Module 6: Deep Learning

Keras is a high-level neural network API written in Python that makes it easy to build and train deep learning models. It supports a wide range of neural network architectures, including CNNs, which are particularly useful for computer vision tasks such as image classification, object detection, and segmentation.

CNNs are a type of neural network that is specifically designed to process image data. They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers, which work together to extract and classify features from images.

Convolutional layers use a set of learnable filters to convolve over the input image, resulting in a set of feature maps that capture different aspects of the image. Pooling layers are used to downsample the feature maps and reduce the dimensionality of the data, while fully connected layers are used to perform the final classification.

To train a CNN, we need a large dataset of labeled images. We can use this data to optimize the weights of the network using techniques such as stochastic gradient descent and backpropagation. Once trained, the CNN can be used to classify new images with high accuracy.

Keras provides a simple and intuitive interface for building and training CNNs. It comes with a wide range of pre-trained models and datasets that can be used for various computer vision tasks, and it also allows for customization and fine-tuning of these models to suit specific needs.

In the following lessons, we will dive deeper into the world of Keras and CNNs, exploring different architectures, techniques, and applications for computer vision. By the end of this course, you will have a solid understanding of how to use Keras to build powerful deep learning models for image classification and other computer vision tasks.

IMDB dataset: a set of 50,000 highly polarized reviews from the Internet Movie Database. They're split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.

IMDB dataset comes packaged with Keras. It has already been preprocessed: the reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary. This enables us to focus on model building, training, and evaluation.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import layers
```

```
from keras.datasets import imdb
from keras.datasets import mnist
```

In [ ]:
```
# Basic Neural Network.  It has 11 input nodes and 1 output node.

model = keras.Sequential([
    layers.Dense(units=1, input_shape=[11])
])
```

In [ ]:
```
model.build()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1)                 12

=================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
w, b = model.weights

print("Weights\n{}\n\nBias\n{}".format(w, b))
```

```
Weights
<tf.Variable 'dense/kernel:0' shape=(11, 1) dtype=float32, numpy=
array([[ 0.42309552],
       [ 0.45354646],
       [ 0.65688664],
       [ 0.14193541],
       [ 0.31000203],
       [ 0.41863483],
       [ 0.05542612],
       [-0.03838784],
       [ 0.35057467],
       [-0.6350689 ],
       [ 0.51117724]], dtype=float32)>

Bias
<tf.Variable 'dense/bias:0' shape=(1,) dtype=float32, numpy=array([0.], dtype=fl
oat32)>
```

MNist dataset:

A set of 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 Neural Networks.

In [ ]:
```
(train_data, train_labels), (test_data, test_labels) = mnist.load_data()
```

In [ ]:
```
train_data.shape
```

Out[ ]: (60000, 28, 28)

In [ ]:
```
train_data.dtype
```

Out[ ]: dtype('uint8')

In [ ]:
```
train_data[0]
```

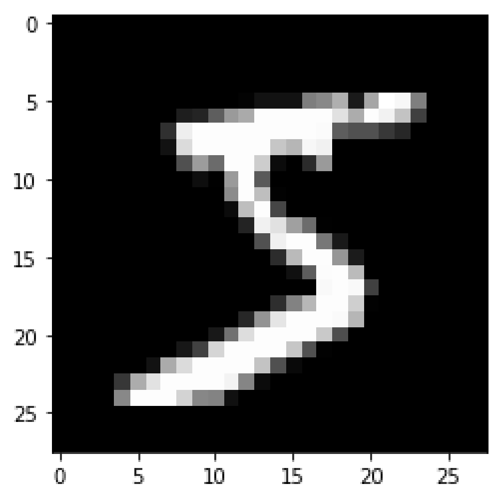
Out[ ]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   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,   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,   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,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   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, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,
```

```
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
       148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
       253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
       253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
       195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
        11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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,   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,   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,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0]], dtype=uint8)
```

