DL0101EN-3-2-Classification-with-Keras-py-v1.0

January 16, 2020

Classification Models with Keras

0.1 Introduction

In this lab, we will learn how to use the Keras library to build models for classificaiton problems. We will use the popular MNIST dataset, a dataset of images, for a change.

The MNIST database, short for Modified National Institute of Standards and Technology database, is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

The MNIST database contains 60,000 training images and 10,000 testing images of digits written by high school students and employees of the United States Census Bureau.

Also, this way, will get to compare how conventional neural networks compare to convolutional neural networks, that we will build in the next module.

0.2 Table of Contents

0.3 Import Keras and Packages

Let's start by importing Keras and some of its modules.

[1]:
```
import keras

from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

Since we are dealing we images, let's also import the Matplotlib scripting layer in order to view the images.

[2]:
```
import matplotlib.pyplot as plt
```

The Keras library conveniently includes the MNIST dataset as part of its API. You can check other datasets within the Keras library here.

So, let's load the MNIST dataset from the Keras library. The dataset is readily divided into a training set and a test set.

[3]:
```
# import the data
from keras.datasets import mnist

# read the data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Let's confirm the number of images in each set. According to the dataset's documentation, we should have 60000 images in X_train and 10000 images in the X_test.

[4]:
```
X_train.shape
```

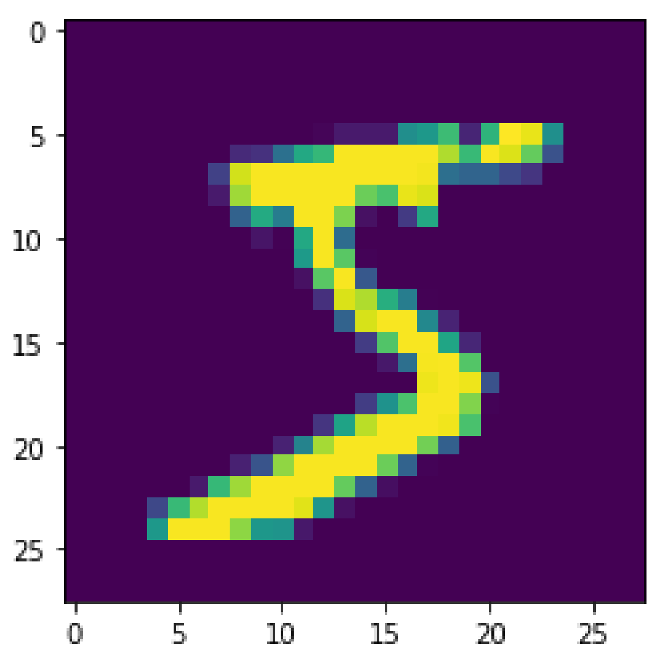
[4]:
```
(60000, 28, 28)
```

The first number in the output tuple is the number of images, and the other two numbers are the size of the images in datset. So, each image is 28 pixels by 28 pixels.

Let's visualize the first image in the training set using Matplotlib's scripting layer.

[5]:
```
plt.imshow(X_train[0])
```

[5]:
```
<matplotlib.image.AxesImage at 0x7fbc1fd3dac8>
```



With conventional neural networks, we cannot feed in the image as input as is. So we need to flatten the images into one-dimensional vectors, each of size 1 x (28 x 28) = 1 x 784.

[6]:
```
# flatten images into one-dimensional vector

num_pixels = X_train.shape[1] * X_train.shape[2] # find size of one-dimensional␣
↪vector

X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32') #␣
↪flatten training images
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32') #␣
↪flatten test images
```

Since pixel values can range from 0 to 255, let's normalize the vectors to be between 0 and 1.

[7]:
```
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
```

Finally, before we start building our model, remember that for classification we need to divide our target variable into categories. We use the to_categorical function from the Keras Utilities package.

[8]:
```
# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

num_classes = y_test.shape[1]
print(num_classes)
```

```
10
```

0.4 Build a Neural Network

[9]:
```
# define classification model
def classification_model():
    # create model
    model = Sequential()
    model.add(Dense(num_pixels, activation='relu', input_shape=(num_pixels,)))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))


    # compile model
    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

```
    return model
```

0.5 Train and Test the Network

[10]:
```
# build the model
model = classification_model()

# fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,␣
 ↪verbose=2)

# evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 26s - loss: 0.1821 - acc: 0.9455 - val_loss: 0.1012 - val_acc: 0.9687
Epoch 2/10
 - 42s - loss: 0.0778 - acc: 0.9757 - val_loss: 0.0868 - val_acc: 0.9717
Epoch 3/10
 - 47s - loss: 0.0535 - acc: 0.9825 - val_loss: 0.0690 - val_acc: 0.9792
Epoch 4/10
 - 48s - loss: 0.0409 - acc: 0.9874 - val_loss: 0.0646 - val_acc: 0.9812
Epoch 5/10
 - 47s - loss: 0.0316 - acc: 0.9903 - val_loss: 0.0876 - val_acc: 0.9751
Epoch 6/10
 - 48s - loss: 0.0253 - acc: 0.9920 - val_loss: 0.0823 - val_acc: 0.9795
Epoch 7/10
 - 48s - loss: 0.0240 - acc: 0.9926 - val_loss: 0.0774 - val_acc: 0.9805
Epoch 8/10
 - 46s - loss: 0.0195 - acc: 0.9939 - val_loss: 0.1015 - val_acc: 0.9781
Epoch 9/10
 - 46s - loss: 0.0174 - acc: 0.9947 - val_loss: 0.0721 - val_acc: 0.9828
Epoch 10/10
 - 46s - loss: 0.0169 - acc: 0.9947 - val_loss: 0.0930 - val_acc: 0.9800
```

Let's print the accuracy and the corresponding error.

[11]:
```
print('Accuracy: {}% \n Error: {}'.format(scores[1], 1 - scores[1]))
```

```
Accuracy: 0.98%
 Error: 0.020000000000000018
```

Just running 10 epochs could actually take over 2 minutes. But enjoy the results as they are getting generated.

Sometimes, you cannot afford to retrain your model everytime you want to use it, especially if you are limited on computational resources and training your model can take a long time. Therefore, with the Keras library, you can save your model after training. To do that, we use the save method.

[12]:
```
model.save('classification_model.h5')
```

Since our model contains multidimensional arrays of data, then models are usually saved as .h5 files.

When you are ready to use your model again, you use the load_model function from keras.models.

[13]:
```
from keras.models import load_model
```

[14]:
```
pretrained_model = load_model('classification_model.h5')
```

0.5.1 Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational. Feel free to contact me if you have any questions!

This notebook is part of a course on **Coursera** called *Introduction to Deep Learning & Neural Networks with Keras*. If you accessed this notebook outside the course, you can take this course online by clicking here.