Deep Learning with Python

Chapter 5: Deep Learning for Computer Vision

Listing 5.1 Instantiating a small convnet

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
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(64, (3, 3), activation='relu'))
```

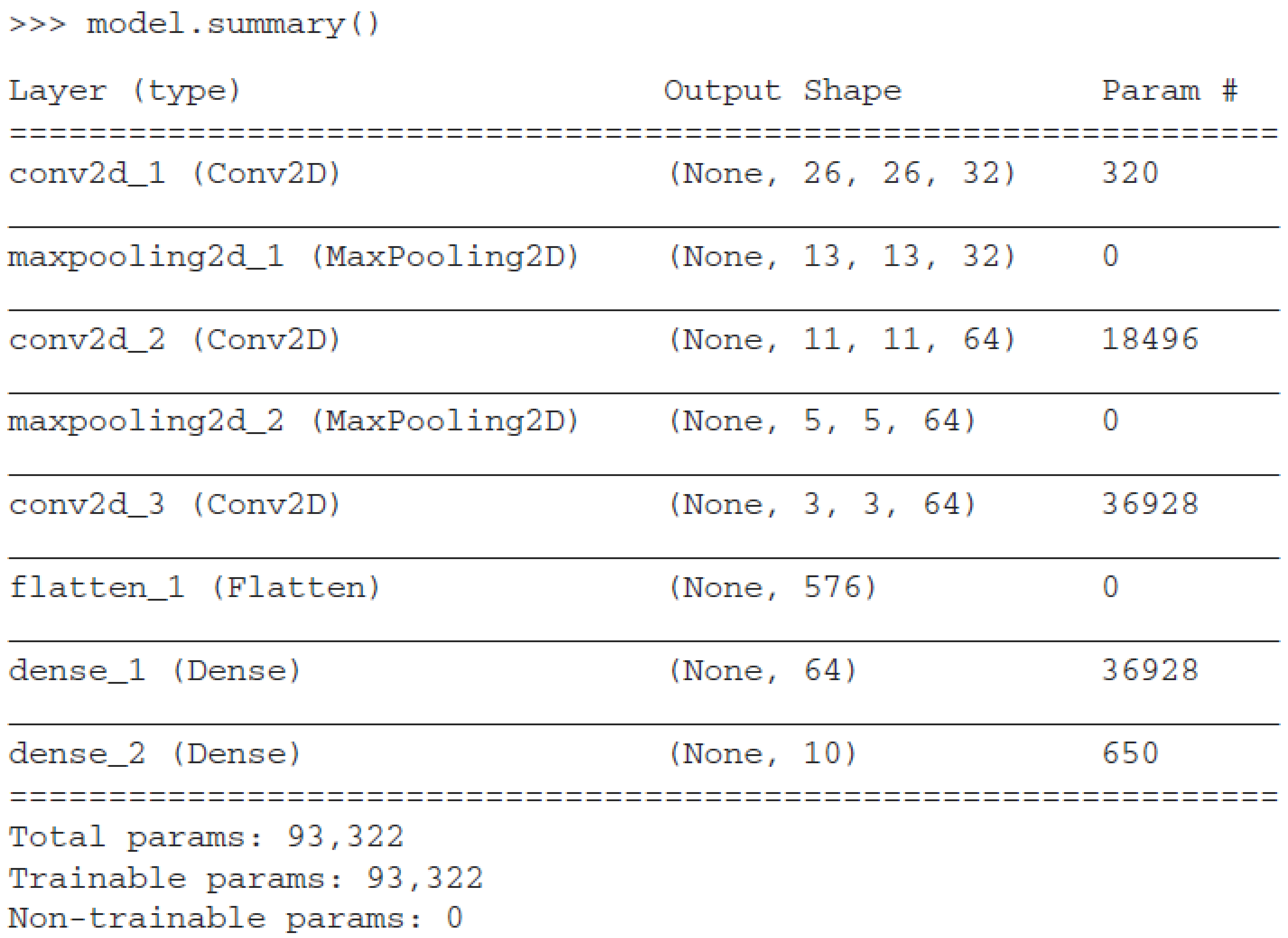
```
>>> model.summary()

________________________________________________________________
Layer (type)                     Output Shape          Param #
================================================================
conv2d_1 (Conv2D)                (None, 26, 26, 32)    320
________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 13, 13, 32)    0
________________________________________________________________
conv2d_2 (Conv2D)                (None, 11, 11, 64)    18496
________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 5, 5, 64)      0
________________________________________________________________
conv2d_3 (Conv2D)                (None, 3, 3, 64)      36928
================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

Listing 5.1 Instantiating a small convnet

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
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(64, (3, 3), activation='relu'))
```

Listing 5.2 Adding a classifier on top of the convnet

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```



```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)   (None, 13, 13, 32)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)   (None, 5, 5, 64)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Listing 5.3 Training the convnet on MNIST images

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

Let's evaluate the model on the test data:

