5.2. Обучение сверточной нейронной сети с нуля на небольшом наборе данных

Необходимость обучения модели классификации изображений на очень небольшом объеме данных — обычная ситуация, с которой вы наверняка столкнетесь в своей практике, если будете заниматься распознаванием образов с помощью технологий компьютерного зрения на профессиональном уровне. Под «небольшим» объемом понимается от нескольких сотен до нескольких десятков тысяч изображений. В качестве практического примера рассмотрим классификацию изображений собак и кошек из набора данных, содержащего 4000 изображений (2000 кошек, 2000 собак). Мы будем использовать 2000 изображений для обучения, 1000 для проверки и 1000 для контроля.

В этом разделе рассматривается одна простая стратегия решения данной задачи: обучение новой модели с нуля при наличии небольшого объема исходных данных. Сначала мы обучим маленькую сверточную нейронную сеть на 2000 обучающих образцах без применения регуляризации, чтобы задать базовый уровень достижимого. Она даст нам точность классификации 71 %. С этого момента начнет проявляться эффект переобучения. Затем вашему вниманию будет представлен эффективный способ уменьшения степени переобучения в распознавании образов — *расширение данных* (data augmentation). С его помощью мы повысим точность классификации до 82 %.

В следующем разделе мы рассмотрим еще два основных приема глубокого обучения на небольших наборах данных: *выделение признаков с использованием предварительно обученной сети* (поможет поднять точность с 90 до 96 %) и *дообучение предварительно обученной сети* (поможет достичь окончательной точности в 97 %). Вместе эти три стратегии — обучение малой модели с нуля, выделение признаков с использованием предварительно обученной модели и дообучение этой модели — станут вашим основным набором инструментов для решения задач классификации изображений с обучением на небольших наборах данных.

5.2.1. Целесообразность глубокого обучения для решения задач с небольшими наборами данных

Иногда можно услышать, что глубокое обучение можно применять только при наличии большого объема данных. Это утверждение верно лишь отчасти: одна из основных характеристик глубокого обучения — возможность самостоятельно находить информативные признаки в обучающих данных, без конструирования признаков вручную, а это достижимо только при наличии большого объема обучающих примеров. Это особенно верно для задач, когда входные образцы имеют много измерений, как, например, изображения.

Однако понятие «большой объем данных» весьма относительно, в первую очередь относительно размера и глубины обучаемой сети. Нельзя обучить сверточную

нейронную сеть решению сложной задачи на нескольких десятках образцов, а вот нескольких сотен вполне может хватить, если модель невелика и регуляризована, а решаемая задача проста. Так как сверточные нейронные сети изучают локальные признаки, инвариантные в отношении переноса, они обладают высокой эффективностью в решении задач распознавания. Обучение сверточной нейронной сети с нуля на очень небольшом наборе изображений дает вполне неплохие результаты, несмотря на относительную нехватку данных, без необходимости конструировать признаки вручную. В данном разделе мы убедимся в этом на практике.

Более того, модели глубокого обучения по своей природе очень гибкие: можно, к примеру, обучить модель для классификации изображений или распознавания речи на очень большом наборе данных и затем использовать ее для решения самых разных задач с небольшими модификациями. В частности, в распознавании образов многие предварительно обученные модели (обычно на наборе данных ImageNet) теперь доступны всем желающим для загрузки и могут использоваться как основа для создания очень мощных моделей распознавания образов на небольших объемах данных. Именно так мы и поступим в следующем разделе. Начнем с получения данных.

5.2.2. Загрузка данных

Набор данных «Dogs vs. Cats», который мы будем использовать, не поставляется в составе Keras. Он был создан в ходе состязаний по распознаванию образов в конце 2013-го, когда сверточные нейронные сети еще не заняли лидирующего положения, и доступен на сайте Kaggle. Этот набор можно получить по адресу: *www.kaggle.com/c/dogs-vs-cats/data* (для этого вам потребуется создать учетную запись на сайте Kaggle, если у вас ее еще нет, но не волнуйтесь, процесс регистрации очень прост).