In [ ]:

```python
plt.imshow(train_data[0], cmap='gray')
```

Out[ ]: <matplotlib.image.AxesImage at 0x7fcfa9fbfeb0>



In [ ]:

```python
train_labels[0]
```

Out[ ]: 5

In [ ]:

```python
#Convert Data in 2D array.  1st dimension is the number of images, 2nd dimension
# Normalize image data to be between 0 and 1.
train_images = train_data.reshape((60000, 28 * 28))
```

```
train_images = train_images.astype('float32') / 255
test_images = test_data.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

In [ ]:
```
train_images.shape
```

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

In [ ]:
```
train_images.dtype
```

Out[ ]:
```
dtype('float32')
```

In [ ]:
```
train_images[0]
```

Out[ ]:
```
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.        ,
       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.        ,
       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.        ,
       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.        ,
       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.        ,
       0.        , 0.        , 0.01176471, 0.07058824, 0.07058824,
       0.07058824, 0.49411765, 0.53333336, 0.6862745 , 0.10196079,
       0.6509804 , 1.        , 0.96862745, 0.49803922, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.11764706, 0.14117648, 0.36862746, 0.6039216 ,
       0.6666667 , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.88235295, 0.6745098 , 0.99215686, 0.9490196 ,
       0.7647059 , 0.2509804 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.19215687, 0.93333334,
       0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.9843137 , 0.3647059 ,
       0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.07058824, 0.85882354, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.7764706 , 0.7137255 ,
0.96862745, 0.94509804, 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.3137255 , 0.6117647 , 0.41960785, 0.99215686, 0.99215686,
0.8039216 , 0.04313726, 0.   , 0.16862746, 0.6039216 ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.05490196,
0.00392157, 0.6039216 , 0.99215686, 0.3529412 , 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.   , 0.54509807,
0.99215686, 0.74509805, 0.00784314, 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.04313726, 0.74509805, 0.99215686,
0.27450982, 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.   , 0.13725491, 0.94509804, 0.88235295, 0.627451 ,
0.42352942, 0.00392157, 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.31764707, 0.9411765 , 0.99215686, 0.99215686, 0.46666667,
0.09803922, 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.1764706 ,
0.7294118 , 0.99215686, 0.99215686, 0.5882353 , 0.10588235,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.0627451 , 0.3647059 ,
0.9882353 , 0.99215686, 0.73333335, 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.   , 0.9764706 , 0.99215686,
0.9764706 , 0.2509804 , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.18039216, 0.50980395,
0.7176471 , 0.99215686, 0.99215686, 0.8117647 , 0.00784314,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.   ,
0.   , 0.   , 0.   , 0.   , 0.15294118,
0.5803922 , 0.8980392 , 0.99215686, 0.99215686, 0.99215686,

```
       0.98039216, 0.7137255 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.09411765, 0.44705883, 0.8666667 , 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.7882353 , 0.30588236, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.09019608, 0.25882354, 0.8352941 , 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.7764706 , 0.31764707,
       0.00784314, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.07058824, 0.67058825, 0.85882354,
       0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.7647059 ,
       0.3137255 , 0.03529412, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.21568628, 0.6745098 ,
       0.8862745 , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.95686275, 0.52156866, 0.04313726, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.53333336, 0.99215686, 0.99215686, 0.99215686,
       0.83137256, 0.5294118 , 0.5176471 , 0.0627451 , 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.        , 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.        , 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.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ], dtype=float32)
```

In [ ]:

```python
model = keras.Sequential([
    layers.Dense(512, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.build((None, 28 * 28))
model.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_52 (Dense)            (None, 512)               401920

 dense_53 (Dense)            (None, 10)                5130
```

```
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(optimizer='rmsprop',loss="sparse_categorical_crossentropy",metrics
```

In [ ]:

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 2s 4ms/step - loss: 0.2556 - accurac
y: 0.9250
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.1035 - accurac
y: 0.9686
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0679 - accurac
y: 0.9789
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0491 - accurac
y: 0.9853
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0359 - accurac
y: 0.9894
```

Out[ ]:

```
<keras.callbacks.History at 0x7fcdaf2f3430>
```

In [ ]:

```
results = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.0696 - accurac
y: 0.9791
```

In [ ]:

```
# Predict the class of the first image in the test set.
model.predict(test_images)[0]
```

Out[ ]:

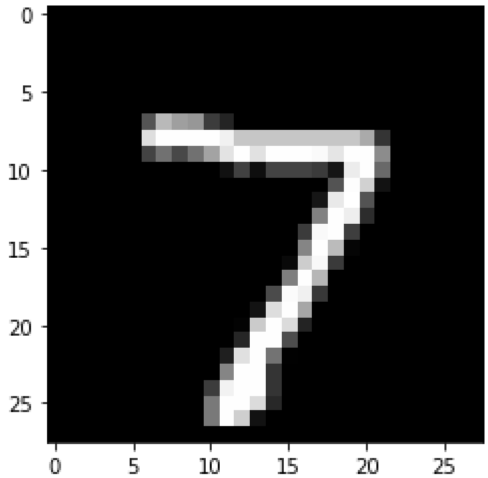
```
array([7.5444611e-09, 4.9639398e-10, 6.8796629e-07, 5.0879051e-05,
       3.3655877e-13, 1.7893797e-07, 3.2480050e-15, 9.9994802e-01,
       2.8783942e-09, 1.2779927e-07], dtype=float32)
```

In [ ]:

```
plt.imshow(test_data[0], cmap='gray')
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7fcf7a9e23a0>
```



IMDB Reviews Dataset

In [ ]:

```
# Load the data
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=
```

In [ ]:

```
# The variables train_data and test_data are lists of reviews; each review is a
# word indices (encoding a sequence of words).
train_data[2]
```

Out[ ]:

```
[1,
 14,
 47,
 8,
 30,
 31,
 7,
 4,
 249,
 108,
 7,
 4,
 5974,
 54,
 61,
 369,
 13,
 71,
 149,
 14,
 22,
 112,
 4,
 2401,
 311,
 12,
 16,
 3711,
 33,
 75,
 43,
 1829,
 296,
 4,
```

86,
320,
35,
534,
19,
263,
4821,
1301,
4,
1873,
33,
89,
78,
12,
66,
16,
4,
360,
7,
4,
58,
316,
334,
11,
4,
1716,
43,
645,
662,
8,
257,
85,
1200,
42,
1228,
2578,
83,
68,
3912,
15,
36,
165,
1539,
278,
36,
69,
2,
780,
8,
106,
14,
6905,
1338,
18,
6,
22,
12,
215,
28,
610,
40,
6,
87,
326,
23,

```
 2300,
 21,
 23,
 22,
 12,
 272,
 40,
 57,
 31,
 11,
 4,
 22,
 47,
 6,
 2307,
 51,
 9,
 170,
 23,
 595,
 116,
 595,
 1352,
 13,
 191,
 79,
 638,
 89,
 2,
 14,
 9,
 8,
 106,
 607,
 624,
 35,
 534,
 6,
 227,
 7,
 129,
 113]
```

In [ ]:

```python
# train_labels and test_labels are lists of 0s and 1s,
# where 0 stands for negative and 1 stands for positive
train_labels[0]
```

Out[ ]: 1

In [ ]:

```python
# We're restricting ourselves to the top 10,000 most frequent words, no word
# index will exceed 10,000.

max([max(sequence) for sequence in train_data])
```

Out[ ]: 9999

Convert the review back into words.

In [ ]:

```
word_index = imdb.get_word_index() # word_index is a dictionary mapping words to
 # Reverses it, mapping integer indices to words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
 #Decodes the review. Note that the indices are offset by 3 because 0, 1, and 2
decoded_review = " ".join([reverse_word_index.get(i - 3, "?") for i in train_dat
```

In [ ]:

```
decoded_review
```

Out[ ]:

```
"? this film was just brilliant casting location scenery story direction everyon
e's really suited the part they played and you could just imagine being there ro
bert ? is an amazing actor and now the same being director ? father came from th
e same scottish island as myself so i loved the fact there was a real connection
with this film the witty remarks throughout the film were great it was just bril
liant so much that i bought the film as soon as it was released for ? and would
recommend it to everyone to watch and the fly fishing was amazing really cried a
t the end it was so sad and you know what they say if you cry at a film it must
have been good and this definitely was also ? to the two little boy's that playe
d the ? of norman and paul they were just brilliant children are often left out
of the ? list i think because the stars that play them all grown up are such a b
ig profile for the whole film but these children are amazing and should be prais
ed for what they have done don't you think the whole story was so lovely because
it was true and was someone's life after all that was shared with us all"
```

Preparing the data You can’t directly feed lists of integers into a neural network. They all have different lengths, but a neural network expects to process contiguous batches of data. You have to turn your lists into tensors. There are two ways to do that:

- Pad your lists so that they all have the same length, turn them into an integer tensor of shape (samples, max_length), and start your model with a layer capable of handling such integer tensors (the Embedding layer, which we’ll cover in detail later in the book).
- Multi-hot encode your lists to turn them into vectors of 0s and 1s. This would mean, for instance, turning the sequence [8, 5] into a 10,000-dimensional vector that would be all 0s except for indices 8 and 5, which would be 1s. Then you could use a Dense layer, capable of handling floating-point vector data, as the first layer in your model. Let’s go with the latter solution to vectorize the data, which you’ll do manually for maximum clarity.

In [ ]:

```
# Multi-hot encoding

def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        # Sets specific indices of results[i] to 1s
        results[i, sequence] = 1.
    return results
x_train = vectorize_sequences(train_data) # Vectorized training data
x_test = vectorize_sequences(test_data) # Vectorized test data
```

In [ ]:

```
x_train[0]
```

Out[ ]: array([0., 1., 1., ..., 0., 0., 0.])

In [ ]:

```
# Vectorize the labels

y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

In [ ]:

```
model = keras.Sequential([
layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"),
layers.Dense(1, activation="sigmoid")
])

model.build((None, 10000))
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_32 (Dense)            (None, 16)                160016

 dense_33 (Dense)            (None, 16)                272

 dense_34 (Dense)            (None, 1)                 17

=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
# Compile the model. RMSprop optimizer and binary_crossentropy loss function.
model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accurac

# Train the model with the training data and labels.  Train on the full training
# Update the weights after each batch of 512 samples. Add history to keep track
history = model.fit(x_train, y_train, epochs=4, batch_size=512)
print("Evaluate on test data")

# Evaluate the model on the test data.
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 1s 9ms/step - loss: 0.4620 - accuracy:
0.8226
Epoch 2/4
49/49 [==============================] - 0s 9ms/step - loss: 0.2685 - accuracy:
0.9064
Epoch 3/4
49/49 [==============================] - 0s 9ms/step - loss: 0.2064 - accuracy:
0.9260
Epoch 4/4
49/49 [==============================] - 0s 9ms/step - loss: 0.1720 - accuracy:
0.9386
Evaluate on test data
782/782 [==============================] - 1s 1ms/step - loss: 0.2942 - accurac
y: 0.8828
```

In [ ]:

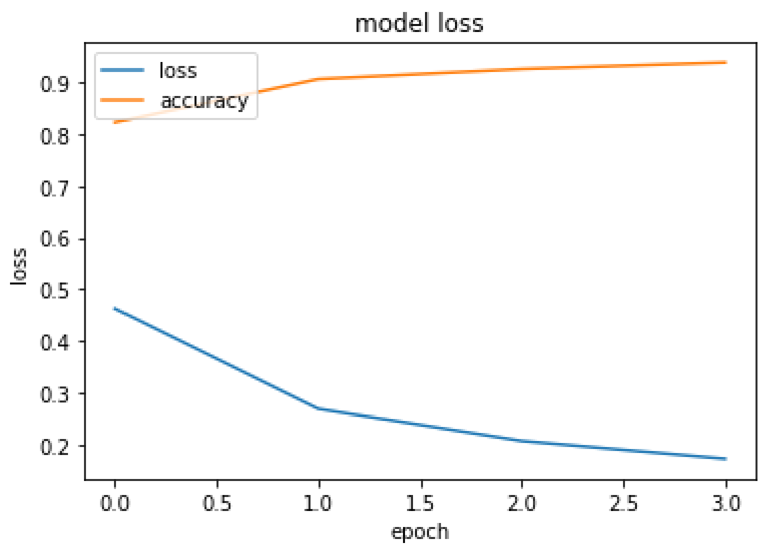
```
# Make predictions on the test data on the first data point.
model.predict(x_test[:1])
```

Out[ ]:

```
array([[0.18801302]], dtype=float32)
```

In [ ]:

```
# Plot the training loss and accuracy
plt.plot(history.history["loss"])
plt.plot(history.history["accuracy"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["loss", "accuracy"], loc="upper left")
plt.show()
```



Here's what you should take away from this example:

- You usually need to do quite a bit of preprocessing on your raw data in order to be able to feed it—as tensors—into a neural network. Sequences of words can be encoded as binary vectors, but there are other encoding options too.
- Stacks of Dense layers with relu activations can solve a wide range of problems (including sentiment classification), and you'll likely use them frequently.
- In a binary classification problem (two output classes), your model should end with a Dense layer with one unit and a sigmoid activation: the output of your model should be a scalar between 0 and 1, encoding a probability.
- With such a scalar sigmoid output on a binary classification problem, the loss function you should use is binary_crossentropy.
- The rmsprop optimizer is generally a good enough choice, whatever your problem. That's one less thing for you to worry about.
- As they get better on their training data, neural networks eventually start overfitting and end up obtaining increasingly worse results on data they've never seen before. Be sure to always monitor performance on data that is outside of the training set.

