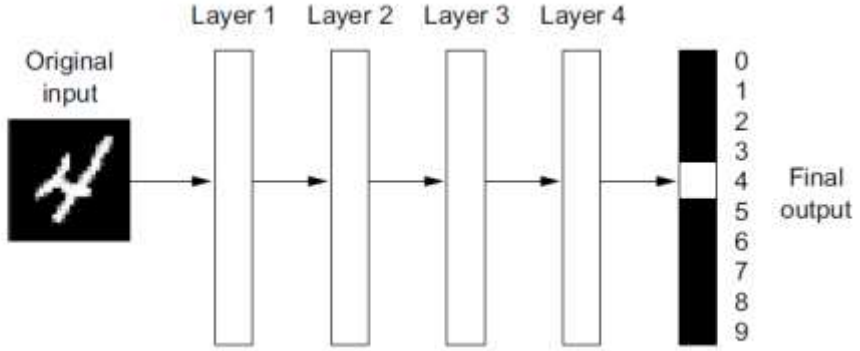
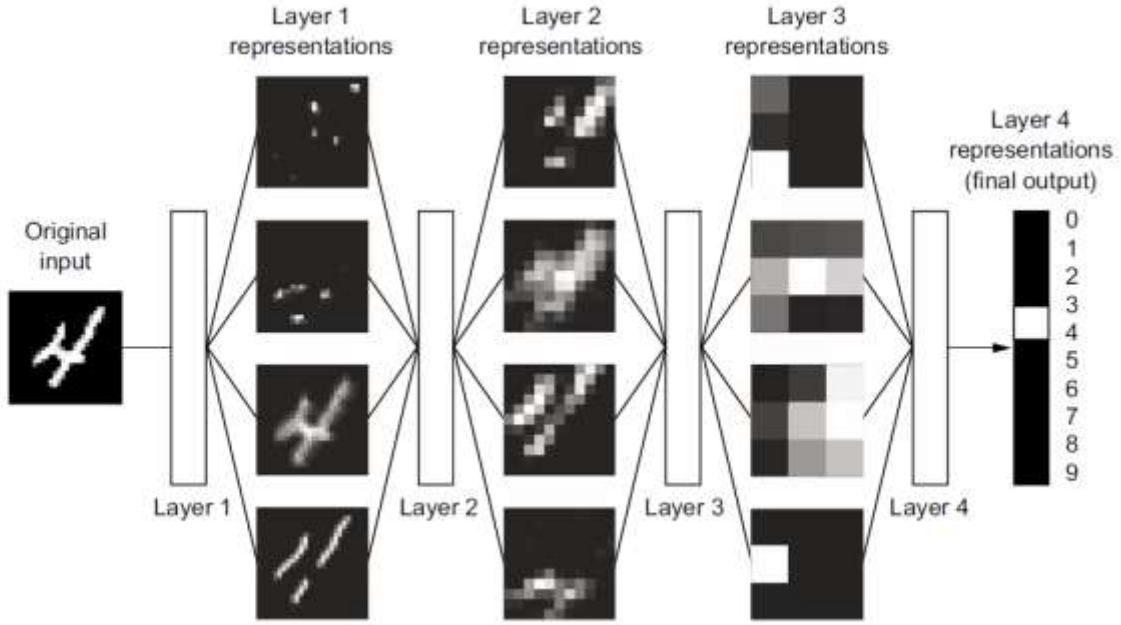

MNIST
Example

The *deep* in Deep Learning





- As you can see in the right figure, the network transforms the digit image into representations that are increasingly different from the original image and increasingly informative about the final result.
- You can think of a deep network as a multistage information distillation process, where information goes through successive filters and comes out increasingly purified

A first look at a neural network

- To classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9).
- MNIST dataset, a classic in the machine-learning community.
- 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s.
- You can think of “solving” MNIST as the “Hello World” of deep learning.
- The MNIST dataset comes preloaded in Keras, in the form of a set of four Numpy arrays.

Figure 2.1 MNIST sample digits

Classification of MNIST Dataset using Keras Library

In [1]:
```
from keras.datasets import mnist
```

Here, train_images and train_labels form the training set, the data that the model will learn from.
The model will then be tested on the test set, test_images and test_labels.