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc
0.99080000000000001
```

Whereas the densely connected network from chapter 2 had a test accuracy of 97.8%, the basic convnet has a test accuracy of 99.3%: we decreased the error rate by 68% (relative). Not bad!

The convolution operation

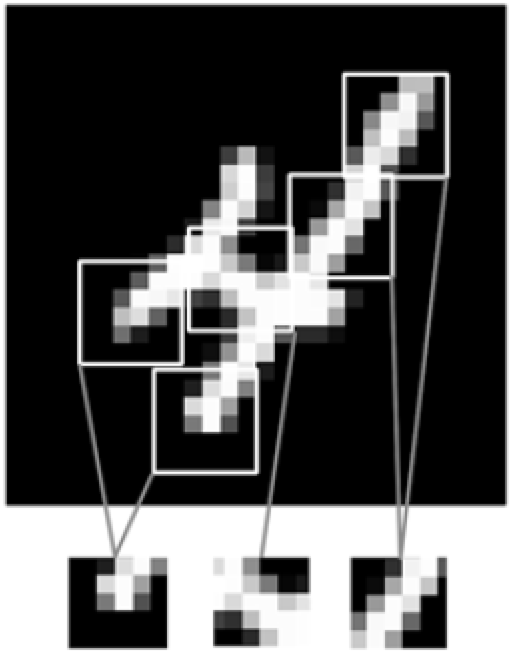


Figure 5.1 Images can be broken into local patterns such as edges, textures, and so on.

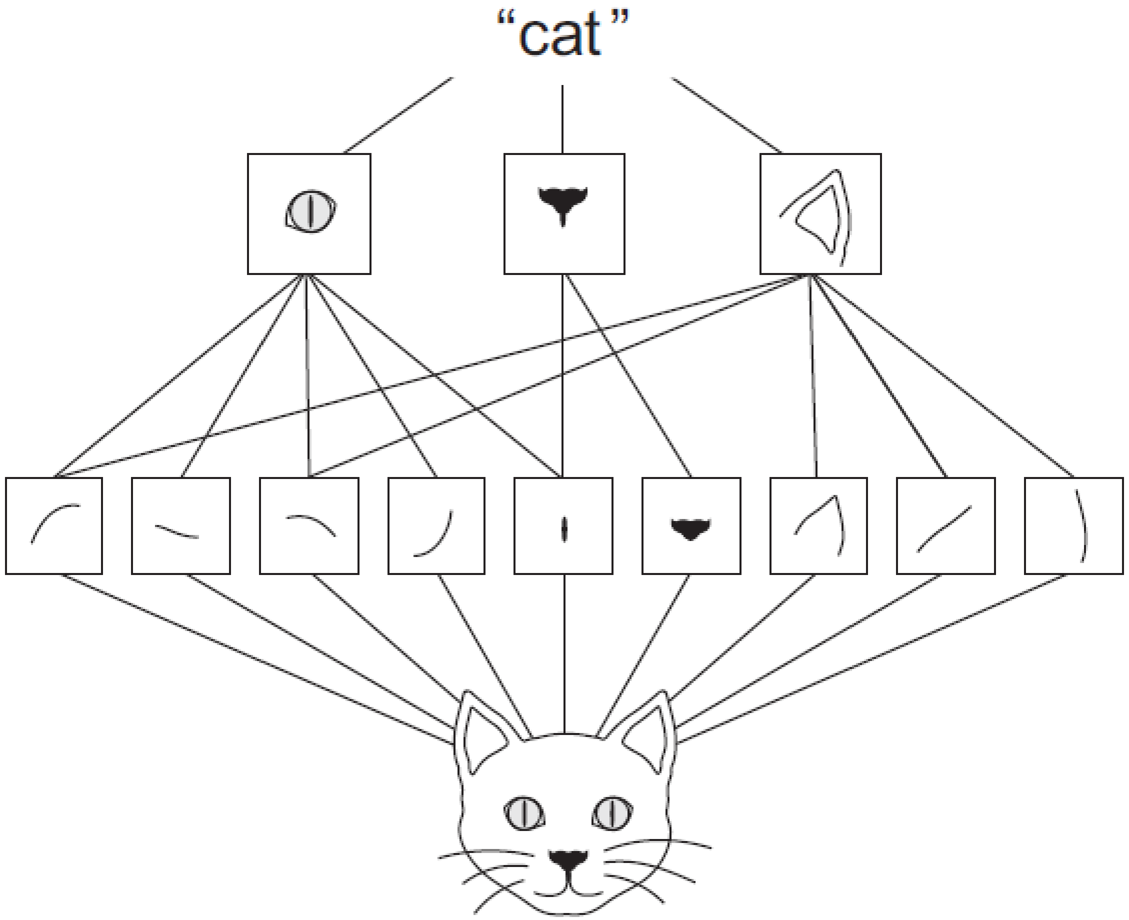


Figure 5.2 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as "cat."

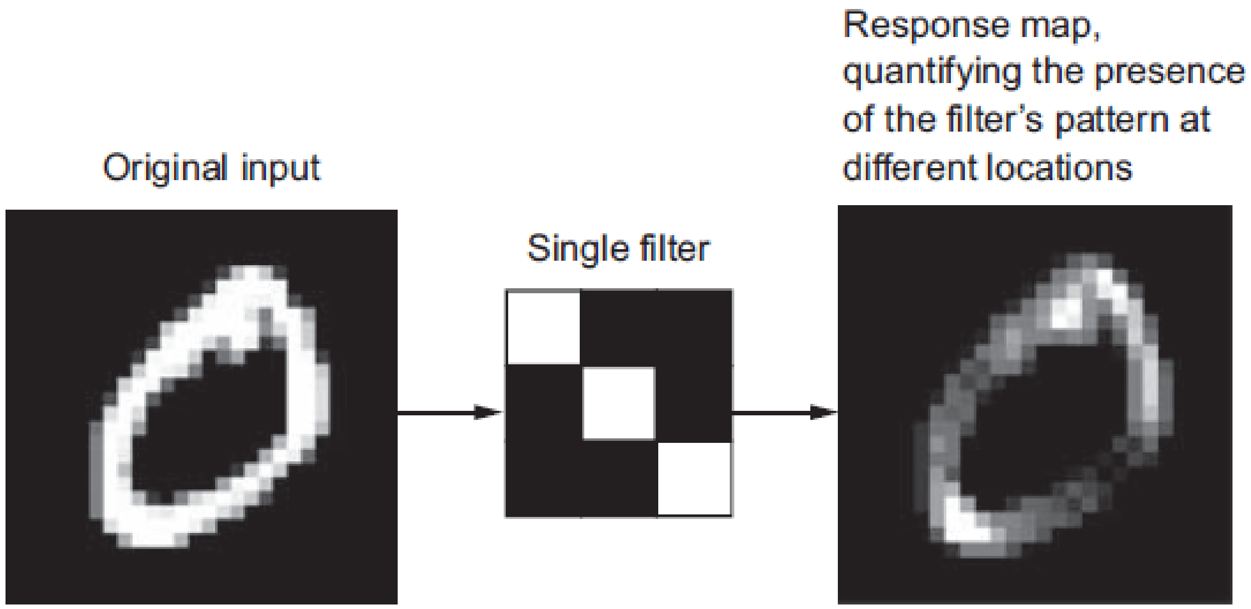

Original input
Single filter
Response map, quantifying the presence of the filter's pattern at different locations

Figure 5.3 The concept of a *response map*: a 2D map of the presence of a pattern at different locations in an input

