Importing Libraries

```
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.callbacks import EarlyStopping
```

Loading Dataset directly into Train & Test parameters

```
(X_train,Y_train),(X_test,Y_test) = keras.datasets.mnist.load_data()
```

Getting to know the data

```
X_train.shape
```

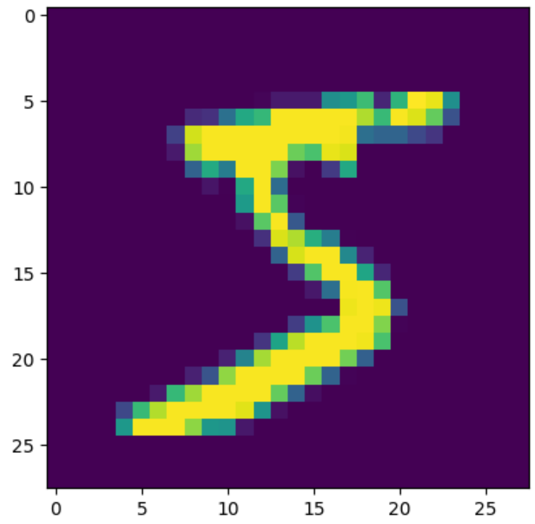
```
(60000, 28, 28)
```

```
X_train[0]
```

```
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,
```

```
# Plotting the image of the 1st row to see how it looks and what digit it is (5)
import matplotlib.pyplot as plt
plt.imshow(X_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f85620cbc40>
```



```
Y_train[0]
```

```
5
```

The values are divided by 255 to standardize them since colors have a range of 0 to 255.

```
X_train = X_train/255
X_test = X_test/255
```

```
X_train[0]
```

```
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, 0.53333333, 0.68627451, 0.10196078,
        0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.11764706, 0.14117647,
        0.36862745, 0.60392157, 0.66666667, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.88235294, 0.6745098 ,
        0.99215686, 0.94901961, 0.76470588, 0.25098039, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.19215686, 0.93333333, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.98431373, 0.36470588, 0.32156863,
        0.32156863, 0.21960784, 0.15294118, 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.07058824, 0.85882353, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.77647059,
        0.71372549, 0.96862745, 0.94509804, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.31372549, 0.61176471,
        0.41960784, 0.99215686, 0.99215686, 0.80392157, 0.04313725,
        0.        . 0.16862745. 0.60392157. 0.        . 0.        .
```

```
# Obj
ann = Sequential()

# Hidden layers
ann.add(Flatten(input_shape=(28,28)))   # Here the image is of (28 x 28) pixels i.e. 28D so using flattening layer to convert it into 1D array
ann.add(Dense(128,activation="relu"))
ann.add(Dense(32,activation="relu"))

# Output layers
ann.add(Dense(10,activation="softmax"))   # As in total possibilities are [0,1,2...8,9] so total neurons are 10 , Softmax is used for multiclass classififcation

# diagram
ann.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense (Dense)               (None, 128)               100480    
                                                                 
 dense_1 (Dense)             (None, 32)                4128      
                                                                 
 dense_2 (Dense)             (None, 10)                330       
                                                                 
=================================================================
Total params: 104,938
Trainable params: 104,938
Non-trainable params: 0
_________________________________________________________________
```

```
ann.compile(loss="sparse_categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
```