Этот набор содержит изображения в формате JPEG с низким разрешением. На рис. 5.8 показано несколько примеров.

Неудивительно, что состязание по классификации изображений кошек и собак на сайте Kaggle в 2013 году выиграли участники, использовавшие сверточные нейронные сети. Лучшие результаты достигали точности в 95 %. В этом примере мы приблизимся к этой точности (в следующем разделе), даже при том, что для обучения моделей будем использовать менее 10% данных, которые были доступны участникам состязаний.

Этот набор содержит 25 000 изображений кошек и собак (по 12 500 для каждого класса) общим объемом 543 Мбайт (в сжатом виде). После загрузки и распаковки архива мы создадим новый набор, разделенный на три поднабора: обучающий набор с 1000 образцами каждого класса, проверочный набор с 500 образцами каждого класса и контрольный набор с 500 образцами каждого класса.

Рис. 5.8. Примеры изображений из набора «Dogs vs. Cats». Размеры не были изменены: изображения имеют разные размеры, ракурсы съемки и т. д.

Все необходимое выполняет код в листинге 5.4.

Листинг 5.4. Копирование изображений в обучающий, проверочный и контрольный каталоги

```
import os, shutil

original_dataset_dir = '/Users/fchollet/Downloads/kaggle_original_data'

base_dir = '/Users/fchollet/Downloads/cats_and_dogs_small'
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)
```

Путь к каталогу с распакованным исходным набором данных

Каталог для сохранения выделенного небольшого набора

Каталоги для обучающего, проверочного и контрольного поднаборов

Каталог для обучающих изображений с кошками

Каталог для обучающих изображений с собаками

Продолжение

Листинг 5.4 (продолжение)

```
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)
```

Каталог для проверочных изображений с кошками

```
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)
```

Каталог для проверочных изображений с собаками

```
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)
```

Каталог для контрольных изображений с кошками

```
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```

Каталог для контрольных изображений с собаками

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)
```

Копирование первых 1000 изображений с кошками в каталог train_cats_dir

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)
```

Копирование следующих 500 изображений с кошками в каталог validation_cats_dir

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)
```

Копирование следующих 500 изображений с кошками в каталог test_cats_dir

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

Копирование первых 1000 изображений с собаками в каталог train_dogs_dir

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

Копирование следующих 500 изображений с собаками в каталог validation_dogs_dir

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

Копирование следующих 500 изображений с собаками в каталог test_dogs_dir

Для проверки подсчитаем, сколько изображений оказалось в каждом поднаборе (обучающем/проверочном/контрольном):

```
>>> print('total training cat images:', len(os.listdir(train_cats_dir)))
total training cat images: 1000
>>> print('total training dog images:', len(os.listdir(train_dogs_dir)))
total training dog images: 1000
>>> print('total validation cat images:', len(os.listdir(validation_cats_
dir)))
total validation cat images: 500
>>> print('total validation dog images:', len(os.listdir(validation_dogs_
dir)))
total validation dog images: 500
>>> print('total test cat images:', len(os.listdir(test_cats_dir)))
total test cat images: 500
>>> print('total test dog images:', len(os.listdir(test_dogs_dir)))
total test dog images: 500
```

Итак, у нас действительно имеется 2000 обучающих, 1000 проверочных и 1000 контрольных изображений. Каждый поднабор содержит одинаковое количество образцов каждого класса: это сбалансированная задача бинарной классификации, соответственно, мерой успеха может служить точность классификации.

5.2.3. Конструирование сети

В предыдущем примере мы сконструировали небольшую сверточную нейронную сеть для данных MNIST, и теперь вы знаете, что это такое. В этом примере мы реализуем ту же самую общую структуру: сверточная нейронная сеть будет организована как стек чередующихся слоев `Conv2D` (с функцией активации `relu`) и `MaxPooling2D`.