In [2]:
```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

The images are encoded as Numpy arrays, and the labels are an array of digits, ranging from 0 to 9. The images and labels hav
one-to-one correspondence.

In [19]:
```
# Print the shape of train_images: total images, each image is represented by 28 x 28 numpy array
train_images.shape
```

Out[19]:
```
(60000, 784)
```

In [16]:
```
# print the first image values
train_images [0]
```

Out[16]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

In [17]:
```
# print the first label
train_labels [0]
```

Out[17]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

In [4]:
```
# print the total length of train set
len(train_labels)
```

Out[4]:
```
60000
```

In [5]:
```
# print all train images' labels
train_labels
```

Out[5]:
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

In [6]:
```
# Print the shape of test_images: total images, each image is represented by 28 x 28 numpy array
test_images.shape
```

Out[6]:
```
(10000, 28, 28)
```

In [7]:
```
# print the total length or size of test labels
len (test_labels)
```

Out[7]:
```
10000
```

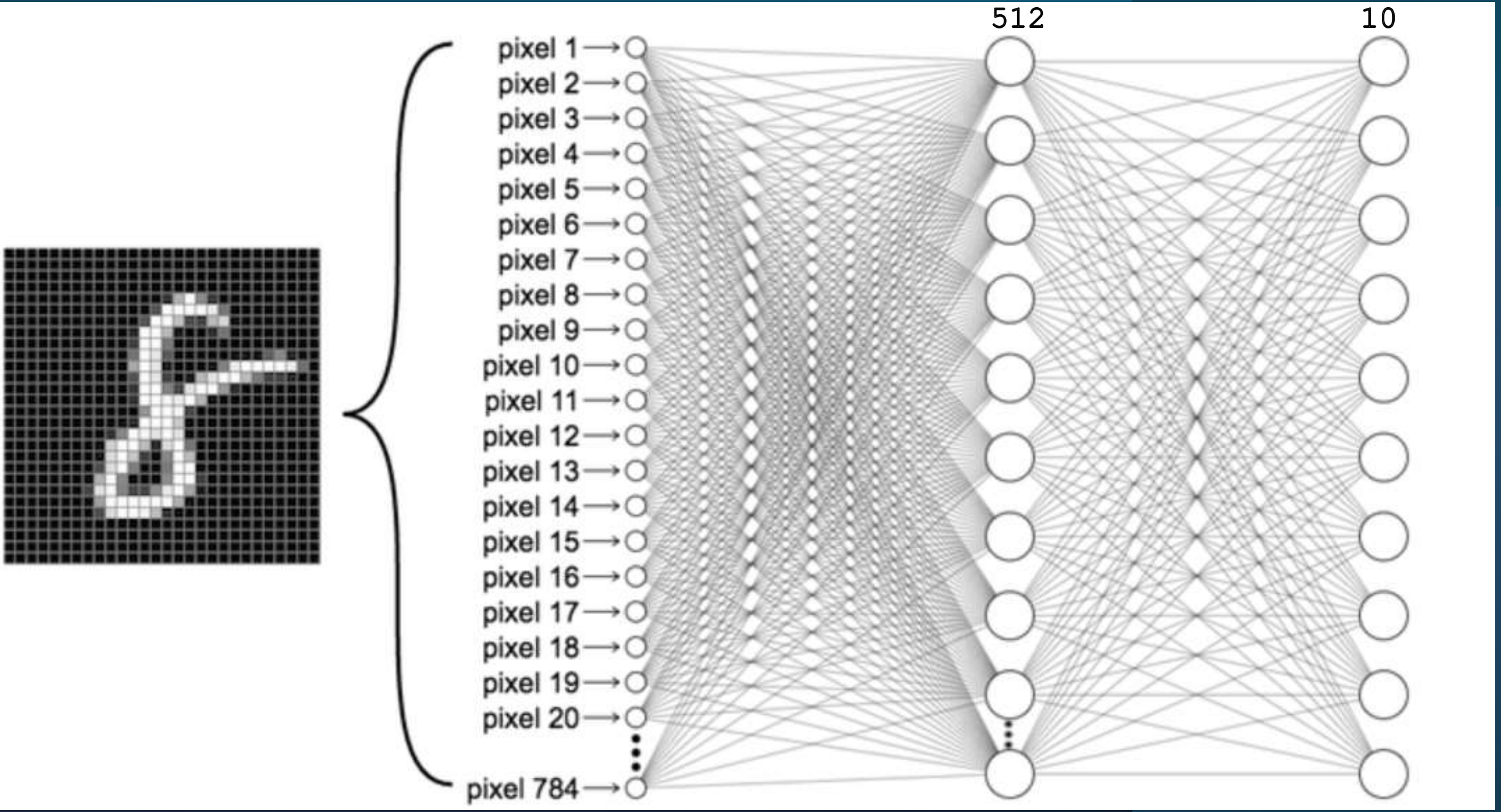
The workflow will be as follows: First, we'll feed the neural network the training data,train_images and train_labels.
The network will then learn to associate images and labels.
Finally, we'll ask the network to produce predictions for test_images, and we'll verify whether these predictions match the labels from test_labels.
Let's build the network–again, remember that you aren't expected to understand everything about this example yet.

The core building block of neural networks is the layer, a data-processing module that you can think of as a filter for data.
Some data goes in, and it comes out in a more useful form.
Specifically, layers extract representations out of the data fed into them–hopefully, representations that are more meaningful for the problem at hand.
Most of deep learning consists of chaining together simple layers that will implement a form of progressive data distillation.
A deep-learning model is like a sieve for data processing, made of a succession of increasingly refined data filters–the layers.

In [9]:
```
#we are using keras library and importing models and layers from it.
from keras import models
from keras import layers
```

Here, our network consists of a sequence of two Dense layers, which are densely connected (also called fully connected) neural layers.
The second (and last) layer is a 10-way softmax layer, which means it will return an array of 10 probability scores (summing to 1).
Each score will be the probability that the current digit image belongs to one of our 10 digit classes.

In [10]:
```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```


