

ОГЛАВЛЕНИЕ

КРАТКОЕ СОДЕРЖАНИЕ	5
ПРЕДИСЛОВИЕ	13
Глубокое обучение в современном мире	13
Что такое глубокое обучение и нужна ли докторская степень, чтобы понять его?	15
PyTorch	16
А как насчет TensorFlow?	16
Типографские соглашения	18
Использование примеров кода	19
Благодарности	19
От издательства	20
ГЛАВА 1. НАЧАЛО РАБОТЫ С PYTORCH	21
Сборка компьютера для глубокого обучения	21
Графический процессор	22
Центральный процессор / материнская плата	22
Оперативная память	23
Хранилище	23
Глубокое обучение в облаке	23
Облачный сервис Google Colaboratory	24
Облачные провайдеры	25
Какой облачный провайдер использовать?	29
Использование Jupyter Notebook	29
Установка PyTorch с нуля	30
Скачивание CUDA	31
Anaconda	31

И наконец, PyTorch! (и Jupyter Notebook)	32
Тензоры	33
Тензорные операции	34
Транслирование тензора	36
Заключение	37
Дополнительная информация	37

ГЛАВА 2. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ PYTORCH 38

Проблема классификации	38
Стандартные трудности	40
Но сначала данные	40
PyTorch и загрузчики данных	41
Создание обучающего набора данных	42
Валидация и контрольные наборы данных	44
И наконец, нейронная сеть!	46
Функции активации	47
Создание нейронной сети.	47
Функции потерь	48
Оптимизация	49
Обучение	52
Работа на GPU	53
Складываем все вместе	54
Прогнозирование	55
Сохранение модели	56
Заключение	57
Дополнительные источники	58

ГЛАВА 3. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ 59

Первая сверточная модель	59
Свертки	60
Субдискретизация	64
Прореживание, или дропаут.	65

История архитектур CNN	66
AlexNet	66
Inception/GoogLeNet.	67
VGG	68
ResNet	70
Другие архитектуры тоже доступны!	71
Использование предварительно обученных моделей в PyTorch	71
Изучение структуры модели	72
Пакетная нормализация (BatchNorm)	75
Какую модель мне использовать?	76
Необходимые покупки: PyTorch Hub	76
Заключение	77
Дополнительные источники	77

ГЛАВА 4. ПЕРЕНОС ОБУЧЕНИЯ И ДРУГИЕ ФОКУСЫ 79

Перенос обучения с помощью ResNet	79
Вычисление скорости обучения	82
Дифференциальная скорость обучения	85
Аугментация данных	87
Преобразования Torchvision	88
Цветовое пространство и лямбда-преобразование	94
Пользовательские классы преобразования	96
Начните с меньшего и получите больше!	97
Ансамбли	98
Заключение	99
Дополнительные источники	100

ГЛАВА 5. КЛАССИФИКАЦИЯ ТЕКСТА 101

Рекуррентные нейронные сети	101
Сети с долгой краткосрочной памятью	104
Управляемые рекуррентные блоки	105
biLSTM	106

Вложения (Embeddings)	107
torchtext	110
Получение наших данных: твиты!.	111
Определение полей	113
Построение словаря.	115
Создание модели	117
Обновление цикла обучения	118
Классификация твитов.	119
Аугментация данных	120
Случайная вставка	121
Случайное удаление	122
Случайная перестановка	122
Обратный перевод.	123
Аугментация и torchtext	124
Перенос обучения?	125
Заключение	125
Дополнительные источники	126
ГЛАВА 6. ПУТЕШЕСТВИЕ В МИР ЗВУКОВ	127
Звук	127
Набор данных ESC-50	129
Получение набора данных	129
Воспроизведение аудио в Jupyter.	130
Исследуя данные ESC-50	130
SoX и LibROSA	131
torchaudio	132
Создание набора данных ESC-50	133
Модель CNN для набора данных ESC-50	135
Частота — моя вселенная	138
Мел-спектрограммы	138
Новый набор данных	140
Появление ResNet	144
Определение скорости обучения	145

Аугментация аудиоданных	147
Преобразования torchaudio	147
Эффекты SoX	148
SpecAugment	149
Новые эксперименты	154
Заключение	155
Дополнительные источники	155
ГЛАВА 7. ОТЛАДКА МОДЕЛЕЙ PYTORCH	156
Три часа ночи. Что делают ваши данные?	156
TensorBoard	157
Установка TensorBoard	158
Отправка данных в TensorBoard	158
Хуки PyTorch	162
Построение графика среднего и стандартного отклонения	163
Карты активаций класса	165
Флеймграфы	168
Установка ru-spy	170
Чтение флеймграфов	171
Решение задачи медленного преобразования	173
Отладка проблем с графическим процессором	177
Проверка графического процессора	177
Градиентное создание контрольных точек	179
Заключение	181
Дополнительные источники	182
ГЛАВА 8. PYTORCH В РАБОЧЕЙ СРЕДЕ	183
Обслуживание модели	183
Построение сервиса Flask	184
Настройка параметров модели	187
Сборка контейнера Docker	188
Локальное и облачное хранилище	191
Логирование и телеметрия	194