Convolutions are defined by two key parameters:

- *Size of the patches extracted from the inputs*—These are typically 3 × 3 or 5 × 5. In the example, they were 3 × 3, which is a common choice.
- *Depth of the output feature map*—The number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.

In Keras `Conv2D` layers, these parameters are the first arguments passed to the layer: `Conv2D(output_depth, (window_height, window_width))`.

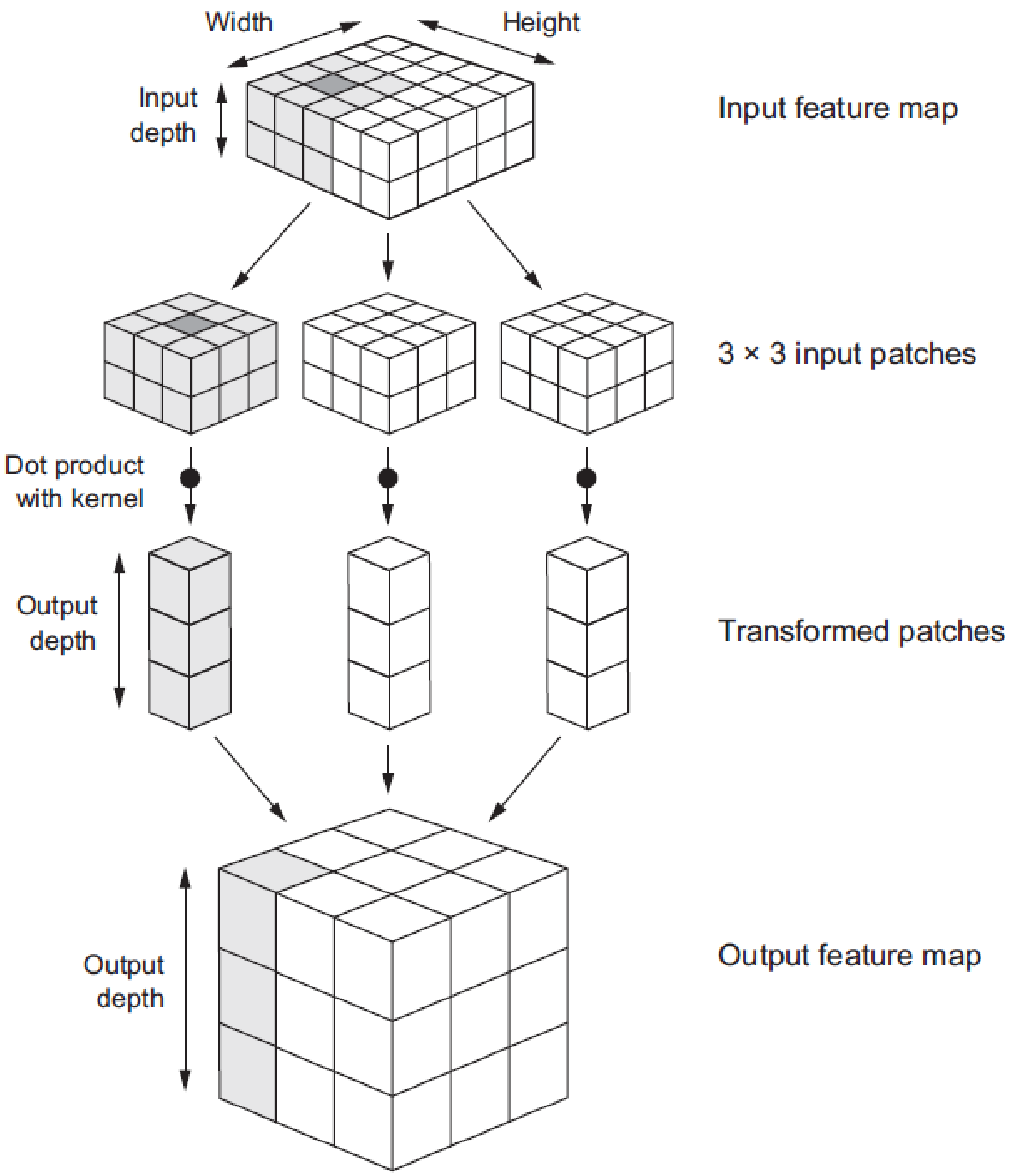
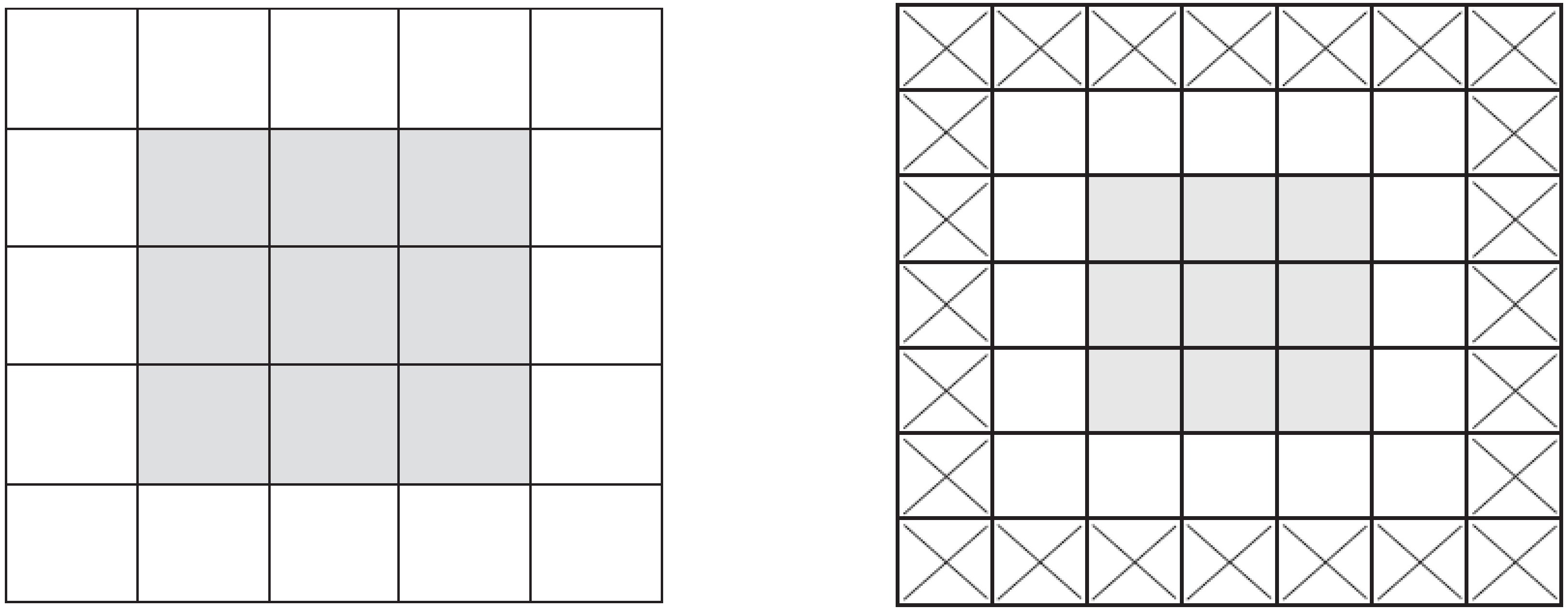


Figure 5.4 How convolution works

UNDERSTANDING BORDER EFFECTS AND PADDING



UNDERSTANDING CONVOLUTION STRIDES

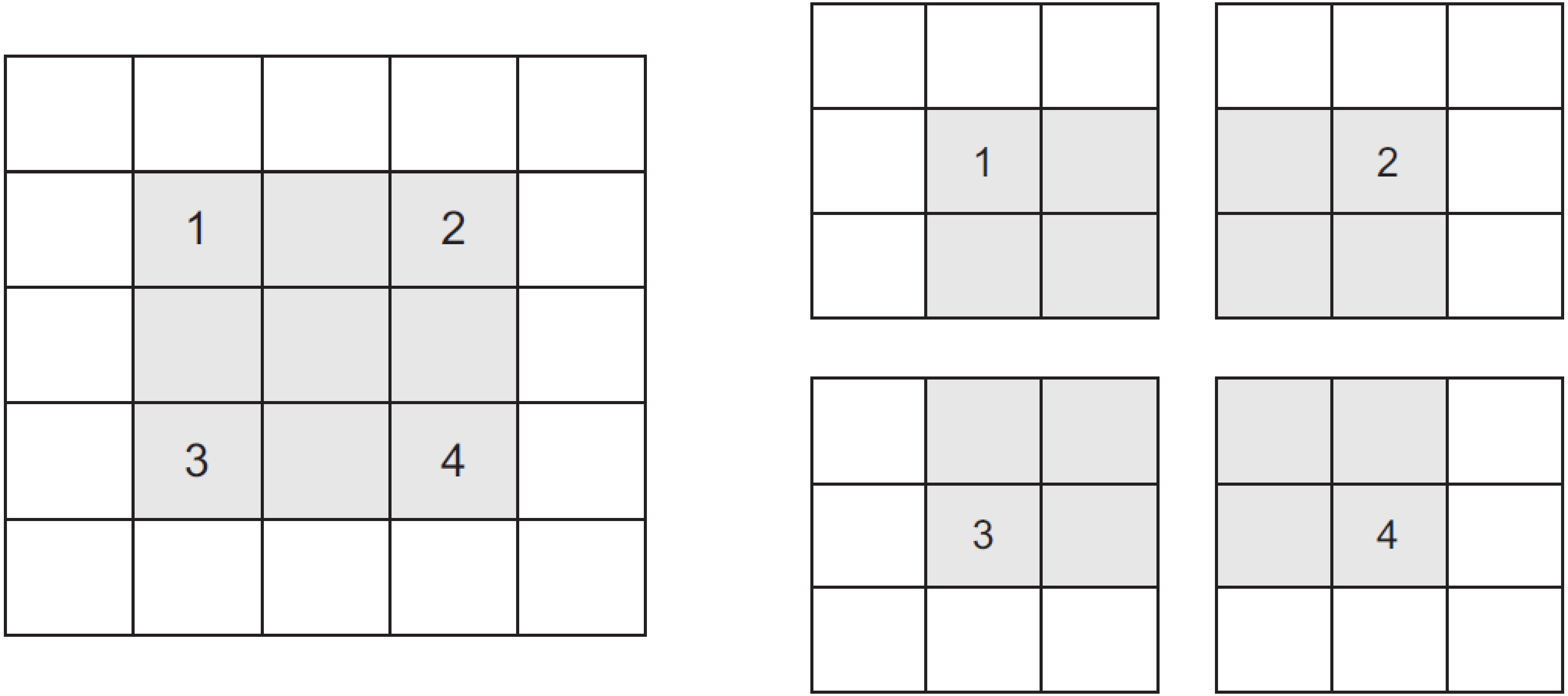


Figure 5.7 3 × 3 convolution patches with 2 × 2 strides

The max-pooling operation

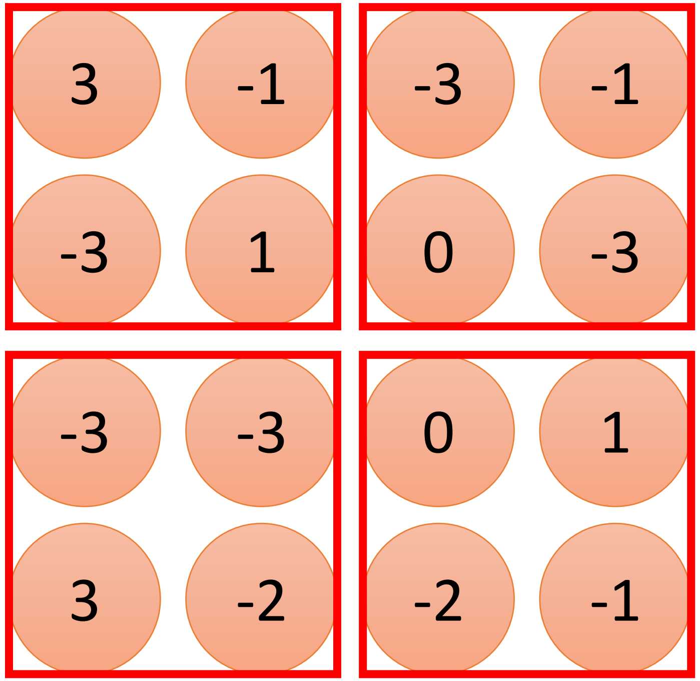
CNN – Max Pooling