512
10
pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 12
pixel 13
pixel 14
pixel 15
pixel 16
pixel 17
pixel 18
pixel 19
pixel 20
pixel 784

To make the network ready for training, we need to pick three more things, as part of the compilation step:
1- A loss function – How the network will be able to measure its performance on the training data, and thus how it will be able to steer itself in the right direction.
2- An optimizer—The mechanism through which the network will update itself based on the data it sees and its loss function.
3- Metrics to monitor during training and testing—Here, we'll only care about accuracy (the fraction of the images that were correctly classified).

In [11]:
```
network.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

Before training, we'll preprocess the data by reshaping it into the shape the network expects and scaling it so that all values are in the [0, 1] interval.
Previously, our training images, for instance, were stored in an array of shape (60000, 28, 28) of type uint8 with values in the [0, 255] interval.
We transform it into a float32 array of shape (60000, 28 * 28) with values between 0 and 1.

In [12]:
```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

We also need to categorically encode the labels

In [13]:
```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

We're now ready to train the network, which in Keras is done via a call to the network's fit method—we fit the model to its training data:
Two quantities are displayed during training: the loss of the network over the training data, and the accuracy of the network over the training data.
We quickly reach an accuracy of 0.989 (98.9%) on the training data.

In [14]:
```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 6s 11ms/step - loss: 0.2587 - accuracy: 0.9244
Epoch 2/5
469/469 [==============================] - 5s 11ms/step - loss: 0.1052 - accuracy: 0.9682
Epoch 3/5
469/469 [==============================] - 5s 11ms/step - loss: 0.0689 - accuracy: 0.9795
Epoch 4/5
469/469 [==============================] - 5s 11ms/step - loss: 0.0500 - accuracy: 0.9843
Epoch 5/5
469/469 [==============================] - 5s 11ms/step - loss: 0.0375 - accuracy: 0.9883
```

Out[14]:
```
<keras.callbacks.History at 0x17ec4f0ece0>
```

check that the model performs well on the test set, too.

In [18]:
```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0722 - accuracy: 0.9776
test_acc: 0.9775999784469604
```

A first look at a neural network

- The test-set accuracy turns out to be 97.8%—that's quite a bit lower than the training set accuracy.
- This gap between training accuracy and test accuracy is an example of *overfitting*: the fact that machine-learning models tend to perform worse on new data than on their training data.
- You just saw how you can build and train a neural network to classify handwritten digits in less than 20 lines of Python code.
- In the subsequent lecture, we will go into detail about every moving piece we just previewed and clarify what's going on behind the scenes.