapes=testIter.provide_data,
67                 label_shapes=testIter.provide_label)
68     модель.set_params (аргумент, вспомогательный)
69
70     # загрузите кодировщик меток, затем создайте одноразовые сопоставления для
71     # точность вычислений
72     le = pickle.loads(open(config.LABEL_ENCODER_PATH, "rb").read())
73     agh = AgeGenderHelper (конфигурация)
74     oneOff = agh.buildOneOffMappings(le)
75
76     # вычислить и отобразить одноразовую оценочную метрику
77     acc = _compute_one_off (модель, testIter, oneOff)
78     print("[INFO] одноразовый: {:.2f}%".format(acc * 100))

```

В строке 60 мы делаем проверку, чтобы убедиться, что мы оцениваем нашу сеть по возрастному набору данных. Если это так, мы распаковываем параметры модели в строках 63 и 64, после чего повторно инициализируем модель как объект модуля , который может быть совместим с нашей одноразовой метрикой точности. (строки 65-68).

Строки 72-74 строят сопоставления словаря oneOff . Вызов _compute_one_off on Line 77 вычисляет однократную точность для нашего тестового набора. Мы используем функцию _compute_one_off здесь, а не функцию one_off_callback , потому что нам нужно оценить точность на

один итератор данных (имейте в виду, что функция `one_off_callback` требует двух итераторов данных, один из которых предположительно является обучающим итератором, а другой — проверочным/тестирующим). Наконец, строка 78 выводит одноразовую точность на наш терминал.

В следующем разделе мы увидим, как и `train.py`, и `test_accuracy.py` используются вместе . для обучения и оценки двух CNN для точности предсказания возраста и пола соответственно.

14.7 Результаты прогнозирования возраста и пола

В предыдущих главах как для Practitioner Bundle, так и для ImageNet Bundle я представил серию экспериментов, чтобы продемонстрировать, как я получил модель с оптимальными характеристиками. Однако из-за длины этой главы я собираюсь представить наилучшие результаты только для возрастных CNN и гендерных CNN.

14.7.1 Возрастные результаты

Мой лучший эксперимент по прогнозированию возраста с использованием MxAgeGenderNet был получен с использованием оптимизатора Adam с начальной скоростью обучения $1e^{-4}$ (что меньше, чем значение по умолчанию $1e^{-3}$). Также применялось затухание веса L2 , равное 0,0005 . Использование небольшой начальной скорости обучения позволило мне обучать сеть дольше без переобучения. Я начал обучение, используя следующую команду:

```
$ python train.py --checkpoints checkpoints/age --prefix agenet
```

Разрыв между обучением и проверкой остается небольшим примерно до эпохи 90, когда начинается расхождение. Я позволил обучению продолжаться до эпохи 120 (рис. 14.3, вверху слева), где остановил обучение, снизил скорость обучения до $1e^{-5}$ и перезапустил обучение:

```
$ python train.py --checkpoints checkpoints/age --prefix agenet \
--начало эпохи 120
```

Я позволил обучению продолжаться в течение 20 эпох (рис. 14.3, вверху справа). На этом этапе мы можем начать видеть, как потери при проверке и точность начинают немного выравниваться. Я не хотел тренироваться слишком долго, так как меня беспокоила переподгонка к тренировочным данным. Я решил снизить скорость обучения до $1e^{-6}$, а затем возобновить обучение еще на 10 эпох:

```
$ python train.py --checkpoints checkpoints/age --prefix agenet \
--начало эпохи 140
```

После 150-й эпохи я остановил процесс обучения и изучил свой проверочный набор, отметив, что точность проверки составляет 71,65 % при ранге 1 и 94,14 % при однократной проверке (рис. 14.3, внизу). Довольный этими результатами, я затем оценил на тестовом наборе:

```
$ python test_accuracy.py --checkpoints контрольные точки/возраст --prefix agenet \
--epoch 150
[INFO] загрузка модели...
[INFO] предсказание по данным теста "возраст"...
[ИНФО] ранг-1: 71,15%
[ИНФО] разовый: 88,28%
```

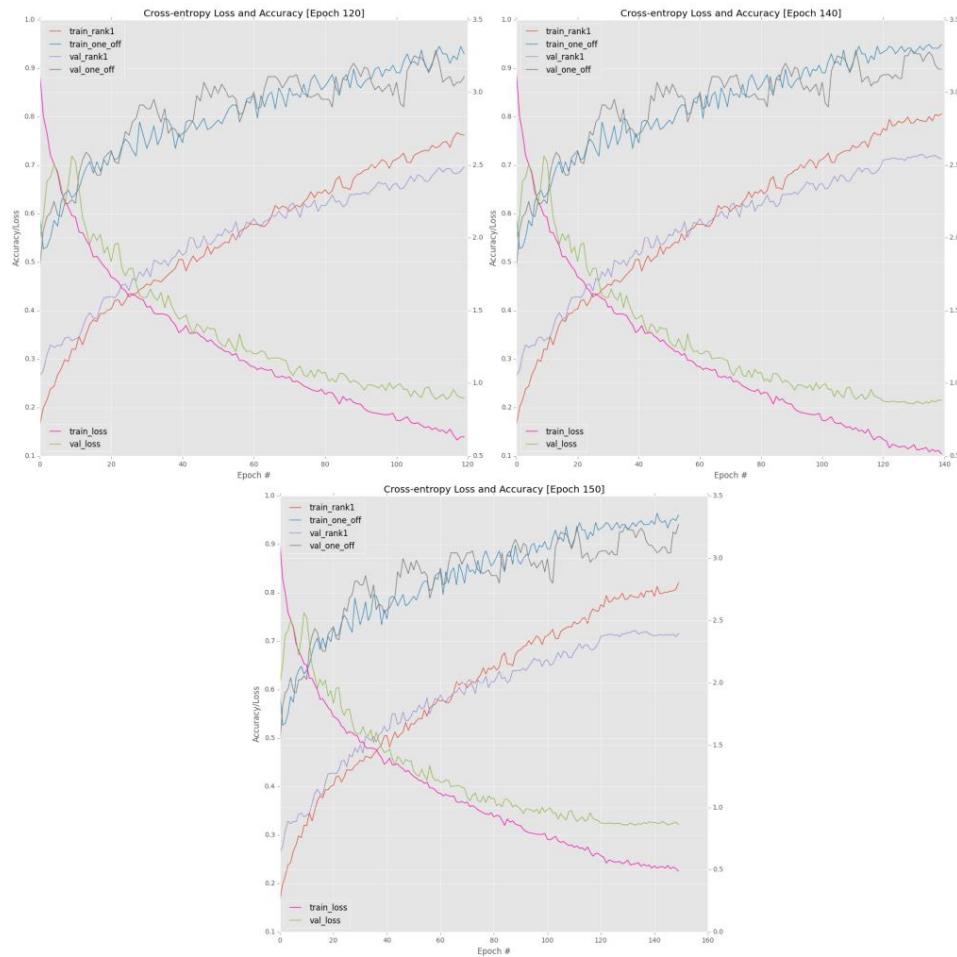


Рисунок 14.3: Верхний левый: использование начальной скорости обучения $\alpha = 1e - 4$ позволяет нам медленно, но неуклонно обучаться до эпохи 120. Верхний правый: в этот момент показатели обучения и проверки начинают отклоняться друг от друга, поэтому мы корректируем α до $1e - 5$ и тренироваться еще 10 эпох. Внизу: использование $\alpha = 1e - 6$ приводит к стагнации обучения.

Здесь вы можете видеть, что этот подход дал 71,15% ранг-1 и 88,28% одноразовую точность на тестовом наборе. По сравнению с оригинальным методом Леви и др., этот результат является существенным улучшением. Их лучший метод (с использованием передискретизации с 10 кропами) достиг только 50,7% точности ранга 1 и 84,70% однократной точности. Одни только результаты ранга 1 в этом эксперименте показывают улучшение на 20,45%!

14.7.2 Гендерные результаты

После применения train.py к набору данных о возрасте я вернулся к age_gender_config и установил для DATASET_TYPE значение пола , чтобы указать, что я хочу обучить архитектуру MxAgeGenderNet на наборе данных о поле. Мои лучшие результаты были получены при обучении с использованием оптимизатора SGD с базовой скоростью обучения $1e - 2$. Я также применил термин импульса 0,9 и уменьшение веса L2 0,0005. Я начал обучение, используя следующую команду:

```
$ python train.py --checkpoints checkpoints/gender --prefix gendernet
```

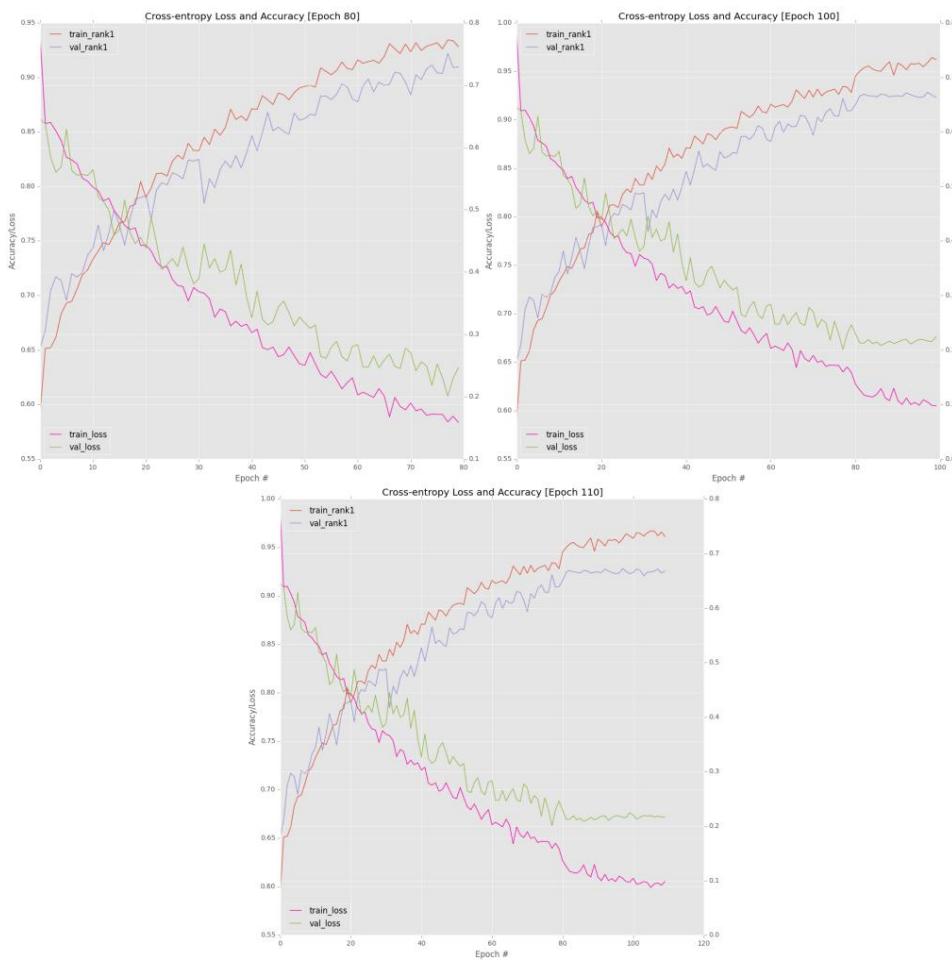


Рисунок 14.4: Верхний левый: первые 80 эпох обучения нашей сети на наборе гендерных данных. Вверху справа: снижение α с $1e-2$ до $1e-3$ на 20 эпох. Внизу: последние 10 эпох при $\alpha = 1e-4$.

Как показано на изображении ниже, обучение проходит медленно и стабильно (рис. 14.4, вверху слева). В эпоху 80 я прекратил обучение, снизил скорость обучения до $1e-3$ и продолжил обучение еще 20 эпох:

```
$ python train.py --checkpoints checkpoints/gender --prefix gendernet \ --start-epoch 80
```

Выходной график эпохи 100 показан на рис. 14.4 (вверху справа). Больше всего меня беспокоило переоснащение. Потери при проверке полностью выровнялись, а потери при обучении продолжали уменьшаться. Я не хотел тренироваться дольше, так как переобучение казалось правдоподобным, поэтому я снизил скорость обучения с $1e-3$ до $1e-4$ и позволил сети обучаться еще 10 эпох:

```
$ python train.py --checkpoints checkpoints/gender --prefix gendernet \ --start-epoch 100
```

Окончательный график всех 120 эпох можно увидеть на рис. 14.4 (внизу). Глядя на график потерь, мы видим стагнацию в проверке, в то время как обучение продолжает снижаться. Однако, когда

исследуя график точности, мы отмечаем, что разрыв между обучением и проверкой относительно стабилен для большей части процесса обучения. Точность проверки после эпохи 120 составила 92,57%.

Затем я использовал эпоху 120 для оценки набора тестов:

```
$ python test_accuracy.py --checkpoints контрольные точки/пол \
--prefix gendernet --epoch 110 [INFO]
загрузка модели...
[INFO] предсказание на основе тестовых данных "поля"...
[ИНФО] Ранг-1: 90,29%
```

Как видите, на тестовом наборе я достиг точности 90,29%. Глядя на результаты Леви и др., обратите внимание, что их подход (с использованием метода передискретизации с 10 кадрами) дал точность 86,8% — мой метод на целых 3,5% выше.

Теперь, когда сверточные нейронные сети нашего возраста и пола обучены и оценены, давайте перейдем к тому, как мы можем подготовить и предварительно обработать изображения для классификации с использованием этих сетей.

14.8 Визуализация результатов

Теперь, когда мы обучили две отдельные CNN — одну для предсказания возраста, а другую для определения пола, — мы можем перейти к теме развертывания. На этапе развертывания нам нужно будет иметь возможность загружать обе наши предварительно обученные CNN, предварительно обрабатывать наши входные изображения, а затем передавать их через две сети для получения наших выходных классификаций.

Вместо использования существующей конфигурации `age_gender_config` давайте откроем новую файл, назовите его `age_gender_deploy.py` и вставьте следующие конфигурации:

```
1 # импортируем необходимые пакеты
2 из age_gender_config import OUTPUT_BASE 3 из os пути
импорта
4
5 # определяем путь к предсказателю лицевых ориентиров dlib 6
DLIB_LANDMARK_PATH = "shape_predictor_68_face_landmarks.dat"
```

Чтобы получить более высокую точность при прогнозировании возраста и пола, часто бывает полезно обрезать и выровнять лицо на данном изображении. До сих пор все наши входные изображения были предварительно обрезаны и выровнены для нас; однако это предположение не будет выполняться в реальном мире. Чтобы еще больше повысить точность, нам нужно применить выравнивание лица, тему, которую мы обсудим в Разделе 14.8.2 ниже. Переменные `DLIB_LANDMARK_PATH` предоставляют путь к предварительно обученному предсказателю лицевых ориентиров, который позволит нам выравнивать лица на входных изображениях.

Далее определим возраст конфигураций CNN:

```
8 # определяем путь к возрастной сети + вспомогательные файлы 9
AGE_NETWORK_PATH = "checkpoints/age"
10 AGE_PREFIX = "агент"
11 AGE_EPOCH = 150 12
AGE_LABEL_ENCODER = path.sep.join([OUTPUT_BASE, "age_le.cpickle"])
13 AGE_MEANS = path.sep.join([OUTPUT_BASE, "age_adience_mean.json"])
```

Здесь мы начинаем с определения `AGE_NETWORK_PATH`, который является путем к контрольным точкам возрастного веса. `AGE_PREFIX` управляет именем сети, в данном случае agenet. мы будем загружать

AGE_EPOCH номер 150 с диска. Чтобы преобразовать необработанные целочисленные метки в удобочитаемые метки, нам понадобится путь к нашему AGE_LABEL_ENCODER. Наконец, для выполнения нормализации среднего нам нужен путь AGE_MEANS.

Мы определяем идентичные переменные для гендерной сети ниже:

```

15 # определяем путь к гендерной сети + вспомогательные файлы 16
GENDER_NETWORK_PATH = "checkpoints/gender"
17 GENDER_PREFIX = "гендернет"
18 GENDER_EPOCH = 110 19
GENDER_LABEL_ENCODER = path.sep.join([OUTPUT_BASE,
20      "gender_le.cpickle"])
21 GENDER_MEANS = path.sep.join([OUTPUT_BASE,
22      "gender_adience_mean.json"])

```

В следующем разделе мы узнаем, как применять наши две сети для классификации и визуализации результатов прогнозов возраста и пола на основе изображений в наборе данных Adience. Но что произойдет, если вы захотите применить эти типы прогнозов к изображениям за пределами Adience? Для этого нам сначала нужно понять процесс выравнивания лица. Оттуда мы будем использовать эти методы (и наши предварительно обученные сети), чтобы применить предсказания возраста и пола к нашим собственным изображениям.

14.8.1 Визуализация результатов Inside Adience

Давайте продолжим и рассмотрим vis_classification.py, скрипт, отвечающий за визуализацию прогнозов в наборе данных Adience. Откройте файл vis_classification.py, и мы приступим к работе:

```

1 # импортировать OpenCV перед mxnet, чтобы избежать ошибки
сегментации 2 import cv2
3
4 # импортировать необходимые пакеты
5 из config import age_gender_config as config 6 from config
import age_gender_deploy as deploy 7 from
pyimagesearch.preprocessing import ImageToArrayPreprocessor 8 from
pyimagesearch.preprocessing import SimplePreprocessor 9 from
pyimagesearch.preprocessing import MeanPreprocessor 10 from pyimagesearch.utils
import AgeGenderHelper 11 import numpy as np 12 import mxnet as mx 13 import
argparse 14 import pickle 15 import imutils 16 import json 17 import os

```

Мы начинаем с импорта наших необходимых пакетов Python в строках 2-17 — обратите внимание, как я снова размещаю импорт cv2 над mxnet из-за проблемы с ошибкой сегмента, о которой я упоминал выше. На вашем компьютере это действие может не потребоваться. Оттуда мы импортируем как нашу стандартную конфигурацию, так и разворачиваем файлы конфигурации.

Строки 7-9 импортируют наши препроцессоры изображений, чтобы мы могли должным образом предварительно обработать наши изображения перед их классификацией. AgeGenderHelper будет использоваться для визуализации распределения вероятностей для каждой метки класса.

Перейдем к аргументам командной строки:

```

19 # построить аргумент parse и разобрать аргументы
20 ap = argparse.ArgumentParser()
21 ap.add_argument("-s", "--sample-size", type=int, default=10,
22                  help="эпоха # для загрузки")
23 аргумента = переменные (ap.parse_args())

```

Здесь нам нужен только один переключатель, --sample-size, который представляет собой целое число, представляющее количество изображений, которые мы хотим взять из тестового набора Adience. Теперь мы можем загрузить нашу метку кодировщики и средние файлы для возрастных и гендерных наборов данных:

```

25 # загрузить кодировщики меток и средние файлы
26 print("[INFO] загрузка кодировщиков этикеток и средних файлов...")
27 ageLE = pickle.loads(open(deploy.AGE_LABEL_ENCODER, "rb").read())
28 полеLE = pickle.loads(open(deploy.GENDER_LABEL_ENCODER, "rb").read())
29 ageMeans = json.loads(open(deploy.AGE_MEANS).read())
30 ПолСредства = json.loads(open(deploy.GENDER_MEANS).read())

```

А также загрузить сериализованные сети с диска:

```

32 # загрузить модели с диска
33 print("[INFO] загрузка моделей...")
34 agePath = os.path.sep.join([deploy.AGE_NETWORK_PATH,
35                             развернуть.AGE_PREFIX])
36 гендерный путь = os.path.sep.join([deploy.GENDER_NETWORK_PATH,
37                             развернуть.GENDER_PREFIX])
38 ageModel = mx.model.FeedForward.load(agePath, deploy.AGE_EPOCH)
39 genderModel = mx.model.FeedForward.load(genderPath,
40                                         развернуть.GENDER_EPOCH)

```

После загрузки моделей нам нужно их скомпилировать:

```

42 # теперь, когда сети загружены, нам нужно их скомпилировать
43 print("[INFO] компиляция моделей...")
44 ageModel = mx.model.FeedForward(ctx=[mx.gpu(0)],
45                                 символ = ageModel.symbol, arg_params = ageModel.arg_params,
46                                 aux_params=ageModel.aux_params)
47 гендерная модель = mx.model.FeedForward(ctx=[mx.gpu(0)],
48                                         символ=genderModel.symbol, arg_params=genderModel.arg_params,
49                                         aux_params=genderModel.aux_params)

```

Прежде чем передать изображение через данную сеть, оно должно быть предварительно обработано. Давайте инициализируйте эти препроцессоры сейчас:

```

51 # инициализировать препроцессоры изображений
52 sp = SimplePreprocessor (ширина = 227, высота = 227, интер = cv2.INTER_CUBIC)
53 ageMP = MeanPreprocessor(ageMeans["R"], ageMeans["G"],
54                           ageMeans["B"])
55 полеMP = MeanPreprocessor(genderMeans["R"], genderMeans["G"],
56                           полОзначает ["B"])
57 iap = ImageToArrayPreprocessor()

```

Обратите внимание, как мы создаем два объекта MeanPreprocessor : один для возраста и один для Пол. Здесь мы используем два средних процессора, потому что возраст и пол имеют два отдельных значения. обучающие наборы и, следовательно, имеют разные средние значения RGB.

В следующем блоке кода представлены образцы строк --sample-size из тестового .lst-файла:

```
59 # загрузить образец тестовых изображений
60 строк = открыть (config.TEST_MX_LIST).read().strip().split("\n")
61 строка = np.random.choice (строки, размер = аргументы ["sample_size"])
```

Давайте пройдемся по каждой из этих строк по отдельности:

```
63 # петля по рядам
64 для строки в строке :
65     # распаковать строку
66     (_ , gtLabel, imagePath) = row.strip().split("\t")
67     изображение = cv2.imread (путь к изображению)
68
69     # предварительно обработаем изображение, одно для возрастной модели, а другое для
70     # гендерная модель
71     ageImage = iap.preprocess(ageMP.preprocess(
72         sp.preprocess(изображение)))
73     гендерное изображение = iap.preprocess(genderMP.preprocess(
74         sp.preprocess(изображение)))
75     ageImage = np.expand_dims (ageImage, ось = 0)
76     гендерное изображение = np.expand_dims (гендерное изображение, ось = 0)
```

Для каждой строки мы распаковываем ее в метку истинности (gtLabel) и соответствующий путь к изображению. (строка 66). Входное изображение загружается с диска в строке 67. Выполняется предварительная обработка изображения. в строках 71-76, создавая два выходных изображения.

Первое выходное изображение — это ageImage, результат обработки изображения препроцессорами возраста. Точно так же у нас есть гендерное изображение, результат прохождения через поле препроцессоры. Теперь, когда наши изображения были предварительно обработаны, мы можем пропустить их через соответствующие сети для классификации:

```
78     # пропустить ROI через соответствующие модели
79     agePreds = ageModel.predict(ageImage)[0]
80     ПолПредс = ПолМодель.предсказ(полИзображение)[0]
81
82     # сортируем прогнозы по их вероятности
83     ageIdxs = np.argsort(agePreds)[::-1]
84     genderIdxs = np.argsort(genderPreds)[::-1]
```

Вызов методов .predict в строках 79 и 80 дает нам наши вероятности для каждого класса . метка. Затем мы сортируем эти метки в порядке убывания с наибольшей вероятностью в начале списка. списки (строки 83 и 84).

Чтобы визуализировать распределение вероятностей каждой метки класса, мы будем использовать наши visualizeAge и методы visualizeGender:

```
86     # визуализировать предсказания возраста и пола
87     ageCanvas = AgeGenderHelper.visualizeAge(agePreds, ageLE)
```



Рисунок 14.5: Различные примеры классификаций из набора данных Adience использовали наш возраст и гендерные сети. В каждом из случаев мы смогли правильно предсказать как возраст, так и пол предмета.

```
88     полеCanvas = AgeGenderHelper.visualizeGender(genderPreds,
89             ПОЛLE)
90     изображение = imutils.resize(изображение, ширина = 400)
```

Для краткости объяснение этих функций не было включено в эту главу. Эти две функции являются просто расширениями визуализаций эмоций/выражений лица в главе 11. Для тех, кто заинтересован, я предоставил объяснение этих двух функций на сопутствующем веб-сайте. к этой книге. В противном случае я оставлю читателю в качестве упражнения исследовать эти функции. поскольку они совершенно не связаны с глубоким обучением и просто функциями OpenCV, используемыми для красивого рисования форматированное распределение вероятностей.

Наш последний кодовый блок рисует верхний прогноз возраста и пола на выходном изображении . с последующим отображением результатов на нашем экране:

```
92     # рисуем фактическое предсказание на изображении
93     gtLabel = ageLE.inverse_transform (int (gtLabel))
94     text = "Факт: {}".format(*gtLabel.split("_"))
95     cv2.putText(изображение, текст, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
96                 0.7, (0, 0, 255), 3)
97
98     # показать выходное изображение
99     cv2.imshow ("Изображение", изображение)
100    cv2.imshow("Возрастные вероятности", ageCanvas)
101    cv2.imshow (" Вероятности пола", гендерный холст)
102    cv2.waitKey(0)
```

Чтобы увидеть наш скрипт vis_classification.py в действии, откройте терминал и выполните команду следующая команда:

```
$ Pythonvis_classification.py
```

Пример выходных результатов можно увидеть на рис. 14.5. Обратите внимание, как мы умеем правильно предсказать возраст и пол человека на фотографии.

14.8.2 Понимание выравнивания лица

Чтобы понять выравнивание лица, нам сначала нужно рассмотреть основы ориентиров лица. Ориентиры лица используются для локализации и представления основных областей лица, таких как:

- Глаза
- Брови • Нос
- Рот
- линия подбородка

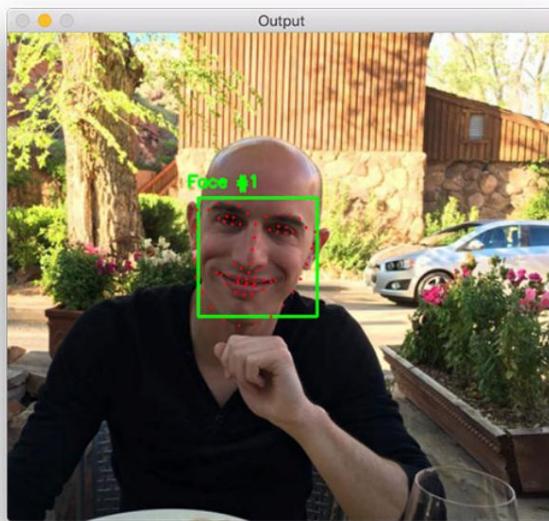


Рисунок 14.6: Пример обнаружения ориентиров лица на лице. Обратите внимание, как локализованы глаза, брови, нос, рот и линия подбородка.

Пример ориентиров лица, обнаруженных на изображении, можно увидеть на рис. 14.6. Обнаружение ориентиров лица является подмножеством проблемы прогнозирования формы. Учитывая входное изображение (обычно ROI, определяющее интересующий объект), предсказатель формы пытается локализовать ключевые точки интереса вдоль формы. В контексте лицевых ориентиров наша цель состоит в том, чтобы обнаружить важные лицевые структуры на лице, используя модели предсказания формы. Таким образом, обнаружение ориентиров лица состоит из двух шагов. процесс:

- Шаг №1: Локализуйте лицо на изображении. •

Шаг № 2: Определите ключевые структуры лица в области интереса лица.

Для выполнения обоих этих шагов мы будем использовать библиотеку dlib [40]. Для шага № 1 dlib использует предварительно обученный детектор HOG + Linear SVM [41, 42] — этот детектор может определять координаты ограничивающей рамки (x, y) лица на изображении. Затем, для шага № 2, используя метод, предложенный Каземи и Салливаном в их статье 2014 года «Выравнивание лица за одну миллисекунду с ансамблем регрессии».

деревья, ключевые лицевые структуры могут быть локализованы. Конечным результатом стал детектор лицевых ориентиров, который можно использовать для обнаружения лицевых ориентиров в режиме реального времени с получением высококачественных результатов.

Имея лицевые ориентиры, мы можем затем применить выравнивание лица, процесс: 1.

Идентификация геометрической структуры лица в цифровых изображениях.

2. Попытка получить каноническое выравнивание лица на основе перевода, масштаба и вращение.

Существует множество форм выравнивания лица. Некоторые методы пытаются наложить (заранее определенную) 3D-модель, а затем применить преобразование к входным изображениям таким образом, чтобы ориентиры лица на входном лице совпадали с ориентирами на 3D-модели. Другие, более простые методы, полагаются на сами ориентиры лица (в частности, области глаз) для получения нормализованного вращения, перемещения и представления лица в масштабе.

Таким образом, выравнивание лица можно рассматривать как еще одну форму нормализации данных. Точно так же, как мы имеем в виду нормализацию изображений перед их передачей через CNN, мы можем выровнять лица для получения большей точности при (1) обучении наших CNN и (2) их оценке. Полный обзор процесса выравнивания лица выходит за рамки этой книги, поскольку он в значительной степени сосредоточен на компьютерном зрении и методах обработки изображений, выходящих за рамки глубокого обучения; тем не менее, я призываю читателей, интересующихся как ориентирами лица, так и выравниванием лица, прочитать мои руководства по этим методам:

- Ориентиры лица: <http://pyimg.co/xkgwd> . • Выравнивание

лиц: <http://pyimg.co/tnbzf> Чтобы применить выравнивание

лица к нашим входным изображениям, мы будем использовать класс FaceAligner , реализованный в моей библиотеке imutils (<http://pyimg.co/7c29j>). Я кратко опишу алгоритм выравнивания лица ниже.

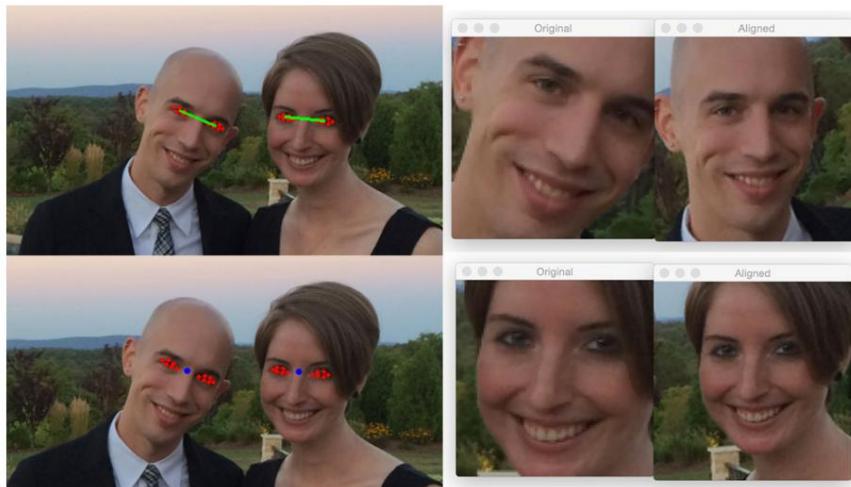


Рисунок 14.7: Вверху: локализация глаз с помощью ориентиров лица (красный цвет) сопровождается вычислением угла между глазами (зеленый цвет). Внизу: чтобы выровнять лицо, нам также нужно вычислить центр (x, y) посередине между глазами. Справа: результаты применения выравнивания лица к двум лицам. Обратите внимание, что оба лица масштабируются примерно одинаково и повернуты так, что глаза лежат вдоль горизонтальной линии.

Во-первых, глаза локализуются с помощью лицевых ориентиров (рис. 14.7, вверху). Зная (x, y)-координаты глаз, мы можем вычислить угол между ними. Оттуда нам также нужна середина между глазами (внизу). Затем применяется аффинное преобразование для преобразования изображений в новое выходное координатное пространство, так что лицо: 1. Центрируется на изображении.

2. Повернуто так, что глаза лежат на горизонтальной линии (т. е. лицо повернуто так, что глаза лежат вдоль одних и тех же координат у).

3. Масштабируется таким образом, чтобы размеры всех лиц в заданном наборе данных были примерно одинаковыми.

Результаты применения выравнивания лица к двум лицам можно увидеть на рис. 14.7 (справа).

Обратите внимание, что в каждом примере лицо центрировано, масштабировано примерно одинаково и повернуто так, что глаза лежат вдоль горизонтальной линии. Опять же, полный обзор лицевых ориентиров выходит за рамки этой книги, поэтому, пожалуйста, обратитесь к ссылкам блога PyImageSearch выше для получения дополнительной информации об обоих этих методах. В оставшейся части этой главы мы будем рассматривать выравнивание лица как алгоритм черного ящика. Те, кто хочет узнать больше об этих методах, должны обратиться к ссылкам.

14.8.3 Применение предсказания возраста и пола к вашим собственным изображениям

Возможность применять предсказание возраста и пола к набору данных Adience — это хорошо, но если мы захотим применить наши CNN к изображениям за пределами Adience? Что мы делаем тогда? И какие шаги необходимо предпринять? Чтобы узнать это, откройте `test_prediction.py` и начните с импорта следующих библиотек:

```
1 # импортируем OpenCV перед mxnet, чтобы избежать ошибки
сегментации 2 import cv2
3
4 # импортируем необходимые пакеты 5
из config import age_gender_deploy as deploy 6 из
pyimagesearch.preprocessing import ImageToArrayPreprocessor 7 из
pyimagesearch.preprocessing import SimplePreprocessor 8 из pyimagesearch.preprocessing
import MeanPreprocessor 9 из pyimagesearch.preprocessing import CropPreprocessor
10 из pyimagesearch.utils import AgeGenderHelper 11 from imutils.face_utils import
FaceAligner 12 из imutils import face_utils 13 из imutils import paths 14 import numpy as
np 15 import mxnet as mx 16 import argparse 17 import pickle 18 import imutils 19
import json 20 import dlib 21 import os
```

Как видите, нам нужно импортировать довольно много пакетов Python. Для начала нам понадобится наша конфигурация развертывания, чтобы мы могли загружать как возрастные, так и гендерные сети. Мы будем использовать несколько препроцессоров изображений. Обратите внимание на строку 9, где мы импортируем препроцессор `CropPreprocessor` — чтобы повысить точность классификации, мы будем выполнять метод 10 культур из главы 10 пакета Practitioner Bundle.

Затем мы импортируем класс `FaceAligner` в строку 11. Метод выравнивания лиц кратко обсуждался в разделе 14.8.2 выше (см. <http://pyimg.co/tnbfz>), для получения дополнительной информации), но на данный момент просто рассматривайте этот класс как инструмент выравнивания лица черного ящика.

Затем у нас есть аргументы командной строки:

```
23 # построить разбор аргумента и разобрать аргументы 24 ap =
argparse.ArgumentParser() 25 ap.add_argument("-i", "-image", required=True,
```

```
26     help="путь к входному изображению (или каталогу)")
27 аргументов = вары (ap.parse_args())
```

Здесь нужен только один аргумент, --image, который может быть:

1. Путь к одному изображению.
2. Путь к каталогу изображений.

В случае, если --image является каталогом, мы будем перебирать каждое из изображений по отдельности и применить возрастную и гендерную классификацию к каждому из них. Наши следующие несколько блоков кода будут похожи на наш предыдущий скрипт vis_classification.py . Мы начнем с загрузки наших кодировщиков меток и подразумеваем файлы:

```
29 # загрузить кодировщики меток и средние файлы
30 print("[INFO] загрузка кодировщиков этикеток и средних файлов...")
31 ageLE = pickle.loads(open(deploy.AGE_LABEL_ENCODER, "rb").read())
32 полеLE = pickle.loads(open(deploy.GENDER_LABEL_ENCODER, "rb").read())
33 ageMeans = json.loads(open(deploy.AGE_MEANS).read())
34 ПолСредства = json.loads(open(deploy.GENDER_MEANS).read())
```

Затем следует загрузить сами сериализованные сети:

```
36 # загрузить модели с диска
37 print("[INFO] загрузка моделей...")
38 agePath = os.path.sep.join([deploy.AGE_NETWORK_PATH,
39     развернуть.AGE_PREFIX])
40 гендерный путь = os.path.sep.join([deploy.GENDER_NETWORK_PATH,
41     развернуть.GENDER_PREFIX])
42 ageModel = mx.model.FeedForward.load(agePath, deploy.AGE_EPOCH)
43 genderModel = mx.model.FeedForward.load(genderPath,
44     развернуть.GENDER_EPOCH)
```

После загрузки моделей их также необходимо скомпилировать:

```
46 # теперь, когда сети загружены, нам нужно их скомпилировать
47 print("[INFO] компиляция моделей...")
48 ageModel = mx.model.FeedForward(ctx=[mx.gpu(0)],
49     символ = ageModel.symbol, arg_params = ageModel.arg_params,
50     aux_params=ageModel.aux_params)
51 гендерная модель = mx.model.FeedForward(ctx=[mx.gpu(0)],
52     символ=genderModel.symbol, arg_params=genderModel.arg_params,
53     aux_params=genderModel.aux_params)
```

Конечно, нам нужно предварительно обработать наши изображения, прежде чем передавать их по сети:

```
55 # инициализировать препроцессоры изображений
56 sp = SimplePreprocessor(ширина = 256, высота = 256,
57     интер=cv2.INTER_CUBIC)
58 cp = CropPreprocessor(ширина = 227, высота = 227, горизонт = True)
59 ageMP = MeanPreprocessor(ageMeans["R"], ageMeans["G"],
60     ageMeans["B"])
61 полеMP = MeanPreprocessor(genderMeans["R"], genderMeans["G"],
62     полОзначает ["B"])
63 iap = ImageToArrayPreprocessor(dataFormat="channels_first")
```

Обратите внимание, как мы инициализировали наш SimplePreprocessor для изменения размера изображения до 256×256 пикселей (строки 56 и 57). После изменения размера изображения мы применим метод 10-кратного кадрирования, извлекая 227 × 227 областей из входного изображения с помощью CropPreprocessor (строка 58). Особая забота берется в строке 63 для создания экземпляра препроцессора ImageToArrayProcessor с «сначала каналы» формат данных. Мы используем порядок «каналы в первую очередь» из-за того, что mxnet представляет изображения с каналы перед пространственными измерениями.

Следующий блок кода обрабатывает инициализацию детектора лиц dlib (на основе HOG [41, 42]), загрузку предиктора ориентиров лица с диска и создание экземпляра нашего FaceAligner:

```
65 # инициализируем детектор лиц dlib (на основе HOG), затем создаем
66 # Предсказатель лицевых ориентиров и выравниватель лица
67 детектор = dlib.get_frontal_face_detector()
68 предиктор = dlib.shape_predictor(deploy.DLIB_LANDMARK_PATH)
69 fa = FaceAligner (предиктор)
```

Затем мы должны определить, загружаем ли мы один образ с диска или нам следует перечислить содержимое каталога и получить пути ко всем входным изображениям:

```
71 # инициализировать список путей к изображениям как одно изображение
72 imagePaths = [args["image"]]
73
74 # если входной путь на самом деле является каталогом, то перечислить все изображения
75 # пути в каталоге
76 , если os.path.isdir(args["image"]):
77     imagePaths = отсортировано (список (пути.list_files (аргументы ["изображение"])))
```

Учитывая наши imagePaths, давайте переберем их по отдельности:

```
79 # цикл по путям изображений
80 для imagePath в imagePaths:
81     # загрузить изображение с диска, изменить его размер и преобразовать в
82     # оттенки серого
83     print("[INFO] обрабатывает {}".format(imagePath))
84     изображение = cv2.imread (путь к изображению)
85     изображение = imutils.resize (изображение, ширина = 800)
86     серый = cv2.cvtColor (изображение, cv2.COLOR_BGR2GRAY)
87
88     # обнаруживать лица на изображении в градациях серого
89     прямоугольники = детектор (серый, 1)
```

Мы начинаем с загрузки нашего изображения с диска (строка 84), изменяя его размер до фиксированной ширины 800. пикселей (строка 85) и преобразование их в оттенки серого (строка 86). Затем в строке 89 используется детектор лиц dlib для определять расположение лиц на изображении.

Давайте перейдем к каждому из обнаруженных лиц:

```
91     # цикл по обнаружению лиц
92     для прямоугольника в прямоугольниках:
93         # определяем лицевые ориентиры для области лица, затем
94         # выровнять лицо
95         shape = предсказатель (серый, прямоугольник)
```

```

96         face = fa.align(изображение, серый, прямоугольник)
97
98         # изменить размер лица до фиксированного размера, затем извлечь 10-кратное кадрирование
99         # патчи от него
100        лицо = sp.preprocess(лицо)
101        патчи = cp.preprocess(лицо)
102
103        # выделяем память для патчей возраста и пола
104        agePatches = np.zeros((patches.shape[0], 3, 227, 227),
105                               dtype="плавающий")
106        гендерные патчи = np.zeros((patches.shape[0], 3, 227, 227),
107                               dtype="плавающий")

```

Для каждого положения лица ограничительной рамки, rect, мы применяем предсказатель ориентира лица, извлеките лицо и выровняйте его (строки 95 и 96). Затем мы предварительно обрабатываем лицо, применяя SimplePreprocessor, чтобы изменить его размер до 256×256 пикселей, а затем CropPreprocessor, чтобы извлеките 10 участков лица.

В строках 104–107 инициализируются пустые массивы NumPy для хранения каждого из десяти урожаев как для возраста, так и для возраста, прогнозы и гендерные прогнозы соответственно. Причина, по которой мы явно определяем эти два массива, потому что каждый ROI в патчах должен быть предварительно обработан как возрастными препроцессорами, так и гендерными препроцессорами. Теперь мы можем применить эти препроцессоры:

```

109        # ЦИКЛ ПО ПАТЧАМ
110        для j в np.arange(0, patches.shape[0]):
111            # выполнить вычитание среднего на патче
112            agePatch = ageMP.preprocess(patches[j])
113            гендерный патч = гендерный MP.preprocess(патчи [j])
114            agePatch = iap.preprocess(agePatch)
115            гендерный патч = iap.preprocess(гендерный патч)
116
117            # обновить соответствующие списки патчей
118            agePatches[j] = agePatch
119            ПолПатчес[j] = ПолПатч

```

В строке 110 мы начинаем перебирать каждый из десяти патчей. Для каждого из них мы применяем средний возраст препроцессор вычитания и препроцессор вычитания среднего пола (строки 112 и 113). И agePatch, и genderPatch затем преобразуются в массивы, совместимые с mxnet, через канал. заказ в строках 114 и 115. После того, как оба прошли предварительную обработку, они могут быть добавлены в массивы agePatches и genderPatches соответственно (строки 118 и 119).

Теперь мы, наконец, готовы делать прогнозы для agePatches и genderPatches:

```

121        # делать прогнозы по возрасту и полу на основе извлеченных
122        # исправления
123        agePreds = ageModel.predict(agePatches)
124        ПолПредс = гендерМодель.предик(гендерПатчес)
125
126        # вычислить среднее значение для каждой метки класса на основе
127        # предсказания для патчей
128        agePreds = agePreds.mean(ось = 0)
129        ПолPreds = ПолPreds.mean(ось = 0)

```

Строки 123 и 124 пропускают ROI по возрасту и полу через соответствующие сети, что дает вероятности классов для каждого патча. Мы усредняем вероятности классов вместе для каждого патча, чтобы получить наши окончательные прогнозы в строках 128 и 129.

Наш последний блок кода обрабатывает отображение распределения меток класса для возраста и пола, как а также рисуя ограничивающую рамку вокруг лица, мы предсказываем возраст и пол для:

```

131      # визуализировать предсказания возраста и пола
132      ageCanvas = AgeGenderHelper.visualizeAge(agePreds, ageLE)
133      полCanvas = AgeGenderHelper.visualizeGender(genderPreds,
134          полLE)
135
136      # рисуем ограничивающую рамку вокруг лица
137      клон = изображение.копировать()
138      (x, y, w, h) = face_utils.rect_to_bb(прямоугольник)
139      cv2.rectangle(клон, (x, y), (x + w, y + h), (0, 255, 0), 2)
140
141      # показать выходное изображение
142      cv2.imshow("Ввод", клон)
143      cv2.imshow("Лицо", лицо)
144      cv2.imshow("Возрастные вероятности", ageCanvas)
145      cv2.imshow("Вероятности пола", гендерный холст)
146      cv2.waitKey(0)

```

Окончательные выходные изображения отображаются на нашем экране в строках 142-146.

Чтобы применить наш сценарий `test_prediction.py` к изображениям за пределами набора данных Adience, просто выполните следующую команду:

```
$ python test_prediction.py --примеры изображений
```

Набор примеров результатов можно найти на рис. 14.8. Обратите внимание, как мы смогли правильно классифицировать объект на фотографии в зависимости от его пола и возраста.

14.9 Резюме

В этой главе мы узнали, как предсказать возраст и пол человека на фотографии, обучив два отдельные сверточные нейронные сети. Чтобы выполнить этот процесс, мы обучили наши сети набор данных Adience, который включает ярлыки для двух полов и восемь отдельных возрастных групп. Нашей целью было повторить работу Levi и др. полученные на этом наборе данных, где они получили:

1. 50,57% точная и 84,70% однократная точность для возраста.
2. Точная точность определения возраста 86,8%.

Мы следовали их архитектуре CNN, вводя (1) слои пакетной нормализации и (2) дополнительный отсев. В процессе обучения мы применяли дополнительную аугментацию данных, чтобы помочь уменьшить переоснащение. В целом удалось получить:

1. 71,15% точная и 88,28% однократная точность для возраста.
2. Точная точность определения возраста 90,29%.

Оба этих результата являются огромным улучшением по сравнению со всеми заявленными исходными показателями точности. После того как мы оценили производительность нашей CNN, затем мы определили собственный скрипт Python для прогнозирования возраста и пол людей на фотографиях не входит в набор данных Adience. Эту задачу выполнил с использованием метода, называемого выравниванием лица, перед передачей области интереса через сеть для классификации.

Стоит отметить, что мы не использовали тот же инструмент выравнивания, что и Levi et al. Вместо этого мы использовали наш собственный (более простой) алгоритм выравнивания лица и добились хороших результатов. При обучении собственного

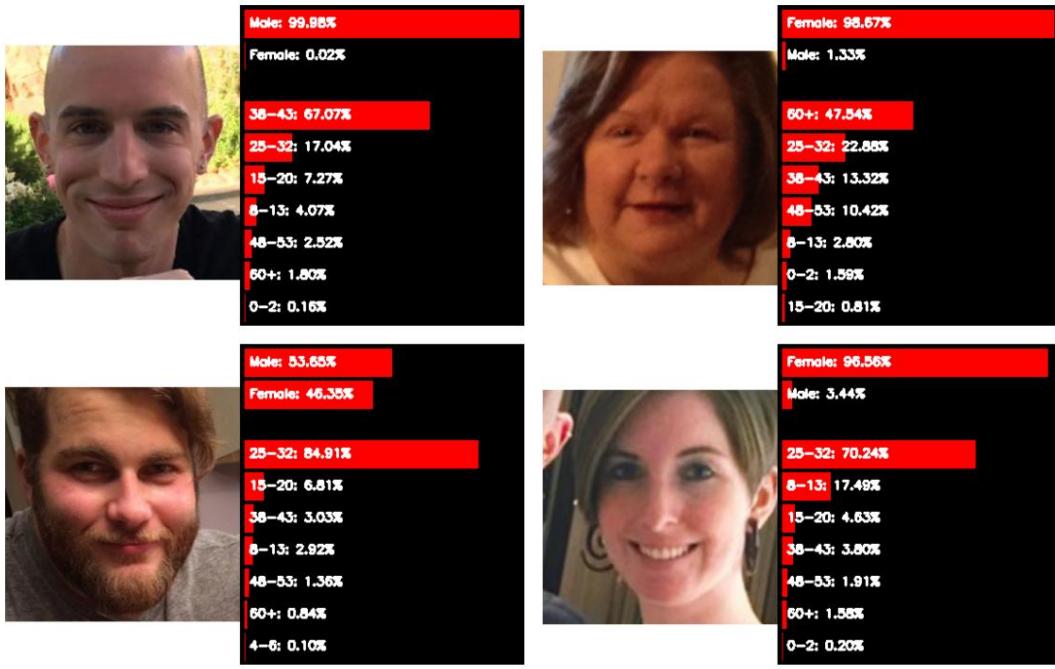


Рисунок 14.8: Пример предсказания возраста и пола для выборки входных изображений вне набора данных Adience .

настраиваемых алгоритмов прогнозирования возраста и пола, я бы предложил использовать один и тот же инструмент выравнивания лица для ваших изображений для обучения, проверки, тестирования и внешней оценки — это обеспечит идентичную предварительную обработку ваших изображений и то, что ваша модель будет лучше обобщаться.

15. Более быстрые R-CNN

Глубокое обучение повлияло почти на все аспекты компьютерного зрения, которые существенным образом зависят от машинного обучения . Классификация изображений, механизмы поиска изображений (также известные как поиск изображений на основе содержимого или CBIR), локализация и сопоставление (SLAM), сегментация изображений и многие другие претерпели изменения с момента последнего возрождения нейронных сетей и глубокого обучения. Обнаружение объекта ничем не отличается.

Одним из самых популярных алгоритмов обнаружения объектов на основе глубокого обучения является семейство алгоритмов R-CNN , первоначально представленное Girshick et al. в 2013 г. [43]. С тех пор алгоритм R-CNN претерпел ряд итераций, улучшая алгоритм с каждой новой публикацией и превосходя традиционные алгоритмы обнаружения объектов (например, каскады Хаара [44]; HOG + Linear SVM [41]) на каждом этапе . шаг пути.

В этой главе мы обсудим алгоритм Faster R-CNN и его компоненты, включая якоря, базовую сеть, сеть региональных предложений (RPN) и объединение областей интереса (ROI). Это обсуждение строительных блоков Faster R-CNN поможет вам понять основной алгоритм, как он работает и как возможно сквозное обнаружение объектов глубокого обучения.

В следующей главе мы рассмотрим API обнаружения объектов TensorFlow [45], в том числе способы его установки, принципы работы и способы использования API для обучения ваших собственных детекторов объектов Faster R-CNN на пользовательских наборах данных.

Эти две главы, посвященные более быстрым R-CNN, наряду со следующими двумя главами, посвященными однократным детекторам (SSD), будут посвящены обнаружению объектов с точки зрения беспилотных автомобилей, демонстрируя, как обучить детекторы объектов локализовать дорожные знаки и транспортные средства на изображениях. и видеопотоки. Вы сможете использовать эти обсуждения и примеры кода в качестве отправной точки для своих собственных проектов.

15.1 Обнаружение объектов и глубокое обучение

Обнаружение объектов, независимо от того, выполняется ли оно с помощью глубокого обучения или других методов компьютерного зрения, имеет три основные цели — учитывая входное изображение, которое мы хотим получить:

1. Список ограничивающих рамок или координат (x, y) для каждого объекта на изображении
2. Метка класса, связанная с каждой ограничивающей рамкой 3. Оценка вероятности/достоверности , связанная с каждой ограничивающей рамкой и меткой класса

В главе 13 пакета «Практик» мы рассмотрели традиционный конвейер обнаружения объектов, в том числе:

- Скользящие окна для локализации объектов в разных местах
- Пирамиды изображений, используемые для обнаружения объектов в различных масштабах
- Классификация с помощью предварительно обученной CNN

Таким образом, мы способны представить обнаружение объекта как классификацию, просто используя скользящие окна и пирамиды изображений. Проблемы с этим подходом многочисленны, но основные из них включают:

- Медленно и утомительно: запуск скользящего окна в каждом месте на каждом слое изображения.

пирамида - это трудоемкий процесс

- Отсутствие соотношения сторон: поскольку наша CNN требует входных данных фиксированного размера, мы не можем закодировать соотношение сторон потенциального объекта в области интереса, извлеченной и переданной в CNN. Это приводит к менее точной локализации.
- Склонен к ошибкам: балансировка скорости (большие шаги скользящего окна и меньшее количество слоев пирамиды изображений) с, как мы надеемся, более высокой точностью (больше шагов скользящего окна и больше слоев пирамиды) невероятно сложна. Эта проблема еще больше усугубляется отсутствием соотношения сторон объекта в ROI.

Что нам действительно нужно, так это сквозной детектор объектов на основе глубокого обучения, где мы вводим изображение в сеть и получаем ограничивающие рамки и метки классов для вывода. Как мы увидим, создание комплексного детектора объектов — непростая задача.

15.1.1 Измерение производительности детектора объектов

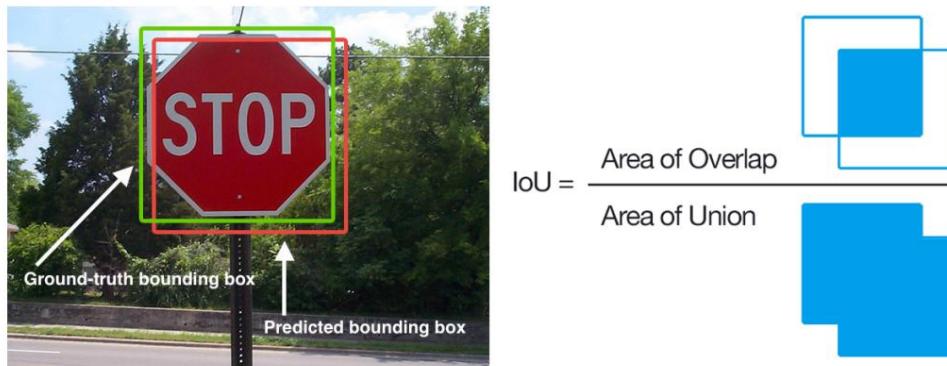


Рисунок 15.1: Слева: пример обнаружения знака остановки на изображении. Предсказанная ограничивающая рамка нарисована красным, а наземная ограничивающая рамка — зеленым. Наша цель — вычислить пересечение Union между этими ограничивающими рамками. Справа: вычислить пересечение Union так же просто, как разделить площадь перекрытия ограничивающих прямоугольников на площадь объединения.

При оценке производительности детектора объектов мы используем оценочную метрику, называемую Intersection over Union (IoU). Обычно вы обнаружите, что IoU используется для оценки производительности детекторов HOG + Linear SVM [41], методов сверточной нейронной сети, таких как Faster R-CNN [46], SSD [47], You Only Look Once (YOLO) [48], [49] и др.; однако имейте в виду, что фактический алгоритм, используемый для создания предсказанных ограничивающих рамок, не имеет значения.

Любой алгоритм, который предоставляет предсказанные ограничивающие рамки (и, возможно, метки классов) в качестве выходных данных, может быть оценен с использованием IoU. Более формально, чтобы применить IoU для оценки произвольного детектора объектов, нам нужны:

1. Наземные ограничивающие рамки (т. е. помеченные вручную ограничивающие рамки из нашего тестового набора, которые указывают, где на изображении находится наш объект).

2. Предсказанные ограничивающие рамки из нашей модели.
3. Если вы хотите вычислить полноту наряду с точностью, вам также понадобится класс истинности метки и предсказанные метки классов.

Пока у нас есть эти два набора ограничивающих рамок, мы можем применять IoU.

На рис. 15.1 (слева) я включил наглядный пример ограничительной рамки, основанной на реальных данных (зеленый), в сравнении с предсказанной ограничительной рамкой (красный). Таким образом, вычисление IoU можно определить с помощью уравнения, показанного на рис. 15.1 (справа).

Изучив это уравнение, вы увидите, что IoU — это просто отношение. В числителе мы вычисляем площадь перекрытия между предсказанной ограничивающей рамкой и ограничивающей рамкой истинности. Знаменатель — это площадь объединения, или, проще говоря, площадь, охватываемая как предсказанной ограничивающей рамкой, так и ограничивающей рамкой истинности. Разделив площадь перекрытия на площадь объединения, мы получим окончательную оценку — пересечение над объединением.

Откуда берутся правдивые примеры?

Прежде чем мы зайдем слишком далеко, вам может быть интересно, откуда берутся правдивые примеры. Ранее в этой главе я упоминал, что наш набор данных должен быть «размечен вручную», но что именно это означает?

При обучении собственного детектора объектов вам понадобится набор данных. Этот набор данных должен быть разбит (как минимум) на две группы: 1. Учебный набор, используемый для обучения вашего детектора объектов 2. Тестовый набор для оценки вашего детектора объектов. У вас также может быть проверочный набор, используемый для настройки гиперпараметров вашей модели.

Как обучающий, так и тестовый набор будут состоять из: 1. Сами фактические изображения 2.

Ограничивающие рамки, связанные с объектом (объектами) на изображении. Ограничительные рамки просто (x, y)-координаты объекта на изображении.

Ограничительные рамки для обучающих и тестовых наборов помечены вручную, поэтому мы называем их «основной правдой». Ваша цель — взять обучающие изображения + ограничивающие рамки, построить детектор объектов, а затем оценить его эффективность на тестовом наборе. Оценка $\text{IoU} > 0,5$ обычно считается «хорошим» прогнозом.

Почему мы используем Intersection вместо Union?

До сих пор в этой книге мы в основном выполняли классификацию, когда наша модель предсказывала набор меток классов для входного изображения — предсказанная метка с наибольшей вероятностью была либо правильной, либо неправильной. Этот тип бинарной классификации упрощает вычисление точности — она либо верна, либо нет; однако для обнаружения объектов все не так просто.

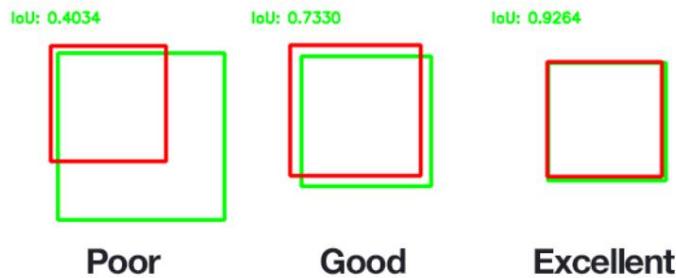


Рисунок 15.2: Пример вычисления Intersection over Unions для различных ограничивающих рамок. Чем больше прогнозируемая ограничительная рамка перекрывается с наземной ограничительной рамкой, тем лучше прогноз и, следовательно, выше показатель IoU.

В действительности крайне маловероятно, что (x, y) -координаты нашего предсказанного ограничивающего прямоугольника будут точно соответствовать (x, y) -координатам истинного ограничивающего прямоугольника. Из-за различных параметров нашей модели, таких как слой, используемый для извлечения признаков, размещение привязки, функция потерь и т. д., полное и полное совпадение между предсказанными и реальными ограничивающими рамками просто нереально.

Поскольку координаты не будут точно совпадать, нам нужно определить оценочную метрику, которая вознаграждает предсказанные ограничивающие рамки за сильное совпадение с земной правдой, как показано на рис . 15.2.

На этой иллюстрации я включил три примера хороших и нулевых оценок IoU. Предсказанные ограничивающие рамки, которые сильно перекрываются с ограничивающими рамками, основанными на реальных данных, имеют более высокие оценки, чем те, которые перекрываются меньше. Такое поведение делает IoU отличной метрикой для оценки пользовательских детекторов объектов.

Опять же, нас не интересует точное совпадение (x, y) -координат, но мы хотим убедиться , что наши предсказанные ограничивающие рамки совпадают как можно точнее — IoU может учитывать этот факт .

Реализация IoU вручную выходит за рамки этой книги, хотя это довольно простой процесс. Если вам интересно узнать больше о IoU, включая пошаговое руководство по коду Python, демонстрирующее, как его реализовать, см. эту запись в блоге PyImageSearch: <http://pyimg.co/ag4o5>

Средняя средняя точность (mAP)

В контексте машинного обучения точность обычно относится к точности, но в контексте обнаружения объектов IoU — это наша точность. Однако нам нужно определить метод для вычисления точности для каждого класса и для всех классов в наборе данных. Для достижения этой цели нам нужна средняя средняя точность (mAP).

Чтобы вычислить среднюю точность для одного класса, мы определяем IoU всех точек данных для определенного класса. Получив IoU, мы делим его на общее количество меток класса для этого конкретного класса, получая среднюю точность. Чтобы вычислить среднюю среднюю точность, мы вычисляем средний IoU для всех N классов, а затем берем среднее значение этих N средних значений, отсюда и термин «средняя средняя точность» .

При чтении документов по обнаружению объектов вы обычно видите результаты, сообщаемые с точки зрения mAP. Это значение mAP получается путем усреднения средней точности для каждого класса для всех классов в наборе данных. Обычно мы сообщаем mAP@0,5, указывая, что для того, чтобы объект в тестовом наборе был помечен как «положительное обнаружение» , он должен иметь не менее 0,5 IoU с достоверностью. Значение 0,5 настраивается, но 0,5 является довольно стандартным для большинства наборов данных обнаружения объектов.

15.2 (более быстрая) архитектура R-CNN

В этом разделе мы рассмотрим архитектуру Faster R-CNN. Мы начнем с краткого обсуждения R-CNN и то, как они развивались в серии из трех публикаций, что привело к архитектуре Faster R-CNN, которую мы обычно используем сейчас. Наконец, мы рассмотрим основные строительные блоки архитектуры Faster R-CNN, сначала на высоком уровне, а затем снова более подробно.

Как мы увидим, архитектура Faster R-CNN является сложной, со многими движущимися частями — мы сосредоточим наши усилия на получении понимания принципов этой архитектуры до обучения сети на наших собственных наборах данных в главе 16.

15.2.1 Краткая история R-CNN Чтобы лучше

понять, как работает алгоритм обнаружения объектов Faster R-CNN, нам сначала нужно рассмотреть историю алгоритма R-CNN, включая его первоначальное воплощение и то, как он развивался.

R-CNN

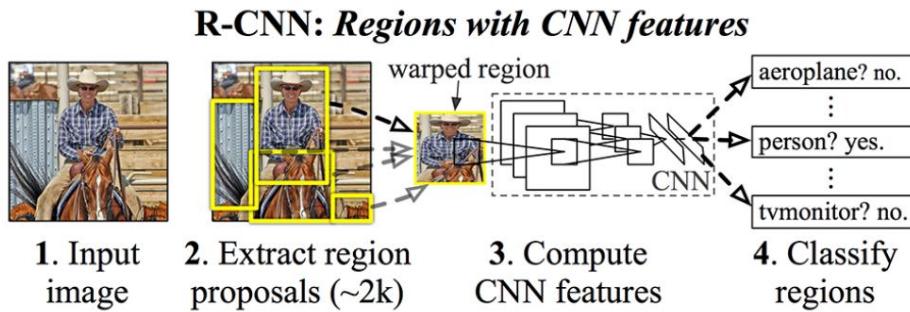


Рисунок 15.3: Первоначальная архитектура R-CNN состояла из (1) приема входного изображения, (2) извлечения ~2000 предложений с помощью алгоритма выборочного поиска, (3) выделения признаков для каждого ROI предложения с использованием предварительно обученной CNN (например, один, обученный на ImageNet), и (4) классификация функций для каждой области интереса с использованием линейного SVM для конкретного класса. (Изображение предоставлено: рисунок 1 Girshick et al. [43])

Первая статья R-CNN «Иерархии с расширенными функциями для точного обнаружения объектов и семантической сегментации» была опубликована в 2013 году Girshick et al. [43] — мы называем эту исходную статью и связанную с ней реализацию просто R-CNN. Обзор исходного алгоритма R-CNN можно найти на рис. 15.3, который включает в себя четырехэтапный процесс:

- Шаг № 1: Введите изображение
- Шаг № 2: Извлеките предложения областей (т. е. области изображения, которые потенциально могут содержать объекты).)

используя такой алгоритм, как выборочный поиск [50].

- Шаг № 3. Используйте трансферное обучение, в частности извлечение признаков, для вычисления признаков для каждое предложение (которое фактически является ROI) с использованием предварительно обученной CNN.
- Шаг № 4. Классифицируйте каждое предложение, используя извлеченные признаки с помощью машины опорных векторов . (CBM).

На первом этапе мы вводим изображение в наш алгоритм. Затем мы запускаем алгоритм предложения региона как выборочный поиск (или его аналог). Алгоритм выборочного поиска заменяет скользящие окна и пирамиды изображений, интеллектуально исследуя входное изображение в различных масштабах и местоположениях, тем самым значительно сокращая общее количество ROI предложений, которые будут отправлены в сеть для классификации. Таким образом, мы можем думать о выборочном поиске как об интеллектуальном скользящем окне и алгоритме пирамиды изображений.

R Обзор алгоритма выборочного поиска выходит за рамки этой книги, поэтому я рекомендую относиться к нему как к «черному ящику», который интеллектуально предлагает вам места ROI. Подробное обсуждение выборочного поиска см. в Uijlings et al. [50].

После того, как у нас есть наши предложения, мы вырезаем каждое из них по отдельности из входного изображения и применяем трансферное обучение с помощью извлечения признаков (глава 3 пакета для практиков). Вместо того, чтобы получать окончательные прогнозы от CNN, мы используем извлечение признаков, чтобы позволить нижестоящему классификатору изучить больше отличительных шаблонов из этих признаков CNN.

Четвертый и последний шаг — обучить серию SVM поверх этих извлеченных функций для каждого класса.

Глядя на этот конвейер для оригинальной R-CNN, мы можем ясно увидеть вдохновение и параллели с традиционными детекторами объектов, такими как исходная платформа HOG Dalal и Triggs + Linear SVM:

- Вместо того, чтобы применять исчерпывающую пирамиду изображений и скользящее окно, мы используем более интеллектуальный алгоритм выборочного поиска.
- Вместо извлечения признаков HOG из каждой области интереса мы теперь извлекаем признаки CNN . окончательная классификация входного ROI, только мы тренируемся этот SVM на функциях CNN, а не на функциях HOG

Основная причина, по которой этот подход работал так хорошо, связана с надежными отличительными функциями, изученными CNN — то, что мы подробно изучили в главах 3 и 5, посвященных трансферному обучению в пакете Practitioner Bundle.

Проблема оригинального подхода R-CNN в том, что он по-прежнему невероятно медленный. Более того, мы на самом деле не учимся локализовать через глубокую нейронную сеть.

Вместо этого мы оставляем локализацию алгоритму выборочного поиска — мы классифицируем ROI только после того, как он будет определен как «интересный» и «заслуживающий изучения» алгоритмом предложения региона, что поднимает вопрос: возможно ли получить сквозное обнаружение объектов на основе глубокого обучения?

Быстрый R-CNN

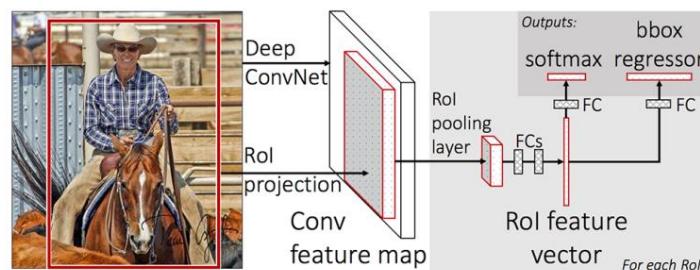


Рисунок 15.4: В архитектуре Fast R-CNN мы по-прежнему используем выборочный поиск для получения наших предложений, но теперь мы применяем ROI Pooling , извлекая окно фиксированного размера из карты объектов и используя эти функции для получения окончательной метки класса и ограничения коробка. (Изображение предоставлено: Рисунок 1 Girshick et al. [51])

Примерно через полтора года после того, как Girshick et al. представил оригинальную публикацию R-CNN в arXiv, Гиршик опубликовал вторую статью, Fast R-CNN [51]. Подобно исходному R-CNN, алгоритм Fast R-CNN по-прежнему использовал выборочный поиск для получения предложений по регионам, но был внесен новый вклад : объединение областей интереса (ROI). Визуализацию новой обновленной архитектуры можно увидеть на рис. 15.4 . В этом новом подходе мы применяем CNN ко всему входному изображению и извлекаем из него карту признаков , используя нашу сеть. Пул ROI работает, извлекая окно фиксированного размера из карты объектов и затем передавая его в набор полностью связанных слоев, чтобы получить выходную метку для ROI.

Мы обсудим ROI Pooling более подробно в Разделе 15.2.5, а пока усвойте, что ROI Pooling работает с картой объектов, извлеченной из CNN, и извлекает из нее окно фиксированного размера .

Основным преимуществом здесь является то, что сеть теперь эффективно обучаема от начала до конца: 1. Мы вводим изображение и связанные с ним наземные ограничивающие рамки 2. Извлекаем карту объектов 3. Применяем объединение ROI и получаем вектор признаков ROI 4. И, наконец, используйте два набора полно связанных слоев, чтобы получить (1) предсказания меток класса и

(2) расположение ограничительной рамки для каждого предложения.

Несмотря на то, что сеть теперь можно обучать от начала до конца, производительность при выводе резко упала.

(т. е. предсказания) за счет зависимости от алгоритма выборочного поиска (или эквивалентного) предложения области. Чтобы сделать архитектуру R-CNN еще быстрее, нам нужно включить предложение региона непосредственно в R-CNN.

Быстрее R-CNN

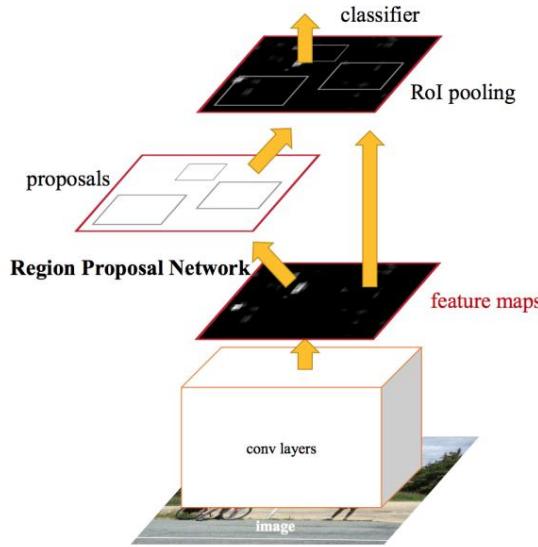


Рисунок 15.5: Последнее воплощение семейства R-CNN, Faster R-CNN, представляет сеть предложений регионов (RPN), которая выпекает предложения регионов непосредственно в архитектуре, устраняя потребность в алгоритме выборочного поиска. (Изображение предоставлено: рисунок 2 Girshick et al. [46])

Спустя чуть более месяца после публикации статьи Fast R-CNN Гиршик сотрудничал с Реном и Саном в статье 2015 года Faster R-NN: на пути к обнаружению объектов в реальном времени с сетями региональных предложений [46].

В этой работе Girshick et al. создал дополнительный компонент к архитектуре R-CNN, сеть региональных предложений (RPN). Как звучит название этого модуля, целью RPN является устранение требования запуска выборочного поиска перед выводом и вместо этого встраивание предложения региона непосредственно в архитектуру R-CNN.

На рис. 15.5 представлена визуализация обновленной архитектуры. Здесь мы видим, что входное изображение представляется сети, а его функции извлекаются с помощью предварительно обученной CNN (т. е. базовой сети). Эти функции параллельно отправляются в два разных компонента архитектуры Faster R-CNN .

Первый компонент, RPN, используется для определения того, где на изображении может находиться потенциальный объект. На данный момент мы не знаем, что это за объект, только то, что потенциально объект находится в определенном месте на изображении.

Предлагаемые ROI ограничивающей рамки основаны на модуле объединения областей интереса (ROI) сети вместе с извлеченными функциями из предыдущего шага. Пул ROI используется для извлечения окон функций фиксированного размера, которые затем передаются в два полностью связанных слоя (один для меток классов и один для координат ограничивающей рамки) для получения наших окончательных локализаций.

Мы подробно обсудим RPN в Разделе 15.2.4, но, по сути, сейчас мы собираемся размещать точки привязки с одинаковыми интервалами по всему изображению в различных масштабах и соотношениях сторон. Затем RPN проверит эти привязки и выдаст набор предложений относительно того, где, по его мнению, существует объект.

Здесь важно отметить, что RPN на самом деле не обозначает ROI; вместо этого он вычисляет свою «оценку объективности» и спрашивает: «Похожа ли эта область на какой-то объект?» Мне лично нравится думать о RPN и оценке объективности как о своего рода бинарном классификаторе, где RPN помечает каждую область интереса как «фон» или «передний план». Если RPN считает, что ROI находится «на переднем плане», то ROI заслуживает дальнейшего рассмотрения с помощью полносвязных слоев ROI Pooling и окончательной метки + ограничивающей рамки.

На этом этапе вся архитектура является сквозной обучаемой, и внутри сети происходит полный конвейер обнаружения объектов, включая: 1. Предложение региона 2. Извлечение признаков

3. Вычисление координат ограничивающей рамки объектов.
4. Предоставление меток классов для каждой ограничивающей рамки . шаг к тому, чтобы сделать обнаружение объектов в реальном времени с помощью глубокого обучения реальностью.

Теперь, когда у нас есть краткое введение в компоненты (более быстрой) архитектуры R-CNN, давайте более подробно рассмотрим каждый из них по отдельности.

15.2.2 Базовая сеть

На рисунке 15.5 мы можем видеть общие модули в архитектуре Faster R-CNN. После того, как мы введем образ в архитектуру, первый компонент, с которым мы столкнемся, — это базовая сеть. Базовая сеть обычно представляет собой CNN, предварительно обученную для конкретной задачи классификации. Эта CNN будет использоваться для трансферного обучения, в частности, для извлечения признаков.

В контексте обнаружения объектов мы обычно используем CNN, предварительно обученную на наборе данных ImageNet. Здесь мы используем предварительно обученную CNN, поскольку функции, изученные определенным слоем, часто можно перенести на задачи классификации, не относящиеся к тому, на чем была обучена исходная сеть. Полный обзор трансферного обучения, извлечения признаков, тонкой настройки и того, как мы можем использовать трансферное обучение через предварительно обученные сети, см. в главе 3 пакета для практиков.

В исходной статье Faster R-CNN в качестве базовых сетей использовались VGG [17] и ZF [52]. Сегодня мы обычно заменяем более глубокую и точную базовую сеть, такую как ResNet [21, 22], или меньшую, более компактную сеть для устройств, содержащих ресурсы, таких как MobileNet [53].

Одним из важных аспектов сетей обнаружения объектов является то, что они должны быть полностью свёрточными, не путать с полносвязными. Полностью сверточная нейронная сеть не содержит полносвязных слоев, которые обычно находятся в конце сети до того, как будут сделаны выходные прогнозы. В контексте классификации изображений удаление полностью связанных слоев обычно достигается путем применения объединения средних значений по всему объему до того, как один плотный классификатор softmax используется для вывода окончательных прогнозов.

Полностью сверточная нейронная сеть обладает двумя основными преимуществами, в том числе: 1.

Быстрота благодаря всем операциям свертки 2. Способность принимать изображения любого пространственного разрешения (т. е. ширины и высоты) при условии, что

образ и сеть могут влезть в память, конечно

При реализации обнаружения объектов с помощью глубокого обучения важно, чтобы мы не полагались на входные изображения фиксированного размера, которые требуют от нас приведения изображения к определенному размеру. Мало того , что фиксированные размеры могут искажать соотношение сторон входного изображения, но они имеют еще худший побочный эффект, делая чрезвычайно сложным для сети обнаружение мелких объектов — если мы слишком сильно уменьшим пространственные размеры нашего изображения, мелкие объекты теперь будут отображаться в виде крошечных пиксельных пятен, слишком маленьких для обнаружения сетью. Поэтому важно, чтобы мы позволяли сети принимать изображения с произвольным пространственным разрешением и позволяли размеру входного изображения быть решением, принимаемым разработчиком или инженером сети.

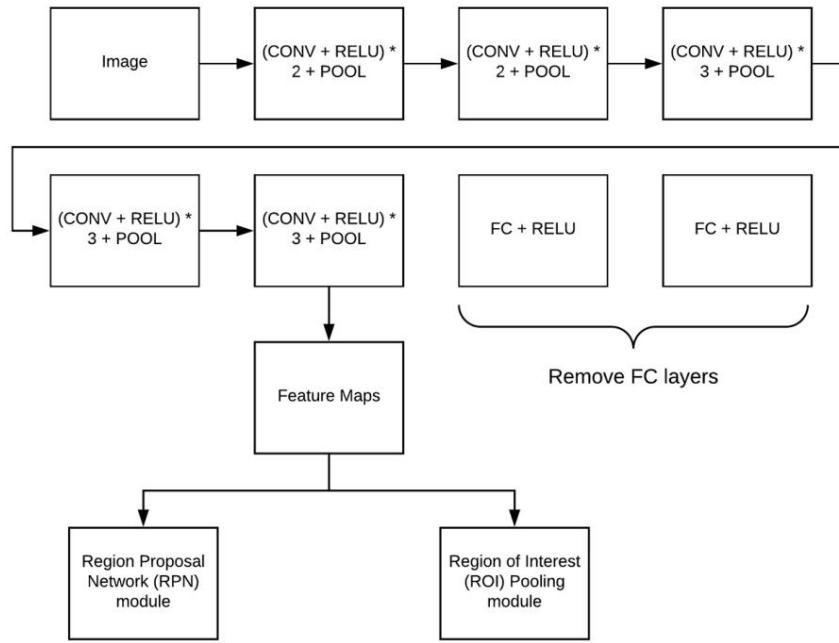


Рисунок 15.6: Базовая сеть используется для извлечения признаков из входного изображения. Мы гарантируем, что наша сеть является полностью сверточной, удаляя полно связные слои в конце сети и вместо этого используя выходные данные слоев CONV и POOL . Этот процесс позволяет нашей сети обрабатывать входные изображения произвольных пространственных размеров. Затем функции передаются в модули RPN и ROI Pooling.

Однако в случае VGG в Girshick et al. мы знаем, что в конце VGG есть полно связные слои — таким образом, это не полностью сверточная сеть. Однако есть способ изменить поведение CNN. Имейте в виду, что нам нужны только выходные данные определенного слоя CONV (или POOL) в сети — эти выходные данные являются нашей картой признаков, процесс которой визуализируется на рис. 15.6.

Мы можем получить эту карту объектов, распространив входное изображение по сети и остановившись на нашем целевом слое (нам не нужно передавать изображение через всю сеть, чтобы получить нашу карту объектов). Тот факт, что VGG изначально принимал только входное изображение 224x 224 , совершенно произведен, поскольку нас интересует только вывод определенного слоя CONV + POOL . Эта карта функций будет использоваться Региональной сетью предложений и модулем ROI Pooling позже в архитектуре Faster R-CNN.

15.2.3 Якоря

В традиционных конвейерах обнаружения объектов мы будем использовать либо (1) комбинацию скользящего окна + пирамида изображений, либо (2) алгоритм, подобный выборочному поиску, для генерации предложений для нашего классификатора. Поскольку наша цель — разработать сквозной детектор объектов с использованием глубокого обучения, который включает модуль предложения, нам необходимо определить метод, который будет генерировать ROI нашего предложения.

Основным разделением между классификацией и обнаружением объектов является предсказание ограничивающих рамок или (x, y)-координат, окружающих объект. Таким образом, мы можем ожидать, что наша сеть вернет кортеж, состоящий из координат ограничивающей рамки определенного объекта. Но с этим подходом есть проблема , а именно:

1. Как нам обращаться с сетью, предсказывающей значения за пределами изображения?
2. Как мы кодируем такие ограничения, как $x_{min} < x_{max}$ и $y_{min} < y_{max}$?

Звучит так, что решить эту проблему практически невозможно. Однако решение, предложенное

Girchick et al., называемые якорями, являются умными и новыми.

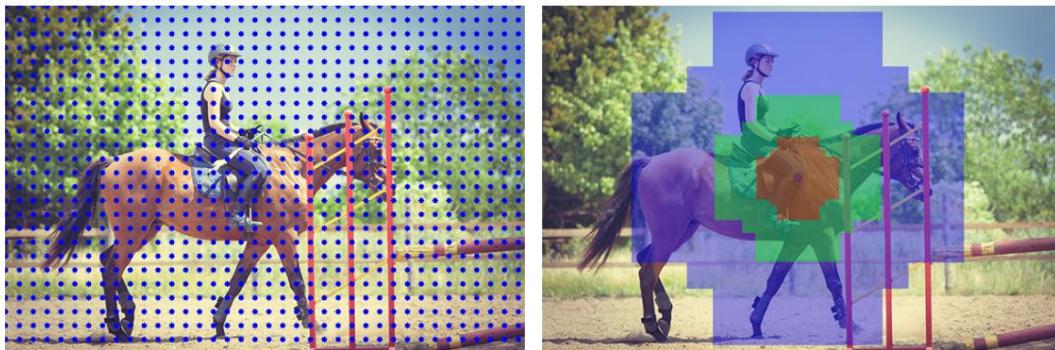


Рисунок 15.7: Слева: создание привязок начинается с процесса выборки координат изображения через каждые r пикселей ($r = 16$ в исходной реализации Faster R-CNN). Справа: мы создаем в общей сложности девять якорей, сосредоточенных вокруг каждой выбранной координаты (x, y) . В этой визуализации $x = 300$, $y = 200$ (центральная синяя координата). Всего девять якорей получены из каждой комбинации масштаба: 64×64 (красный), 128×128 (зеленый), 256×256 (синий); и соотношение сторон: $1 : 1$, $2 : 1$, $1 : 2$.

Вместо того, чтобы пытаться предсказать необработанные (x, y) -координаты ограничивающих рамок, мы можем вместо этого научиться предсказывать их смещения от эталонных рамок, а именно: x -центр, y -центр, ширина и высота. Эти значения дельты позволяют нам получить лучшее соответствие нашему эталонному прямоугольнику без необходимости прогнозировать фактические необработанные (x, y) -координаты, что позволяет нам обойти потенциально неразрешимую проблему кодирования «правил» ограничивающего прямоугольника в сети.

Итак, откуда берутся эти ограничивающие рамки ссылок? Нам нужно генерировать якоря самостоятельно, не используя алгоритм выборочного поиска. Чтобы выполнить этот процесс, нам сначала нужно равномерно выбрать точки на входном изображении (рис. 15.7, слева). Здесь мы видим входное изображение размером 600×400 пикселей — мы пометили каждую точку регулярно выбираемого целого числа (с интервалом в шестнадцать пикселей) синим кружком.

Следующим шагом является создание набора привязок в каждой из точек выборки. Как и в оригинальной публикации Faster R-CNN, мы создадим девять привязок (которые представляют собой фиксированные ограничивающие рамки) с различными размерами и соотношениями сторон, окружающими заданную точку выборки.

Цвета ограничивающих прямоугольников — это наши масштабы/размеры, а именно: 64×64 , 128×128 и 256×256 . Для каждого масштаба у нас также есть соотношение сторон: $1 : 1$, $1 : 2$ и $2 : 1$. Каждая комбинация масштаба и соотношения сторон дает всего девять якорей. Эта комбинация масштаба и соотношения сторон дает нам значительный охват всех возможных размеров и масштабов объектов на входном изображении (рис. 15.7, справа).

Однако здесь возникает проблема, когда мы разбиваем общее количество сгенерированных якорей:

- Если мы используем шаг 16 пикселей (по умолчанию для Faster R-CNN) на изображении 600×800 , мы получить в общей сложности 1989 позиций.
- С девятью якорями, окружающими каждую из 1989 позиций, мы теперь имеем в общей сложности $1989 \times 9 = 17\,901$ позиция ограничительной рамки для нашей CNN для оценки.

Если бы наша CNN классифицировала каждую из 17 901 ограничивающей рамки, наша сеть была бы лишь немногого быстрее, чем исчерпывающий цикл по каждой комбинации скользящего окна и пирамиды изображений.

К счастью, с помощью региональной сети предложений (RPN) мы можем значительно сократить количество окон предложений кандидатов, оставив нам гораздо более управляемый размер.

15.2.4 Региональная сеть предложений (RPN)

Если цель создания привязок состоит в том, чтобы получить хороший охват всех возможных масштабов и размеров объектов на изображении, цель Сети предложений по регионам (RPN) состоит в том, чтобы сократить количество сгенерированных ограничивающих рамок до более приемлемого размера.

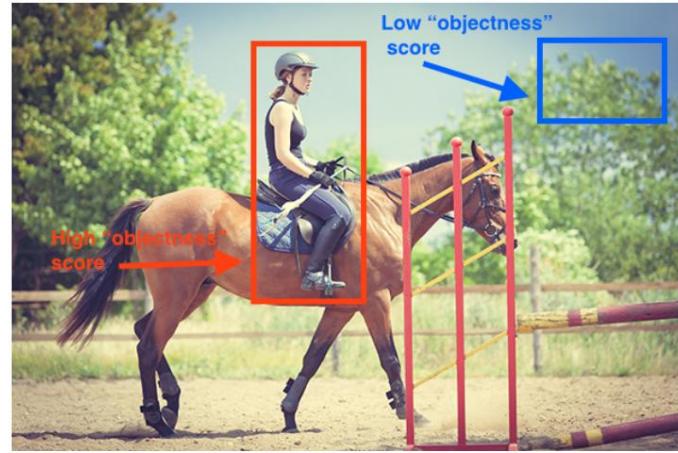


Рисунок 15.8: Цель модуля «Предложение региона» — принять набор привязок и быстро определить «объектность» региона на изображении. А именно, это включает в себя маркировку области интереса как переднего плана или фона. Фоновые области привязки отбрасываются, в то время как объекты переднего плана распространяются в модуль ROI Pooling.

Модуль RPN прост, но мощен и состоит из двух выходов. Верхняя часть модуля RPN принимает входные данные, которые являются нашей сверточной картой признаков из раздела 15.2.2. Затем мы применяем преобразование 3×3 , изучая 512 фильтров.

Эти фильтры подаются в два тракта параллельно. Первый вывод (слева) RPN — это оценка, которая указывает, считает ли RPN область интереса передним планом (заслуживает дальнейшего изучения) или фоновым (отбрасывание). На рис. 15.8 представлена визуализация маркировки «объектности» входной области интереса .

Опять же, RPN на самом деле не маркирует ROI — он просто пытается определить, является ли ROI фоновым или передним планом. Фактическая маркировка ROI будет иметь место позже в архитектуре (раздел 15.2.6). Размерность этого вывода равна $2 \times K$, где K — общее количество привязок, один вывод для вероятности переднего плана и второй вывод для вероятности фона.

Второй вывод (справа) — это наш регрессор ограничительной рамки, используемый для настройки привязок, чтобы лучше соответствовать объекту, который он окружает. Регулировка привязок снова выполняется с помощью свертки 1×1 , но на этот раз с выходом объема $4 \times K$. Результат равен $4 \times K$, так как мы предсказываем четыре значения дельты (т. е. смещения): x-центр, у-центр, ширина, высота.

При условии, что наша вероятность переднего плана достаточно велика, мы затем применяем:

- Немаксимальное подавление для подавления перекрытия
- Выбор предложения. Естественно, будет много перекрывающихся якорных местоположений в соответствии с разделом 15.2.3 выше — немаксимальное подавление помогает уменьшить количество местоположений, для перехода к модулю ROI Pooling. Мы дополнительно сокращаем количество местоположений, которые передаются в модуль пула ROI посредством выбора предложений. Здесь мы берем только лучшие N предложений и отбрасываем остальные.

В оригинальной публикации Faster R-CNN Girshick et al. установить $N = 2000$, но мы можем уйти с гораздо меньшим N , например $N = 10, 50, 100, 200$, и по-прежнему получать хорошие прогнозы.

Следующим шагом в конвейере будет распространение ROI и дельт в интересующую область. (ROI) Модуль объединения (раздел 15.2.5), но давайте сначала обсудим, как мы можем обучить RPN.

Обучение РПН

Во время тренировки мы берем наши якоря и кладем их в два разных ведра:

- Ведро № 1 — передний план: все привязки, которые имеют 0,5 IoU с наземным объектом.
Ограничительная рамка.
- Bucket #2 — Background: Все привязки, которые имеют < 0,1 IoU с наземным объектом.

На основе этих сегментов мы случайным образом выбираем между ними, чтобы сохранить равное соотношение между фоном и передним планом.

Оттуда нам нужно рассмотреть наши функции потерь. В частности, модуль RPN имеет две связанные с ним функции потерь. Первая функция потерь предназначена для классификации, которая измеряет точность предсказания RPN переднего плана по сравнению с фоном (здесь прекрасно работает бинарная кросс-энтропия).

Вторая функция потерь предназначена для нашей регрессии ограничивающей рамки. Эта функция потерь работает только с якорями переднего плана, поскольку якоря фона не имеют смысла ограничивающей рамки (и мы должны были уже обнаружить «фон» и отбросить его).

Для функции потерь регрессии ограничивающей рамки Girshick et al. использовал вариант или потерю L1, называемую плавной потерей L1. Поскольку для нас нереально на 100 % точно предсказать координаты наземной истины ограничивающего прямоугольника (рассматривается в разделе 15.1.1), слаженная потеря L1 позволяет ограничивающим рамкам, которые «достаточно близки» к соответствующим наземным рамкам, быть существенно исправить и тем самым уменьшить влияние потери.

15.2.5 Объединение областей интереса (ROI)

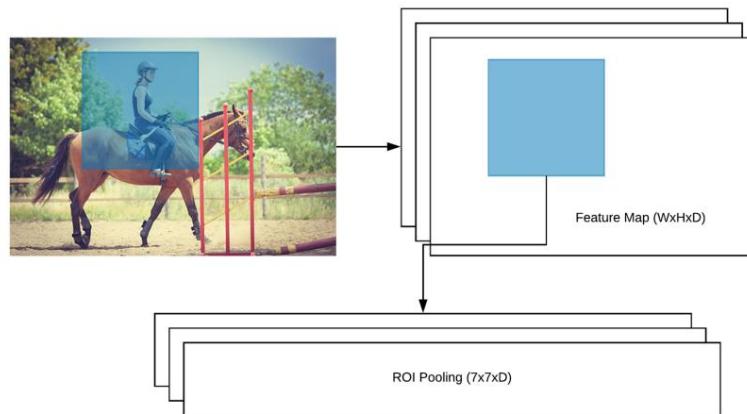


Рисунок 15.9: Цель модуля объединения областей интереса ROI состоит в том, чтобы взять все N местоположений предложений, а затем извлечь соответствующие функции ROI из сверточной карты признаков. Затем модуль ROI Pooling изменяет размеры извлеченных объектов для ROI до $7 \times 7 \times D$ (где D — глубина карты объектов). Этот фиксированный размер подготавливает функцию к двум будущим слоям FC в следующем модуле.

Цель модуля ROI Pooling — принять все N местоположений предложений из модуля RPN и вырезать векторы признаков из сверточной карты признаков в разделе 15.2.2 (рис. 15.9).

Обрезка векторов признаков выполняется следующим

- образом: • Использование срезов массива для извлечения соответствующего фрагмента из карты объектов • Изменение его размера до $14 \times 14 \times D$, где D — глубина карты объектов •

Применение операции максимального объединения с шагом 2x2, что дает вектор признаков $7 \times 7 \times D$.

Окончательный вектор признаков, полученный из модуля ROI Pooling, теперь можно передать в сверточную нейронную сеть на основе региона (описанную в следующем разделе), которую мы будем использовать для получения окончательных координат ограничивающей рамки для каждого объекта вместе с соответствующей меткой класса.

Для получения дополнительной информации о том, как реализовано объединение ROI, см. оригинал Girshick et al. публикация [46], а также превосходный учебник Faster R-CNN: Down the Rabbit Hole of Modern Object Detection [54].

15.2.6 Сверточная нейронная сеть на основе регионов

Последним этапом нашего конвейера является региональная сверточная нейронная сеть, или, как мы ее знаем, R-CNN. Этот модуль служит двум целям:

1. Получите окончательные прогнозы меток класса для каждого местоположения ограничивающей рамки на основе обрезанной карты объектов из модуля ROI Pooling. 2. Дальнейшее уточнение прогнозируемых координат (x, y) ограничивающей рамки для повышения точности прогноза. Компонент реализован через два полно связанных слоя, как показано на рис. 15.10. В крайнем левом углу у нас есть входные данные для нашей R-CNN, которые представляют собой векторы признаков, полученные из модуля ROI Pooling. Эти функции проходят через два полно связанных слоя (каждый 4096-d), прежде чем будут переданы в последние два слоя FC, которые дадут нашу метку класса (или фоновый класс) вместе со значениями дельты ограничивающей рамки.

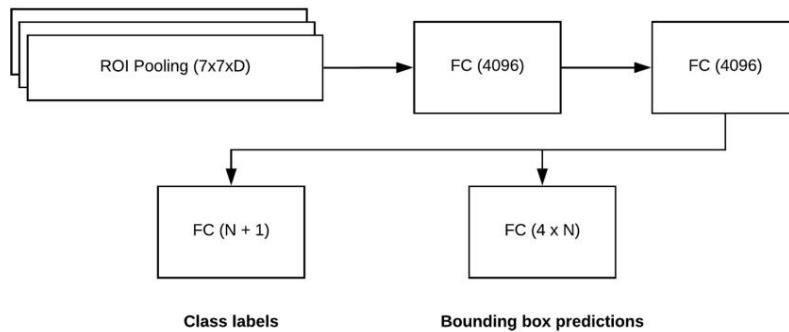


Рисунок 15.10: Последний модуль в архитектуре Faster R-CNN берет выходные данные компонента ROI Pooling и пропускает их через два уровня FC (каждый 4096-d), подобно последним уровням сетей классификации, обученных в ImageNet. Выходные данные этих уровней FC передаются на последние два уровня FC в сети. Один слой FC — это $N + 1 - d$, узел для каждой из меток класса плюс дополнительная метка для фона. Второй слой FC — это $4 \times N$, который представляет дельты для окончательных предсказанных ограничивающих рамок.

Один слой FC имеет $N + 1$ узлов, где N — общее количество меток классов. Добавление дополнительного измерения используется для обозначения фонового класса на тот случай, если наш модуль RPN пропускает фоновую область.

Второй слой FC имеет размер $4 \times N$. N здесь снова наше общее количество меток классов. Четыре значения являются нашими соответствующими дельтами для $x\text{-center}$, $y\text{-center}$, $width$, $height$, которые будут преобразованы в наши окончательные ограничивающие рамки.

Эти два результата снова подразумевают, что у нас будет две функции потерь: 1.

Категориальная кросс-энтропия для классификации 2. Гладкая потеря L1 для регрессии ограничивающей рамки Мы используем здесь категориальную кросс-

энтропию, а не бинарную кросс-энтропию, так как мы вычисляем вероятности для каждого из наших N классов по сравнению с двоичным случаем (фон против переднего плана) в модуле RPN. Последним шагом является применение подавления немаксимумов по классам к нашему набору ограничивающих значений.

прямоугольники — эти ограничивающие прямоугольники и связанные с ними метки классов считаются окончательными прогнозами нашей сети.

15.2.7 Полный конвейер обучения У нас есть

выбор при обучении всего конвейера Faster R-CNN. Первый выбор — обучить модуль RPN, получить удовлетворительные результаты, а затем перейти к обучению модуля R-CNN.

Второй вариант — объединить четыре функции потерь (две для модуля RPN, две для модуля R-CNN) с помощью взвешенной суммы, а затем совместно обучить все четыре. Какая из них лучше?

Почти во всех ситуациях вы обнаружите, что совместное сквозное обучение всей сети путем минимизации взвешенной суммы четырех функций потерь не только занимает меньше времени, но и обеспечивает более высокую точность.

В нашей следующей главе мы узнаем, как использовать API обнаружения объектов TensorFlow для обучения наших собственных сетей Faster R-CNN. Если вас интересуют более подробные сведения о конвейере Faster R-CNN, модулях RPN и ROI Pooling, а также дополнительные примечания о том, как мы совместно минимизируем четыре функции потерь, обязательно обратитесь к оригинальной публикации Faster R-CNN [46].], а также статью TryoLabs [54].

15.3 Резюме

В этой главе мы начали с обзора архитектуры Faster R-CNN вместе с ее более ранними вариантами, разработанными Girshick et al. [43, 46, 51]. Архитектура R-CNN претерпела несколько итераций и улучшений, но с последней архитектурой Faster R-CNN мы можем обучать сквозные детекторы объектов глубокого обучения.

Сама архитектура включает четыре основных компонента. Первый компонент — это базовая сеть (т. е. ResNet, VGGNet и т. д.), которая используется в качестве экстрактора признаков.

Затем у нас есть Сеть предложений регионов (RPN), которая принимает набор привязок и выводит предложения относительно того, где, по ее мнению, находятся объекты на изображении. Важно отметить, что RPN не знает, что представляет собой объект на изображении, а только то, что потенциальный объект существует в заданном месте.

Объединение регионов интереса используется для извлечения карт объектов из каждого региона предложения.

Наконец, сверточная нейронная сеть на основе региона используется для (1) получения окончательных прогнозов меток классов для предложения и (2) дальнейшего уточнения местоположений предложений для большей точности.

Учитывая большое количество движущихся частей в архитектуре R-CNN, нецелесообразно реализовывать всю архитектуру вручную. Вместо этого рекомендуется использовать существующие реализации, такие как TensorFlow Object Detection API [55] или Luminoth от TryoLabs [56].

В нашей следующей главе мы узнаем, как обучить Faster R-CNN с помощью объекта TensorFlow.

API обнаружения для обнаружения и распознавания распространенных уличных и дорожных знаков в США.

16. Обучение более быстрой R-CNN с нуля

В нашей предыдущей главе мы обсудили архитектуру Faster R-CNN и ее многочисленные компоненты, включая сеть региональных предложений (RPN) и модуль ROI Pooling. В то время как предыдущая глава была сосредоточена исключительно на внутренних рабочих частях, в этой главе мы переключимся на реализацию и на то, как на самом деле обучить Faster R-CNN на пользовательском наборе данных.

Мы собираемся использовать подход «глубокое обучение для беспилотных автомобилей» как в этой главе, и Глава 18 об однократных детекторах. К концу этой главы вы сможете: 1. Установить и настроить TensorFlow Object Detection API в своей системе. 2. Создать набор изображений + аннотации изображений в форме TensorFlow Object Detection API. 3. Обучить более быструю R-CNN. в наборе данных LISA Traffic Signs (или в вашем собственном наборе данных). 4. Оцените точность и примените обученный Faster R-CNN для ввода изображений и видео.

16.1 Набор данных о дорожных знаках LISA

Набор данных о дорожных знаках LISA был подготовлен и собран Mogelmose et al. и полностью подробно описаны в их статье 2012 года «Обнаружение и анализ дорожных знаков на основе зрения для интеллектуальных систем помощи водителю: перспективы и обзор» [57].

Набор данных состоит из 47 различных типов дорожных знаков США, включая знаки остановки, знаки пешеходного перехода и т. д. Пример изображения из набора данных LISA Traffic Signs можно увидеть на рис. 16.1. Набор данных изначально был захвачен с помощью видео, но также включены отдельные кадры и связанные с ними аннотации.

Всего 7855 аннотаций на 6610 кадрах. Дорожные знаки различаются по разрешению от 6x6 до 167x168 пикселей. Кроме того, некоторые изображения были сняты на камеру с более низким разрешением 640×480 , а другие — на камеру с более высоким разрешением 1024×522 пикселей. Некоторые изображения имеют оттенки серого, а другие — цветные. Различия в качестве камеры и цветовом пространстве захвата делают этот набор данных интересным для изучения с точки зрения обнаружения объектов.

Полный набор данных LISA Traffic Signs занимает более 7 ГБ, поэтому, чтобы быстрее обучить нашу сеть (и быстрее изучить основы обнаружения объектов), мы будем выбирать три класса дорожных знаков: знак остановки, пешеходный переход и сигнальные знаки впереди. Мы будем обучать наш Faster R-CNN этому



Рисунок 16.1: Наборы данных LISA Traffic Signs состоят из 47 различных дорожных знаков США с 7855 аннотациями в 6610 кадрах. Здесь мы можем увидеть пример изображения, содержащего два знака остановки. Наша цель будет заключаться в обнаружении и маркировке дорожных знаков, таких как этот знак остановки.

выборочный набор данных из трех классов, но вы также можете потренироваться на полном наборе данных (но я бы рекомендовал воспроизвести наши результаты в этой главе, прежде чем вносить какие-либо изменения).

Домашнюю страницу LISA Traffic Signs можно найти по этой ссылке: <http://pyimg.co/lp6xo>. Оттуда вам нужно прокрутить страницу вниз до раздела «Доступ» и загрузить ZIP - архив набора данных. Я решил хранить набор данных в каталоге с соответствующим названием lisa.

После завершения загрузки разархивируйте его:

```
$ mkdir lisa $  
cd lisa $ mv  
signDatabasePublicFramesOnly.zip ./ $ unzip  
signDatabasePublicFramesOnly.zip
```

16.2 Установка API обнаружения объектов TensorFlow

При составлении этой книги я оценил множество реализаций обнаружения объектов на основе глубокого обучения, включая чистые библиотеки на основе Keras, пакеты на основе mxnet, реализации Caffe и даже библиотеки Torch.

Обнаружение объектов не только намного сложнее для обучения сети, но и значительно сложнее для реализации, поскольку существует гораздо больше компонентов, некоторые из которых требуют пользовательских слоев и функций потерь. После прочтения главы ?? Что касается основ Faster R-CNN, должно быть ясно, что существует множество модулей, которые необходимо реализовать вручную.

Внедрение всей архитектуры Faster R-CNN не может быть рассмотрено в этой статье. книги по ряду причин, в том числе:

1. Внедрение и объяснение архитектуры Faster R-CNN вручную с использованием того же стиля , что и в остальной части книги (кодовые блоки и подробные объяснения), заняло бы сотни (если не больше) страниц.
2. Библиотеки и пакеты обнаружения объектов имеют тенденцию быть хрупкими по своей природе, поскольку используются пользовательские уровни и методы потери. Когда выпускается новая версия связанной с ними серверной библиотеки, высок риск поломки такого пользовательского модуля.

Цель здесь состоит в том, чтобы вместо этого обсудить библиотеку, которая является наименее хрупкой, насколько это возможно, предоставляя вам лучшее из обоих миров: знание архитектуры Faster R-CNN и возможность обучать сеть с нуля.

Из-за хрупкости пользовательских реализаций с нуля мы собираемся использовать API обнаружения объектов TensorFlow (TFOD API) в этой главе и в главе 18, посвященной детекторам одиночных выстрелов. Мало того, что TensorFlow является одной из наиболее часто используемых библиотек глубокого обучения, но и сам Google поддерживает API TFOD — над проектом работает большое сообщество как разработчиков с открытым исходным кодом, так и исследовательских и корпоративных разработчиков, помогая уменьшить хрупкий характер библиотеки.

Если вам интересно узнать больше об API TFOD, обязательно загляните на их официальную страницу GitHub [45], а также на их документ CVPR 2017 года «Соотношение скорости и точности для современных детекторов сверточных объектов» [55].

Чтобы установить и настроить TensorFlow Object Detection API, перейдите на следующую страницу сопутствующего веб-сайта для этой книги: <http://pyimg.co/7b7xm>. Если у вас нет доступа к сопутствующему веб-сайту, обратитесь к первым нескольким страницам этой книги, чтобы найти ссылку для регистрации на сопутствующем веб-сайте.

Использование сопутствующего веб-сайта для инструкций по установке TFOD API гарантирует, что их всегда можно будет поддерживать в актуальном состоянии, чего не может гарантировать печатная книга. Следуйте инструкциям, используя ссылку выше, чтобы настроить вашу машину, и оттуда мы сможем обучить ваш первый Faster R-CNN!

16.3 Обучение вашего более быстрого R-CNN

В первой части этого раздела будет подробно рассмотрена структура проекта. Важно, чтобы вы понимали, как правильно структурировать свой проект, иначе вы можете столкнуться с трудными для диагностики ошибками во время обучения или оценки.

API TensorFlow требует, чтобы наши изображения и аннотации были в определенном формате записи — мы обсудим, как взять набор данных изображения + связанные аннотации и поместить их в требуемый формат. Оттуда мы можем обучить нашу архитектуру Faster R-CNN, оценить ее производительность и применить ее к образцам тестовых изображений.

16.3.1 Структура каталога проекта

Между API TFOD и нашими пользовательскими скриптами, используемыми для создания нашего набора входных данных, обучения сети, оценки производительности и применения наших моделей к пользовательским изображениям, есть ряд движущихся частей. Кроме того, некоторые из этих скриптов живут исключительно внутри структуры каталогов TFOD API, в то время как другие скрипты должны быть реализованы нами вручную.

Давайте начнем с перечисления скриптов TFOD API Python, которые мы будем использовать. Мой TFOD API был клонирован в мой домашний каталог, поэтому мой корневой путь к TFOD API — /home/adrian/models/research:

```
$ cd ~/models/research/
pwd /home/adrian/models/
research $ ls -l adversarial_crypto
adversarial_text adv_imagenet_models
```

```
...
объект_обнаружение
...
```

Ваш корневой путь TFOD может отличаться в зависимости от того, где вы его клонировали.

Здесь довольно много подкаталогов, но нас больше всего интересует тот, объект_обнаружение:

```
$ ls -A1 объектное_обнаружение/*.py
object_detection/eval.py
object_detection/evaluator.py
object_detection/eval_util.py
object_detection/exporter.py
object_detection/exporter_test.py
object_detection/export_inference_graph.py
object_detection/__init__.py
object_detection/trainer.py
object_detection/trainer_test.py
object_detection/train.py
```

Мы будем использовать три из этих скриптов Python, в том числе:

1. train.py: используется для обучения наших детекторов объектов на основе глубокого обучения.
2. eval.py: выполняется одновременно с train.py для оценки набора тестов во время обучения.
3. export_inference_graph.py: после того, как наша модель полностью обучена, нам нужно заморозить веса и экспорттировать модель, чтобы мы могли импортировать ее в наши собственные приложения.

Опять же, эти три скрипта являются частью API TFOD и не являются частью какого-либо пользовательского кода или кода. реализацию мы будем писать. Структура каталогов и структура проекта для нашего Faster

Пользовательские сценарии R-CNN и конфигурации обучения/оценки можно увидеть ниже:

```
| --- ssds_and_rcnn
| |--- build_lisa_records.py
| |--- конфигурация
| | |--- __init__.py
| | |--- lisa_config.py
| |--- Лиза/
| | |--- allAnnotations.csv
| | |--- категории.txt
...
| |--- вид8/
| |--- вид9/
| |--- видеоисточники.txt
| --- прогнозировать.py
| --- прогнозирование_видео.py
```

Для создания нашего пользовательского детектора объектов требуется ряд конфигураций, поэтому мы создайте файл с именем lisa_config.py для хранения любых конфигураций, таких как пути к файлам, метки классов, и т. д.

build_lisa_records.py будет использоваться для приема входного набора изображений, создавая тренировочные и тестовые сплиты, а затем создавать файлы записей, совместимые с TensorFlow, которые можно использовать для обучения.

После того, как наша сеть будет обучена, мы можем использовать и прогнозирование.py , и прогнозирование_видео.py для применить нашу сеть к входным изображениям или видеофайлам/потокам соответственно.

Сценарий setup.sh содержит единственную команду, которую можно использовать для простого обновления PYTHONPATH для включения импорта API TFOD при выполнении скриптов Python через командную строку.

Каталог lisa содержит все файлы, относящиеся к набору данных LISA Traffic Signs, такие как сами необработанные изображения и сериализованные файлы записей. Я решил сохранить набор данных LISA на отдельном жестком диске в моей системе (где у меня больше места), а затем создайте символическую ссылку к набору данных в структуре моего проекта, аналогично тому, что мы делали в предыдущих главах этого

книга. Опять же, это мое личное предпочтение, но у вас могут быть свои собственные рекомендации, которые вам удобны.

Внутри каталога набора данных LISA я бы также предложил создать следующие каталоги для ваших экспериментов и данных:

```
$ cd lisa $  
mkdir записывает эксперименты $  
mkdir Experiments/обучающие эксперименты/оценочные эксперименты/exported_model
```

В каталоге записей будут храниться три файла: 1.

training.record: сериализованный набор данных изображений, ограничивающих прямоугольников и меток.
используется для
обучения. 2. testing.record: изображения, ограничивающие рамки и метки, используемые
для тестирования. 3.classes.pbtxt: открытый текстовый файл, содержащий имена меток классов и их уникальные
целочисленные идентификаторы.

Затем у нас есть подкаталог Experiments, в котором будут храниться все файлы, используемые для наших
обучающих экспериментов. Внутри каталога экспериментов есть три дополнительных подкаталога: обучение, оценка
и экспортированная_модель.

В учебном каталоге будет храниться специальный файл конфигурации конвейера, используемый для указания API TFOD,
как обучать нашу модель, предварительно обученную модель, которую мы будем использовать для тонкой настройки, и любые
контрольные точки модели, созданные во время обучения.

Пока мы обучаем нашу сеть, мы также будем запускать сценарий оценки, предоставляемый TFOD API — любые журналы,
сгенерированные сценарием оценки, будут храниться в каталоге оценки, что позволит нам использовать инструменты
TensorFlow для построения графиков и графиков процесса обучения с использованием специальная утилита TensorBoard.

Наконец, exported_model сохранит окончательную экспортованную замороженную модель веса после
завершения обучения.

Когда я впервые начал использовать TFOD API, я был немного ошеломлен количеством файлов, каталогов и подкаталогов,
необходимых для его использования. Путем проб и ошибок я разработал эту предлагаемую структуру каталогов, поскольку она
правильно сочетает простоту использования с тем, что требует API TFOD.

Для вашего удобства я включил шаблон структуры каталогов в загружаемый код, связанный с этой книгой.
Не стесняйтесь изменять эту предложенную структуру по своему усмотрению, но пока вы не успешно обучите
свою первую сеть, рассмотрите возможность использования предложенного мной метода.

16.3.2 Конфигурация

Первый файл, который мы собираемся изучить, — это файл setup.sh:

```
#!/бин/ш  
  
экспортировать PYTHONPATH=$PYTHONPATH:/home/adrian/  
модели/исследования:/дом/адриан/модели/исследования/тонкий
```

Все, что делает этот файл, — обновляет нашу переменную PYTHONPATH, чтобы включить импорт API TFOD.

 Вы захотите обновить эти пути, чтобы они указывали на место, где вы клонировали репозиторий TFOD API в
своей системе. Не копируйте и не вставляйте мои пути к файлам. Кроме того, PYTHONPATH должен
располагаться в одной строке — мне нужно было использовать две строки (из-за ограничений по ширине
текста в книге), чтобы уместить всю команду.

Если мы не обновим наш PYTHONPATH , чтобы включить API TFOD, наш импорт TensorFlow завершится ошибкой при выполнении наших скриптов. Всякий раз, когда я работаю с TFOD API, я всегда открываю оболочку и загружаю файл:

\$ ИСТОЧНИК setup.sh

Поиск сценария setup.sh и обновление моего PYTHONPATH для включения файлов API TFOD гарантирует, что мой импорт TensorFlow будет успешным.

Далее давайте определим файл конфигурации lisa_config.py:

```
1 # импортируем необходимые пакеты 2
импортируем ОС
3
4 # инициализируем базовый путь для набора данных LISA
5 BASE_PATH = "Лиза"
6
7 # построить путь к файлу аннотаций 8 ANNOT_PATH =
os.path.sep.join([BASE_PATH, "allAnnotations.csv"])
```

Строки 5 определяют BASE_PATH к каталогу дорожных знаков LISA (т. е. там, где находятся необработанные изображения, записи и журналы экспериментов). Здесь каталог lisa будет находиться в том же каталоге, что и наши сценарии build_lisa_records.py и predict.py.

Строка 8 использует BASE_PATH для получения пути к файлу аннотаций, предоставленному с помощью LISA.

Загрузка набора данных о дорожных знаках.

Используя BASE_PATH , мы также можем получить путь к нашим выходным записям обучения и тестирования. (75% для обучения, 25% для тестирования) вместе с открытым текстовым файлом меток классов:

```
10 # построить путь к выходным файлам записей обучения и тестирования, 11 # вместе
с файлом меток классов 12 TRAIN_RECORD = os.path.sep.join([BASE_PATH, "records/
training.record"])
13
14 TEST_RECORD = os.path.sep.join([BASE_PATH, "records/
testing.record"])
15
16 CLASSES_FILE = os.path.sep.join([BASE_PATH,
17         "записи/классы.pbtxt"])
18
19 # инициализируем размер тестового
сплита 20 TEST_SIZE = 0.25
21
22 # инициализируем словарь меток класса
23 КЛАССЫ = {"пешеходный переход": 1, "сигнал впереди": 2, "стоп": 3}
```

При обучении нашего Faster R-CNN нас будут интересовать только три класса: 1. Пешеходные переходы 2. Сигнал впереди 3. Знаки остановки Мы, конечно, могли бы тренироваться на всех 47 классах дорожных знаков, включенных в LISA, но ради этого например, мы будем тренироваться только на трех классах, что позволит нам быстрее обучить сеть и изучить результаты. Я оставлю вам обучение работе с полным набором данных LISA в качестве упражнения.

16.3.3 Класс аннотации TensorFlow

При работе с TFOD API нам необходимо построить набор данных, состоящий как из изображений, так и из связанные с ними ограничивающие рамки. Но прежде чем мы сможем приступить к созданию набора данных, нам нужно рассмотреть что составляет «точку данных» для обнаружения объекта? Согласно TFOD API, нам необходимо предоставить ряд атрибутов, в том числе:

- Изображение в кодировке TensorFlow.
- Ширина и высота изображения
- Кодировка файла изображения (например, JPG, PNG и т. д.)
- Имя файла
- Список координат ограничивающей рамки, нормализованных в диапазоне [0,1], для изображения.
- Список меток классов для каждой ограничивающей рамки.
- Флаг, используемый для кодирования того, является ли ограничивающая рамка «сложной» или нет (вы почти всегда захотите оставить это значение равным «0» или «несложно», чтобы TensorFlow тренировался на нем — флаг сложности — это остаток провокации ЛОС [58]).

Для кодирования всей этой информации в объекте требуется довольно много кода, поэтому для того, чтобы:

1. Держите наш скрипт сборки в чистоте и порядке
2. Повторно используйте код и, следовательно, уменьшите вероятность внесения ошибок в наш код.

Мы собираемся создать класс TFAnnotation для инкапсуляции кодирования данных обнаружения объекта. точка в формате TensorFlow. Наш класс TFAnnotation будет жить в tfannotation.py внутри utils подмодуль pyimagesearch:

```

| --- pyimagesearch
| |--- __init__.py
| |--- обратные вызовы
| |--- ио
...
| |--- утилиты
| | |--- __init__.py
| | |--- agegenderhelper.py
...
| | |--- tfannotation.py

```

Откройте tfannotation.py и вставьте следующий код:

```

1 # импортируем необходимые пакеты
2 из object_detection.utils.dataset_util импортировать bytes_list_feature
3 из object_detection.utils.dataset_util импортировать float_list_feature
4 из object_detection.utils.dataset_util импортировать int64_list_feature
5 из object_detection.utils.dataset_util импортировать int64_feature
6 из object_detection.utils.dataset_util импортировать bytes_feature

```

Строки 2–6 импортируют служебные функции набора данных TensorFlow, используемые для сериализации и кодирования различных Атрибуты и объекты Python — мы будем использовать эти функции для кодирования целых чисел, строк, списков и т. д. Затем мы можем определить конструктор для TFAnnotation:

```

8 класс TFAnnotation:
9     защита __init__(сам):
10         # инициализируем ограничивающую рамку + списки меток
11         self.xMins = []
12         self.xMакс = []

```

```

13         self.yMins = []
14         self.yMaxs = []
15         self.textLabels = []
16         самостоятельные классы = []
17         self.difficult = []

18
19         # инициализируем дополнительные переменные, включая изображение
20         # сама, пространственные размеры, кодировка и имя файла
21         self.image = Нет
22         self.width = Нет
23         self.height = Нет
24         самокодирование = Нет
25         self.filename = Нет

```

Как мы видим, наш конструктор просто выполняет серию инициализаций. `xMins`, `xMaxs`, `yMins` и `yMaxs` будут хранить (x, y)-координаты для наших ограничивающих рамок соответственно. Список `textLabels` — это список удобочитаемых меток классов для каждой ограничивающей рамки. Сходным образом, у нас есть классы, список целочисленных идентификаторов для каждой метки класса.

В строке 21 будет храниться изображение в кодировке TensorFlow вместе с его шириной, высотой и кодировкой изображения. тип и имя файла.

Как мы увидим в Разделе 16.3.4 ниже, мы сначала создадим экземпляр объекта `TFAnnotation`, а затем установить каждый из этих атрибутов. Когда мы будем готовы построить фактическую точку данных TensorFlow, мы можем вызвать метод сборки:

```

27     сборка защищы (сама):
28
29         # кодировать атрибуты, используя их соответствующие TensorFlow
30         # функции кодирования
31         w = int64_feature (собственная ширина)
32         h = int64_feature (собственная высота)
33         имя_файла = bytes_feature(self.filename.encode("utf8"))
34         кодировка = bytes_feature(self.encoding.encode("utf8"))
35         изображение = bytes_feature (self.image)
36         xMins = float_list_feature (self.xMins)
37         xMaxs = float_list_feature (self.xMaxs)
38         yMins = float_list_feature (self.yMins)
39         yMaxs = float_list_feature (self.yMaxs)
40         textLabels = bytes_list_feature (self.textLabels)
41         классы = int64_list_feature (self.classes)
42         сложный = int64_list_feature (самостоятельный. сложный)

```

Приведенный выше блок кода вызывает функции кодирования TensorFlow для каждого из наших атрибутов. Мы можем затем создайте словарь данных из этих закодированных объектов:

```

43     # создать словарь данных, совместимый с TensorFlow
44     данные = {
45         "изображение/высота": ч,
46         "изображение/ширина": w,
47         "изображение/имя файла": имя файла,
48         "image/source_id": имя файла,
49         "изображение/закодированное": изображение,
50         "изображение/формат": кодировка,
51         "изображение/объект/bbox/xmin": xMins,
52         "изображение/объект/bbox/xmax": xMaxs,

```

```

53         "изображение/объект/bbox/ymin": yMins,
54         "изображение/объект/bboxymax": yMaxs,
55         "изображение/объект/класс/текст": textLabels,
56         "изображение/объект/класс/метка": классы,
57         «образ/объект/сложный» : сложный,
58     }
59
60     # вернуть словарь данных
61     возвращаемые данные

```

Ключи к этому словарю данных не являются произвольными — это ключи и значения, которые TensorFlow ожидает от отдельной точки данных. Полный обзор API набора данных TensorFlow выходит за рамки этой книги. Если вам интересно узнать больше об этой кодировке, пожалуйста, обратитесь к документации TensorFlow [45].

Строка 61 возвращает объект, совместимый с TensorFlow, нашей вызывающей функции, что позволяет нам добавить объект в набор данных записи.

16.3.4 Создание набора данных LISA + TensorFlow

Чтобы обучить сеть с помощью TFOD API, нам сначала нужно преобразовать наши изображения и обозначения в формат записи TensorFlow, аналогично тому, как мы преобразовали набор данных изображения в формат mxnet. Давайте продолжим и преобразуем наш образец набора данных LISA — откройте build_lisa_records.py и вставьте следующий код:

```

1 # импортируем необходимые пакеты
2 из конфига импортировать lisa_config как конфиг
3 из pyimagesearch.utils.tfannotation импортировать TFAnnotation
4 из sklearn.model_selection импорта train_test_split
5 из изображения импорта PIL
6 импортировать тензорный поток как tf
7 импорт ОС

```

Строки 2-7 импортируют необходимые пакеты Python. Стока 2 импортирует наш специальный файл конфигурации поэтому у нас есть доступ к нашим путям к входным изображениям, аннотациям, файлам и файлам выходных записей. На линии 3 мы импортируйте наш пользовательский класс TFAnnotation, чтобы сделать наш фактический build_lisa_records.py чище и легче читать.

Далее давайте начнем определять основной метод, который будет выполняться при запуске нашего скрипта:

```

9 ndef основной():
10     # открываем выходной файл классов
11     f = открыть (config.CLASSES_FILE, "w")
12
13     # цикл по классам
14     для (k, v) в config.CLASSES.items():
15         # создать информацию о классе и записать в файл
16         элемент = ("элемент \n"
17                     "\tid: " + str(v) + "\n"
18                     "\tname: " + k + "\n"
19                     "}\n")
20         f.написать (элемент)
21
22     # закрыть файл выходных классов

```

```

23     e.закрыть()
24
25     # инициализировать словарь данных, используемый для отображения каждого имени файла изображения
26     # ко всем ограничивающим рамкам, связанным с изображением, затем загрузите
27     # содержимое файла аннотаций
28     Д = {}
29     строки = открыть (config.ANOT_PATH).read().strip().split("\n")

```

В строке 11 мы открываем указатель файла на наш файл меток выходного класса. Строки 13-20 , затем цикл над каждой из меток класса, записывая каждую в файл. API TFOD ожидает этот файл в определенном JSON/YAML-подобный формат.

Каждый класс должен иметь объект элемента с двумя атрибутами: id и name. Имя _ _ удобочитаемое имя для метки класса, в то время как идентификатор представляет собой уникальное целое число для метки класса. Держать Имейте в виду, что идентификаторы классов должны начинать отсчет с 1 , а не с 0 , поскольку ноль зарезервирован для «фоновый» класс.

Строка 28 инициализирует словарь данных, используемый для сопоставления каждого отдельного имени файла изображения с его соответствующим ограничивающие рамки и аннотации. Страна 29 затем загружает содержимое нашего файла аннотаций.

Формат этого файла аннотаций станет ясен в следующем блоке кода:

```

31     # цикл по отдельным строкам, пропуская заголовок
32     для строки в строках [1:]:
33         # разбить строку на компоненты
34         строка = строка.split(",") [0].split(";")
35         (imagePath, метка, startX, startY, endX, endY, _) = строка
36         (startX, startY) = (float(startX), float(startY))
37         (конецX, конецY) = (с плавающей запятой (конецX), с плавающей запятой (конецY))
38
39         # если нас не интересует метка, игнорируем ее
40         если метка не в config.CLASSES:
41             Продолжать

```

В строке 32 мы начинаем перебирать каждую из отдельных строк в файле аннотаций, пропуская заголовок. Страна 34 разбивает строку, разделенную запятыми, на список. Страна 35 затем извлекает значения из строк, что позволяет нам увидеть шесть важных компонентов наших аннотаций CSV. файл:

1. Входной путь к файлу изображения
2. Метка класса
3. Начальная координата x ограничивающей рамки
4. Начальная координата Y ограничивающей рамки
5. Конечная координата x ограничивающей рамки
6. Конечная координата Y ограничивающей рамки

Строки 36 и 37 преобразуют координаты ограничивающей рамки из строк в числа с плавающей запятой. Если ряд мы в настоящее время рассмотрение не включено в наш список КЛАССОВ, мы его игнорируем (строки 40 и 41).

Поскольку изображение может содержать несколько дорожных знаков и, следовательно, несколько ограничивающих рамок, нам нужно использовать словарь Python для сопоставления пути изображения (как ключа) со списком меток и связанные ограничивающие рамки (значение):

```

43     # построить путь к входному изображению, затем взять любой другой
44     # ограничивающие рамки + метки, связанные с изображением
45     # списки путей, меток и ограничивающих рамок соответственно

```

```

46         p = os.path.sep.join([config.BASE_PATH, imagePath])
47         b = D.get(p, [])
48
49         # построить кортеж, состоящий из метки и ограничивающей рамки,
50         # затем обновить список и сохранить его в словаре
51         b.append((метка, (startX, startY, endX, endY)))
52     D[p] = b

```

Строка 46 строит полный путь к текущему imagePath, используя наш BASE_PATH из конфигурации. файл. Затем мы извлекаем все ограничивающие рамки + метки классов для текущего пути к изображению p (строка 47). Мы добавьте координаты нашей метки и ограничивающей рамки в список b, а затем сохраните их обратно в сопоставлении словарь, D (строки 51 и 52).

Это сопоставление может показаться неважным, но имейте в виду, что нам все равно нужно:

1. Создайте сплит для обучения и тестирования
2. Убедитесь, что наше разделение обучения и тестирования выполняется на изображениях, а не на ограничении коробки

Наша цель здесь состоит в том, чтобы убедиться, что если конкретное изображение помечено как «обучающее» , то все ограничивающие коробки для этого изображения включены в обучающий набор. Точно так же, если изображение помечено как «тестирование» , затем мы хотим, чтобы все ограничивающие рамки для этого изображения были связаны с тестовым набором.

Мы хотим избежать ситуаций, когда изображение содержит ограничивающие рамки как для обучения, так и для тестовые наборы. Такое поведение не только неэффективно, но и создает более серьезную проблему — обнаружение некоторых объектов Алгоритмы используют интеллектуальный анализ с жестким отрицанием, чтобы повысить их точность, боясь немаркированные области изображения и рассматривать их как негативы.

Поэтому возможно, что наша сеть могла подумать, что конкретный ROI не был дорожным знаком, когда на самом деле это было так, что сбивало с толку нашу сеть и наносило ущерб точности. Из-за этого нюанса вы всегда должен ассоциировать изображение, а не ограничивающую рамку, с тренировочным или тестовым набором.

Кстати говоря, давайте сейчас создадим наш сплит обучение/тестирование:

```

54     # создаем тренировочные и тестовые сплиты из нашего словаря данных
55     (trainKeys, testKeys) = train_test_split (список (D.keys()),
56                                         test_size=config.TEST_SIZE, random_state=42)
57
58     # инициализируем файлы разделения данных
59     наборы данных = [
60         ("поезд", trainKeys, config.TRAIN_RECORD),
61         ("тест", testKeys, config.TEST_RECORD)
62     ]

```

Мы используем train_test_split scikit -learn для создания нашего разделения. Обратите внимание, как мы разделяемся на ключи словаря отображения D , гарантируя, что мы разделим пути изображения, а не ограничивающие коробки, избегая потенциально опасного сценария, упомянутого выше.

Строки 59-62 определяют список наборов данных , связывая обучающие и тестовые сплиты с их связанные выходные файлы.

На данный момент мы готовы создать наши файлы записей TensorFlow:

```

64     # цикл по наборам данных
65     for (dType, keys, outputPath) в наборах данных:
66         # инициализируем модуль записи TensorFlow и инициализируем общий
67         # количество примеров, записанных в файл
68         print("[INFO] обрабатывает '{}'".format(dType))
69         писатель = tf.python_io.TFRecordWriter (outputPath)

```

```

70     всего = 0
71
72     # цикл по всем ключам в текущем наборе
73     для k в ключах:
74         # загружаем входное изображение с диска как объект TensorFlow
75         закодировано = tf.gfile.GFile(k, "rb").read()
76         закодировано = байты(закодировано)
77
78         # снова загрузить образ с диска, на этот раз как PIL
79         # объект
80         pilImage = Image.open(k)
81         (ш, ч) = pilImage.size[:2]

```

Мы начинаем зацикливаться на нашем обучении и тестировании сплитов в строке 65. Мы создаем экземпляр нашего писателя , TFRewriter, который указывает на наш текущий outputPath в строке 69.

Строка 73 начинает цикл по ключам нашего словаря отображения D для текущего разделения. Держать Имейте в виду, что k на самом деле является путем к изображению — используя этот путь, мы можем загрузить закодированное изображение в формат TensorFlow в строках 75 и 76.

Можно извлечь ширину и высоту из объекта, закодированного TensorFlow, но это немного утомительно — я предпочитаю просто загружать изображение в формате PIL/Pillow и извлекать размеры (строки 80 и 81). Загрузка образа во второй раз требует дополнительного вызова ввода-вывода, но это сценарий выполняется только один раз для каждого набора данных — в этом случае я предпочитаю чистый, понятный код с меньшим количеством ошибок. обстоятельство.

Теперь мы можем создать наш объект tfAnnot:

```

83     # разобрать имя файла и кодировку из входного пути
84     имя файла = k.split(os.path.sep)[-1]
85     кодировка = имя файла[имя файла.rfind(".") + 1:]
86
87     # инициализируем объект аннотации, используемый для хранения
88     # информация об ограничительной рамке + метки
89     tfAnnot = TFAAnnotation()
90
91     tfAnnot.image = закодировано
92     tfAnnot.encoding = кодировка
93     tfAnnot.filename = имя файла
94     tfAnnot.width = ш
95     tfAnnot.height = h

```

Строки 84 и 85 извлекают имя файла и кодировку изображения (например, JPG, PNG и т. д.) из Путь файла.

В строках 89–94 создается экземпляр объекта tfAnnot и задаются кодированное изображение TensorFlow, кодировка файла, имя файла, ширина и высота. Мы не добавили информацию о ограничивающей рамке в tfAnnot, так что давайте позаботимся об этом сейчас:

```

96     # цикл по ограничивающим рамкам + метки, связанные с
97     # Изображение
98     for (метка, (startX, startY, endX, endY)) в D[k]:
99         # TensorFlow предполагает, что все ограничивающие рамки находятся в
100        # диапазон [0, 1], поэтому нам нужно их масштабировать
101        xMin = началоХ/ш
102        xМакс = конецХ/ш
103        yMin = пускY/ч

```

```

104         yMax = конецY/ч
105
106         # обновить списки ограничивающих рамок + меток
107         tfAnnot.xMins.append(xMin)
108         tfAnnot.xMакс.append(xMax)
109         tfAnnot.yMins.append(yMin)
110         tfAnnot.yMакс.append(yMax)
111         tfAnnot.textLabels.append(label.encode("utf8"))
112         tfAnnot.classes.append(config.CLASSES[метка])
113         tfAnnot.difficult.append(0)
114
115         # увеличить общее количество примеров
116         всего += 1

```

Строка 98 повторяет все метки и ограничивающие рамки, связанные с путем изображения, к. TensorFlow предполагает, что все ограничивающие рамки находятся в диапазоне [0,1], поэтому мы выполняем шаг масштабирования на Строки 101-104 путем деления координат ограничивающей рамки на их связанную с или высоту.

Строки 107-113 добавляют метку класса и информацию о ограничивающей рамке к объекту tfAnnot. Хорошо также увеличиваем наш общий счет, чтобы мы могли отслеживать общее количество ограничивающих рамок для каждого тренировочный/тестовый сплит соответственно.

Наш последний блок кода обрабатывает атрибуты точек данных в формате TensorFlow, добавляя пример писателю и убедиться, что наш основной метод выполняется через TensorFlow, когда мы запустим наш скрипт build_lisa_records.py:

```

118     # кодируем атрибуты точек данных с помощью TensorFlow
119     # вспомогательные функции
120     функции = tf.train.Features(feature=tfAnnot.build())
121     пример = tf.train.Example(функции = функции)
122
123     # добавляем пример в писатель
124     Writer.write(пример.SerializeToString())
125
126     # закрыть модуль записи и распечатать диагностическую информацию в
127     # пользователь
128     писатель.close()
129     print("[INFO] {} примеров, сохраненных для '{}'".format(total,
130         дТип))
131
132 # проверить, должен ли быть запущен основной поток
133 , если __name__ == "__main__":
134     tf.app.run()

```

Чтобы создать файл записей LISA, откройте терминал и выполните следующую команду:

```

$ время Python build_lisa_records.py
[INFO] обрабатывает "поезд"...
[INFO] 2876 примеров сохранено для слова "поезд"
[INFO] обработка "теста"...
[INFO] 955 примеров сохранено для «теста»

```

настоящий	0м4.879с
пользователь	0м3.117с
система	0м2.580с

Здесь вы можете видеть, что весь процесс занял всего четыре секунды. Изучение вашей лизы / записей каталоге вы должны увидеть наши обучающие и тестовые файлы:

```
$ ls лиза/записи/
классы.pbtxt тестирование.запись обучение.запись
```

Как видите, мы успешно создали наши файлы записей и связанный с ними файл меток классов в формат TensorFlow.

16.3.5 Критический шаг перед тренировкой

На этом этапе мы можем приступить к обучению нашего Faster R-CNN; однако я хочу закончить словами предупреждение:

Одна из самых больших ошибок, которую я наблюдаю у разработчиков глубокого обучения, студентов и исследователей, заключается в следующем. спешить и не перепроверять свою работу при построении датасета.

Если ваш школьный проект должен быть сдан сегодня в полночь, ваш босс дышит вам в затылок. для последней и лучшей модели, или вы просто возитесь с глубоким обучением в качестве хобби, будьте предупредил: спешка вызовет только проблемы, особенно в контексте обнаружения объектов.

Имейте в виду, что мы работаем не только с входным изображением и меткой класса — мы теперь есть четыре дополнительных компонента: сами координаты ограничительной рамки. Слишком часто я видеть, как люди предполагают, что их код правильный, и если сценарий выполняется и завершается без ошибок, то все должно быть в порядке. Не попадайтесь в эту ловушку.

Прежде чем вы даже подумаете об обучении своей сети, дважды проверьте свой код. В частности, или через строки 98-113, когда мы строим ограничивающую рамку и метку класса для данного изображения. Ты должен визуально подтвердить, что ваш код работает так, как вы думаете:

1. Загрузка образа с диска через OpenCV
2. Нанесение координат ограничивающей рамки на изображение путем масштабирования xMin, xMax, yMin и yMax. вернуться к стандартным целочисленным координатам
3. Нанесение метки класса также на изображение

Ниже я включил фрагмент кода о том, как я выполняю этот процесс:

```
96      # цикл по ограничивающим рамкам + метки, связанные с
97      # Изображение
98      for (метка, (startX, startY, endX, endY)) в D[k]:
99          # TensorFlow предполагает, что все ограничивающие рамки находятся в
100         # диапазон [0, 1], поэтому нам нужно их масштабировать
101         xMin = началоЧ/w
102         xМакс = конецЧ/ш
103         yMin = пускY/ч
104         yMax = конецY/ч
105
106         # загрузим входное изображение с диска и денормализуем
107         # координаты ограничивающей рамки
108         изображение = cv2.imread (k)
109         startX = интервал (xMin * w)
110         startY = интервал (yMin * ч)
111         конецХ = интервал (xMax * w)
112         конецY = интервал (yMax * ч)
113
114         # рисуем ограничивающую рамку на изображении
115         cv2.rectangle (изображение, (startX, startY), (endX, endY),
116                         (0, 255, 0), 2)
```

```

117
118     # показать выходное
119     изображение cv2.imshow("Image",
120     image) cv2.waitKey(0)

```

Я не буду проверять каждое изображение в своем наборе данных, но я выберу несколько сотен для каждого класса и потрачу 10-20 минут на поиск ошибок. Эта визуальная проверка имеет решающее значение, и ее нельзя пропускать ни при каких обстоятельствах.

Не поддавайтесь ложному предположению, что только потому, что ваш код выполняется и завершается, все работает нормально. Вместо этого используйте пессимистический подход: ничто не работает нормально, пока вы не потратите время на перепроверку своей работы.

Если вы хотите работать со своими собственными пользовательскими наборами данных и создавать связанные файлы записей, вы должны использовать этот сценарий в качестве отправной точки и изменить синтаксический анализ метки класса/ограничивающей рамки. Страйтесь не изменять реальную конструкцию объекта TFAnnotation настолько хорошо, насколько это возможно, чтобы свести к минимуму возможность вставки ошибок.

Хотя вы, безусловно, можете анализировать информацию из файлов записей TensorFlow после их создания, это утомительный и трудоемкий процесс, и ваше время и усилия лучше потратить до того, как объект аннотации будет добавлен в запись.

Опять же, я не могу не подчеркнуть важность достаточной проверки ограничивающих рамок — возьмите время, избегайте дорогостоящих ошибок, как с точки зрения времени, так и финансовых, потраченных на обучение вашей сети.

Если вы не получаете удовлетворительной точности для определенного набора данных обнаружения объектов, ни одна из моих рекомендаций по настройке гиперпараметров не имеет значения и не окажет влияния, если вы не потратите время на проверку своего набора данных перед тем, как перейти к обучению — пожалуйста, ради вас, помните об этом при работе с собственными данными.

16.3.6 Настройка Faster R-CNN Обучение нашего Faster

R-CNN на наборе данных LISA представляет собой четырехэтапный процесс:

1. Загрузите предварительно обученный Faster R-CNN, чтобы мы могли точно настроить сеть.
2. Загрузите образец файла конфигурации TFOD API и измените его, чтобы он указывал на наши файлы записей . 3. Запустите процесс обучения и отслеживайте. 4. Экспортируйте замороженный график модели после завершения обучения

 Ссылки R могут и будут меняться со временем. Я сделаю все возможное, чтобы поддерживать эту книгу в актуальном состоянии, но если вы обнаружите, что ссылка имеет ошибку 404 или результирующая страница не похожа на снимок экрана, включенный в эту книгу, обратитесь к сопутствующему веб-сайту (<http://pyimg.co/fnkxk>) самые свежие ссылки.

Чтобы загрузить нашу предварительно обученную модель, вам сначала нужно отправиться в Зоопарк моделей обнаружения объектов TensorFlow: <http://pyimg.co/1234r>. Здесь вы найдете таблицу, в которой перечислены различные модели, обученные на общих объектах.

в наборе данных Context (COCO) [59].

Набор данных COCO содержит более 200 000 помеченных изображений. Сети, обученные тесту COCO, могут обнаруживать 20 ярлыков классов, включая самолеты, велосипеды, автомобили, собак, кошек и многое другое.

Сети, обученные на большом наборе данных COCO, обычно можно настроить для распознавания объектов и в других наборах данных.

Чтобы скачать модель, найдите ссылку Faster R-CNN + ResNet-101 + COCO в таблице (рис. 16.2) и загрузите ее. На момент написания этой статьи имя файла было fast_rcnn_resnet101_coco_2018_01_28.tar.gz. Опять же, убедитесь, что вы проверили таблицу на странице TFOD API GitHub, чтобы загрузить последнюю модель). Дополнительные сведения о загрузке предварительно обученной модели и настройке среды разработки см. в разделе 16.2 выше.

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes

Рисунок 16.2: В этом примере мы будем использовать архитектуру Faster R-CNN + ResNet-101, обученную на наборе данных COCO. Мы будем настраивать эту модель для обнаружения дорожных знаков.

После того, как вы загрузили модель, переместите ее в подкаталог Experiments /training и распакуйте ее:

```
$ cd lisa/experiments/training $ mv
~/Downloads/faster_rcnn_resnet101_coco_2018_01_28.tar.gz ./ $ tar -zxf
faster_rcnn_resnet101_coco_2018_01_28.tar.gz $ ls -
faster_rcnn_resnet101_coco_2018_01_28 checkpoint frozen_inference_graph.pb
model.ckpt.data-00000-of-00001 model .ckpt.index model.ckpt.meta pipe.config
save_model
```

Файлы внутри fast_rcnn_resnet101_coco_2018_01_28 составляют предварительно обученный TensorFlow Faster R-CNN в наборе данных COCO. Точная структура этого каталога станет ясна по мере работы с главой — мы отложим объяснение каждого файла до соответствующих разделов соответственно.

Теперь, когда у нас есть веса модели, нам также нужен файл конфигурации, чтобы указать API TFOD, как обучать/тонко настраивать нашу сеть. Попытка определить свой собственный файл конфигурации TFOD API с нуля была бы чрезвычайно трудным, утомительным и трудным процессом. Вместо этого рекомендуется взять один из существующих файлов конфигурации и обновить пути и любые другие параметры, которые вы считаете подходящими.

Полный список примеров файлов конфигурации TFOD API можно найти на этой странице: <http://pyimg.co/r2xql>. Вы увидите четыре файла для архитектуры Faster R-CNN + ResNet-101 (рис. 16.3, обведено красным) — обязательно загрузите файл fast_rcnn_resnet101_pets.config и сохраните его в каталоге Experiments / training:

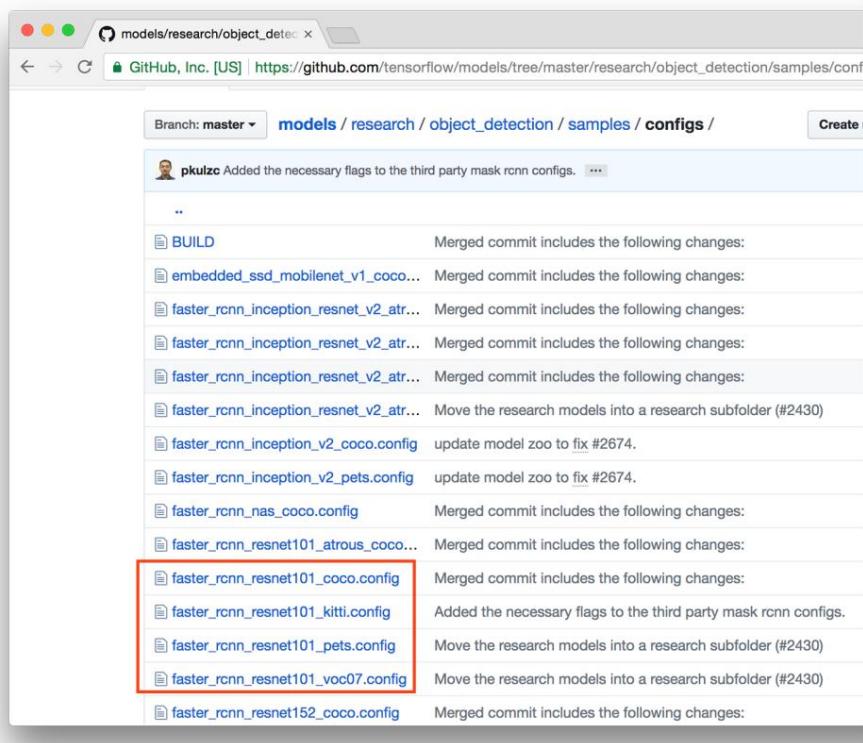


Рисунок 16.3: Наряду с замороженными весами модели нам также нужен начальный файл конфигурации. API TFOD предоставляет ряд файлов конфигурации для архитектуры Faster R-CNN + ResNet-101.

```
$ cd lisa/experiments/training $ mv ~/Downloads/faster_rcnn_resnet101_pets.config fast_rcnn_lisa.config
```

Я также предпринял дополнительный шаг, переименовав файл конфигурации в `fast_rcnn_lisa.config`. Вы можете быть сбиты с толку, почему мы используем конфигурацию, связанную с набором данных Oxford-IIIT «Pets» [60], а не с набором данных COCO, на котором изначально обучалась сеть. Короткий ответ заключается в том, что конфигурация Pets требует меньше изменений, чем конфигурация COCO (по крайней мере, на момент написания этой статьи) — конфигурация Pets, как правило, является лучшей отправной точкой.

Открыв нашу конфигурацию `fast_rcnn_lisa.config`, вы увидите большое количество данных в конфигурации, подобной JSON/YAML:

```
$ vi более быстрая_rcnn_lisa.config
модель { более быстрая_rcnn
    { num_classes: 37 image_resizer
        { keep_aspect_ratio_resizer
            { min_dimension: 600
                max_dimension: 1024
            }
        }
    }
} feature_extractor
{ тип: 'faster_rcnn_resnet101'
```

```

        first_stage_features_stride: 16
    }
...

```

К счастью, нам нужно обновить только семь конфигураций, пять из которых — это пути к файлам, можно легко скопировать и вставить.

Начнем с установки количества `num_classes` — изначально это значение равно 37, но мы изменим до 3, общее количество классов (пешеходный переход, сигнал впереди и знак остановки) в нашей LISA образец набора данных:

```

8   быстрее_rcnn {
9     количество_классов: 3
10    image_resizer {
11      keep_aspect_ratio_resizer {
12        мин_размер: 600
13        макс_размер: 1024
14      }
15    }

```

Затем вы захотите обновить `num_steps`, то есть количество шагов `batch_size` для выполнять при тренировке:

```

84 train_config: {
85   размер партии: 1
86 ...
87   количество_шагов: 50000
88   data_augmentation_options {
89     random_horizontal_flip {
90   }
91 }
92 }

```

По умолчанию это значение равно 200 000, но я рекомендую использовать 50 000 для этого конкретного набора данных. За большинство наборов данных вы захотите обучить минимум 20 000 шагов. Вы, безусловно, можете тренироваться больше шаги, особенно если вы видите, что производительность вашей сети со временем улучшается, но я обычно даже не прекращаю тренировки, пока не пройдете 20 000 шагов.

Обычно вы увидите `batch_size=1` для более быстрых R-CNN, использующих API TFOD. Ты сможешь отрегулируйте это значение по своему усмотрению в своих собственных экспериментах, при условии, что у вашего графического процессора достаточно памяти.

Вы заметите, что API конфигурации удобно имеет текст `PATH_TO_BE_CONFIGURED`. размещается в файле конфигурации везде, где вам нужно обновить путь к файлу. Первый из них `Fine_tune_checkpoint`, который является путем к нашему скаченному Faster R-CNN + ResNet 101 Базовый файл `model.ckpt`:

```

109  градиент_отсечения_по_норме: 10.0
110  Fine_tune_checkpoint: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/lisa/
111      эксперименты/обучение/faster_rcnn_resnet101_coco_2018_01_28/model.ckpt"
112  from_detection_checkpoint: правда

```

Убедитесь, что вы указали полный путь к этому файлу без каких-либо ярлыков консоли, таких как ~ оператор каталога.

 Если вы изучите содержимое файла `fast_rcnn_resnet101_coco_2018_01_28`, вы заметите фактического файла `model.cpkt` нет, но есть три файла, которые начинаются с `model.cpkt` — это ожидается. Файл `model.cpkt` — это базовое имя файла, которое API TFOD использует для получить остальные три файла.

Следующие два обновления `PATH_TO_BE_CONFIGURED` можно найти внутри `train_input_reader`, в частности, `input_path` и `label_map_path`:

```
123 train_input_reader: {
124     tf_record_input_reader {
125         input_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/lisa/
126             записи/обучение.запись"
127     }
128     label_map_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/lisa/
129         записи/классы.pbtxt"
130 }
```

Вы захотите обновить эти пути, чтобы они указывали на ваш файл `training.record` и `class.pbtxt`. файл соответственно. Опять же, вполне вероятно, что ваша структура каталогов будет немного отличаться от моей. так что не копируйте и не вставляйте мои пути — убедитесь, что вы перепроверили свои пути и включили их в файл конфигурации.

`eval_input_reader` требует, чтобы мы обновили `input_path` и `label_map_path` как хорошо:

```
137 eval_input_reader: {
138     tf_record_input_reader {
139         input_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/lisa/
140             записи/тестирование.запись"
141     }
142     label_map_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/lisa/
143         записи/классы.pbtxt"
144     перемешивание: ложь
145     число_читателей: 1
146 }
```

Но на этот раз убедитесь, что вы указали путь к файлу `testing.record`, а не файл `training.record`.

Я бы также предложил обновить ваш `eval_config`:

```
130 eval_config: {
131     число_примеров: 955
132     # Примечание. Стока ниже ограничивает процесс оценки 10 оценками.
133     # Удалите строку ниже, чтобы оценивать бесконечно.
134     #max_evals: 10
135 }
```

Установите `num_examples` как общее количество ограничивающих рамок в тестовом наборе. Мне также нравится закомментировать `max_evals`, так как это гарантирует, что скрипт оценки будет работать бесконечно, пока мы остановить его вручную, но это решение зависит от вас и вашего оборудования. Мы обсудим, как вам следует правильно установите `max_evals` и запустите сценарий оценки в разделе 16.3.8.

16.3.7 Обучение более быстрой R-CNN

Теперь мы готовы обучить нашу архитектуру Faster R-CNN + ResNet-101 на наборе данных LISA Traffic Signs! Может показаться, что для достижения этой цели было сделано много шагов, но имейте в виду два ключевых аспекта: 1. TFOD API предназначен для расширения, повторного использования и настройки, поэтому по определению будет больше файлов конфигурации. Использование этих файлов конфигурации на самом деле выгодно вам, конечному пользователю, поскольку позволяет писать меньше кода и сосредоточиться на фактических параметрах модели.

2. Процесс становится значительно проще, чем больше и больше вы проводите этих экспериментов.

Поскольку мы используем API TFOD, нам не нужно писать какой-либо фактический код для запуска процесса обучения — единственный пользовательский код, необходимый с нашей стороны, — это создание набора данных записи, как мы делали в предыдущих разделах. Оттуда сценарии, представленные в каталоге `models/research/object_detection` API TFOD, можно использовать для обучения и оценки наших сетей.

Обучение можно начать с помощью следующей команды (убедитесь, что вы используете файл `setup.sh` script для экспорта вашего PYTHONPATH перед запуском этой команды):

```
$ python object_detection/train.py --logtostderr \ --pipeline
ssds_and_rcnn/lisa/experiments/training/faster_rcnn_lisa.config \ --train_dir ssds_and_rcnn/
lisa/experiments/training
ИНФОРМАЦИЯ:tensorflow:Начало сеанса.
ИНФОРМАЦИЯ:tensorflow:Начальные очереди.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Сводка записи на шаге 0.
ИНФОРМАЦИЯ: тензорный поток: глобальный шаг 1: потеря = 2,3321 (5,715 с/шаг)
ИНФОРМАЦИЯ: тензорный поток: глобальный шаг 2: потеря = 2,1641 (0,460 сек/шаг)
...
```

Обратите внимание, как переключатель `--pipeline` указывает на наш файл `fast_rcnn_lisa.config`. Переключатель `-train-dir` затем указывает на наш учебный подкаталог внутри данных.

 Пути к входным файлам и каталогам могут быть довольно длинными, поэтому для удобства я создал символическую ссылку из каталога `ssds_and_rcnn` в каталог `models/research`, что позволяет мне тратить меньше времени на ввод и отладку длинных путей к файлам. Вы также можете применить эту тактику.

В то время как приведенная выше команда начнет обучение сети, нам нужна вторая команда, чтобы начать оценку. Откройте второй терминал и выполните следующую команду:

```
$ python object_detection/eval.py --logtostderr \ --
pipeline_config_path ssds_and_rcnn/lisa/experiments/training/faster_rcnn_lisa.config \ --checkpoint_dir
ssds_and_rcnn/lisa/experiments/training \ --eval_dir ssds_and_rcnn/lisa/experiments/evaluation
```

...

Здесь мы

указываем: • `--pipeline_config_path`, который указывает на наш файл `fast_rcnn_lisa.config` • `--checkpoint_dir`, который является путем к подкаталогу обучения и где все контрольные точки модели будут сохранены во время обучения • `--eval_dir`, который является путем к оценке подкаталог, в котором хранятся журналы оценки будет храниться

Обратите внимание, как две приведенные выше команды были выполнены внутри каталога models/research API TFOD (то есть там, где живут скрипты object_detection/train.py и object_detection/eval.py).

Последняя команда может быть выполнена в любом месте вашей системы, если вы укажете полный путь в каталог данных, содержащий наши подкаталоги для обучения и оценки:

```
$ cd ~/pyimagesearch/dlbook/ssds_and_rcnn/ $  
tensorboard --logdir lisa/experiments TensorBoard  
0.1.8 по адресу http://annalee:6006 (нажмите CTRL+C, чтобы выйти)
```

Команду tensorboard можно использовать для создания хорошей визуализации процесса обучения в нашем веб-браузере (т. е. без разбора журналов, как в mxnet). На моей машине я могу визуализировать процесс обучения, открыв веб-браузер и указав в нем адрес http://annalee:6006.

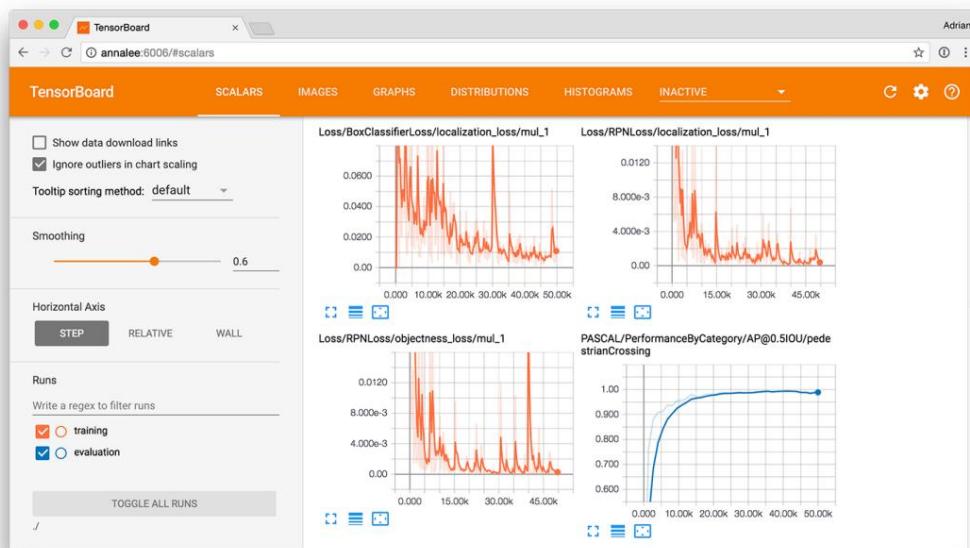


Рисунок 16.4: Пример скриншота моей панели управления TensorBoard после 24 часов обучения.

На вашей машине порт все равно должен быть 6006 , но имя хоста (например, annalee) будет либо IP-адрес вашей машины или имя самой машины.

Кроме того, для заполнения панели инструментов TensorBoard может потребоваться несколько минут — и train.py, и eval.py должны работать некоторое время, чтобы сгенерировать журналы для TensorBoard для анализа и отображения. Кроме того, некоторые значения TensorBoard, такие как mAP, будут невидимы до тех пор, пока скрипт eval.py не запустится. В зависимости от скорости вашей машины и процесса оценки отображение значений mAP может занять более 15-30 минут.

На рис. 16.4 показан снимок экрана моей приборной панели TensorBoard. Здесь вы можете видеть потерю стабильно уменьшается по мере роста моего mAP, увеличивается по мере обучения нашей сети.

Обучение будет продолжаться до тех пор, пока не будет выполнено num_steps или пока вы не выйдете из скрипта , нажав Ctrl + C. — вам также нужно нажать ctrl + c из eval.py и tensorboard.

К тому времени, когда мы достигаем шага 50 000, значение моего убытка составляет 0,008 с mAP 98%, что очень хорошо.

Я завершу этот раздел, сказав, что если вы впервые пытаетесь обучить сеть с помощью TFOD API, вы, скорее всего, столкнетесь с ошибками. Обязательно прочтайте Раздел 16.3.8 ниже, так как я

перечисленные рекомендации по обучению ваших сетей, чтобы помочь вам (1) полностью избежать ошибок и (2) устранить их при появлении ошибок.

16.3.8 Предложения по работе с TFOD API

Этот раздел включает в себя ряд рекомендаций и лучших практик, которые я лично использую при работе с TFOD API. Уделите время внимательному прочтению этого раздела, так как он поможет вам диагностировать ошибки и лучше обучать собственные сети обнаружения объектов.

Примите, что существует кривая обучения .

Первый факт, который вам нужно принять, заключается в том, что при использовании TFOD API существует кривая обучения, и нет никакого способа обойти это. Когда вы впервые используете API TFOD, не ожидайте, что все скрипты будут работать «из коробки» .

Откровенно говоря, это нереально. Мое предложение состоит в том, чтобы выделить по крайней мере один-два дня (в зависимости от вашего уровня опыта работы с системами Python, Unix и диагностики ошибок) для ознакомления с API.

Если вы ожидаете, что весь ваш конвейер будет запущен менее чем за час, особенно если вы впервые используете API TFOD, ваши ожидания не совпадают с реальностью — это не то, как эти современные инструменты глубокого обучения работают.

Не расстраивайтесь. Не раздражайтесь. И, несмотря на замешательство или разочарование, не покидайте свое рабочее место и сразу же бегите за помощью к своему коллеге, сослуживцу или однокурснику. Ошибки с этими инструментами будут происходить. Это ожидаемо.

Начните с глубокого вдоха, а затем найдите время, чтобы диагностировать ошибку, и снова примите, что существует кривая обучения — трассировки стека, созданные TensorFlow, обычно полезны и включают в себя то, что ошибка (хотя вам может потребоваться прокрутить вывод, чтобы точно определить, в чем ошибка , поскольку вывод может быть немного подробным).

Во многих ситуациях ошибки, с которыми вы столкнетесь, скорее всего, будут связаны с неправильным обновлением пути Python или путей к файлам. Всегда дважды проверяйте пути к файлам!

Сначала повторите известные

результаты Моя вторая рекомендация — обучить вашу первую Faster R-CNN на наборе данных LISA Traffic Sign, как мы это делаем в этой главе. Будете ли вы обучать сеть более 50 000 шагов, зависит от вас, но я бы обучил не менее 2500 шагов, чтобы вы могли убедиться, что ваш конвейер обучения и оценки работает правильно — точность на этом этапе все еще будет низкой, но вы по крайней мере, иметь возможность определить , действительно ли сеть обучается.

Слишком часто я вижу студентов, инженеров, исследователей и практиков, изучающих глубокое обучение, которые сразу же пытаются: 1. Заменить примеры наборов данных 2. Вставить свои собственные данные 3.

Пытаться обучить сеть плохие результаты Не делайте этого. Слишком много движущихся частей, чтобы сразу поменять их в собственном наборе данных. Вам нужно научиться использовать инструменты TFOD API, прежде чем вы сможете обучать сеть на своем собственном наборе данных.

Независимо от того, сталкиваетесь ли вы с крайним сроком проекта, пытаетесь ли вы выполнить требования к выпускному/заключительному камню или просто играете с хобби-проектом, простой факт заключается в следующем: нет никакого оправдания тому, что вы сначала не воспроизвели примеры результатов, обсуждаемые здесь, прежде чем вы начнете работать со своим проектом. собственные данные. Обоснование этой мысли восходит к научному методу — не меняйте слишком много переменных одновременно.

Прямо сейчас ваша цель — добраться до вершины кривой обучения TFOD API. Лучший способ добраться до вершины кривой обучения — это работать с набором данных и примером, где вы можете ясно увидеть ожидаемые результаты — воспроизведите эти результаты, прежде чем переходить к другим экспериментам.

Экспортируйте свой PYTHONPATH

Еще одна распространенная ошибка, которую я вижу, — это сбой при обновлении вашего PYTHONPATH при попытке выполнить либо (1) скрипты в каталоге models/research API TFOD, либо (2) попытку импортировать файлы из API TFOD. Если вы забудете обновить PYTHONPATH , ваша ошибка, скорее всего, будет выглядеть примерно так:

```
$ python build_lisa_records.py
Трассировка (самый последний вызов последним):
    Файл «build_lisa_records.py» , строка 6, в <module>
        from pyimagesearch.utils.tfannotation import TFAnnotation File "/home/
        adrian/.virtualenvs/dl4cv/lib/python3.4/site-packages/pyimagesearch/utils/
        tfannotation.py", строка 2, в <module>
            из object_detection.utils.dataset_util импортировать bytes_list_feature
ImportError: нет модуля с именем «object_detection»
```

Здесь мы пытаемся выполнить build_lisa_records.py , который пытается импортировать функции из подмодуля object_detection API TFOD; однако, поскольку наш PYTHONPATH не установлен должным образом, импорт завершается ошибкой.

В этом случае экспортируйте PYTHONPATH либо с помощью сценария setup.sh , либо вручную через вашу оболочку:

```
$ source setup.sh $
python build_lisa_records.py [INFO]
обрабатывает «{train}» ...
[INFO] обработка '{test}'...
```

Оттуда Python сможет найти библиотеки TFOD API.

Очистите свой «тренировочный»

каталог При обучении своих собственных сетей я столкнулся со странной ошибкой, из-за которой мое обучение сети резко замедлялось, если я время от времени не удалял все файлы в моем учебном каталоге, кроме файлов fast_rcnn_resnet101_coco_2018_01_28.tar.gz и fast_rcnn_lisa.config.

Я не совсем уверен, почему это так, но если вы обнаружите, что ваши сети тренируются медленнее или ваш результаты не кажутся правильными, остановите обучение, а затем очистите каталог обучения:

```
$ cd lisa/experiments/training/ $ rm
checkpoint graph.pbtxt pipeline.config $ rm events./*
model.* $ rm -rf faster_rcnn_resnet101_coco_2018_01_28
$ tar -zxf faster_rcnn_resnet101_coco_2018_01_28.tar.gz $ ls
-faster_rcnn_lisa.config faster_rcnn_resnet101_coco_2018_01_28
faster_rcnn_resnet101_coco_2018_01_28.tar .gz
```

Здесь вы можете видеть, что я удалил все контрольные точки (включая оригинальную модель Faster R-CNN без архива , предварительно обученную на COCO) и файлы событий. Оттуда возобновите тренировку. Опять же, эта ошибка/поведение может быть специфичным для моей машины или моей конкретной настройки, но я также хотел поделиться ею с вами.

Мониторинг потерь и mAP

При обучении ваших собственных детекторов объектов обязательно измерьте свои значения потерь и mAP (mAP будет вычисляться только в том случае, если вы запускаете скрипт eval.py).

МАД должно начинаться с низкого уровня, быстро повышаться в течение первых 10 000–50 000 шагов (в зависимости от того, как долго вы тренируетесь), а затем выравниваться. В это время ваши потери также должны уменьшиться. Ваша цель здесь — снизить потери как можно ниже, в идеале < 1,5. Потеря < 1 желательна при обучении ваших собственных детекторов объектов.

Обучение и оценка с помощью GPU и CPU

Обучение сети с помощью API TFOD можно выполнять с использованием нескольких GPU, но на момент написания этой статьи рекомендуется использовать один GPU, пока вы не освоитесь с инструментами API TFOD. При запуске сценария оценки вы также должны использовать один графический процессор (если он доступен).

Если у вас есть только один графический процессор на вашей машине, вы не должны пытаться обучать и оценивать его на тот же GPU — вы, скорее всего, столкнетесь с ошибкой исчерпания памяти, и ваши скрипты завершатся.

Вместо этого вы должны запустить процесс обучения на своем графическом процессоре, а затем запустить оценку на своем процессоре. Загвоздка здесь в том, что оценка с помощью вашего процессора будет невероятно медленной; поэтому вы должны установить num_examples в очень маленькое число (например, 5-10) внутри eval_config конфигурации вашего конвейера:

```

130 eval_config:
131 { num_examples: 5 #
132   Примечание. Следующая строка ограничивает процесс оценки до 10 оценок.
133   # Удалите строку ниже, чтобы оценивать бесконечно. #max_evals:
134   10
135 }
```

Это обновление гарантирует, что для оценки будет использоваться небольшая часть вашего оценочного набора данных (что позволит вам вычислить приблизительное значение mAP), а не весь тестовый набор (что заняло бы много времени на одном процессоре).

Постарайтесь максимально снизить нагрузку на ЦП, чтобы вы могли использовать эти циклы для перемещения данных. Включать и выключать GPU, тем самым сокращая время, необходимое для обучения сети.

Забыв настроить видимые устройства CUDA

Продолжая раздел 16.3.8 выше, я обнаружил, что некоторые читатели знают, что им нужно использовать графический процессор для обучения и ЦП (если нет доступных дополнительных графических процессоров) для оценки, но не знаю, как это сделать.

Читая документацию TensorFlow (или почти любую документацию по библиотекам глубокого обучения), вы столкнетесь с переменной среды CUDA_VISIBLE_DEVICES. По умолчанию TensorFlow выделяет вашу модель графическим процессорам на вашем компьютере при запуске скрипта, но для читателей, плохо знакомых с TensorFlow, несколько сбивает с толку то, что библиотека не будет обучать сеть на всех графических процессорах — она будет выделять память только на каждом из графических процессоров. Такое поведение проблематично, если вы хотите, чтобы сценарий выполнялся на одном графическом процессоре или вообще без графического процессора.

На моей машине есть четыре графических процессора Titan X, которые по умолчанию TensorFlow попытается выделить память на всех этих четырех графических процессорах. Чтобы предотвратить это нежелательное распределение, я могу установить свой CUDA_VISIBLE_DEVICES перед выполнением скрипта:

```
$ export CUDA_VISIBLE_DEVICES="0" $
python object_detection/train.py --logtostderr \
    ssds_and_rcnn/lisa/experiments/training/faster_rcnn_lisa.config \
    --train_dir ssds_and_rcnn/
lisa/experiments/training
...
```

Здесь вы можете видеть, что я указываю моему текущему сеансу оболочки выставлять только GPU 0 для ТензорФлоу. Оттуда при запуске train.py будет выделен только GPU 0.

Затем я могу открыть второй терминал и настроить CUDA_VISIBLE_DEVICES для отображения только GPU 1:

```
$ export CUDA_VISIBLE_DEVICES="1" $  
python object_detection/eval.py --logtostderr \--  
    pipeline_config_path ssds_and_rcnn/lisa/experiments/training/faster_rcnn_lisa.config \--checkpoint_dir  
    ssds_and_rcnn/lisa/experiments/training \--eval_dir ssds_and_rcnn/lisa /эксперименты/оценка  
  
...
```

Здесь для TensorFlow виден только GPU 1, поэтому TensorFlow будет использовать только GPU 1 для eval.py — таким образом я могу выделить определенные графические процессоры для определенных скриптов.

Конечно, если вы не хотите, чтобы TensorFlow использовал GPU для скрипта, типичное поведение, когда ваша машина имеет только один GPU и этот GPU используется для обучения, вы можете установить CUDA_VISIBLE_DEVICES пустым:

```
$ export CUDA_VISIBLE_DEVICES="" $  
python object_detection/eval.py --logtostderr \--  
    pipeline_config_path ssds_and_rcnn/lisa/experiments/training/faster_rcnn_lisa.config \--checkpoint_dir  
    ssds_and_rcnn/lisa/experiments/training \--eval_dir ssds_and_rcnn/lisa /эксперименты/оценка  
  
...
```

Приведенная выше команда сообщит TensorFlow, что нет графических процессоров (подразумевается пустой строкой в команде экспорта) и что при выполнении eval.py вместо этого следует использовать ЦП.

Параметры оптимизатора

Глядя на конфигурацию нашего конвейера, вы увидите, что мы не возились ни с одним из параметров оптимизатора. В случае Faster R-CNN + ResNet-101 файл конфигурации по умолчанию использует оптимизатор импульса SGD + Нестерова со скоростью обучения 0,0003 — вы должны сохранять эти параметры оптимизатора как есть, пока вы не освоитесь с набором инструментов и в идеале воспроизвели результаты этой главы несколько раз.

Кроме того, обязательно прочитайте комментарии в конфигурации конвейера от сообщества TensorFlow, так как они также дают вам рекомендуемые значения для параметров. Полный обзор всех возможных параметров оптимизатора в TFOD API выходит далеко за рамки этой книги, но если вы прочитали Starter Bundle и Practitioner Bundle, вы обнаружите, что параметры, используемые в TFOD API, близко соответствуют Keras и mxnet.. Обязательно обратитесь к документации TFOD API по вопросам, касающимся этих параметров: <http://pyimg.co/pml2d> [45].

Уважайте инструментарий

После изучения этого раздела вы увидите ряд рекомендаций и лучших практик, которые я предлагаю, но ни один из них не является таким важным, как раздел 16.3.8 выше — есть кривая обучения, и вам нужно посвятить время и энергии, чтобы подняться на нее.

TFOD API — чрезвычайно мощный инструмент, и как только вы к нему привыкнете, вы сможете быстро и легко обучать свои собственные детекторы объектов на основе глубокого обучения. Но чтобы добраться до этого момента, вам нужно набраться терпения.

Неизбежно вы столкнетесь с ошибкой. Потратите время, чтобы прочитать его, переварить и даже потратить некоторое время на поиск ошибки в Google и чтение документации TensorFlow. Не ждите «быстрого решения», это ожидание, как правило, нереалистично.

Тем не менее, чаще всего вы обнаружите, что ваша ошибка связана с:

1. Забыли экспортить PYTHONPATH 2.
- Опечатка в пути к файлу в файле конфигурации
3. Проблема с созданием ваших файлов записей, например недопустимые ограничивающие рамки уважаю тебя.

16.3.9 Экспорт графика замороженной модели

Теперь, когда наша модель обучена, мы можем использовать export_inference_graph.py для создания модели Tensor Flow, которую мы можем импортировать в наши собственные скрипты. Чтобы экспортить свою модель, сначала измените каталог на каталог models/research на своем компьютере, а затем выполните export_inference_graph.py:

```
$ cd ~/models/research $  
python object_detection/export_inference_graph.py --input_type  
image_tensor \ --pipeline_config_path ~ssds_and_rcnn/lisa/  
experiments/training/faster_rcnn_lisa.config \ --trained_checkpoint_prefix ~ssds_and_rcnn/lisa/experiments/  
training/model.ckpt-50000 \ --output ~/ssds_and_rcnn/lisa/experiments/exported_model
```

Здесь вы можете видеть, что я экспортировал контрольную точку модели на шаге 50 000 в файл exported_model . каталог. Проверяя содержимое exported_model , вы можете увидеть файлы, сгенерированные TensorFlow:

```
$ ls lisa/experiments/exported_model/  
контрольная точка Frozen_inference_graph.pb model.ckpt.data-00000-of-00001  
model.ckpt.index model.ckpt.meta save_model
```

Обязательно сохраните каталог exported_model и файлы внутри него — нам нужно будет импортировать эти файлы в наш скрипт Python в следующем разделе.

16.3.10 Быстрее R-CNN на изображениях и видео

Мы обучили наш Faster R-CNN. Мы оценили его точность. Но нам еще предстоит программно применить ее к входному изображению — как нам применить нашу сеть к входному изображению за пределами набора данных, на котором она обучалась? Чтобы ответить на этот вопрос, откройте predict.py и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2  
из object_detection.utils import label_map_util 3 import tensorflow  
as tf 4 import numpy as np 5 import argparse 6 import imutils 7  
import cv2  
  
8  
9 # создаем аргумент parse и анализируем аргументы 10 ap =  
argparse.ArgumentParser() 11 ap.add_argument("-m", "--model",  
required=True, help="базовый путь для графа обнаружения зависящих  
12     контрольных точек")  
13 ap.add_argument("-l", "--labels", required=True, help="файл  
14     меток") 15 ap.add_argument("-i", "--image", required=True,  
16     help="путь к входному изображению")
```

```

17 ap.add_argument("-n", "--num-classes", type=int, required=True,
18                 help="количество меток классов")
19 ap.add_argument("-c", "--min-confidence", type=float, default=0.5,
20                 help="минимальная вероятность, используемая для фильтрации слабых обнаружений")
21 аргумент = варс (ap.parse_args())

```

Строки 2-8 импортируют наши библиотеки Python. label_map_util TensorFlow — вспомогательная функция это позволит нам легко загрузить наш файл меток классов с диска.

Затем мы анализируем аргументы командной строки в строках 10-21. Нам нужны четыре командные строки аргументы, за которыми следует один необязательный:

1. --model: Здесь мы указываем путь к нашему графу обнаружения замороженных контрольных точек (т.е. Сама модель TensorFlow).
2. --labels: метки не встроены в модель TensorFlow, поэтому нам нужно указать путь к нашему файлу class.pbtxt.
3. --image: этот переключатель управляет путем к нашему входному изображению, к которому мы хотим применить объект обнаружение к.
4. --num-classes: К сожалению, TensorFlow не может автоматически определять количество классы из файла --labels, поэтому нам также нужно вручную указать это число.
5. --min-confidence (необязательно): минимальная вероятность, используемая для фильтрации слабых обнаружений.

Теперь, когда наши аргументы командной строки проанализированы, мы можем загрузить модель с диска:

```

23 # инициализируем набор цветов для меток нашего класса
24 ЦВЕТА = np.random.uniform(0, 255, size=(args["num_classes"], 3))
25
26 # инициализируем модель
27 модель = tf.Graph()
28
29 # создайте контекстный менеджер, который сделает эту модель моделью по умолчанию для
30 # исполнение
31 с помощью модель.as_default():
32     # инициализируем определение графа
33     graphDef = tf.GraphDef()
34
35     # загружаем график с диска
36     c tf.gfile.GFile(args["model"], "rb") как f:
37         сериализованныйграф = f.read()
38         graphDef.ParseFromString (сериализованный график)
39         tf.import_graph_def (graphDef, имя = "")

```

Строка 24 случайным образом инициализирует набор цветов RGB для каждой ограничивающей рамки. Случайная инициализация сделана из соображений удобства — вы можете изменить этот скрипт, чтобы использовать фиксированные цвета для каждой метки, как хорошо.

Строка 27 инициализирует модель, которую мы будем загружать с диска. Строки 31-39, затем загрузите сериализованная сеть TensorFlow с использованием вспомогательных утилит TensorFlow.

Давайте также загрузим наши метки классов с диска:

```

41 # загрузить метки классов с диска
42 labelMap = label_map_util.load_labelmap(аргументы["метки"])
43 категории = label_map_util.convert_label_map_to_categories(
44     labelMap, max_num_classes=args["num_classes"],
45     use_display_name = Истина)
46 categoryIdx = label_map_util.create_category_index(категории)

```

Строка 42 загружает необработанный файл pbtxt с диска. Затем мы можем использовать convert_label_map_to_categories вместе с нашим переключателем --numclasses для создания набора категорий. Стока 46 создает отображение из целочисленного идентификатора метки класса (т.е. что TensorFlow вернет при прогнозировании) к удобочитаемой метке класса.

Чтобы предсказать ограничивающие рамки для нашего входного изображения, нам сначала нужно создать TensorFlow. сеанс и захват ссылок на каждый из тензоров изображения, ограничивающей рамки, вероятности и классов внутри сети:

```

48 # создать сеанс для выполнения логического вывода
49 с помощью model.as_default():

50     c tf.Session(graph=model) как sess:
51         # получить ссылку на тензор входного изображения и поля
52         # тензор
53         imageTensor = model.get_tensor_by_name("image_tensor:0")
54         boxTensor = model.get_tensor_by_name("detection_boxes:0")

55         # для каждой ограничивающей рамки мы хотели бы знать счет
56         # (т.е. вероятность) и метка класса
57         scoresTensor = model.get_tensor_by_name("detection_scores:0")
58         classesTensor = model.get_tensor_by_name("detection_classes:0")
59         numDetections = model.get_tensor_by_name («num_detections: 0» )

```

Эти ссылки позволяют нам получить доступ к связанным с ними значениям после передачи изображения через сеть.

Далее загрузим наш образ с диска и подготовим его к детектированию:

```

62     # загружаем образ с диска
63     изображение = cv2.imread (аргументы [ "изображение" ])
64     (В, Ш) = изображение.форма[:2]

65
66     # проверяем, должны ли мы изменять размер по ширине
67     если Ш > В и Ш > 1000:
68         изображение = imutils.resize (изображение, ширина = 1000)

69
70     # в противном случае проверяем, должны ли мы изменять размер вдоль
71     # высота
72     elif H > W и H > 1000:
73         изображение = imutils.resize (изображение, высота = 1000)

74
75     # подготавливаем изображение к обнаружению
76     (В, Ш) = изображение.форма[:2]
77     вывод = изображение.копировать()
78     изображение = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2RGB)
79     изображение = np.expand_dims (изображение, ось = 0)

```

Этот блок кода должен быть похож на то, что мы делали ранее с Keras и mxnet. Мы начните с загрузки изображения с диска, захватите его размеры и измените его размер так, чтобы самый большой размер не более 1000 пикселей (строки 63-73). Вы можете изменить размер изображения как больше, так и меньше, как вы считаете нужным, но 300-1000 пикселей — хорошая отправная точка. Строки 75-79 затем подготавливают изображение для обнаружения.

Получение ограничивающих рамок, вероятностей и меток классов так же просто, как вызов .run метод сеанса:

```

81      # выполнить вывод и вычислить ограничивающие рамки,
82      # вероятности и метки классов
83      (ящики, баллы, метки, N) = sess.run(
84          [boxesTensor, scoresTensor, classTensor, numDetections],
85          feed_dict={imageTensor: изображение})

86
87      # сжать списки в одно измерение
88      коробки = np.squeeze(коробки)
89      баллы = np.squeeze(баллы)
90      метки = np.squeeze(метки)

```

Здесь мы передаем наш список ограничивающей рамки, баллов (то есть вероятностей), меток классов и числа тензоров обнаружений в метод sess.run . feed_dict инструктирует TensorFlow установить imageTensor к нашему изображению и запускаем прямой проход, получая наши ограничивающие рамки, оценки (т.е. вероятности) и метки классов.

Ящики , баллы и метки — это многомерные массивы, поэтому мы сжимаем их до одномерного изображения. массив, что позволяет нам легко перебирать их.

 Если вы не знакомы с методом сжатия NumPy и с тем, как он работает, обязательно обратитесь к документации: <http://pyimg.co/fnbgd>



Рисунок 16.5: Примеры применения нашего Faster R-CNN для обнаружения и маркировки присутствия трафика в США знаки на изображениях.

Последний шаг — зациклить наши обнаружения, нарисовать их на выходном изображении и отобразить результат на наш экран:

```

92         # цикл по предсказаниям ограничивающей рамки
93         for (box, score, label) в zip(box, scores, labels):
94             # если прогнозируемая вероятность меньше минимума
95             # уверенность, игнорируй
96             если оценка < args["min_confidence"]:
97                 Продолжать
98
99             # масштабируем ограничивающую рамку от диапазона [0, 1] до [W, H]
100            (startY, startX, endY, endX) = поле
101            старtX = интервал (началоЧ * W)
102            начало Y = интервал (начало Y * H)
103            конецХ = интервал (конецХ * W)
104            конец Y = интервал (конец Y * H)
105
106            # рисуем предсказание на выходном изображении
107            метка = Idx_категории[метка]
108            idx = интервал (метка ["id"]) - 1
109            label = "{}: {:.2f}".format(label["name"], оценка)
110            cv2.rectangle (выход, (startX, startY), (endX, endY),
111                           ЦВЕТА[idx], 2)
112            y = startY - 10, если startY - 10 > 10 , иначе startY + 10
113            cv2.putText (выход, метка, (startX, y),
114                         cv2.FONT_HERSHEY_SIMPLEX, 0.3, ЦВЕТА[idx], 1)
115
116            # показать выходное изображение
117            cv2.imshow ("Выход", выход)
118            cv2.waitKey(0)

```

В строке 93 мы начинаем перебирать каждую из отдельных ограничивающих рамок, оценок и прогнозируемых значений. этикетки. Если оценка ниже минимальной достоверности, мы игнорируем ее, тем самым отфильтровывая слабые предсказания.

Имейте в виду, что API TFOD требует, чтобы наши ограничивающие рамки находились в диапазоне [0,1] — чтобы нарисовать их на нашем изображении, нам сначала нужно вызвать их обратно в диапазон [0,W] и [0,H], соответственно (строки 100-104).

Строки 107-114 обрабатывают зацикливание удобочитаемой метки в словаре categoryIdx . а затем рисование метки и связанной с ней вероятности на изображении. Наконец, строки 117 и 118 отображать выходное изображение на наш экран.

Чтобы выполнить наш сценарий predict.py , откройте терминал и выполните следующую команду: убедившись, что вы указали правильный путь к входному изображению:

```
$ питон предсказать.py \
--model lisa/experiments/exported_model/frozen_inference_graph.pb \
--labels lisa/records/classes.pbtxt \
--image путь/k/input/image.png \
--num-классы 3
```

Я запустил predict.py для нескольких примеров изображений из тестового набора LISA. А монтаж этих обнаружений можно увидеть на рис. 16.5.

16.4 Резюме

В этой главе мы узнали, как использовать API обнаружения объектов TensorFlow для обучения Faster R-CNN. + Архитектура ResNet-101 для обнаружения наличия дорожных знаков на изображениях. В целом, мы смогли

тренируйте очень высокое 98% mAP @ 0,5 IoU после в общей сложности 50 000 шагов.

Обычно вы хотите обучить свои сети как минимум 20 000 шагов — выше.

точность может быть получена с использованием 50 000-200 000 шагов в зависимости от набора данных.

После того, как мы обучили нашу Faster R-CNN с помощью TFOD API, мы создали скрипт Python, чтобы применить нашу сеть к нашим собственным входным изображениям. Этот же скрипт можно использовать в ваших собственных проектах. Пример сценария, используемого для применения моделей к входным видеопотокам/файлам, можно найти в файлах для загрузки, связанных с этой книгой.

В следующей главе мы продолжим тему глубокого обучения беспилотных автомобилей, обучив SSD для обнаружения переднего и заднего вида транспортных средств.

17. Детекторы одиночного выстрела (SSD)

В наших предыдущих двух главах мы обсудили структуру Faster R-CNN, а затем обучили Faster R-CNN обнаруживать дорожные знаки на изображениях. Хотя более быстрые R-CNN позволили нам достичь нашей цели, мы обнаружили две проблемы, которые необходимо было решить:

1. Структура сложная, включает несколько движущихся частей 2. Мы достигли примерно 7–10 кадров в секунду — приемлемо для объекта, основанного на глубоком обучении. детектор, но не в том случае, если нам нужна истинная производительность в реальном времени.

Чтобы решить эти проблемы, мы рассмотрим структуру Single Shot Detector (SSD) для обнаружения объектов. Детектор объектов SSD является полностью сквозным, не содержит сложных движущихся частей и способен работать в сверхреальном времени.

17.1 Общие сведения об однократных детекторах (SSD)

Фреймворк SSD был впервые представлен Liu et al. в своей статье 2015 года SSD: Single Shot MultiBox Detector [61]. Компонент Multibox, используемый для алгоритма регрессии ограничивающей рамки, взят из статьи Szegedy (тот же Кристиан Сегеди из сети Google Inception) и др., опубликованной в 2015 году, Масштабируемое обнаружение объектов высокого качества [62].

Объединив метод регрессии Multibox с платформой SSD, мы можем создать детектор объектов, который достигает точности, сравнимой с Faster R-CNN, и обеспечивает до 59+ кадров в секунду — более быстрый FPS можно получить и с использованием более мелких и эффективных базовых сетей.

В первой части этого раздела мы обсудим некоторые мотивы использования твердотельных накопителей, причины их создания и проблемы, которые они решают в контексте обнаружения объектов. Оттуда мы рассмотрим архитектуру SSD, включая концепцию MultiBox Priors. Наконец, мы обсудим, как обучаются SSD.

17.1.1 Мотивация

Работа Girchick et al. в их трех публикациях R-CNN [43, 46, 51] действительно замечательны и позволили обнаружению объектов на основе глубокого обучения стать реальностью. Однако есть несколько проблем, которые исследователи и практики обнаружили с R-CNN.

Во-первых, обучение требовало нескольких этапов. Сеть предложений по регионам (RPN) необходимо было обучить, чтобы генерировать предлагаемые ограничивающие рамки, прежде чем мы сможем обучить фактический классификатор распознавать объекты на изображениях. Позже проблема была смягчена путем сквозного обучения всей архитектуры R-CNN, но до этого открытия был введен утомительный процесс предварительного обучения.

Вторая проблема заключается в том, что обучение заняло слишком много времени. (Быстрее) R-CNN состоит из нескольких компонентов, в том числе: 1.

Сеть региональных предложений 2.

Модуль объединения ROI 3.

Окончательный классификатор

Несмотря на то, что все три компонента объединены в одну структуру, они по-прежнему являются движущимися частями, которые замедляют работу. весь процесс обучения.

Последняя проблема, и, возможно, самая важная, заключается в том, что время вывода было слишком медленным — мы могли еще не получить обнаружение объектов в реальном времени с помощью глубокого обучения.

На самом деле мы можем увидеть, как SSD пытаются решить каждую из этих проблем, изучая само имя . Термин Single Shot подразумевает, что и локализация, и обнаружение выполняются за один прямой проход сети в течение времени вывода — сети достаточно «посмотреть» на изображение только один раз, чтобы получить окончательные прогнозы.

Важно понимать значение термина одиночный выстрел. В отличие от R-CNN, которые требуют повторной выборки пикселей из исходного изображения или фрагментов из карты объектов, твердотельные накопители вместо этого продолжают распространять карты объектов вперед, соединяя карты объектов новым способом, чтобы можно было обнаруживать объекты различных размеров и масштабов. Как мы увидим, и согласно Liu et al., фундаментальное улучшение скорости твердотельных накопителей происходит за счет устранения предложений ограничивающей рамки и субдискретизации пикселей или функций [61].

Термин Multibox относится к оригинальному алгоритму Multibox Сегеди и др., который использовался для регрессии ограничивающей рамки [62]. Этот алгоритм позволяет твердотельным накопителям локализовать объекты разных классов, даже если их ограничивающие рамки перекрываются. Во многих алгоритмах обнаружения объектов перекрывающиеся объекты разных классов часто поддавляются в единую ограничивающую рамку с наивысшей достоверностью.

Наконец, термин « детектор » подразумевает, что мы будем не только локализовать (x, y)-координаты набора объектов на изображении, но и возвращать их метки классов.

17.1.2 Архитектура

Как и Faster R-CNN, SSD начинается с базовой сети. Эта сеть обычно предварительно обучена, как правило, на большом наборе данных, таком как ImageNet, что позволяет ей изучить богатый набор отличительных признаков. Опять же, мы будем использовать эту сеть для передачи обучения, распространения входного изображения на заранее заданный слой, получения карты объектов, а затем перехода к слоям обнаружения объектов.

В работе Liu et al. использовалась VGG16 [17], так как на момент публикации было обнаружено, что VGG обеспечивает лучшую точность/результаты обучения переносу, чем другие популярные сетевые архитектуры. Сегодня вместо этого мы будем использовать более глубокую архитектуру ResNet [21, 22] или DenseNet [55] для получения более высокой точности, или мы можем заменить SqueezeNet [30] или MobileNet [53] для дополнительной скорости. Для объяснения структуры SSD в этом разделе мы будем использовать VGG в качестве базовой сети.

Рисунок 17.1 иллюстрирует архитектуру SSD. Мы используем слои VGG вплоть до conv_6, а затем отсоединяем все остальные слои, включая полно связные слои. Затем в архитектуру добавляется набор новых слоев CONV — это уровни, которые делают возможной структуру SSD. Как видно из схемы, каждый из этих слоев также является слоем CONV. Такое поведение подразумевает, что наша сеть является полностью сверточной: мы можем принимать входное изображение произвольного размера — мы больше не ограничены входными требованиями VGG 224× 224.

Мы рассмотрим, что делают эти дополнительные вспомогательные слои позже в этой главе, но для

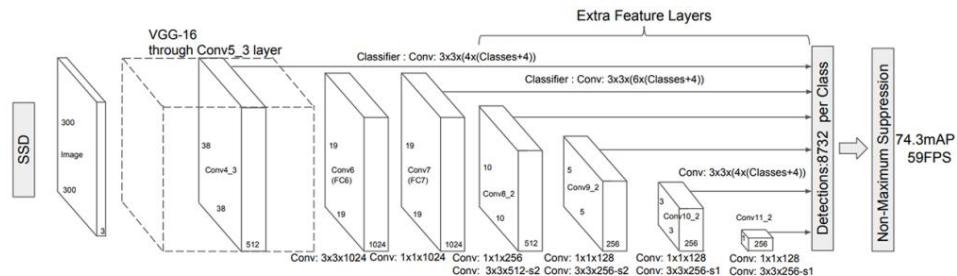


Рисунок 17.1: Схема архитектуры Single Shot Detector (SSD). SSD начинается с базовой сети (обычно предварительно обученной в ImageNet). Набор новых слоев CONV используется для замены более поздних слоев CONV и POOL . Каждый слой CONV соединяется с выходным слоем FC . В сочетании с (модифицированным) алгоритмом Multibox это позволяет твердотельным накопителям обнаруживать объекты разного масштаба на изображении за один прямой проход. (Источник: рисунок 2 Луи и др. [61])

пока обратите внимание на две важные составляющие:

1. Мы постепенно уменьшаем размер тома в более глубоких слоях, как в стандартной CNN . 2. Каждый из слоев CONV соединяется с последним слоем обнаружения.

Важен тот факт, что каждая карта объектов подключается к последнему слою обнаружения — это позволяет сети обнаруживать и локализовать объекты на изображениях в различных масштабах. Кроме того, эта локализация масштаба происходит в прямом проходе. Не требуется повторная выборка карт функций, что позволяет твердотельным накопителям работать в режиме полной прямой связи — именно этот факт делает твердотельные накопители такими быстрыми и эффективными.

17.1.3 Мультибокс, Приоры и Фиксированные Приоры

Фреймворк SSD использует модифицированную версию алгоритма MultiBox Сегеди и др. [62] для предложений ограничительной рамки. Этот алгоритм вдохновлен предыдущей работой Сегеди над сетью Inception — MultiBox использует как (1) серию ядер 1×1 , чтобы помочь уменьшить размерность (с точки зрения ширины и высоты объема), так и (2) серию ядер 3×3 . для более многофункционального обучения.

Алгоритм MultiBox начинается с априорных значений , похожих на якоря из фреймворка Faster R-CNN, хотя и не совсем такие. Априорные значения представляют собой ограничивающие рамки фиксированного размера, размеры которых были предварительно рассчитаны на основе размеров и местоположений наземных ограничивающих рамок для каждого класса в наборе данных.

Мы называем их «априорными» , поскольку полагаемся на байесовский статистический вывод или, точнее, на априорное распределение вероятностей того, где на изображении появятся местоположения объектов. Априоры выбираются таким образом, чтобы их пересечение по объединению (IoU) превышало 50% с объектами истинности земли. Оказывается, этот метод вычисления априорных значений лучше, чем случайный выбор координат из входного изображения; однако проблема заключается в том, что теперь нам нужно предварительно обучить предиктор MultiBox, что подрывает нашу цель сквозного обучения полного детектора объектов на основе глубокого обучения.

К счастью, есть решение: фиксированные априорные значения , и оно включает ту же технику выбора привязки, что и Faster R-CNN.

Чтобы наглядно представить концепцию фиксированных априорных значений, взгляните на рис. 17.2. Слева у нас есть исходное входное изображение с ограничивающими прямоугольниками. Наша цель — создать карты признаков, которые дискретизируют входное изображение на ячейки (среднюю и правую) — этот процесс будет происходить естественным образом по мере того, как изображение проходит через CNN, и выходные пространственные размеры наших объемов становятся все меньше и меньше.

Каждая ячейка на карте объектов, подобно якорю, имеет небольшой набор ограничивающих рамок по умолчанию (четыре из них) с различными соотношениями сторон. Разбивая входное изображение на карты признаков разного размера, мы можем обнаруживать объекты в разных масштабах на изображении.

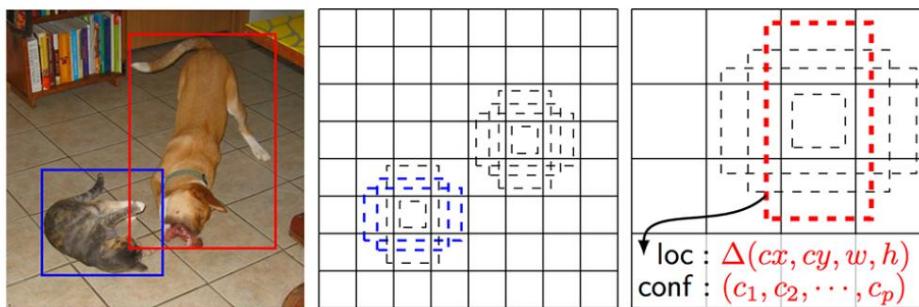


Рисунок 17.2: Слева: Исходное входное изображение с ограничивающими прямоугольниками. Посередине и справа: создание карты объектов, которая дискретизирует входное изображение на ячейки разного размера. Каждая ячейка (аналогично якорю в терминологии Faster R-CNN) имеет набор связанных с ней ограничивающих рамок по умолчанию. Чем меньше количество ячеек, тем крупнее объект, который можно обнаружить. Чем больше количество ячеек, тем меньший объект может быть обнаружен.

(Источник: рисунок 1 Луи и др. [61])

Чтобы лучше понять этот процесс, рассмотрим среднюю карту признаков 8×8 . Здесь наши окружающие априоры ограничительной рамки маленькие, способные локализовать небольшие объекты. Однако на правильной карте признаков 4×4 наши априорные значения ограничивающей рамки больше, что позволяет локализовать более крупные объекты.

В примере из Liu et al. Карта признаков 8×8 , в частности, выделенная синим цветом ограничительная рамка, может использоваться для локализации кошки на изображении, в то время как карта признаков 4×4 с выделенной красным ограничительной рамкой может использоваться для локализации кошки на изображении. локализовать собаку — собака намного больше кошки и поэтому требует дискретной карты признаков с меньшим количеством ячеек.

Этот процесс дискретизации карт объектов (пространственные размеры которых впоследствии уменьшаются позже в CNN) в сочетании с различными соотношениями сторон позволяет нам эффективно локализовать объекты различных масштабов, точек обзора и соотношений.

Кроме того, как и в Faster R-CNN, вместо того, чтобы предсказывать необработанные координаты (x, y) , мы предсказываем смещения ограничивающей рамки (т. е. дельты) для каждой метки класса.

Чтобы прояснить этот момент, рассмотрим ограничивающую рамку, состоящую из двух наборов (x, y) -координат с b фиксированными априорными значениями по умолчанию для каждой ячейки карты объектов, а также с метками общего класса. Если пространственные размеры нашей карты объектов равны $f = M \times N$, то мы должны вычислить в общей сложности значения $t = f \times b \times (4 + c)$ для каждой карты объектов [61, 63].

Опять же, важно иметь в виду, что для каждой предсказанной ограничивающей рамки мы также вычисляем вероятность во всех метках классов внутри региона, а не сохраняем только ограничивающую рамку с наибольшей вероятностью среди всех классов. Вычисление и сохранение вероятности ограничивающих рамок по классам позволяет нам также обнаруживать потенциально перекрывающиеся объекты.

17.1.4 Методы обучения

При обучении SSD нам необходимо учитывать функцию потерь алгоритма MultiBox, которая включает в себя два компонента: 1. Потеря уверенности.

2. Потеря местоположения

Категориальная кросс-энтропийная потеря используется для уверенности, поскольку она измеряет, были ли мы правы в нашем прогнозе метки класса для ограничивающей рамки. Потеря местоположения, как и Faster R-CNN, использует сглаженную потерю L1, что дает SSD больше свободы для близких, но не идеальных локализаций (т. е. окончательная прогнозируемая ограничивающая рамка не обязательно должна быть идеальной, просто «достаточно близко»). Имейте в виду, что предсказать ограничивающие рамки, которые соответствуют действительности, просто нереально.

Перед обучением вашей сети вы также захотите рассмотреть количество ограничивающих рамок по умолчанию.

используется SSD — оригинальная статья Liu et al. рекомендует использовать четыре или шесть ограничивающих рамок по умолчанию. Вы можете настроить эти числа по своему усмотрению, но обычно вы хотите оставить их на рекомендуемых значениях по умолчанию. Добавление дополнительных вариаций масштаба и соотношения сторон может потенциально (но не всегда) позволить вам обнаруживать больше объектов, но также значительно замедлит время вывода .

Та же концепция верна и для количества карт объектов — вы можете включать в сеть дополнительные блоки CONV , тем самым увеличивая глубину, что приводит к увеличению вероятности того , что объект будет правильно обнаружен и классифицирован. Компромисс здесь снова скорость. Чем больше слоев CONV вы добавите, тем медленнее будет работать сеть.

Лю и др. предоставьте обширное исследование компромиссов между добавлением/удалением количества ограничивающих рамок по умолчанию и блоков CONV в своей статье [61] — пожалуйста, обратитесь к ней, если вы заинтересованы в более подробной информации об этих компромиссах.

Фреймворк SSD также включает в себя общую концепцию обнаружения объектов с жесткими отрицательными значениями для повышения точности обучения. В процессе обучения ячейки с низким IoU с наземными объектами рассматриваются как отрицательные примеры.

По определению, большинство объектов изображения будет содержаться в небольшой части изображения. Поэтому возможно, что большая часть изображения будет считаться отрицательными примерами — если бы мы использовали все отрицательные примеры для входного изображения, наши пропорции положительных и отрицательных обучающих примеров были бы крайне несбалансированными.

Чтобы количество отрицательных примеров не превышало количество положительных, Liu et al. рекомендует сохранять соотношение отрицательных и положительных примеров около 3:1. Большинство реализаций SSD , которые вы будете использовать, либо делают эту выборку по умолчанию, либо предоставляют ее в качестве настраиваемого параметра для сети.

В исходной статье для сквозного обучения использовался оптимизатор SGD, но можно использовать и другие оптимизаторы. Как правило, вы увидите RMSprop или Adam, особенно при точной настройке существующей модели обнаружения объектов.

Во время прогнозирования подавление немаксимумов используется по классам, что дает окончательные прогнозы из сети. После обучения Лю и др. продемонстрировали, что SSD получили точность, сравнимую с их аналогами Faster R-CNN, в ряде наборов данных обнаружения объектов, включая PASCAL VOC [58], COCO [59] и ILSVRC [9].

Однако основной вклад здесь заключается в том, что твердотельные накопители могут работать со скоростью примерно 22 кадра в секунду на входных изображениях 512×512 и 59 кадров в секунду на изображениях 300×300 . Еще более высокий FPS можно получить , заменив базовую сеть VGG на более эффективную с вычислительной точки зрения MobileNet (хотя точность может немного снизиться).

17.2 Резюме

В этой главе мы рассмотрели основы структуры Single Shot Detector (SSD).

