**Digit Recognition**

1. **Introduction**

Handwritten Digit Recognition is an inserting machine learning problem in which we have to identify the handwritten digits through various classification algorithms. There are a number of ways and algorithms to recognize handwritten digits, including Deep Learning/ CNN, SVM, Gaussian Naive Bayes, KNN, Decision Trees, Rando Forests, etc.

1. **Dataset**

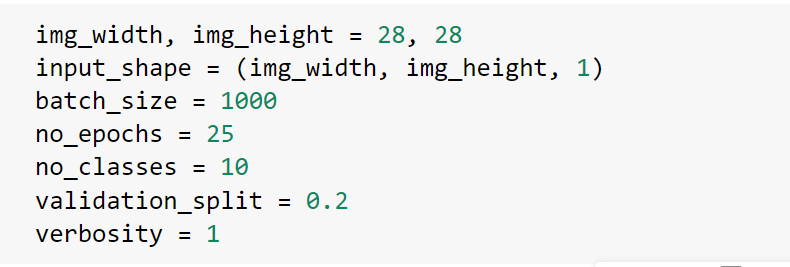
We will use the MNIST dataset. It is a set of 70,000 small images of digits handwritten by various people. All images are labelled with the respective digit they represent. MNIST is the hello world of ML. There are 70,000 images, and each image has 784 features. Each image is 28 x 28 pixels, and each feature simply represents one pixel’s intensity, from 0 (white) to 255 (black).

For importing the libraries, we follow the steps given below:

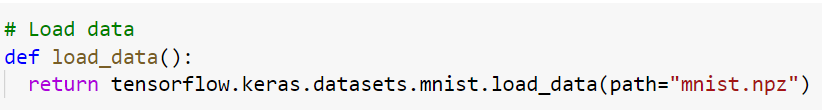
First, we import tensorflow. Further from tensorflow.keras, we import the Sequential API. The Sequential API is used for stacking layers of neural network on top of each other. Then we import Dense, Conv2D and Flatten from tensorflow.keras.layers. These are the general components of CNN.

1. **Methods Section**

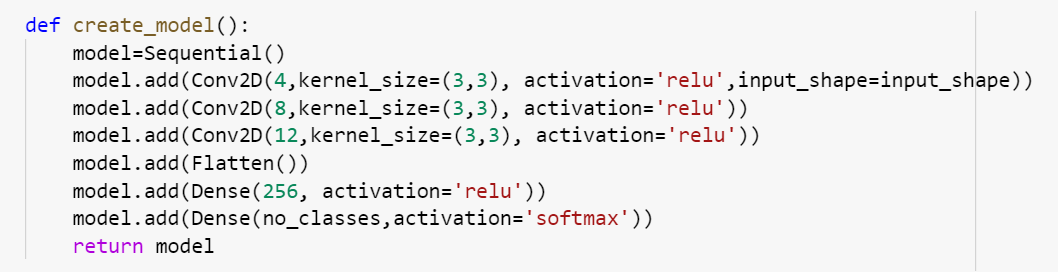
Image width and height are set to 28 pixels. Here, we use grayscale images, which are one channel and has 1 as the third dimension. We will train the data for 25 iterations and instruct the model that it should take into account 10 classes 🡪 the digits 0 to 9. 20% of the data will be used for steering the training process away from a process called overfitting. Model training process to be specified as verbose, i.e., to specify all the possible output in the terminal.



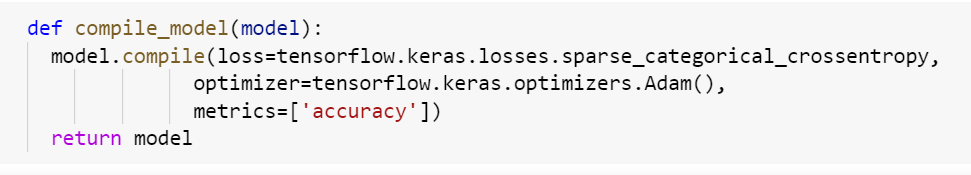
For loading the dataset, we are using the mnist.load\_data function. We need the path as “mnist.npz” for our analysis.



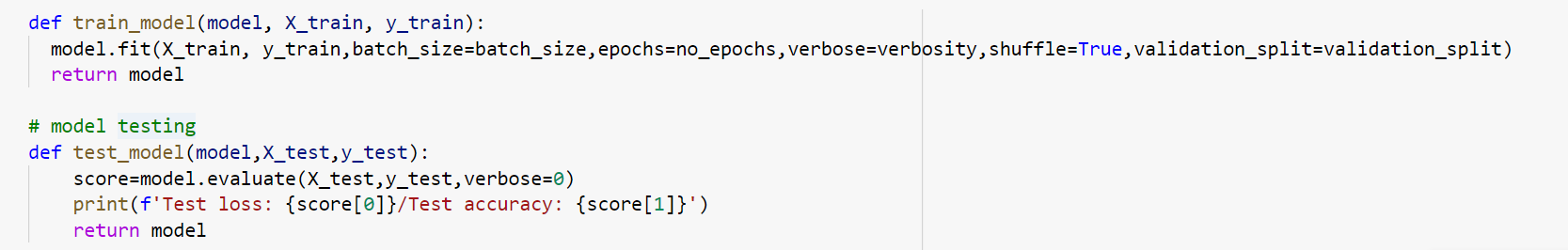
Using model.add, we will stack a few layers on top of each other. Starting the convolution layers for extracting features and Dense layers for actually converting the presence of features into a prediction.



We just created a skeleton of the model. Compiling involves specifying an optimization mechanism and loss function.

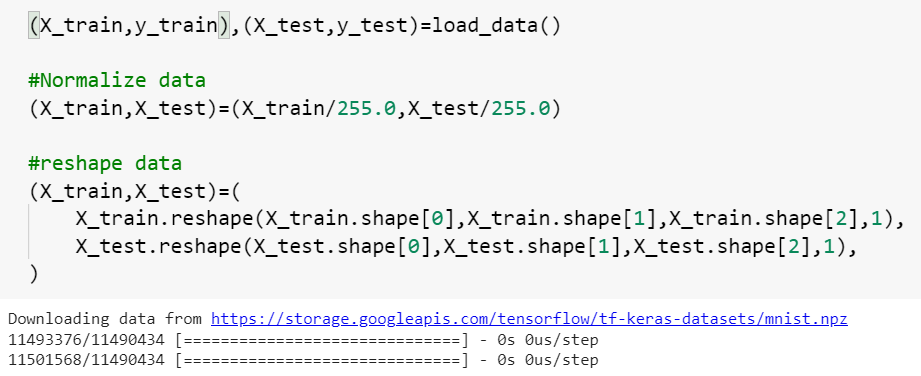


When a model is trained, it must be tested against data that it hasn't seen before. This ensures that model can be generalized, meaning that it is also effective on data that it has not seen before. Our goal is to create a model that can both predict and generalize.

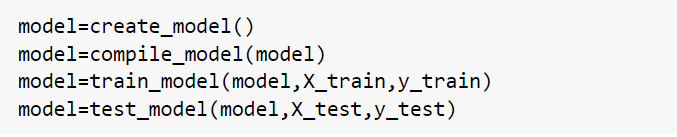


1. **Analysis Section**

We can now add code which truly loads our data based on the definition we created before. Before all of that, we must normalize it to [0,1] range and reshape it, because tensorflow expects our input to have a specific structure.



All we are left with is creating and training the model. So let’s do that now.



1. **Results**

