**M Owais Tahir | Analysing Handwritten Digits Using Keras and Tensorflow**

The MNIST dataset comes pre-loaded in Keras, in the form of a set of four Numpy arrays:

**import keras**

**from keras import models**

**from keras import layers**

**from keras.datasets import mnist**

**from keras.utils import to\_categorical**

**train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()**

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. Our images are encoded as Numpy arrays, and the labels are simply an array of digits, ranging from 0 to 9. There is a one-to-one correspondence between the images and the labels.

**Input:**

**train\_images.shape**

**Output:**

**(60000, 28, 28)**

Our workflow will be as follow: first we will present our neural network with the training data, train\_images and train\_labels. The network will then learn to associate images and labels. Finally, we will ask the network to produce predictions for test\_images, and we will verify if these predictions match the labels from test\_labels.

Let's build our network -- again, remember that you aren't supposed to understand everything about this example just yet.

**Input:**

**model = models.Sequential()**

**model.add(layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))**

**model.add(layers.Dense(10, activation='softmax'))**

To make our network ready for training, we need to pick three more things, as part of "compilation" step:

* A loss function: the is how the network will be able to measure how good a job it is doing on its training data, and thus how it will be able to steer itself in the right direction.
* An optimizer: this is the mechanism through which the network will update itself based on the data it sees and its loss function.
* Metrics to monitor during training and testing. Here we will only care about accuracy (the fraction of the images that were correctly classified).

**Input:**

**model.compile(optimizer='rmsprop', loss='categorical\_crossentropy', metrics=['accuracy'])**

Before training, we will preprocess our data by reshaping it into the shape that 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.

**Input:**

**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**

**train\_labels = to\_categorical(train\_labels)**

**test\_labels = to\_categorical(test\_labels)**

We are now ready to train our network, which in Keras is done via a call to the fit method of the network: we "fit" the model to its training data.

**Input:**

**model.fit(train\_images, train\_labels, epochs=5, batch\_size=128)**

**Output:**

**Epoch 5/5**

**60000/60000 [==============================] - 5s 85us/step - loss: 0.0370 - acc: 0.9887**

**Input:**

**test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)**

**Output:**

**10000/10000 [==============================] - 1s 64us/step**

Two quantities are being 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 (i.e. 98.9%) on the training data. Now let's check that our model performs well on the test set too:

**Input:  
print ('Test Accuracy: ', test\_acc)**

**Output:**

**Test Accuracy: 0.9804**