The first argument being passed to each Dense layer is the number of units in the layer: the dimensionality of representation space of the layer. You remember from chapters 2 and 3 that each such Dense layer with a relu activation implements the following chain of tensor operations: output = relu(dot(input, W) + b)

Convolutional Neural Networks

Let's walk through an example of how to build a CNN using Keras. We will use the Minst dataset, which contains 60,000 training images and 10,000 test images of handwritten digits. The goal is to train a CNN to classify these images into their correct digit categories.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.datasets import mnist

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [ ]:

```
x_train.shape
```

Out[ ]:

```
(60000, 28, 28)
```

In [ ]:

```
x_train.dtype
```

Out[ ]:

```
dtype('uint8')
```

In [ ]:

```
x_train[0]
```

Out[ ]:

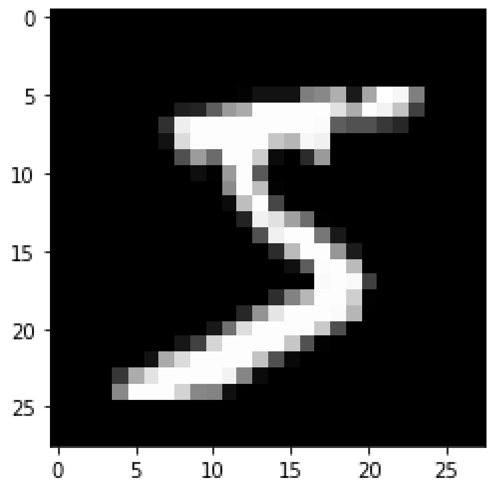
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   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,   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,   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,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
```

```
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   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, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
        148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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,   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,   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,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

In [ ]:

```
plt.imshow(x_train[0], cmap='gray')
```

Out[ ]: <matplotlib.image.AxesImage at 0x7fcfae840e50>



In [ ]:
```
y_train.shape
```

Out[ ]: (60000,)

In [ ]:
```
print(y_train[0])
```

5

In [ ]:
```
#Prepare the data for training

# Convert the data to float32. This is the data type tf.keras layers expect.  Th
x_train = x_train.astype(np.float32) / 255
x_test = x_test.astype(np.float32) / 255

# Reshape the data to four-dimensional tensor (sample, rows, columns, channels).
x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28,28, 1)
```

In [ ]:
```
y_train[0]
```

Out[ ]: 5

In [ ]:
```
#Prepare the labels for training

# How many classes are there?
num_classes = 10


# Convert the labels to binary class matrices. This is called one-hot encoding.
# This is done because CNN uses categorical cross entropy as its loss function.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

print(y_train[0])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In [ ]:

```python
model = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 13, 13, 32)       0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0         
 2D)                                                             
                                                                 
 flatten_2 (Flatten)         (None, 1600)              0         
                                                                 
 dropout_2 (Dropout)         (None, 1600)              0         
                                                                 
 dense_35 (Dense)            (None, 10)                16010     
                                                                 
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```python
batch_size = 128
epochs = 10

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accur

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_spl
```

```
Epoch 1/10
422/422 [==============================] - 17s 39ms/step - loss: 0.3697 - accura
cy: 0.8885 - val_loss: 0.0804 - val_accuracy: 0.9792
Epoch 2/10
422/422 [==============================] - 589s 1s/step - loss: 0.1100 - accurac
y: 0.9657 - val_loss: 0.0587 - val_accuracy: 0.9848
Epoch 3/10
422/422 [==============================] - 17s 41ms/step - loss: 0.0822 - accura
cy: 0.9749 - val_loss: 0.0489 - val_accuracy: 0.9870
Epoch 4/10
422/422 [==============================] - 16s 38ms/step - loss: 0.0696 - accura
cy: 0.9782 - val_loss: 0.0469 - val_accuracy: 0.9875
```

```
Epoch 5/10
422/422 [==============================] - 17s 39ms/step - loss: 0.0609 - accura
cy: 0.9809 - val_loss: 0.0368 - val_accuracy: 0.9910
Epoch 6/10
422/422 [==============================] - 16s 39ms/step - loss: 0.0535 - accura
cy: 0.9828 - val_loss: 0.0345 - val_accuracy: 0.9910
Epoch 7/10
422/422 [==============================] - 16s 39ms/step - loss: 0.0510 - accura
cy: 0.9846 - val_loss: 0.0335 - val_accuracy: 0.9913
Epoch 8/10
422/422 [==============================] - 16s 39ms/step - loss: 0.0461 - accura
cy: 0.9859 - val_loss: 0.0322 - val_accuracy: 0.9907
Epoch 9/10
422/422 [==============================] - 16s 39ms/step - loss: 0.0437 - accura
cy: 0.9866 - val_loss: 0.0326 - val_accuracy: 0.9910
Epoch 10/10
422/422 [==============================] - 16s 39ms/step - loss: 0.0422 - accura
cy: 0.9866 - val_loss: 0.0311 - val_accuracy: 0.9913
```

Out[ ]:

```
<keras.callbacks.History at 0x7fcfa1985a90>
```

In [ ]:

```python
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.025207605212926865
Test accuracy: 0.9915000200271606
```