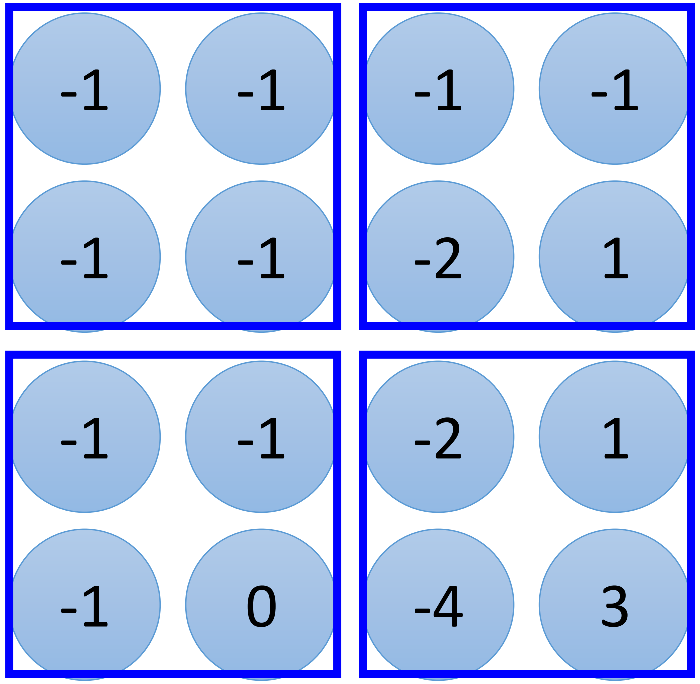
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



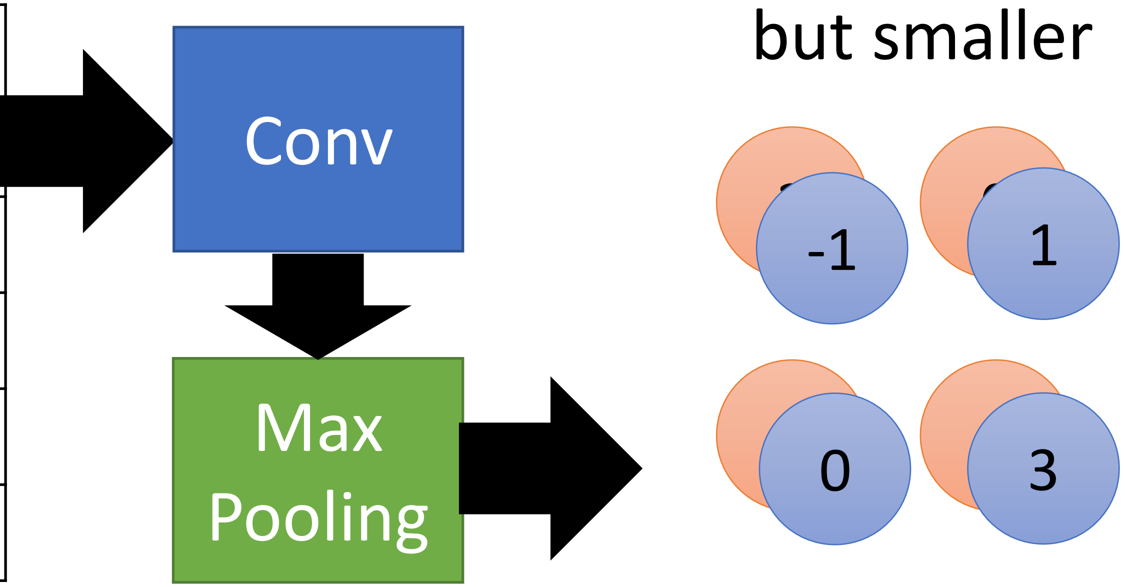


CNN – Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

New image but smaller



2 x 2 image

Each filter is a channel

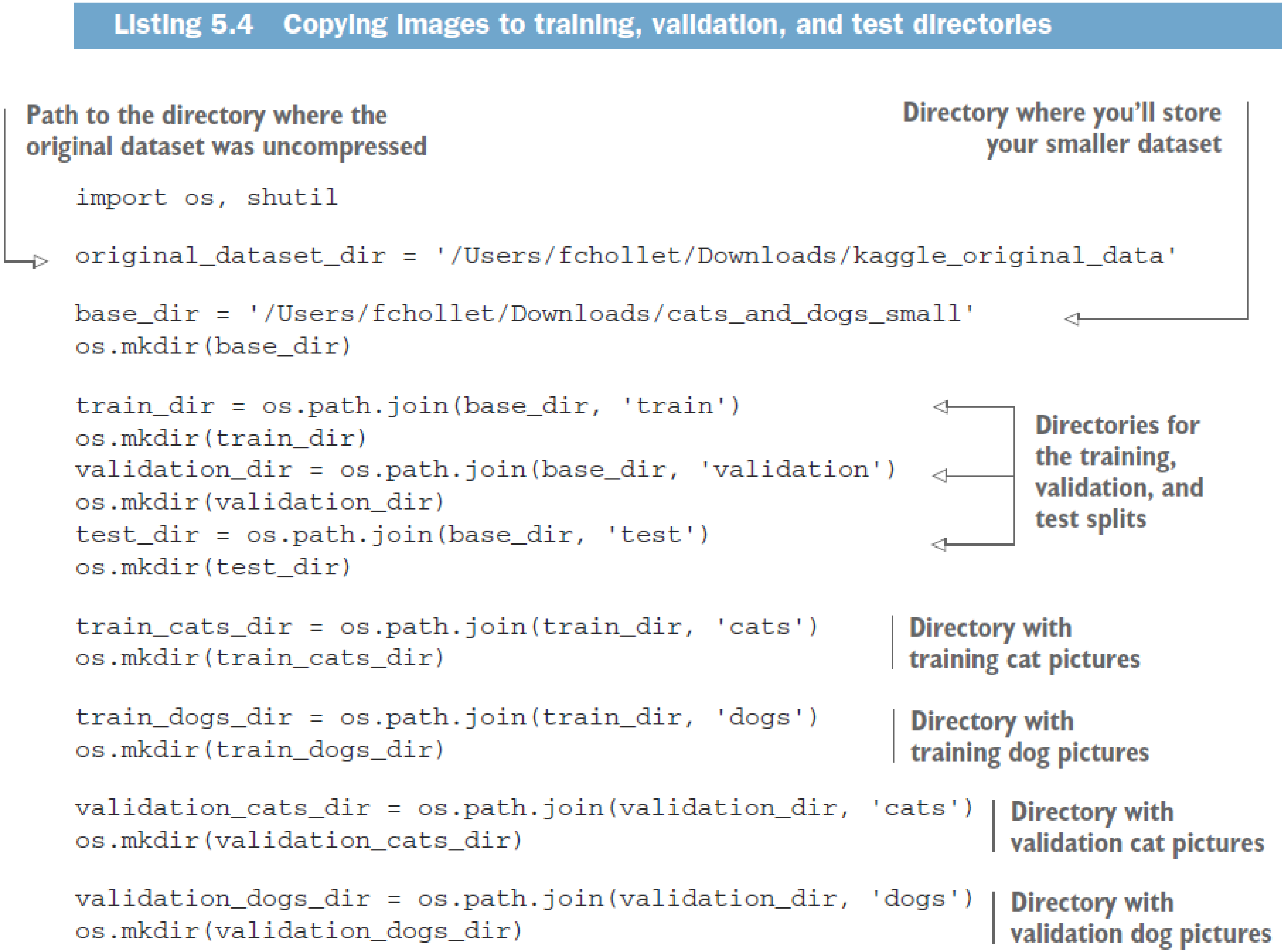
Training a convnet from scratch on a small dataset

Downloading the data

The Dogs vs. Cats dataset that you'll use isn't packaged with Keras. It was made available by Kaggle as part of a computer-vision competition in late 2013, back when convnets weren't mainstream. You can download the original dataset from www.kaggle.com/c/dogs-vs-cats/data (you'll need to create a Kaggle account if you don't already have one—don't worry, the process is painless).

Unsurprisingly, the dogs-versus-cats Kaggle competition in 2013 was won by entrants who used convnets. The best entries achieved up to 95% accuracy. In this example, you'll get fairly close to this accuracy (in the next section), even though you'll train your models on less than 10% of the data that was available to the competitors.

This dataset contains 25,000 images of dogs and cats (12,500 from each class) and is 543 MB (compressed). After downloading and uncompressing it, you'll create a new dataset containing three subsets: a training set with 1,000 samples of each class, a validation set with 500 samples of each class, and a test set with 500 samples of each class.



Listing 5.4 Copying images to training, validation, and test directories

Path to the directory where the original dataset was uncompressed

Directory where you'll store your smaller dataset