Однако, так как мы имеем дело с большими изображениями и решаем более сложную задачу, мы сделаем сеть больше: она будет иметь на одну пару слоев `Conv2D` + `MaxPooling2D` больше. Это увеличит ее емкость и обеспечит дополнительное снижение размеров карт признаков, чтобы они не оказались слишком большими, когда достигнут слоя `Flatten`. С учетом того, что мы начнем с входов, имеющих размер 150×150 (выбор был сделан совершенно произвольно), в конце, точно перед слоем `Flatten`, получится карта признаков размером 7×7.

ПРИМЕЧАНИЕ

Глубина карт признаков в сети будет постепенно увеличиваться (с 32 до 128), а их размеры — уменьшаться (со 148×148 до 7×7). Этот шаблон вы будете видеть почти во всех сверточных нейронных сетях.

Так как перед нами стоит задача бинарной классификации, сеть должна заканчиваться единственным признаком (слой `Dense` с размером 1 и функцией активации `sigmoid`). Этот признак будет представлять собой вероятность принадлежности рассматриваемого изображения одному из двух классов.

Листинг 5.5. Создание небольшой сверточной нейронной сети для классификации изображений кошек и собак

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
          input_shape=(150, 150, 3)))

model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Посмотрим, как изменяются размеры карт признаков с каждым последующим слоем:

```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)(None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)(None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
maxpooling2d_3 (MaxPooling2D)(None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
maxpooling2d_4 (MaxPooling2D)(None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

На этапе компиляции, как обычно, используем оптимизатор `RMSprop`. Так как сеть заканчивается единственным признаком, используем функцию потерь `binary_crossentropy` (для напоминания: в табл. 4.1 приводится шпаргалка по использованию разных функций потерь в разных ситуациях).

Листинг 5.6. Настройка модели для обучения

```
from keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

5.2.4. Предварительная обработка данных

Как вы уже знаете, перед передачей в сеть данные должны быть преобразованы в тензоры с вещественными числами. В настоящее время данные хранятся в виде файлов JPEG, поэтому их нужно подготовить для передачи в сеть, выполнив следующие шаги:

1. Прочитать файлы с изображениями.
2. Декодировать содержимое из формата JPEG в таблицы пикселов RGB.
3. Преобразовать их в тензоры с вещественными числами.
4. Масштабировать значения пикселов из диапазона [0, 255] в диапазон [0, 1] (как вы уже знаете, нейронным сетям предпочтительнее передавать небольшие значения).

Этот порядок действий может показаться немного сложным, но, к счастью, в Keras имеются утилиты, способные выполнить его автоматически. Во фреймворке Keras имеется модуль `keras.preprocessing.image` с инструментами для обработки изображений. В частности, в нем вы найдете класс `ImageDataGenerator`, который позволит быстро настроить генераторы Python для автоматического преобразования файлов с изображениями в пакеты готовых тензоров. В листинге 5.7 показано, как им воспользоваться.

Листинг 5.7. Использование ImageDataGenerator для чтения изображений из каталогов

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
```

Масштабировать значения с коэффициентом 1/255

Целевой каталог

Привести все изображения к размеру 150 × 150

Продолжение

Листинг 5.7 (продолжение)

```
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

Так как используется функция потерь binary_crossentropy, метки должны быть бинарными

Генераторы Python

Генератором в языке Python называется объект, действующий как итератор: его можно использовать с инструкцией цикла `for ... in`. Работа генераторов основана на использовании инструкции `yield`.

Вот пример генератора, возвращающего целые числа:

```
def generator():
    i = 0
    while True:
        i += 1
        yield i

for item in generator():
    print(item)
    if item > 4:
        break
```