Развертывание в Kubernetes	195
Установка на Google Kubernetes Engine.	196
Создание кластера k8s.	196
Сервисы масштабирования.	198
Обновления и очистка	198
TorchScript	199
Трассировка.	200
Выполнение скриптов	203
Ограничения TorchScript.	205
Работа с libTorch	207
Получение libTorch и Hello World.	207
Импорт модели TorchScript	209
Заключение	211
Дополнительные источники	212

ГЛАВА 9. PYTORCH НА ПРАКТИКЕ 213

Аугментация данных: смешанная и сглаженная	213
mixup.	213
Сглаживание маркировок.	218
Компьютер, улучшай!	219
Введение в сверхвысокое разрешение	220
Введение в GAN	223
Фальсификатор и критик	224
Обучение GAN	225
Схлопывание мод распределения.	226
ESRGAN	227
Запуск ESRGAN	227
Новые приключения в распознавании образов	228
Обнаружение объектов	228
Faster R-CNN и Mask R-CNN.	231
Состязательные семплы	233
Black-Vox-атаки	236
Защита от состязательных атак	237

Больше, чем кажется: архитектура Transformer	238
Механизмы внимания.	238
Все, что нужно, — это внимание	240
BERT	240
FastBERT	241
GPT-2.	243
Генерация текста с помощью GPT-2	244
ULMFiT.	246
Что выбрать?	249
Заключение	249
Дополнительные источники	251

ОБ АВТОРЕ	252
----------------------------	------------

ОБ ОБЛОЖКЕ	253
-----------------------------	------------

Аугментация данных

Одна из самых страшных фраз, которые только можно услышать в науке о данных: «Черт, моя модель переобучена!». Как я уже говорил в главе 2, переобучение происходит в том случае, когда модель решает отразить данные, представленные в обучающем наборе, а не сгенерировать обобщенное решение. Часто приходится слышать, что конкретная модель *запомнила набор данных*, то есть модель выучила ответы и продолжила плохо обрабатывать рабочие данные.

Традиционным способом предотвращения этой проблемы является накопление большого количества данных. Чем больше данных будет видеть модель, тем более общее представление она получит о задаче, которую пытается решить. При рассмотрении задачи стягивания, если вы не дадите модели просто сохранять все ответы (перегружая ее память таким большим количеством данных), то она будет вынуждена *сжимать* входные данные и, следовательно, создавать решение, которое не сможет просто сохранить ответы внутри нее. На практике это работает хорошо, но предположим, что у нас есть только тысяча изображений и мы делаем перенос обучения. Что же делать в таком случае?

Одним из подходов, который мы можем использовать, является *аугментация данных*. Если у нас есть изображение, мы можем проделать с ним несколько манипуляций, которые должны предотвратить переобучение и сделать модель более общей. Рассмотрим изображения кошки Гельветики на рис. 4.2 и 4.3.

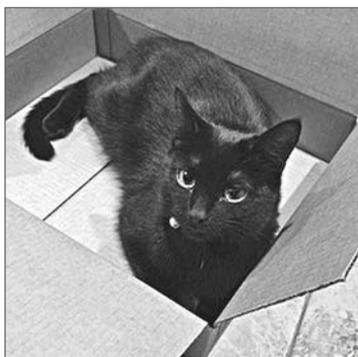


Рис. 4.2. Исходное изображение



Рис. 4.3. Перевернутая Гельветика

Для нас очевидно, что это один и тот же образ. Второе фото — это просто зеркальное отображение исходного изображения. Тензорное представление будет другим, так как RGB-значения будут находиться в разных местах трехмерного изображения. Но на фото все та же кошка, поэтому мы рассчитываем, что модель, обучаемая на этом изображении, научится распознавать форму кошки с левой или правой стороны кадра, а не будет просто связывать все изображение с *кошкой*. В PyTorch все просто. Возможно, вы помните этот фрагмент кода из главы 2:

```
transforms = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225] )
])
```

Он формирует конвейер преобразования, через который проходят все изображения при входе в модель для обучения. Но библиотека `torchvision.transforms` содержит много других функций преобразования, которые можно использовать для аугментации набора данных для обучения. Давайте рассмотрим некоторые из наиболее полезных и посмотрим, что происходит с Гельветикой при некоторых менее очевидных преобразованиях.

