```
import keras
keras.__version__
```

```
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'))
```

Let's display the architecture of our convnet so far:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_10 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_16 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 5, 5, 64)         0
 g2D)

 conv2d_17 (Conv2D)           (None, 3, 3, 64)          36928

=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

You can see above that the output of every `Conv2D` and `MaxPooling2D` layer is a 3D tensor of shape `(height, width, channels)`. The width and height dimensions tend to shrink as we go deeper in the network. The number of channels is controlled by the first argument passed to the `Conv2D` layers (e.g. 32 or 64).

The next step would be to feed our last output tensor (of shape `(3, 3, 64)`) into a densely-connected classifier network like those you are already familiar with: a stack of `Dense` layers. These classifiers process vectors, which are 1D, whereas our current output is a 3D tensor. So first, we will have to flatten our 3D outputs to 1D, and then add a few `Dense` layers on top:

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

We are going to do 10-way classification, so we use a final layer with 10 outputs and a softmax activation. Now here's what our network looks like:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_10 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_16 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 5, 5, 64)         0
 g2D)

 conv2d_17 (Conv2D)           (None, 3, 3, 64)          36928

 flatten_7 (Flatten)          (None, 576)               0

 dense_14 (Dense)             (None, 64)                36928
```

```
 dense_15 (Dense)            (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

As you can see, our `(3, 3, 64)` outputs were flattened into vectors of shape `(576,)`, before going through two `Dense` layers.

Now, let's train our convnet on the MNIST digits. We will reuse a lot of the code we have already covered in the MNIST example from Chapter 2.

```
from keras.datasets import mnist
from tensorflow.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='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=7, batch_size=100)
```

```
Epoch 1/7
600/600 [==============================] - 43s 70ms/step - loss: 0.0939 - accuracy: 0.9714
Epoch 2/7
600/600 [==============================] - 42s 70ms/step - loss: 0.0532 - accuracy: 0.9837
Epoch 3/7
600/600 [==============================] - 42s 70ms/step - loss: 0.0370 - accuracy: 0.9880
Epoch 4/7
600/600 [==============================] - 42s 70ms/step - loss: 0.0285 - accuracy: 0.9909
Epoch 5/7
600/600 [==============================] - 42s 70ms/step - loss: 0.0230 - accuracy: 0.9926
Epoch 6/7
600/600 [==============================] - 42s 69ms/step - loss: 0.0180 - accuracy: 0.9940
Epoch 7/7
600/600 [==============================] - 42s 69ms/step - loss: 0.0163 - accuracy: 0.9942
<keras.callbacks.History at 0x7f548b433a90>
```

Let's evaluate the model on the test data:

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

```
313/313 [==============================] - 3s 8ms/step - loss: 0.0379 - accuracy: 0.9890
```

```
test_acc
```

```
0.9890000224113464
```

```
test_loss
```

```
0.037858348339796066
```

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

✓ 0s completed at 10:22 PM