На экран будет выведено следующее:

```
1
2
3
4
5
```

Рассмотрим вывод одного из таких генераторов: он возвращает пакеты изображений 150 × 150 в формате RGB (с формой `(20, 150, 150, 3)`) и бинарные метки (с формой `(20,)`). В каждом пакете имеется 20 образцов (размер пакета). Обратите внимание на то, что этот генератор возвращает пакеты до бесконечности: он выполняет бесконечный цикл перебора изображений в целевом каталоге. По этой причине мы должны прервать цикл в некоторый момент:

```
>>> for data_batch, labels_batch in train_generator:
>>>     print('data batch shape:', data_batch.shape)
>>>     print('labels batch shape:', labels_batch.shape)
>>>     break
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

Передадим исходные данные в модель с помощью генератора. Используем для этого метод `fit_generator`, эквивалент метода `fit` для генераторов данных, подобных этому. В первом аргументе он принимает генератор Python, бесконечно возвращающий

пакеты входных данных и целей, как это делает данный генератор. Так как данные генерируются до бесконечности, модель Keras должна знать, сколько образцов извлечь из генератора, прежде чем объявить эпоху завершенной. Эту функцию выполняет аргумент `steps_per_epoch`: после извлечения `steps_per_epoch` пакетов из генератора, то есть после выполнения `steps_per_epoch` шагов градиентного спуска, процесс обучения переходит к следующей эпохе. В данном случае пакеты содержат по 20 образцов, поэтому для получения 2000 образцов модель извлечет 100 пакетов.

При использовании метода `fit_generator` вы можете передать аргумент `validation_data`, так же как методу `fit`. Важно отметить, что этот аргумент может быть не только генератором данных, но также кортежем массивов Numpy. Если в `validation_data` передать генератор, предполагается, что он будет возвращать пакеты проверочных данных до бесконечности; поэтому вы должны также передать аргумент `validation_steps`, определяющий количество пакетов, извлекаемых из генератора проверочных данных для оценки.

Листинг 5.8. Обучение модели с использованием генератора пакетов

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

Хорошей практикой считается всегда сохранять модели после обучения.

Листинг 5.9. Сохранение модели

```
model.save('cats_and_dogs_small_1.h5')
```

Создадим графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения (рис. 5.9 и 5.10).

Листинг 5.10. Формирование графиков изменения потерь и точности в процессе обучения

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

Продолжение

Листинг 5.10 (продолжение)

```
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

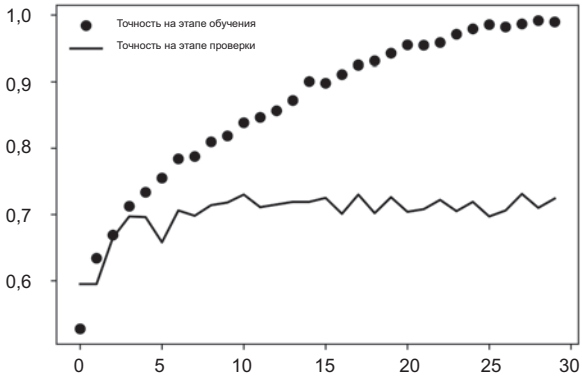

Рис. 5.9. Точность на этапах обучения и проверки

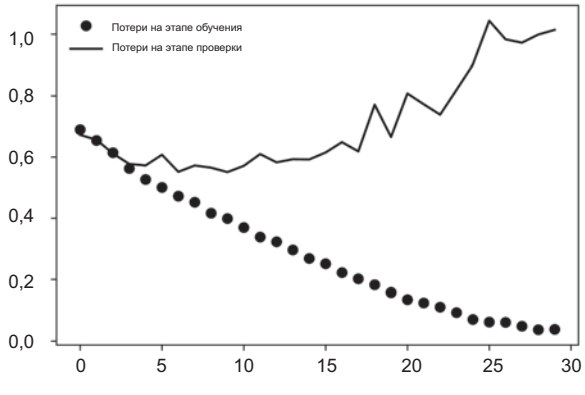

Рис. 5.10. Потери на этапах обучения и проверки

На графиках четко наблюдается эффект переобучения. Точность на обучающих данных линейно растет и приближается к 100 %, тогда как точность на проверочных данных останавливается на отметке 70–72 %. Потери на этапе проверки достигают минимума всего после пяти эпох и затем замирают, а потери на этапе обучения продолжают линейно уменьшаться, почти достигая 0.

Поскольку у нас относительно немного обучающих образцов (2000), переобучение становится проблемой номер один. Вы уже знаете несколько методов, помогающих смягчить эту проблему, таких как прореживание и сокращение весов