```
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)

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

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

Directories for the training, validation, and test splits

Directory with training cat pictures

Directory with training dog pictures

Directory with validation cat pictures

Directory with validation dog pictures

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

Directory with test cat pictures

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

Directory with test dog pictures

```
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)
```

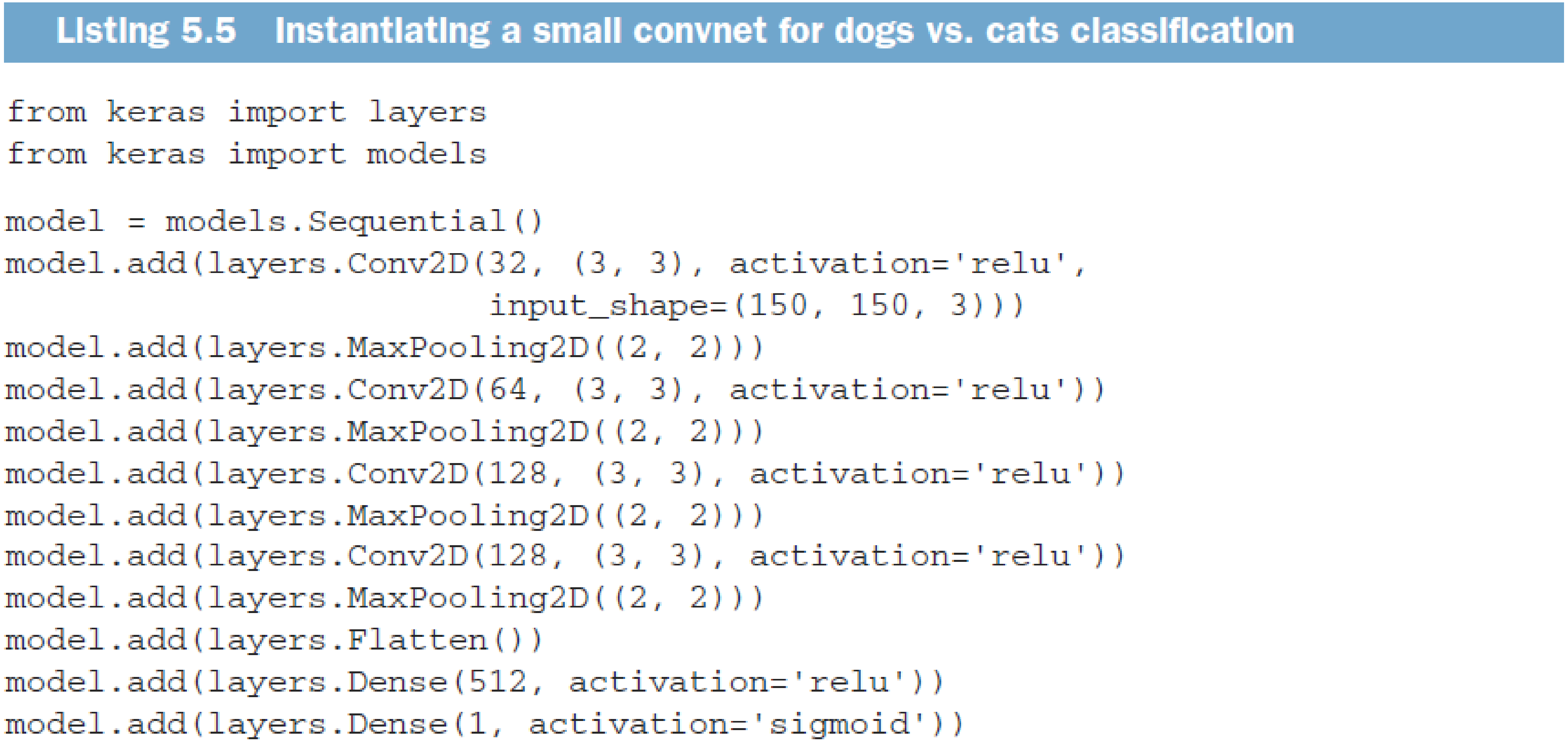
Copies the first 1,000 cat images to train_cats_dir

```
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)
```

Copies the next 500 cat images to validation_cats_dir

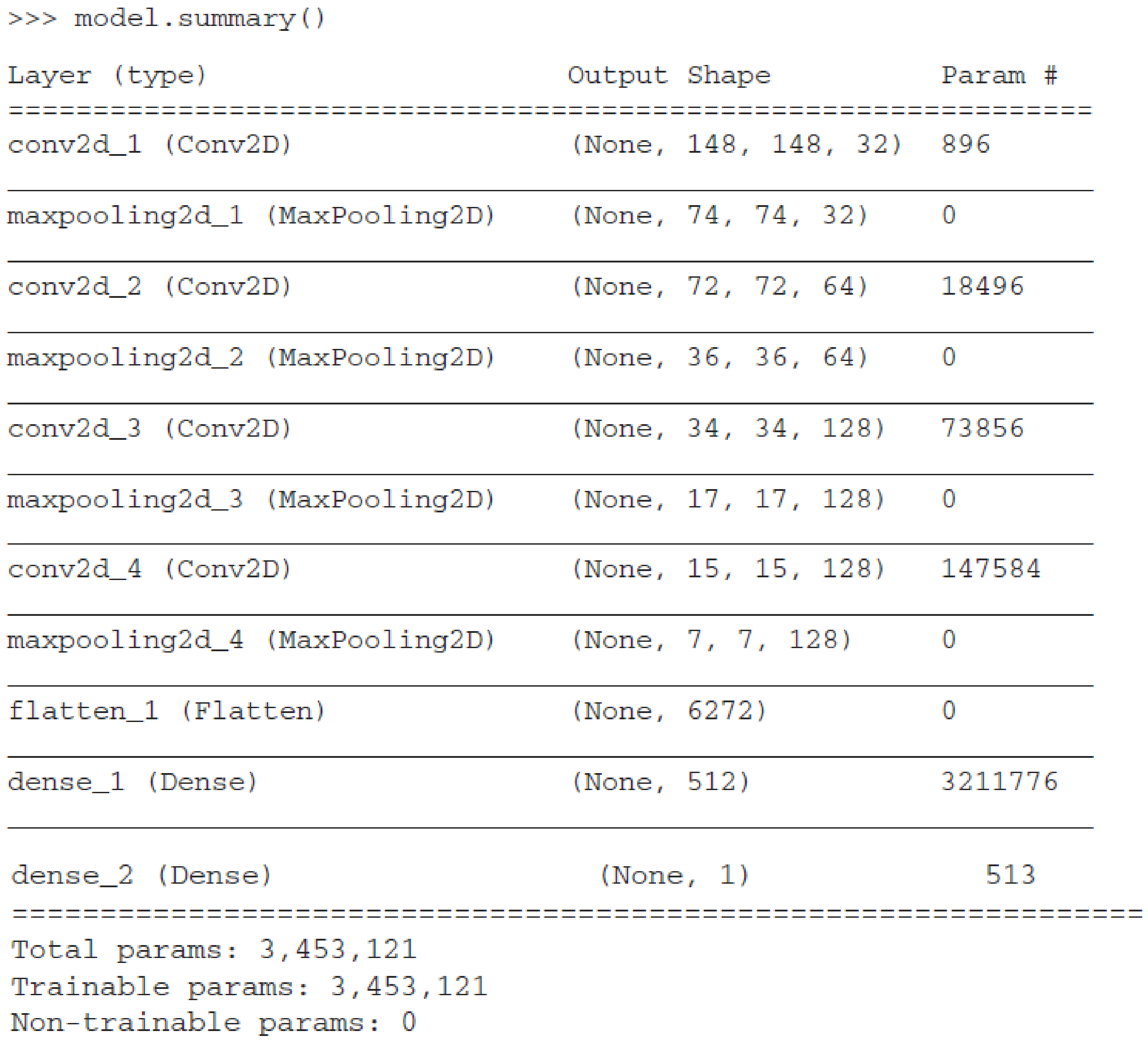
```
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)
```