В отличие от Faster R-CNN, которые содержат несколько движущихся частей и компонентов, твердотельные накопители унифицированы, инкапсулированы в единую сквозную сеть, что упрощает обучение твердотельных накопителей и позволяет обнаруживать объекты в реальном времени, сохраняя при этом сопоставимую точность.

Основная критика твердотельных накопителей заключается в том, что они, как правило, плохо работают с небольшими объектами, главным образом потому, что маленькие объекты могут отображаться не на всех картах объектов — чем больше объект отображается на карте объектов, тем больше вероятность того, что алгоритм MultiBox сможет его обнаружить.

Распространенным способом решения этой проблемы является увеличение размера входного изображения, но это (1) снижает скорость, с которой может работать SSD, и (2) не полностью устраняет проблему обнаружения мелких объектов. Если вы пытаетесь обнаружить объекты, которые малы по сравнению с размером входного изображения, вам следует вместо этого рассмотреть возможность использования Faster R-CNN. Для получения дополнительной информации о твердотельных накопителях см. публикацию Лю и др. [61], а также прекрасную вводную статью Эдди Форсона [63].

Продолжая нашу тему глубокого обучения беспилотных автомобилей, в следующей главе мы узнаем, как с нуля обучить SSD распознавать передний и задний вид транспортных средств с помощью Google Object Detection API.

18. Обучение SSD с нуля

В нашей последней главе мы обсудили внутреннюю работу детектора одиночных выстрелов (SSD) от Liu et al. Теперь, когда у нас есть понимание структуры SSD, мы можем использовать API обнаружения объектов TensorFlow (API TFOD) для обучения SSD на наборе данных, аналогично главе 16, где мы обучали Faster R-CNN с использованием API TFOD.

Мы продолжим тему глубокого обучения для беспилотных автомобилей, обучив SSD распознавать виды спереди и сзади транспортных средств, используя набор данных о транспортных средствах, созданный и помеченный Дэвисом Кингом из dlib [40]. Этот процесс не только дает нам больше возможностей для работы с TFOD API и набором инструментов, но также дает нам дополнительное представление об ограничениях TFOD API и о том, как некоторые типы наборов данных могут быть более сложными для обучения, чем другие, из-за жестких условий. -отрицательный майнинг.

18.1 Набор данных автомобиля

Набор данных, который мы используем для этого примера, взят из библиотеки dlib Дэвиса Кинга [40] и был вручную аннотирован Кингом для использования в демонстрации его алгоритма обнаружения объектов с максимальной маржой [64, 65].

Каждое изображение в этом наборе данных было снято с камеры, установленной на приборной панели автомобиля. Для каждого изображения все видимые виды транспортных средств спереди и сзади помечены соответствующим образом. Пример таких аннотаций транспортных средств в наборе данных можно найти на рис. 18.1.

Изображения были помечены с помощью инструмента imglab , включенного в dlib [66]. Одним из преимуществ использования imglab для аннотаций является то, что мы можем помечать потенциально «запутанные» области изображения как «игнорировать». Если вы используете библиотеку dlib для обучения собственного детектора HOG + Linear SVM или детектора CNN MMOD (с использованием C++), dlib исключит из обучения области, помеченные как игнорируемые.

Примеры областей, которые могут и должны быть помечены как игнорируемые в контексте обнаружения транспортных средств, включают:

1. Районы, где большое количество автомобилей находится на компактной территории, где не понятно, где начинается один автомобиль и заканчивается другой
2. Транспортные средства на расстоянии слишком малы, чтобы их можно было распознать визуально на 100%, но CNN все еще может «видеть» их и изучать по ним закономерности.

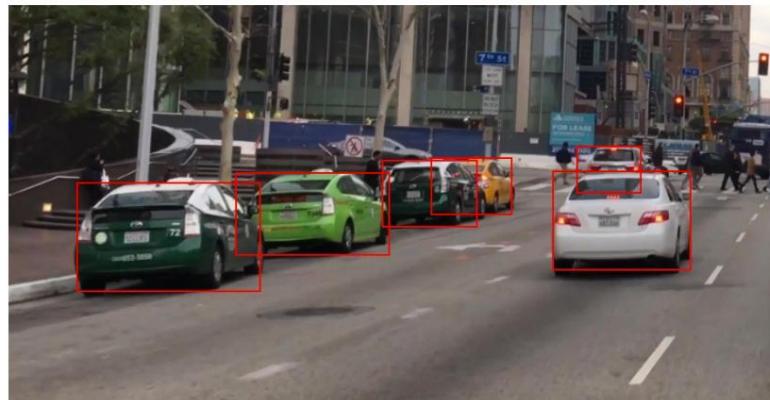


Рисунок 18.1: Пример вида сзади транспортных средств и связанных с ними ограничивающих рамок в dlib

Набор данных о транспортных средствах [67].

К сожалению, в TFOD API нет понятия «игнорируемых» областей изображений, которые могут навредить нашему производительность обнаружения объектов. Цель использования набора данных транспортных средств dlib двояка:

1. Обучить высококачественный детектор объектов SSD локализовать транспортные средства спереди и сзади в картинки
2. Чтобы продемонстрировать, как комбинация (1) жесткого отрицательного майнинга SSD с (2) отсутствием «игнорируемых» области в TFOD API могут сбить с толку наш детектор объектов

К концу этой главы у вас будет четкое понимание обеих целей, что позволит вам принимать разумные решения относительно того, какую структуру обнаружения объектов использовать при обучении собственные детекторы на пользовательских наборах данных.

18.2 Обучение вашего SSD

В этом разделе будет показано, как обучать SSD с помощью TFOD API, аналогично тому, как мы обучали Быстрее R-CNN с использованием TFOD API в главе 16. Поскольку мы уже рассмотрели TFOD API подробно, здесь мы будем тратить меньше времени на обсуждение API и связанных с ним инструментов. Для более подробный обзор API и инструментария TFOD см. в главе 16.

18.2.1 Структура каталогов и конфигурация

Как и в главе 16, где мы обучали Faster R-CNN, у нас будет почти идентичный каталог. структура этого проекта:

```

| --- ssds_and_rcnn
| | --- build_vehicle_records.py
| | --- конфигурация
| | | --- __init__.py
| | | --- dlib_front_rear_config.py
| | --- dlib_front_and_rear_vehicles_v1/
| | | --- image_metadata_stylesheet.xlsx
| | --- input_videos
...
| | --- тестирование.xml
| | --- training.xml
| | --- youtube_frames
| --- прогнозировать.py
| --- прогнозирование_видео.py

```

Мы создадим модуль с именем config , в котором мы будем хранить всю необходимую конфигурацию на основе Python . внутри dlib_front_rear_config.py.

build_vehicle_records.py будет использоваться для создания набора данных транспортных средств в формате записи TensorFlow . Как мы увидим, build_vehicle_records.py очень похож на build_lisa_records.py, только с несколькими модификациями.

И предсказать.py , и предсказать_видео.py идентичны главе 16. Сценарий setup.sh будет использоваться для настройки нашего PYTHONPATH для доступа к импорту и библиотекам TFOD API, опять же, идентично главе 16.

Каталог dlib_front_and_rear_vehicles_v1 содержит набор данных транспортных средств. Вы можете скачать набор данных по следующей ссылке (<http://pyimg.co/9gq7u>) а затем щелкните файл dlib_front_and_rear_vehicles_v1.tar.

Кроме того, вы можете использовать wget и tar для загрузки и разархивирования файлов:

```
$ wget http://dlib.net/files/data/dlib_front_and_rear_vehicles_v1.tar $ tar -xvf  
dlib_front_and_rear_vehicles_v1.tar
```

Внутри dlib_front_and_rear_vehicles_v1 вы найдете файлы изображений, а также два XML-файлы, training.xml и testing.xml:

```
$ cd dlib_front_and_rear_vehicles_v1 $ ls *.xml  
testing.xml training.xml
```

Эти файлы содержат наши аннотации ограничительной рамки + метки классов как для обучающего, так и для тестового набора соответственно. Теперь вам нужно потратить время на создание записей и каталогов данных , как мы это делали в главе 16:

```
$ mkdir записывает эксперименты $  
mkdir эксперименты/тренировочные эксперименты/оценочные эксперименты/exported_model
```

Создание этих каталогов гарантирует, что структура нашего проекта соответствует главе 16, посвященной Faster R-CNN, что позволяет нам повторно использовать большую часть нашего кода и примеры команд.

Откройте файл dlib_front_rear_config.py, и мы просмотрим его содержимое:

```
1 # импортируем необходимые пакеты 2  
импортируем OS  
3  
4 # инициализировать базовый путь для набора данных переднего/заднего автомобиля  
5 BASE_PATH = "dlib_front_and_rear_vehicles_v1"  
6  
7 # строим путь к входным файлам XML для обучения и тестирования 8 TRAIN_XML =  
os.path.sep.join([BASE_PATH, "training.xml"])  
9 TEST_XML = os.path.sep.join([BASE_PATH, "testing.xml"])
```

Строка 5 определяет BASE_PATH для набора данных транспортных средств в нашей системе. Затем мы определяем TRAIN_XML и TEST_XML, пути к файлам обучения и тестирования, предоставленным с набором данных транспортных средств, соответственно.

Если вы откроете один из этих файлов, вы заметите, что это XML-файл. Каждый элемент изображения в файле включает путь к файлу вместе с рядом объектов ограничивающей рамки. Мы узнаем, как анализировать этот XML-файл, в Разделе 18.2.2.

Оттуда мы определяем путь к нашим файлам записей:

```
11 # построить путь к выходным файлам записей обучения и тестирования, 12 #
вместе с файлом меток классов 13 TRAIN_RECORD = os.path.sep.join([BASE_PATH,
"records/training.record"])
14
15 TEST_RECORD = os.path.sep.join([BASE_PATH, "records/
16     testing.record"])
17 CLASSES_FILE = os.path.sep.join([BASE_PATH,
18     "записи/классы.pbtxt"])
```

Наряду со словарем меток классов:

```
20 # инициализируем словарь меток класса
21 КЛАСС = {"тыл": 1, "перед": 2}
```

Опять же, имея в виду, что идентификатор метки 0 зарезервирован для фонового класса (поэтому мы начинаем считать с 1, а не с 0).

18.2.2 Создание набора данных о транспортных

средствах Подавляющее большинство файла build_vehicle_records.py основано на файле build_lisa_records.py из главы 16 — основная модификация заключается в анализе XML-файла набора данных о транспортных средствах, а не CSV-файла, поставляемого с набором данных LISA Traffic Sign.

Существует ряд библиотек, используемых для разбора XML-файлов, но моя любимая — BeautifulSoup. [68]. Если в вашей системе не установлен BeautifulSoup, вы можете установить его через pip:

```
$ pip установить BeautifulSoup4
```

Если вы используете виртуальную среду Python, убедитесь, что вы используете команду workon для доступа к соответствующей виртуальной среде Python перед установкой BeautifulSoup. Я предполагаю, что вы разбираетесь в основах XML-файлов, включая теги и атрибуты — углубленное знание XML-файлов, конечно же, не требуется.

 Если вы никогда раньше не использовали XML-файлы, я предлагаю вам прочитать следующий учебник, который поможет вам освоиться, прежде чем продолжить: <http://pyimg.co/5wzvd>

Давайте начнем — откройте build_vehicle_records.py и вставьте следующий код:

```
1 # импортируем необходимые пакеты 2
из config import dlib_front_rear_config as config 3 из
pyimagesearch.utils.tfannotation import TFAnnotation 4 из bs4 import
BeautifulSoup 5 из PIL import Image 6 из tensorflow as tf 7 из os
```

Строка 2 импортирует наш файл конфигурации, чтобы мы могли получить к нему доступ через сценарий Python. Наш Класс TFAAnnotation (строка 3) позволит нам эффективно создавать точки данных TensorFlow, которые будут быть записаны в наш выходной файл записи. Мы также импортируем BeautifulSoup , чтобы мы могли анализировать наше изображение. пути, ограничивающие рамки и метки классов из наших файлов XML.

Наш следующий блок кода обрабатывает запись CLASSES в файл в формате, подобном JSON/YAML, который TensorFlow ожидает (что опять же идентично главе 16):

9 `def основной():`

```

10     # открываем выходной файл классов
11     f = открыть (config.CLASSES_FILE, "w")
12
13     # цикл по классам
14     для (k, v) в config.CLASSES.items():
15         # создать информацию о классе и записать в файл
16         элемент = ("Элемент {\n"
17             "\tid: " + стр(v) + "\n"
18             "\tname: " + k + "\n"
19             "}\n")
20         f.написать (элемент)
21
22     # закрыть файл выходных классов
23     е.закрыть()
24
25     # инициализируем файлы разделения данных
26     наборы данных = [
27         ("поезд", config.TRAIN_XML, config.TRAIN_RECORD),
28         ("тест", config.TEST_XML, config.TEST_RECORD)
29     ]

```

Строки 26-29 определяют кортеж наборов данных, состоящий из входного XML-файла для обучения/тестирования и выходной файл записи обучения/тестирования. В Главе 16 нам пришлось вручную создавать нашу программу обучения и тестирования. разделены, но для набора данных транспортных средств наши данные предварительно разделены — от нас не требуется дополнительной работы по этот фронт.

Далее, давайте начнем перебирать каждый из тренировочных и тестовых сплитов соответственно:

```

31     # цикл по наборам данных
32     для (dType, inputPath, outputPath) в наборах данных:
33         # сварить суп
34         print("[INFO] обрабатывает '{}'...".format(dType))
35         содержимое = открыть (входной путь).читать ()
36         суп = BeautifulSoup(содержимое, "html.parser")
37
38         # инициализируем модуль записи TensorFlow и инициализируем общий
39         # количество примеров, записанных в файл
40         писатель = tf.python_io.TFRecordWriter (outputPath)
41         всего = 0

```

Строка 35 загружает содержимое текущего inputPath , а строка 36 строит суп , применяя BeautifulSoup для анализа содержимого и построения XML-дерева, которое мы можем легко перемещать в памяти.

Строка 40 создает средство записи , экземпляр TFRecordWriter , который мы можем использовать для записи изображения и ограничивающие рамки в выходной файл записи TensorFlow .

Давайте пройдемся по XML-документу, перебирая все элементы изображения:

```

43     # цикл по всем элементам изображения
44     для изображения в soap.find_all("image"):
45         # загружаем входное изображение с диска как объект TensorFlow
46         p = os.path.sep.join([config.BASE_PATH, изображение["файл"]])
47         закодировано = tf.gfile.GFile(p, "rb").read()
48         закодировано = байты (закодировано)

49
50         # снова загрузить образ с диска, на этот раз как PIL
51         # объект
52         pilImage = Image.open(p)
53         (ш, ч) = pilImage.size[:2]

54
55         # разобрать имя файла и кодировку из входного пути
56         имя_файла = изображение["файл"].split(os.path.sep)[-1]
57         кодировка = имя файла[имя файла.rfind(".") + 1:]

58
59         # инициализируем объект аннотации, используемый для хранения
60         # информации об ограничительной рамке + метки
61         tfAnnot = TFAAnnotation()
62         tfAnnot.image = закодировано
63         tfAnnot.encoding = кодировка
64         tfAnnot.filename = имя файла
65         tfAnnot.width = ш
66         tfAnnot.height = ч

```

Для каждого элемента изображения мы точно определяем атрибут пути к файлу (строка 46). Строки 47 и 48 загружают изображение с диска в закодированном формате TensorFlow. Строки 52 и 53 загружают изображение снова с диска, на этот раз в формате PIL/Pillow, что позволяет нам извлечь размеры изображения.

Строки 56 и 57 извлекают имя файла из пути к изображению, а затем используют имя файла для получить кодировку изображения (например, JPG, PNG и т. д.). Оттуда мы можем инициализировать наш объект tfAnnot. (строки 61-66). Более подробный обзор этого кода см. в Разделе 16.3.3 в Faster

Глава R-CNN.

Каждый элемент изображения в дереве XML также имеет ряд дочерних элементов ограничивающей рамки — давайте теперь переберем все элементы блока для определенного изображения:

```

68     # цикл по всем ограничивающим рамкам, связанным с изображением
69     для коробки в image.find_all("box"):
70         # проверяем, следует ли игнорировать ограничивающую рамку
71         если box.has_attr("игнорировать"):
72             Продолжать

73
74         # извлечь информацию об ограничительной рамке + метку,
75         # убедиться, что все размеры ограничивающей рамки подходят
76         # внутри изображения
77         startX = max(0, float(box["left"]))
78         startY = max(0, float(box["top"]))
79         endX = min(w, float(box["ширина"]) + startX)
80         endY = min(h, float(box["высота"]) + startY)
81         метка = коробка.найти("метка").текст

```

Строка 71 проверяет, установлен ли атрибут ignore для ящика , и если да, то мы не делаем этого . далее обработайте ограничительную рамку. В идеальном мире API TFOD позволил бы нам отмечать определенные

ROI как «игнорировать», как в dlib, тем самым предписывая нашей сети не обучаться в определенных областях Изображение. К сожалению, этот процесс «игнорирования» определенных областей изображения невозможен. с API TFOD.

Строки 77-81 извлекают информацию о ограничивающей рамке и метке класса из элемента `box`. координаты ограничивающей рамки в наших соответствующих файлах XML могут быть отрицательными или больше, чем фактические ширина и высота изображения — мы принимаем особые меры предосторожности, чтобы гарантировать, что мы отсекаем эти значения, используя функции `max` и `min` соответственно.

Итак, почему именно наш набор данных имеет координаты ограничивающей рамки, которые находятся за пределами размеров изображения? Причина кроется в том, как работают библиотека dlib и инструмент imglab.

Библиотека dlib требует, чтобы во время обучения объекты имели одинаковые пропорции ограничивающей рамки — если ограничивающие рамки не имеют одинаковых пропорций, алгоритм выдаст ошибку. Поскольку транспортные средства могут появляться на границах изображения, чтобы сохранить одинаковые пропорции, ограничивающие рамки может фактически выходить за границы изображения. Проблема с соотношением сторон не является проблемой API TFOD, поэтому мы просто обрезаем эти значения.

Затем мы можем масштабировать координаты ограничивающей рамки до диапазона [0,1], что и делает TensorFlow. ожидает:

```

83             # TensorFlow предполагает, что все ограничивающие рамки находятся в
84             # диапазон [0, 1], поэтому нам нужно их масштабировать
85             xMin = началоЧ/w
86             xMакс = конецЧ/ш
87             yMin = пускY/ч
88             yMax = конецY/ч

```

Наш следующий блок кода проверяет, что наши максимальные значения `x` и `y` всегда больше, чем их соответствующие минимумы (и наоборот):

```

90             # из-за ошибок в аннотации возможно
91             # что минимальные значения больше максимальных
92             # значения -- в этом случае считать это ошибкой во время
93             # аннотацию и игнорировать ограничивающую рамку
94             если xMin > xMax или yMin > yMax:
95                 Продолжать
96
97             # аналогично мы можем столкнуться с противоположным случаем
98             # где максимальные значения меньше минимальных
99             # ценности
100            elif xMax < xMin или yMax < yMin:
101                Продолжать

```

Если какая-либо из этих проверок не пройдена, мы игнорируем ограничивающую рамку и предполагаем, что это ошибка во время процесса аннотации.

 Когда я впервые обучил сеть набору данных транспортных средств с помощью TFOD API, я потратил больше дня, пытаясь диагностировать, почему моя сеть выдала ошибку во время процесса обучения. оказывается проблема была связана с ограничивающими рамками, имеющими неверные координаты, где максимальный `x` или значение `y` на самом деле было меньше соответствующего минимума (и наоборот). Применение этих проверки в строках 90-101 отбросили недопустимые ограничивающие рамки (две из них) и разрешили сеть для обучения.

Оттуда мы можем продолжить создание нашего объекта `tfAnnot` и добавить его в наш выходной файл записи:

```

103         # обновить списки ограничивающих рамок + меток
104         tfAnnot.xMins.append(xMin)
105         tfAnnot.xMaxs.append(xMax)
106         tfAnnot.yMins.append(yMin)
107         tfAnnot.yMaxs.append(yMax)
108         tfAnnot.textLabels.append(label.encode("utf8"))
109         tfAnnot.classes.append(config.CLASSES[метка])
110         tfAnnot.difficult.append(0)

111
112         # увеличить общее количество примеров
113         всего += 1

114
115         # кодируем атрибуты точек данных с помощью TensorFlow
116         # вспомогательные функции
117         функции = tf.train.Features(feature=tfAnnot.build())
118         пример = tf.train.Example(функции = функции)

119
120         # добавляем пример в писатель
121         Writer.write(пример.SerializeToString())

```

Наш последний блок кода закрывает объект записи и запускает основной поток выполнения:

```

123     # закрыть модуль записи и распечатать диагностическую информацию в
124     # пользователь
125     писатель.close()
126     print("[INFO] {} примеров, сохраненных для '{}'".format(total,
127         дТип))
128
129 # проверить, должен ли быть запущен основной поток
130 , если __name__ == "__main__":
131     tf.app.run()

```

Чтобы создать набор данных транспортного средства, откройте терминал, убедитесь, что вы выполнили setup.sh для настройте свой PYTHONPATH и выполните следующую команду:

```
$ время python build_vehicle_records.py
[INFO] обрабатывает "поезд"...
[INFO] 6133 примера сохранено для слова "поезд"
[INFO] обработка "теста"...
[INFO] 382 примера сохранено для «теста»

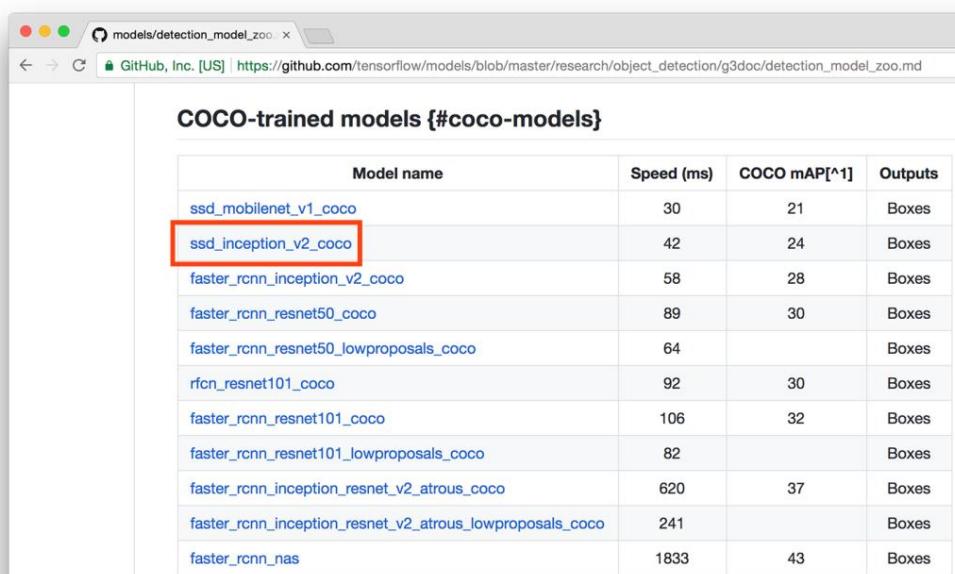
настоящий      0м2.749с
пользователь    0м2.411с
система        0м1.163с
```

Перечисляя содержимое каталога моих записей , мы видим, что наша запись обучения и тестирования файлы были успешно созданы:

```
$ ls dlib_front_and_rear_vehicles_v1/записи/
классы.pbtxt тестирование.запись обучение.запись
```

18.2.3 Обучение SSD

Чтобы обучить наш SSD на наборе данных транспортных средств, нам сначала нужно вернуться в зоопарк модели обнаружения объектов TensorFlow (<http://pyimg.co/1z34r>). и загрузите SSD + Inception v2, обученный на COCO. На рис. 18.2 показан скриншот двух доступных нам SSD: • SSD + MobileNet • SSD + Inception



COCO-trained models {#coco-models}			
Model name	Speed (ms)	COCO mAP[*1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes

Рисунок 18.2: Мы будем обучать SSD с базовой сетью Inception для обнаружения транспортных средств. Убедитесь, что вы загружаете веса модели SSD + Inception, предварительно обученные на COCO, из зоопарка моделей TensorFlow.

В этой главе мы будем использовать архитектуру SSD + Inception, поскольку она обеспечивает более высокую точность обнаружения, но при желании вы можете использовать SSD + MobileNet. Идите вперед и загрузите сеть SSD + Inception, используя ссылку выше, или используйте wget и tar для загрузки и разархивирования архива, который на момент написания этой статьи является версией от 17 ноября 2017 года (ssd_inception_v2_coco_2017_11_17.tar.gz).

```
$ cd dlib_front_and_rear_vehicles_v1/experiments/training $ mv ~/Downloads/ssd_inception_v2_coco_2017_11_17.tar.gz ./ $ tar -xvzf ssd_inception_v2_coco_2017_11_17.tar.gz $ ls -l ssd_inception_v2_coco_2017_11_17 checkpoint frozen_inference_graph.pb model.ckpt.data-00000-of-00001 model .ckpt.index модель.ckpt.meta сохраненная_модель
```

Далее нам нужен наш базовый файл конфигурации. Вернитесь к примерам страниц конфигурации (<http://pyimg.co/r2xql>) и загрузите файл ssd_inception_v2_pets.config.

После того, как вы загрузили файл конфигурации, переместите его в каталог набора данных dlib Vehicles и переименуйте его:

```
$ cd dlib_front_and_rear_vehicles_v1/эксперименты/обучение
$ mv ~/Загрузки/ssd_inception_v2_pets.config ssd_vehicles.config
```

Как видно из приведенных выше команд, я переименовал файл ssd_vehicles.config.

Точно так же, как мы обновили конфигурацию в главе 16 о Faster R-CNN, нам нужно сделать то же самое.

для нашего SSD. Первое изменение состоит в том, чтобы установить num_classes равным 2 , так как у нас есть две метки класса, «передняя» и «задний» вид транспортных средств соответственно:

```
7 модель {
8     передней модели;
9     количество_классов: 2
10    box_coder {
11        более быстрый_rcnn_box_coder {
12            y_шкала: 10,0
13            x_шкала: 10,0
14            высота_шкала: 5,0
15            ширина_шкала: 5,0
16        }
17    }
```

Затем вы захотите установить Fine_tune_checkpoint в train_config так , чтобы он указывал на ваше падение . загруженный SSD + контрольная точка начальной модели:

```
150    Fine_tune_checkpoint: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/
151        dlib_front_and_rear_vehicles_v1/эксперименты/обучение/
152            ssd_inception_v2_coco_2017_11_17/model.ckpt"
153    from_detection_checkpoint: правда
```

Имейте в виду, что ваш путь будет отличаться от моего, поэтому потратьте несколько минут, чтобы подтвердить это. ваш путь правильный — этот шаг избавит вас от головной боли в будущем.

Я позволю своей модели пройти максимум 200 000 шагов; тем не менее, вы могли бы получить примерно такую же точность на 75 000-100 000 шагов:

```
134 train_config: {
135     размер партии: 24
136     ...
137     количество_шагов: 200000
138 ...
139 }
```

Далее мы обновим наш train_input_reader , чтобы он указывал на наши training.record и class.pbtxt . файл:

```
167 train_input_reader: {
168     tf_record_input_reader {
169         input_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/
170             dlib_front_and_rear_vehicles_v1/records/training.record"
171     }
172     label_map_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/
173             dlib_front_and_rear_vehicles_v1/records/classes.pbtxt"
174 }
```

А так же наш eval_input_reader указать на testing.record и class.pbtxt файл:

```

181 eval_input_reader: {
182     tf_record_input_reader {
183         input_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/
184             dlib_front_and_rear_vehicles_v1/records/training.record"
185     }
186     label_map_path: "/home/adrian/pyimagesearch/dlbook/ssds_and_rcnn/
187             dlib_front_and_rear_vehicles_v1/records/classes.pbtxt"
188     перемешивание: ложь
189     число_читателей: 1
190 }
```

Опять же, я бы порекомендовал обновить ваш eval_config , чтобы использовать полные num_examples (при использовании графического процессора) или 5-10 при использовании процессора:

```

174 eval_config: {
175     число_примеров: 382
176     # Примечание. Стока ниже ограничивает процесс оценки 10 оценками.
177     # Удалите строку ниже, чтобы оценивать бесконечно.
178     #max_evals: 10
179 }
```

Если вы правильно указали пути к файлам, все должно быть готово! Откройте новый терминал (обеспечив выполнение setup.sh из каталога ssds_and_rcnn для установки PYTHONPATH) и измените каталог на models/research и выполните следующую команду:

```

$ ИСТОЧНИК setup.sh
$ cd ~/модели/исследования
$ python object_detection/train.py --logtostderr \
    -pipeline ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/
    эксперименты/обучение/ssd_vehicles.config \
    --train_dir ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/
    эксперименты/обучение
ИНФОРМАЦИЯ:tensorflow:Начальные очереди.
ИНФОРМАЦИЯ:тензорный поток:глобальный_шаг/сек: 0
INFO:tensorflow:Сводка записи на шаге 0.
ИНФОРМАЦИЯ: тензорный поток: глобальный шаг 1: потеря = 18,8345 (14,528 сек/шаг)
ИНФОРМАЦИЯ: тензорный поток: глобальный шаг 2: потеря = 17,7939 (0,966 с/шаг)
ИНФОРМАЦИЯ: тензорный поток: глобальный шаг 3: потеря = 17,3796 (0,885 сек/шаг)
...
```

Процесс обучения уже начался, но нам также нужен наш оценочный скрипт.

```

$ python object_detection/eval.py --logtostderr \
    -pipeline_config_path ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/
    эксперименты/обучение/ssd_vehicles.config \
    --checkpoint_dir ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/
    эксперименты/обучение \
    --eval_dir ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/experiments
```

А также наша панель инструментов TensorBoard:

```
$ cd ~/pyimagesearch/dlbook/ssds_and_rcnn/ $ tensorboard
--logdir dlib_front_and_rear_vehicles_v1/experiments TensorBoard 0.4.0rc3 по адресу http://
annalee:6006 (нажмите CTRL+C, чтобы выйти)
```

 Пути к входным файлам и каталогам могут быть довольно длинными, поэтому для удобства я создал символьическую ссылку из каталога `ssds_and_rcnn` в каталог `models/research`, что позволяет мне тратить меньше времени на ввод и отладку длинных путей к файлам. Вы также можете применить эту тактику.

На моей машине я могу визуализировать процесс обучения, открыв веб-браузер и указав в нем адрес `http://annalee:6006`. На вашей машине порт по-прежнему должен быть 6006, но имя хоста (например, `annalee`) будет либо IP-адресом вашей машины, либо именем самой машины.

Опять же, убедитесь, что вы правильно указали источник файла `setup.sh`, чтобы установить `PYTHONPATH`. перед выполнением любой из вышеуказанных команд.

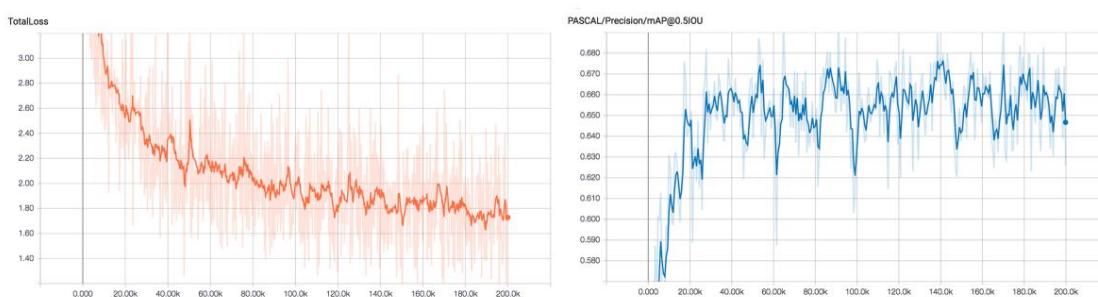


Рисунок 18.3: Слева: общие потери при обучении SSD + Inception на наборе данных dlib Vehicles. Потери начинаются высокими, затем постепенно уменьшаются в течение 200 000 шагов. Дальнейшее обучение дает незначительный выигрыш в точности и потерях. Справа: общее mAP@0,5 для передней и задней этикеток. Мы достигаем точности 65,5%, приличной точности для обнаружения объектов.

На рис. 18.3 показан мой процесс обучения, состоящий из более чем 200 000 шагов, который занял примерно 50 часов на одном графическом процессоре. Потеря закончилась примерно на уровне 1,7 с mAP@0,5 около 65,5%. Дальнейшее обучение лишь незначительно улучшит результаты, если вообще. В идеале я хотел бы видеть убыток ниже 1 и, конечно, ниже 1,5 — мы сохраним обсуждение того, почему мы не можем снизить убыток так низко, позже в разделе 18.2.5.