Преобразования Torchvision

В состав `torchvision` входит большой набор потенциальных преобразований, которые можно использовать для аугментации данных, а также два

способа создания новых преобразований. В этом разделе мы рассмотрим наиболее полезные из них, а также познакомимся с парой отдельных преобразований, которые вы можете использовать в своих собственных приложениях.

```
torchvision.transforms.ColorJitter(brightness=0, contrast=0, saturation=0,  
                                   hue=0)
```

`ColorJitter` случайным образом меняет яркость, контрастность, насыщенность и оттенок изображения.

Для яркости, контраста и насыщенности вы можете задать либо число с плавающей точкой, либо кортеж с плавающей точкой, все неотрицательные числа в диапазоне от 0 до 1, и будет использоваться либо случайность между 0 и заданным значением с плавающей точкой, либо кортеж для генерации случайности между заданной парой значений с плавающей точкой. Для оттенка требуется значение с плавающей точкой или кортеж с плавающей точкой от $-0,5$ до $0,5$, он будет генерировать случайные корректировки оттенка от $[-hue, hue]$ или $[min, max]$ (см. рис. 4.4).

Два преобразования случайным образом отражают изображение по горизонтальной или вертикальной оси:

```
torchvision.transforms.RandomHorizontalFlip(p=0.5)  
torchvision.transforms.RandomVerticalFlip(p=0.5)
```

Либо задайте значение с плавающей точкой от 0 до 1 для вероятности отражения, либо по умолчанию примите значение с вероятностью отражения 50%. Вертикально перевернутая кошка показана на рис. 4.5.



Рис. 4.4. `ColorJitter` применяется при 0,5 для всех параметров



Рис. 4.5. Вертикально перевернутое изображение

`RandomGrayscale` — это аналогичный тип преобразования, за исключением того, что он случайным образом преобразует изображения в оттенках серого в зависимости от параметра p (обработка по умолчанию 10 %):

```
torchvision.transforms.RandomGrayscale(p=0.1)
```

`RandomCrop` и `RandomResizeCrop`, как вы можете предполагать, обрезают картинку случайным образом, размер может быть либо переменной `int` для высоты и ширины, либо кортежем, содержащим разную высоту и ширину. На рис. 4.6 показан пример работы `RandomCrop`.

```
torchvision.transforms.RandomCrop(size, padding=None,
pad_if_needed=False, fill=0, padding_mode='constant')
torchvision.transforms.RandomResizedCrop(size, scale=(0.08, 1.0),
ratio=(0.75, 1.3333333333333333), interpolation=2)
```

Теперь нужно быть внимательнее, потому что если кадрированное изображение будет слишком маленьким, вы рискуете вырезать важные детали и модель выучит что-то неверно. Например, если на картинке изображена кошка, которая играет на столе, программа обрежет кошку и просто оставит часть стола, который будет классифицирован как *кошка*. Так себе результат. `RandomResizeCrop` изменит размер кадрирования, чтобы заполнить заданный размер, `RandomCrop` может сделать кадрирование близко к краю и в темных местах за пределами изображения.

Как вы уже знаете из главы 3, мы можем добавить отступ, чтобы сохранить необходимый размер изображения. По умолчанию отступ является

константой и заполняет пустые пиксели за пределами изображения тем значением, которое задано в параметре `fill`. Тем не менее советую использовать вместо него отступ `reflect`, поскольку практика показывает, что он работает немного лучше, а не просто заполняет пустое константное пространство.



`RandomResizeCrop` использует билинейную интерполяцию, но вы также можете выбрать ближайшую соседнюю или бикубическую интерполяцию, изменив параметр интерполяции. Более подробно см. на странице фильтров PIL.

Если вы хотите повернуть изображение случайным образом, `RandomRotation` будет варьироваться между `[-degrees, degrees]`, если `degrees` — это одно число с плавающей точкой или `int`, либо `(min,max)`, если это кортеж:

```
torchvision.transforms.RandomRotation(degrees, resample=False, expand=False,
                                       center=None)
```

Если для `expand` выбрано значение `True`, эта функция расширит выходное изображение так, чтобы оно могло включать весь поворот; оно обрезается по размерам входных данных по умолчанию.

Вы можете задать фильтр передискретизации PIL и при желании ввести `(x,y)` кортеж для центра вращения; в противном случае преобразование будет вращаться вокруг центра изображения.

Рисунок 4.7 — преобразование `RandomRotation` с заданными 45 градусами.



Рис. 4.6. `RandomCrop`. Размер=100