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

May 23, 2020

0.1 Table of Contents

1. Import Keras and Packages
2. Build a Neural Network
3. Train and Test the Network

0.2 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.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:521: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:522: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:528: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

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()
```

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.

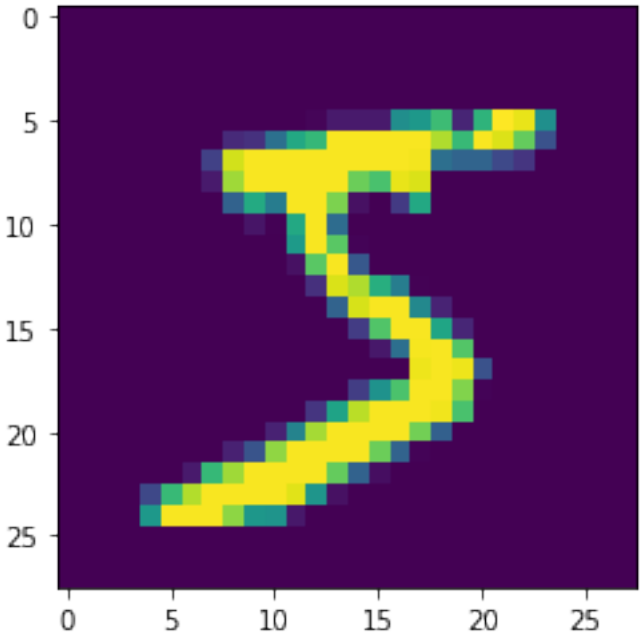
[4]:
```
X_train.shape
```

[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.

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

[5]:
```
<matplotlib.image.AxesImage at 0x7f8aeb584668>
```



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
```

[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
```

0.3 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=1)

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/1
60000/60000 [==============================] - 243s 4ms/step - loss: 0.1844 -
acc: 0.9445 - val_loss: 0.1053 - val_acc: 0.9666
```

Let's print the accuracy and the corresponding error.

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

```
Accuracy: 0.9666%
 Error: 0.033399999999999985
```

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')
```