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.

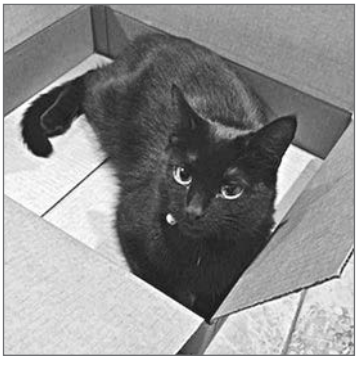

Рис. 4.2. Исходное изображение

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 `torchivision.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 пикселы 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