Copies the next 500 cat images to test_cats_dir



Listing 5.5 Instantiating a small convnet for dogs vs. cats classification

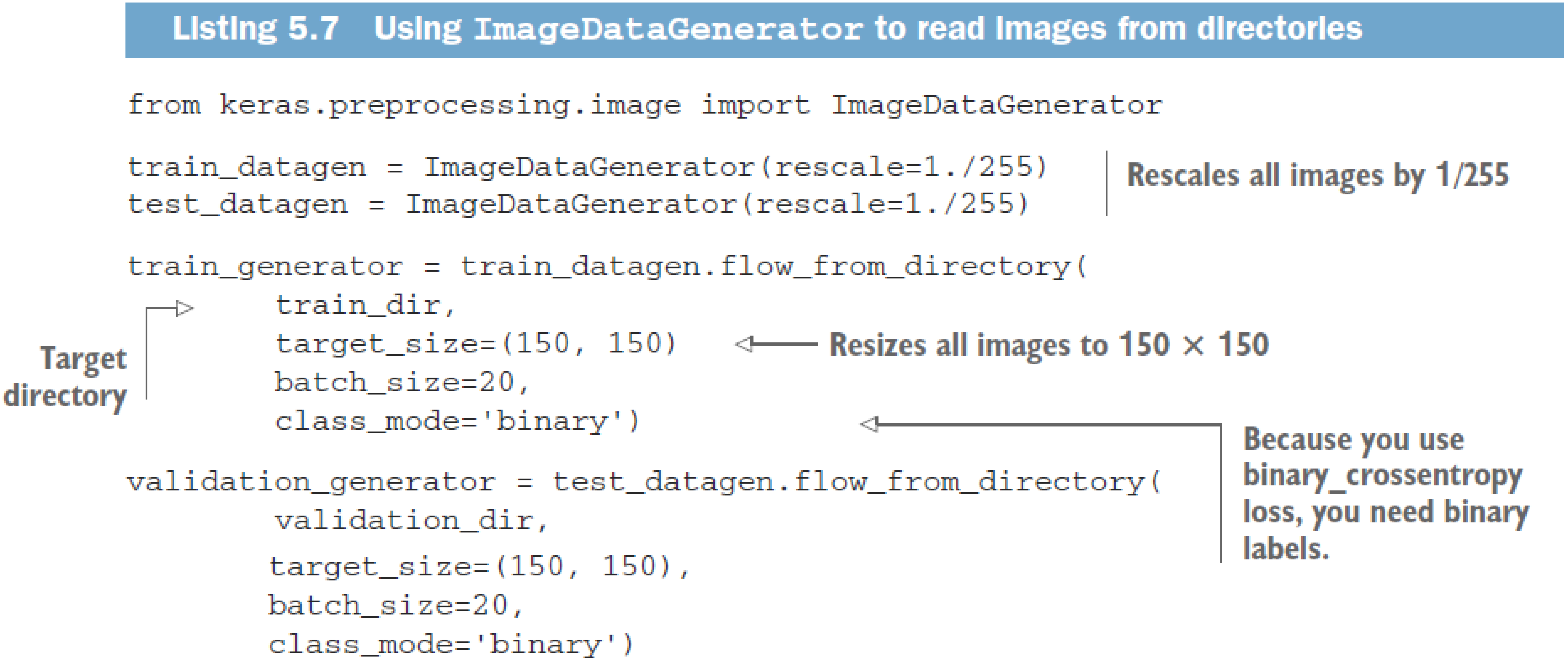
```
from keras import layers
from keras import models

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'))
```



```
>>> 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
```

Listing 5.6 Configuring the model for training

```
from keras import optimizers

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



Listing 5.7 Using `ImageDataGenerator` to read images from directories

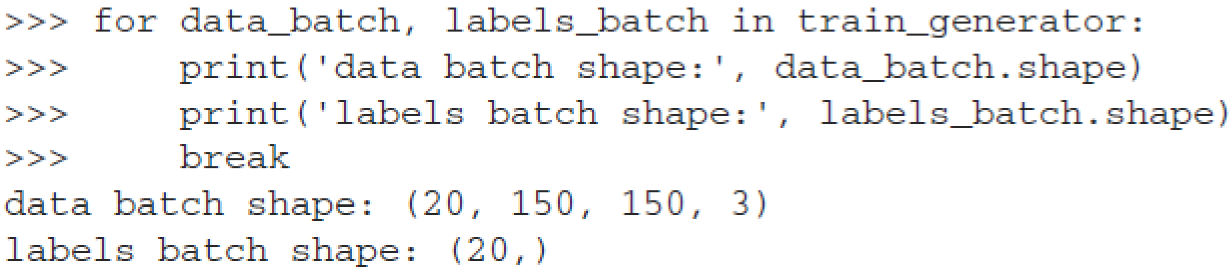
```
from keras.preprocessing.image import ImageDataGenerator

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,
        class_mode='binary')