18.2.4 Результаты SSD

Теперь, когда наш SSD обучен, давайте экспортим его:

```
$ cd ~/models/research $
python object_detection/export_inference_graph.py --input_type image_tensor \
--pipeline_config_path ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/experiments/training/
ssd_vehicles.config \
--trained_checkpoint_prefix ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/
эксперименты/обучение/model.ckpt-200000 \
--output ~/ssds_and_rcnn/dlib_front_and_rear_vehicles_v1/experiments/exported_model
```

Здесь вы можете видеть, что я экспорттировал контрольную точку модели на шаге 200 000 в файл `exported_model`. справочник; однако вы могли бы получить примерно такую же точность на 75 000-100 000 шагов:

Теперь, когда наша модель экспортирована, мы можем применить ее к образцу изображения, убедившись, что вы предоставили допустимый путь к входному изображению:

```
$ python predict.py \--model dlib_front_and_rear_vehicles_v1/experiments/ exported_model/frozen_inference_graph.pb \--labels dlib_front_and_rear_vehicles_v1/records/classes.pbtxt \--image path/to/input/image.png \--num-classes 2
```

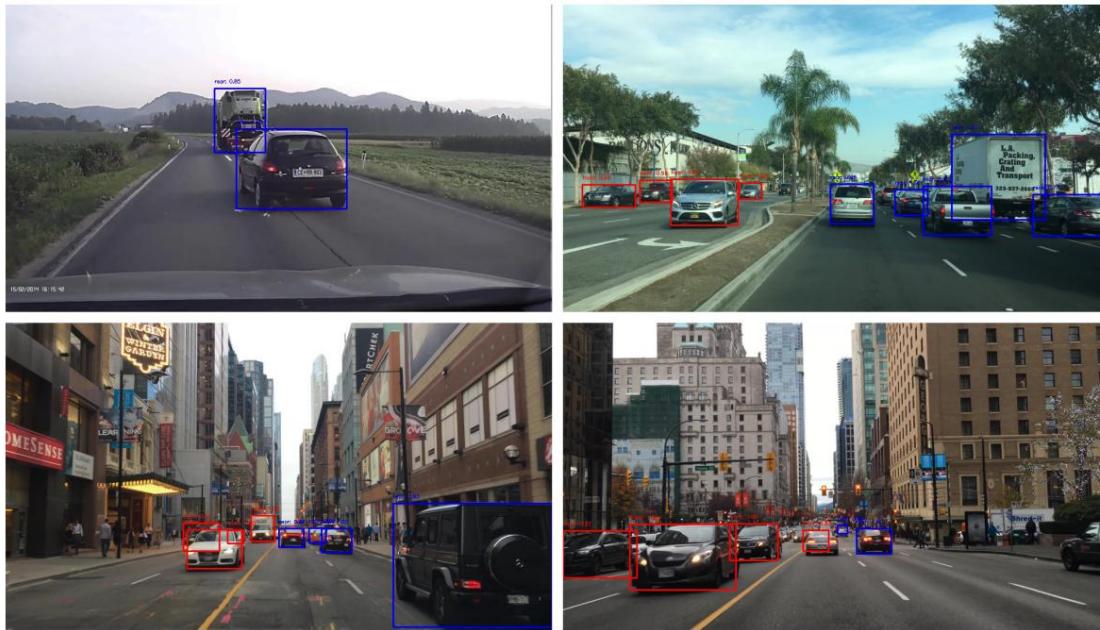


Рисунок 18.4: Примеры нашей сети SSD + Inception, классифицирующей входные тестовые кадры из набора данных транспортных средств dlib. Наша архитектура делает очень хорошую работу по обнаружению + маркировке, за исключением нижнего правого угла, где мы ошибочно классифицируем вид автомобиля сзади как вид спереди. В некоторых случаях, как бы вы ни настраивали свои гиперпараметры, вы не можете преодолеть ограничения вашего набора данных (раздел 18.2.5).

Я запустил predict.py для нескольких примеров изображений из тестового набора dlib Vehicles . На рис. 18.4 показан монтаж этих примеров обнаружения. Здесь мы видим, что наш SSD обычно способен правильно определять и маркировать вид автомобиля «спереди» и «сзади» ; однако в некоторых случаях обнаружение правильное, но метка отключена.

Проблема с нашим детектором? Должны ли мы использовать другой оптимизатор? Нам нужно настроить наши гиперпараметры?

Иногда, независимо от того, как вы настраиваете свои гиперпараметры, вы не можете преодолеть ограничения вашего набора данных и/или среды, которую вы используете для обучения вашего детектора объектов, что является темой нашего следующего раздела.

18.2.5 Возможные проблемы и ограничения

У наших результатов есть два ограничения и недостатки. Первый можно различить, изучив результаты и прочитав главу 17, где мы обсуждали основы детекторов одиночных выстрелов — твердотельные накопители плохо работают с небольшими объектами.

Глядя на вывод наших результатов, вы увидите, что детектор пропускает небольшие транспортные средства. Мы могли бы помочь уменьшить эту проблему, увеличив разрешение наших входных изображений, но если мы действительно хотим обнаруживать небольшие транспортные средства на расстоянии, мы могли бы рассмотреть возможность использования фреймворка Faster R-CNN. Конечно, есть компромисс — если мы будем использовать Faster R-CNN, мы пожертвуем скоростью нашего конвейера обнаружения объектов.

В контексте обнаружения транспортных средств и этого конкретного проекта распознавание небольших транспортных средств на расстоянии вряд ли будет проблемой. Транспортные средства, которые находятся далеко, не являются для нас непосредственной заботой.

По мере их приближения наш SSD сможет распознать их присутствие и дать указание беспилотному транспортному средству предпринять корректирующие действия. Учитывая, что наш SSD может работать со скоростью более 20 кадров в секунду, наши проблемы с распознаванием мелких объектов могут быть смягчены, но нам все еще нужно беспокоиться о транспортных средствах, работающих далеко, на очень высоких скоростях, которые могут быстро сократить разрыв.

Вторая проблема, с которой мы можем столкнуться, — это то, что наш SSD путает виды «спереди» и «сзади» транспортных средств. Эти неправильные классификации частично связаны с тем, что передняя и задняя части транспортных средств, хотя и различаются семантически, по-прежнему имеют много визуально схожих характеристик, но более серьезная проблема заключается в невозможности пометить изображения как «игнорировать» с помощью TFOD API.

Транспортные средства на расстоянии по-прежнему имеют характеристики, подобные транспортным средствам, которые могут сбить с толку TFOD API во время обучения. Эта проблема еще больше усугубляется алгоритмом твердотельного отрицательного майнинга SSD, который пытается изучить негативные шаблоны из ячеек, которые, вероятно, содержат транспортные средства.

Тем не менее, эта проблема не относится к твердотельным накопителям. Я призываю вас вернуться и обучить Faster R-CNN на наборе данных транспортных средств — вы заметите такое же поведение. Если вы обнаружите, что вам нужно пометить области изображений как «игнорировать», а области необходимо игнорировать на большинстве изображений в вашем наборе данных (как в случае с нашим набором данных о транспортных средствах), вы можете рассмотреть возможность использования другого библиотеки обнаружения объектов, чем набор инструментов TFOD.

18.3 Резюме

В этой главе мы узнали, как обучить детектор одиночного выстрела распознавать виды спереди и сзади транспортных средств, используя набор данных Дэвиса Кинга [65]. В целом, мы смогли получить 65,5% mAP в наборе данных — визуальная проверка результатов подтвердила, что прогнозы были хорошиими и даже могли работать в режиме реального времени.

Прогнозирование в реальном времени является одним из основных преимуществ платформы SSD. Зоопарк модели обнаружения объектов TensorFlow сообщает, что наш SSD может работать со скоростью примерно 24 кадра в секунду, в то время как наш более быстрый R-CNN может достигать только 9 кадров в секунду.

Однако обучение нашего SSD на наборе данных транспортных средств выявило некоторые проблемы, с которыми мы можем столкнуться при обучении наших собственных детекторов пользовательских объектов. Поскольку в TFOD API нет понятия «игнорировать эту область изображения», мы не можем исключить потенциально «запутанные» части изображения из процесса обучения. Примерами такой «запутанной» области могут быть:

1. Большое количество машин на компактной территории, где непонятно, где начинается одна машина, а где другая. другие цели
2. Транспортные средства на расстоянии слишком малы, чтобы их можно было 100% визуально распознать, но CNN все еще может «видеть» их и изучать по ним закономерности. спектакль. Я надеюсь, что в будущем API TFOD будет включать метод пометки ограничивающих рамок как «игнорируемых», как это делает инструмент dlib imglab [66], но до этого момента важно знать об этих ограничениях.

19. Выводы

Поздравляем с завершением работы над пакетом ImageNet для глубокого обучения компьютерному зрению с помощью Python! Это было довольно сложное путешествие, и я чувствую себя очень привилегированным и польщенным, что проделал это путешествие с вами. Вы многому научились за последние 15 глав, так что давайте на секунду подытожим ваши новые знания. В этом пакете вы узнали, как:

- Обучайте сети в средах с несколькими GPU.
- Получите и загрузите набор данных ImageNet (вместе с соответствующими предположениями о лицензировании), которые сопровождают набор данных и получившиеся обученные модели.
- Подготовьте набор данных ImageNet, эффективно сжимая изображения и метки классов. упакованные файлы записи.
- Обучайте современные сетевые архитектуры на ImageNet, включая AlexNet, VGGNet, GoogLeNet, ResNet и SqueezeNet, и в каждом случае повторяйте результаты соответствующих авторов. • Внедрите систему «Глубокое обучение как услуга», которую можно использовать для эффективного создания глубоких изучение конечных точек API (и, возможно, использование для создания собственной компании).
- Обучите модель, которая может распознавать выражения лица и эмоции в видеопотоке в реальном времени. • Мы переносим обучение и извлечение признаков для автоматического прогнозирования и исправления ориентации изображения.
- Применяйте тонкую настройку, чтобы распознавать более 164 марок и моделей автомобилей с более чем 96,54% точность.
- Обучить модель глубокого обучения, способную предсказывать возраст и пол человека в фотография.
- Обучить Faster R-CNN и детектор одиночных выстрелов (SSD) для беспилотных автомобилей, включая обнаружение дорожных знаков и обнаружение транспортных средств с видом спереди/сзади.

Каждый из этих проектов был сложным и отражал типы методов, которые вам нужно будет применять при выполнении глубокого обучения в реальном мире. Кроме того, точно такие же методы, лучшие практики и эмпирические правила используются исследователями глубокого обучения, выполняющими современную работу. Если вы проработали все примеры в этой книге, я уверен, что вы можете присвоить себе титул специалиста по глубокому обучению, если не глубокого обучения.

эксперт по обучению. Однако, чтобы сохранить этот статус, вы захотите продолжить учебу . . .

19.1 Куда теперь?

Глубокое обучение — это быстро развивающаяся область, в которой каждый день публикуются новые методы. Быть в курсе всех новых тенденций в исследованиях может показаться сложной задачей, поэтому, чтобы помочь вам, я создал набор ресурсов, которые я лично использую, чтобы быть в курсе последних событий. Эти ресурсы также можно использовать для дальнейшего изучения глубокого обучения. Для начала я очень рекомендую отличный курс cs231n Стэнфордского университета «Сверточные нейронные сети для визуального распознавания» :

<http://cs231n.stanford.edu/>

Каждый весенний семестр проводится новое занятие в классе со слайдами и заметками, размещенными на странице программы: <http://cs231n.stanford.edu/syllabus.html>. Там же можно найти ссылки на заметки предыдущего семестра.

Для более теоретического рассмотрения глубокого обучения Deep Learning Гудфеллоу и др. это должен прочитать:

<http://www.deeplearningbook.org/> Книга

бесплатна для чтения в Интернете и может быть куплена на Amazon здесь:

<http://pyimg.co/v6b1i> Хотя глубокое обучение не относится к компьютерному

зрению, оно предоставляет больше теоретической информации , чем описано в этой практической книге. Кроме того, если вы намерены применять глубокое обучение к областям за пределами компьютерного зрения, вам обязательно нужно прочитать работу Гудфеллоу и др.

Когда дело доходит до последних новостей в области глубокого обучения, я настоятельно рекомендую следить за разделами /r/machinelearning и /r/deeplearning: • <https://www.reddit.com/r/MachineLearning/> • <https://www.reddit.com/r/deeplearning/> Я часто посещаю эти сабреддиты и, как правило, просматриваю большинство сообщений каждые 24-48 часов, добавляя в закладки самые интересные темы, чтобы прочитать их позже.

Преимущество подписки на эти сабреддиты заключается в том, что вы станете частью реального сообщества, с которым сможете обсуждать различные методы, алгоритмы и последние публикации.

Я рекомендую еще два сообщества: LinkedIn Groups, Computer Vision Online и Computer Vision and Pattern Recognition соответственно: • <http://pyimg.co/s1tc3> . • <http://pyimg.co/6er60> Хотя эти группы не относятся к глубокому обучению, они постоянно содержат интересные дискуссии и служат центром для всех, кто интересуется изучением компьютерного зрения на стыке глубокого обучения.

Когда дело доходит до публикаций по глубокому обучению, обратите внимание на Arxiv Sanity Preserver Андрея Карпати: <http://www.arxiv-sanity.com/> Этот веб-сайт позволяет вам фильтровать предварительные публикации глубокого обучения, отправленные в arXiv на основе

по ряду условий, в том числе: •

Последнее представление

- Топ самых последних представленных работ
- Рекомендовать статьи •

Недавно обсуждавшиеся статьи Этот

сайт является бесценным инструментом для всех, кто намеревается проводить исследования в области глубокого обучения.

Когда дело доходит до программного обеспечения, я бы посоветовал использовать репозитории GitHub для обоих Keras. и мкснет:

• <https://github.com/fchollet/keras> • <https://github.com/dmlc/mxnet>

Следуя указанным выше репозиториям, вы будете получать уведомления по электронной почте об обновлении библиотек (и связанных с ними функций). Оставаясь на вершине этих пакетов, вы сможете быстро повторять свои собственные проекты на основе новых функций, которые появятся в процессе разработки. Хотя всегда интересно обсудить теорию, лежащую в основе глубокого обучения, не секрет, что я сторонник фактического внедрения того, что вы изучаете — следование за Keras и mxnet позволит вам улучшить свои навыки внедрения.

Чтобы получать еженедельные новости о глубоком обучении общего назначения, я порекомендуйте This Wild Week in AI, куратор Денни Бритц: <http://www.wildml.com/newsletter/> Я бы также рекомендовал Еженедельник глубокого обучения, составленный Яном Бассом и Матле Бауманном: <http://www.deeplearningweekly.com/> Оба этих информационных бюллетеня содержат контент, который имеет отношение к более широкому сообществу глубокого обучения, но, учитывая привлекательность и интерес глубокого обучения, применяемого к компьютерному зрению, вы увидите контент, связанный с компьютерным зрением, почти в каждом выпуске.

Наконец, следите за обновлениями на PyImageSearch.com. блог и следите за новыми сообщениями. Хотя не все публикации могут быть связаны с глубоким обучением, вы сможете использовать эти учебные пособия, чтобы лучше понять компьютерное зрение и применить эти методы в реальных практических проектах реального мира.

Немного знаний о компьютерном зрении может иметь большое значение при изучении глубокого обучения, поэтому, если вы еще этого не сделали, обязательно ознакомьтесь с практическим Python и OpenCV [34]:

<http://pyimg.co/prao> И

более интенсивный курс PyImageSearch Gurus [35]: <http://pyimg.co/gurus>

Оба этих ресурса позволят вам улучшить свои общие навыки

компьютерного зрения, а в

свою очередь, поможет вам в вашей карьере глубокого обучения.

Еще раз спасибо за то, что позволили мне сопровождать вас на пути к тому, чтобы стать экспертом в области глубокого обучения. Если у вас есть какие-либо вопросы, пожалуйста, напишите мне по адресу adrian@pyimagesearch.com. И несмотря ни на что, продолжайте практиковаться и внедрять — глубокое обучение — это отчасти наука, отчасти искусство. Чем больше вы будете практиковаться, тем лучше вы станете.

Ура, —

Адриан Роузброк

Список используемой литературы

- [1] Дидерик П. Кингма и Джимми Ба. «Адам: метод стохастической оптимизации» . В: CoRR abs/1412.6980 (2014). URL: <http://arxiv.org/abs/1412.6980> (цитируется на стр. 13).
- [2] Джеффри Хинтон. Нейронные сети для машинного обучения. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (цитируется на стр. 13).
- [3] Команда Kaggle. Kaggle: Собаки против кошек. <https://www.kaggle.com/c/dogs-vs-cats> (цитируется на стр. 14).
- [4] Андрей Карпаты. Крошечный вызов ImageNet. <http://cs231n.stanford.edu/project.html> (цитируется на стр. 14).
- [5] Тяньци Чен и др. «MXNet: гибкая и эффективная библиотека машинного обучения для гетерогенных распределенных систем» . В: arXiv.org (декабрь 2015 г.), arXiv: 1512.01274. arXiv: 1512.01274 [cs.DC] (цитируется на стр. 17).
- [6] Сообщество кафе. Caffe: использование нескольких графических процессоров. <https://github.com/BVLC/caffe/blob/master/docs/multigpu.md> (цитируется на стр. 19).
- [7] Янцин Цзя и др. «Caffe: сверточная архитектура для быстрого встраивания функций» . В: Материалы 22-й Международной конференции ACM по мультимедиа. MM '14. Орландо, Флорида, США: ACM, 2014, страницы 675–678. ISBN: 978-1-4503-3063-3. DOI: 10.1145/2647868.2654889. URL: <http://doi.acm.org/10.1145/2647868.2654889> (цитируется на стр. 19).
- [8] Алекс Крижевский, Илья Суцкевер и Джеффри Э. Хинтон. «Классификация ImageNet с глубокими сверточными нейронными сетями» . В: Достижения в системах обработки нейронной информации 25. Под редакцией F. Pereira et al. Curran Associates, Inc., 2012 г., страницы 1097–1105. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf> (цитируется на стр. 20, 21, 54).

- [9] Ольга Русаковская и др. «Масштабная задача визуального распознавания ImageNet». В: Международный журнал компьютерного зрения (IJCV) 115.3 (2015), страницы 211–252. DOI: 10.1007/s11263-015-0816-y (цитируется на стр. 21, 297).
- [10] Ворднет. О Ворднет. <http://wordnet.princeton.edu>. 2010 (цитируется на стр. 21, 31).
- [11] Алисон Грей. NVIDIA и IBM Cloud поддерживают масштабную программу визуального распознавания изображений ImageNet . [https://devblogs.nvidia.com/parallelforall/nvidia-ibm-cloud support — imagenet — крупномасштабный — визуальный — распознавание — вызов/](https://devblogs.nvidia.com/parallelforall/nvidia-ibm-cloud-support---imagenet--- крупномасштабный---визуальный---распознавание---вызов/) (цитируется на стр. 22).
- [12] М. Эверингем и др. «Вызов классов визуальных объектов Pascal (VOC)». In: Международный Журнал Computer Vision 88.2 (июнь 2010 г.), страницы 303–338 (цитируется на странице 22).
- [13] Сообщество AcademicTorrents и кураторы. ImageNet LSVRC 2015. <http://academictorrents.com/collection/imagenet-lsvrc-2015> (цитируется на стр. 24).
- [14] Янцин Цзя и Эван Шелхамер. Caffe Model Zoo: лицензия модели BAIR. http://кафе.berkeleyvision.org/model_zoo.html#bvlc-model-license (цитируется на стр. 26).
- [15] Томаш Малисевич. Глубокое обучение против больших данных: кому что принадлежит? <http://www.computervisionblog.com/2015/05/глубокое обучение-против-больших-данных-кто-чем-владеет>. HTML (цитируется на стр. 27).
- [16] Джек Донахью. Справочник BVLC CaffeNet. https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet (цитируется на стр. 67).
- [17] Карен Симонян и Эндрю Зиссерман. «Очень глубокие сверточные сети для крупномасштабного распознавания изображений» . В: CoRR abs/1409.1556 (2014). URL: <http://arxiv.org/abs/1409.1556> (цитируется на стр. 75, 76, 86, 183, 199, 254, 294).
- [18] Ксавье Глорот и Йошуа Бенжио. «Понимание сложности обучения нейронных сетей с глубокой прямой связью» . В: Материалы Международной конференции по искусственному интеллекту и статистике (AISTATS'10). Общество искусственного интеллекта и статистики. 2010 (цитируется на стр. 77).
- [19] Дмитрий Мишкин и Иржи Матас. «Все, что вам нужно, это хорошее начало» . В: CoRR abs/1511.06422 (2015). URL: <http://arxiv.org/abs/1511.06422> (цитируется на стр. 77, 86).
- [20] Кайминг Хе и соавт. «Углубление в выпрямители: превосходство на уровне человека в классификации ImageNet» . В: CoRR abs/1502.01852 (2015). URL: <http://arxiv.org/abs/1502.01852> (цитируется на стр. 77, 85).
- [21] Кайминг Хе и др. «Глубокое остаточное обучение для распознавания изображений» . В: CoRR abs/1512.03385 (2015). URL: <http://arxiv.org/abs/1512.03385> (цитируется на стр. 86, 90, 105, 107, 254, 294).
- [22] Кайминг Хе и соавт. «Отображение идентичности в глубоких остаточных сетях» . В: CoRR abs/1603.05027 (2016). URL: <http://arxiv.org/abs/1603.05027> (цитируется на стр. 86, 105, 254, 294).
- [23] Кристиан Сегеди и др. «Погружаемся глубже с извилинами» . В: Компьютерное зрение и распознавание образов (CVPR). 2015. URL: <http://arxiv.org/abs/1409.4842> . (цитируется на стр. 89, 90, 103).
- [24] Сообщество VLFeat. VLFeat: предварительно обученные модели. http://www.vlfeat.org/matconvnet/predварительно_обученный/ (цитируется на стр. 89, 99, 103, 104, 120).
- [25] Йост Тобиас Спрингенберг и др. «Стремление к простоте: полностью сверточная сеть» . В: CoRR abs/1412.6806 (2014). URL: <http://arxiv.org/abs/1412.6806> (цитируется на стр. 90, 109).

- [26] Имиджнет. Крупномасштабный конкурс визуального распознавания 2014 (ILSVRC2014). <http://image-net.org/challenges/LSVRC/2014/results> (цитируется на стр. 99).
- [27] Каймин Хэ. Глубокие остаточные сети. <https://github.com/KaimingHe/deep> — остаточные сети (цитируется на стр. 108).
- [28] Вэй Ву. Реснет. <https://github.com/tornadomeet/ResNet> (цитируется на стр. 108).
- [29] Каймин Хэ. ResNet: Должны ли слои свертки иметь смещения? <https://github.com/KaimingHe/deep-residual-networks/issues/10#issuecomment-194037195> (цитируется на стр. 109).
- [30] Форрест Н. Яндоля и др. «SqueezeNet: точность уровня AlexNet с в 50 раз меньшим количеством параметров и размером модели <1 МБ» . В: CoRR abs/1602.07360 (2016). URL: <http://arxiv.org/abs/1602.07360> (цитируется на стр. 121, 123, 133, 294).
- [31] Команда Kaggle. Задачи в репрезентативном обучении: задача распознавания выражения лица . <https://www.kaggle.com/c/challenges-in-representation-learning-face-expression-recognition-challenge> (цитируется на стр. 141, 159).
- [32] Ян Дж. Гудфеллоу и др. «Проблемы в репрезентативном обучении: отчет о трех конкурсах по машинному обучению» . В: Обработка нейронной информации: 20-я международная конференция, ICONIP 2013, Тэгу, Корея, 3-7 ноября 2013 г. Материалы, часть III. Под редакцией Минху Ли и др. Берлин, Гейдельберг: Springer Berlin Heidelberg, 2013, страницы 117–124. ISBN: 978-3-642-42051-1. DOI: 10.1007/978-3-642-42051-1_16. URL: https://doi.org/10.1007/978-3-642-42051-1_16 (цитируется на стр. 141).
- [33] Джостин Хо. Распознавание эмоций лица. <https://github.com/JostineHo/mememoji>. 2016 (цитируется на стр. 143, 156).
- [34] Адриан Роузброк. Практические Python и OpenCV + тематические исследования. PyImageSearch.com, 2016. URL: <https://www.pyimagesearch.com/practical-python-opencv/> (цитируется на стр. 160, 162, 315).
- [35] Адриан Роузброк. Гуру PyImageSearch. https://www.pyimagesearch.com/guru_pyimagesearch/. 2016 (цитируется на стр. 160, 315).
- [36] А. Куаттони и А. Торральба. «Распознавание внутренних сцен» . В: Компьютерное зрение и распознавание образов, Конференция IEEE Computer Society on. Лос-Аламитос, Калифорния, США: Компьютерное общество IEEE, 2009 г., стр. 413–420 (цитируется на стр. 165, 166).
- [37] Джонатан Краузе и др. «Трехмерные представления объектов для мелкозернистой категоризации» . В: 4-й международный семинар IEEE по трехмерному представлению и распознаванию (3dRR-13). Сидней, Австралия, 2013 г. (цитируется на стр. 179).
- [38] Г. Леви и Т. Хасснер. «Возрастно-половая классификация с использованием сверточных нейронных сетей » . В: Конференция IEEE 2015 г. по компьютерному зрению и семинарам по распознаванию образов (CVPRW). Июнь 2015 г., страницы 34–42. DOI: 10.1109/CVPRW.2015.7301352 (цитируется на стр. 203).
- [39] Эран Эйдингер, Рон Энбар и Тал Хасснер. «Возрастная и гендерная оценка нефильтрованных лиц» . В: Пер. Информация. За. сек. 9.12 (декабрь 2014 г.), страницы 2170–2179. ISSN: 1556-6013. DOI: 10.1109/TIFS.2014.2359646. URL-адрес: <http://dx.doi.org/10.1109/TIFS.2014.2359646> (цитируется на стр. 204).
- [40] Дэвис Э. Кинг. «Dlib-ml: набор инструментов для машинного обучения» . В: Дж. Мах. Учиться. Рез. 10 (декабрь 2009 г.), страницы 1755–1758. ISSN: 1532-4435. URL-адрес: <http://dl.acm.org/citation.cfm?id=1577069.1755843> (цитируется на стр. 238, 299).

- [41] Навнит Далал и Билл Триггс. «Гистограммы ориентированных градиентов для обнаружения человека» . В: Материалы конференции IEEE Computer Society 2005 г. по компьютерному зрению и распознаванию образов (CVPR'05) - Том 1 - Том 01. CVPR '05. Вашингтон, округ Колумбия, США: Компьютерное общество IEEE, 2005 г., страницы 886–893. ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.177 . (цитируется на стр. 238, 242, 247, 248).
2005.177. URL-адрес: <http://dx.doi.org/10.1109/CVPR.2005.177> . (цитируется на стр. 238, 242, 247, 248).
- [42] Педро Ф. Фельценшвалб и др. «Обнаружение объектов с дискриминационно обученными моделями на основе частей» . В: IEEE Trans. Аналитический узор. Max. Интел. 32.9 (сентябрь 2010 г.), страницы 1627–1645.
ISSN: 0162-8828. DOI: 10.1109/TПАМИ.2009.167. URL-адрес: <http://dx.doi.org/10.1109/TПАМИ.2009.167> (цит. по стр. 238, 242).
- [43] Росс Б. Гиршик и др. «Богатые иерархии функций для точного обнаружения объектов и семантической сегментации» . В: CoRR abs/1311.2524 (2013). URL: <http://arxiv.org/abs/1311.2524> (цитируется на стр. 247, 251, 260, 293).
- [44] Пол Виола и Майкл Джонс. «Быстрое обнаружение объектов с использованием расширенного каскада простых функций» . В: 2001, страницы 511–518 (цитируется на странице 247).
- [45] Сообщество TensorFlow. API обнаружения объектов Tensorflow. https://github.com/tensorflow/models/tree/master/research/object_detection. 2017 (цит. по страницам 247, 263, 269, 285).
- [46] Шаоцин Рен и др. «Быстрее R-CNN: к обнаружению объектов в реальном времени с сетями региональных предложений» . В: CoRR abs/1506.01497 (2015). URL: <http://arxiv.org/abs/1506.01497> (цитируется на стр. 248, 253, 259, 260, 293).
- [47] Л. Фей-Фей, Р. Фергус и Пьетро Перона. «Изучение генеративных визуальных моделей на нескольких обучающих примерах: поэтапный байесовский подход, протестированный на 101 категории объектов» . В: 2004 (цитируется по стр. 248).
- [48] Джозеф Редмон и др. «Вы только посмотрите один раз: унифицированное обнаружение объектов в реальном времени» . В: CoRR abs/1506.02640 (2015). URL: <http://arxiv.org/abs/1506.02640> (цитируется на стр. 248).
- [49] Джозеф Редмон и Али Фархади. «YOLO9000: лучше, быстрее, сильнее» . В: Препринт arXiv arXiv:1612.08242 (2016) (цитируется на стр. 248).
- [50] JRR Uijlings et al. «Выборочный поиск для распознавания объектов» . В: Международный журнал компьютерного зрения (2013). DOI: 10.1007/s11263-013-0620-5 . URL-адрес: <http://www.huppelen.nl/publications/selectiveSearchDraft.pdf> (цитируется на стр. 251).
- [51] Росс Б. Гиршик. «Быстрый R-CNN» . В: CoRR abs/1504.08083 (2015). архив : 1504.08083.
URL: <http://arxiv.org/abs/1504.08083> (цитируется на стр. 252, 260, 293).
- [52] Мэтью Д. Зейлер и Роб Фергус. «Визуализация и понимание работы сверточной сети» . В: CoRR abs/1311.2901 (2013). URL: <http://arxiv.org/abs/1311.2901> (цитируется на стр. 254).
- [53] Эндрю Г. Ховард и др. «MobileNets: эффективные сверточные нейронные сети для приложений мобильного зрения» . В: CoRR abs/1704.04861 (2017). URL: <http://arxiv.org/abs/1704.04861> (цитируется на стр. 254, 294).
- [54] Хавьер Рей. Faster R-CNN: вниз по кроличьей норе современного обнаружения объектов. <https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>. 2018 (цитируется на стр. 259, 260).
- [55] Джонатан Хуанг и др. «Компромисс между скоростью и точностью для современных сверточных детекторов объектов» . В: CoRR abs/1611.10012 (2016). URL: <http://arxiv.org/abs/1611.10012> (цитируется на стр. 260, 263, 294).

- [56] TryoLabs и сообщество Open Source. *luminoth* — набор инструментов глубокого обучения для компьютера Зрение. <https://github.com/tryolabs/luminoth>. 2017 (цитируется на стр. 260).
- [57] Andreas Mogelmoose, Мохан Манубхай Триведи и Томас Б. Мёслунд. « Обнаружение и анализ дорожных знаков на основе зрения для интеллектуальных систем помощи водителю: перспективы и обзор » . В: Пер. Интел. Транспорт. Сис. 13.4 (декабрь 2012 г.), страницы 1484–1497. ISSN: 1524-9050 (цитируется на стр. 261).
- [58] М. Эверингем и др. «Вызов классов визуальных объектов Pascal: ретроспектива» . В: International Journal of Computer Vision 111.1 (январь 2015 г.), страницы 98–136 (цитируется на страницах 267, 297).
- [59] Цунг-Йи Лин и др. «Microsoft COCO: общие объекты в контексте» . В: CoRR абс/1405.0312 (2014). URL: <http://arxiv.org/abs/1405.0312> (цитируется на стр. 275, 297).
- [60] OM Parkhi и соавт. "Кошки и собаки". В: Конференция IEEE по компьютерному зрению и шаблонам Признания. 2012 г. (цитируется на стр. 277).
- [61] Вэй Лю и др. «SSD: Однокадровый детектор MultiBox» . В: CoRR абс/1512.02325 (2015). URL: <http://arxiv.org/abs/1512.02325> (цитируется на стр. 293–297).
- [62] Кристиан Сегеди и др. «Масштабируемое высококачественное обнаружение объектов» . В: CoRR абс/1412.1441 (2014). URL: <http://arxiv.org/abs/1412.1441> (цитируется на стр. 293–295).
- [63] Эдди Форсон. Понимание SSD MultiBox — обнаружение объектов в реальном времени в глубоком обучении. <https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab>. 2017 (цитируется на стр. 296, 297).
- [64] Дэвис Э. Кинг. «Обнаружение объектов с максимальным запасом» . В: CoRR абс/1502.00046 (2015). URL-адрес: <http://arxiv.org/abs/1502.00046> (цитируется на стр. 299).
- [65] Дэвис Кинг. Выпущен Dlib 18.6: Создайте свой собственный детектор объектов! <http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html>. 2014 (цитируется на стр. 299, 312).
- [66] Дэвис Кинг. dlib - imglab. <https://github.com/davisking/dlib/tree/master/инструменты/imglab>. 2016 (цитируется на стр. 299, 312).
- [67] Дэвис Кинг. Обнаружение транспортных средств с помощью Dlib 19.5. http://блог.dlib.net/2017/08/обнаружение транспортных средств с dlib-195_27.html. 2017 (цитируется на стр. 300).
- [68] Леонард Ричардсон. Beautiful Soup: Мы назвали его Черепахой, потому что он научил нас. <https://www.crummy.com/software/BeautifulSoup/>. 2004 г. (цитируется на стр. 302).