Creating an object of early stopping to save the resources

```
ES = EarlyStopping(mode="min",monitor="val_loss",patience=25,verbose=1)
```

```
ann.fit(X_train,Y_train,validation_split=0.2,epochs=25,batch_size=64,callbacks=ES)
```

```
Epoch 1/25
750/750 [==============================] - 6s 7ms/step - loss: 0.3259 - accuracy: 0.9071 - val_loss: 0.1704 - val_accuracy: 0.9518
Epoch 2/25
750/750 [==============================] - 6s 8ms/step - loss: 0.1408 - accuracy: 0.9588 - val_loss: 0.1222 - val_accuracy: 0.9637
Epoch 3/25
750/750 [==============================] - 9s 12ms/step - loss: 0.0961 - accuracy: 0.9715 - val_loss: 0.1039 - val_accuracy: 0.9700
Epoch 4/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0731 - accuracy: 0.9779 - val_loss: 0.0984 - val_accuracy: 0.9717
Epoch 5/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0563 - accuracy: 0.9830 - val_loss: 0.0980 - val_accuracy: 0.9703
Epoch 6/25
750/750 [==============================] - 5s 6ms/step - loss: 0.0446 - accuracy: 0.9861 - val_loss: 0.1057 - val_accuracy: 0.9699
Epoch 7/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0370 - accuracy: 0.9885 - val_loss: 0.0932 - val_accuracy: 0.9741
Epoch 8/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0288 - accuracy: 0.9910 - val_loss: 0.1017 - val_accuracy: 0.9744
Epoch 9/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0256 - accuracy: 0.9921 - val_loss: 0.0946 - val_accuracy: 0.9745
Epoch 10/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0196 - accuracy: 0.9940 - val_loss: 0.0975 - val_accuracy: 0.9750
Epoch 11/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0176 - accuracy: 0.9942 - val_loss: 0.0994 - val_accuracy: 0.9738
Epoch 12/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0167 - accuracy: 0.9943 - val_loss: 0.1109 - val_accuracy: 0.9741
Epoch 13/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0132 - accuracy: 0.9957 - val_loss: 0.1236 - val_accuracy: 0.9723
Epoch 14/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0124 - accuracy: 0.9960 - val_loss: 0.1058 - val_accuracy: 0.9783
Epoch 15/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0075 - accuracy: 0.9980 - val_loss: 0.1112 - val_accuracy: 0.9752
Epoch 16/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0121 - accuracy: 0.9959 - val_loss: 0.1170 - val_accuracy: 0.9747
Epoch 17/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0110 - accuracy: 0.9965 - val_loss: 0.1197 - val_accuracy: 0.9748
Epoch 18/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0100 - accuracy: 0.9962 - val_loss: 0.1340 - val_accuracy: 0.9734
Epoch 19/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.1232 - val_accuracy: 0.9765
Epoch 20/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0056 - accuracy: 0.9984 - val_loss: 0.1267 - val_accuracy: 0.9752
Epoch 21/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0090 - accuracy: 0.9970 - val_loss: 0.1432 - val_accuracy: 0.9731
Epoch 22/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0091 - accuracy: 0.9970 - val_loss: 0.1417 - val_accuracy: 0.9743
```

```
Epoch 23/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0072 - accuracy: 0.9976 - val_loss: 0.1465 - val_accuracy: 0.9731
Epoch 24/25
750/750 [==============================] - 5s 7ms/step - loss: 0.0057 - accuracy: 0.9982 - val_loss: 0.1326 - val_accuracy: 0.9778
Epoch 25/25
750/750 [==============================] - 4s 5ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.1414 - val_accuracy: 0.9771
<keras.callbacks.History at 0x7f855f2a44c0>
```

```
Y_prob = ann.predict(X_test)
```

```
313/313 [==============================] - 1s 2ms/step
```

Assigning the class based on the highest probability values for the numbers 0 to 9.

```
Y_pred = Y_prob.argmax(axis=1)
```

The probability that the first row belongs to each of the [0-9] classes.

```
Y_prob[0]
```

```
array([2.1568794e-13, 3.7061783e-15, 1.6564383e-10, 5.2114744e-09,
       3.5921284e-20, 4.5142032e-15, 2.0634833e-23, 9.9999994e-01,
       2.1030237e-14, 4.5286639e-12], dtype=float32)
```

```
Y_pred
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

Classification Report

```
# Scores of all classes are pretty good so we can say model has learnt well on all the classes
from sklearn.metrics import classification_report
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.98      0.97      1032
           3       0.97      0.97      0.97      1010
           4       0.98      0.97      0.97       982
           5       0.98      0.96      0.97       892
           6       0.97      0.97      0.97       958
           7       0.98      0.97      0.98      1028
           8       0.96      0.97      0.96       974
           9       0.96      0.97      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```