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

Rescales all images by 1/255

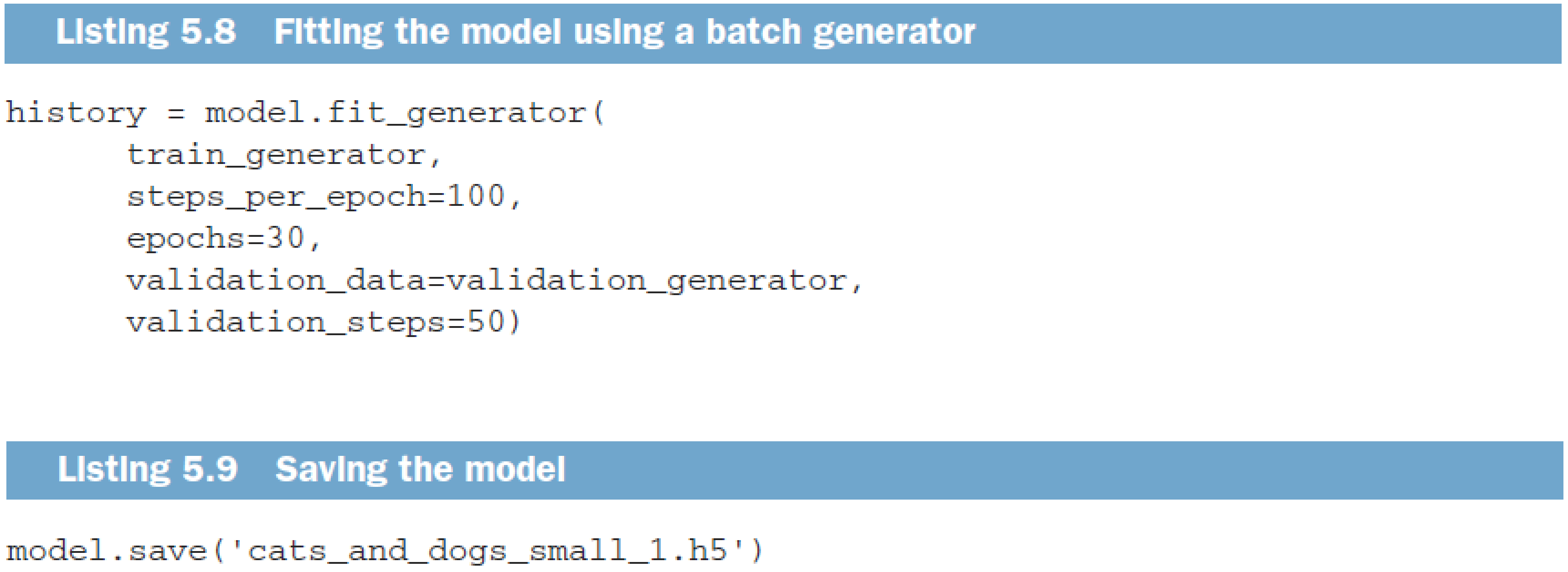
Target directory

Resizes all images to 150 × 150

Because you use binary_crossentropy loss, you need binary labels.



```
>>> 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,)
```



Listing 5.8 Fitting the model using a batch generator

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

Listing 5.9 Saving the model

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

Listing 5.10 Displaying curves of loss and accuracy during training

```
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()

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()
```

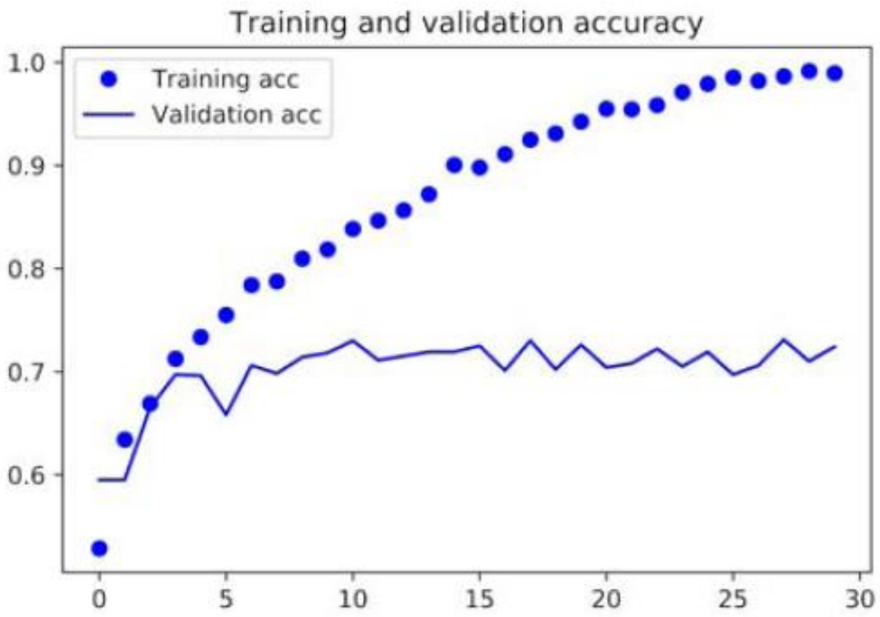


Figure 5.9 Training and validation accuracy

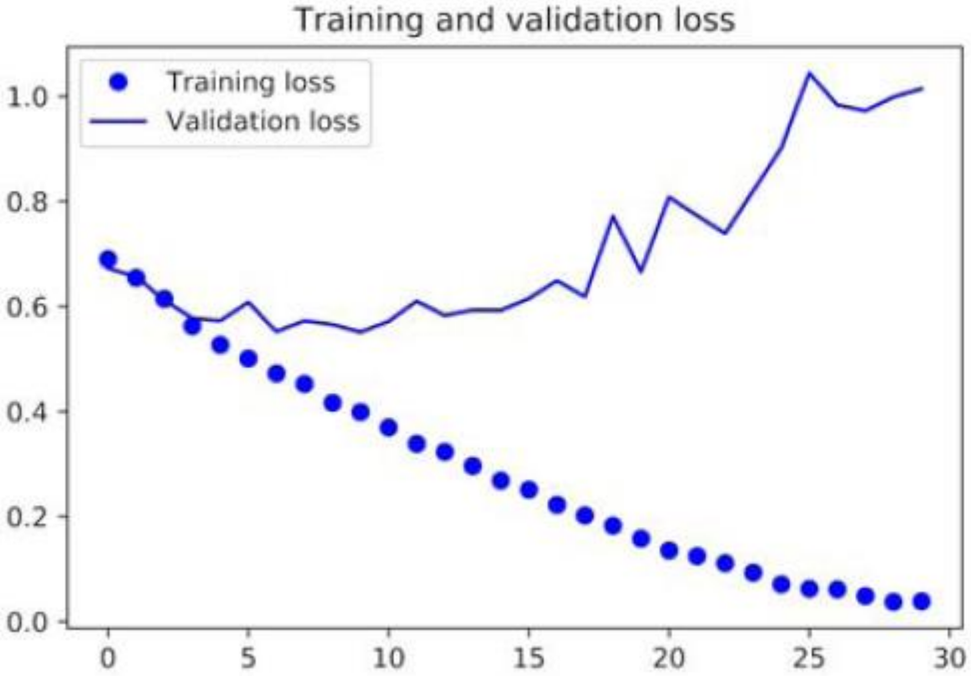


Figure 5.10 Training and validation loss