

MAJOR PROJECT

IN MACHINE LEARNING



DONE BY (GROUP 6)

PAAVANA M RAO , GONDIPALLI. HARI PRIYA

GONUGUNTLA. SHARMILA

KORRAPATI. MAHENDRA SUNNY GOUD

**INTRODUCTION**

Human Visual System is a marvel of the world. People can readily recognise digits. But it is not as simple as it looks like. The human brain has a million neurons and billions of connections between them, which makes this exceptionally complex task of image

However, it turns into a challenging task for computers to recognize digits. Simple hunches about how to recognize digits become difficult to express algorithmically. Moreover, there is a significant variation in writing from person to person, which makes it immensely complex



A Handwritten digit recognition system is the working of a machine to train itself so that it can recognize digits from different sources like emails, bank cheque, papers, images, etc.

**MNIST DIGIT RECOGNIZER**

The [MNIST database](http://yann.lecun.com/exdb/mnist/) (Modified National Institute of Standard Technology database) of handwritten digits consists of a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. Additionally, the black and white images from NIST were size-normalized and centred to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

This database is well liked for training and testing in the field of machine learning and image processing. It is a remixed subset of the original NIST datasets. One half of the 60,000 training images consist of images from NIST's testing dataset and the other half from Nist's training set. The 10,000 images from the testing set are similarly assembled.



The MNIST dataset is used by researchers to test and compare their research results with others. The lowest error rates in literature are as low as 0.21 percent.1

**REQUIREMENTS**

*GOOGLE COLAB –*

Google Colab has been used to implement the network. It is a free cloud service that can be used to develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and Open CV

The most important feature that distinguishes Colab from other free cloud services is; it provides GPU and is totally free. Thus, if PC is incompatible with hardware requirements or does not support GPU, then it is the best option because a stable internet connection is the only requirement.

*DATASETS –*

[..\Downloads\train.csv](../Downloads/train.csv)

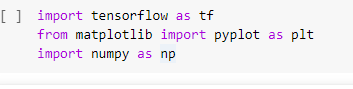
[..\Downloads\test.csv](../Downloads/test.csv)

**PHASES OF IMPLEMENTATION**

* ***IMPORTING THE LIBRARIES***
* ***LOADING THE DATASET***
* ***PRE-PROCESSING THE DATA***
* ***CREATE THE MODEL***
* ***COMPILE THE MODEL***
* ***TRAIN THE MODEL***
* ***EVALUATE AND TEST THE MODEL***

**IMPORTING THE LIBRARIES**

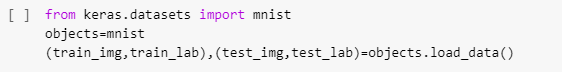
First, we imported all the libraries that we are going to use.



We imported TensorFlow which is an open-source free library that is used for machine learning applications such as neural networks etc. Further, we imported pyplot function, which is basically used for plotting, from the matplotlib library which is used for visualisation purposes. After that, we imported NumPy i.e. Numerical Python which is used to perform various mathematical operations.

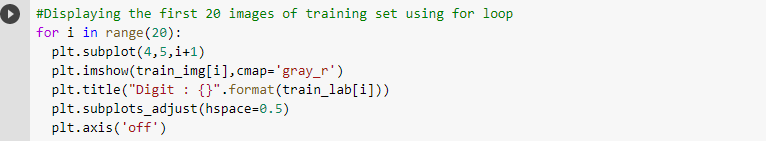
**LOADING DATASET**

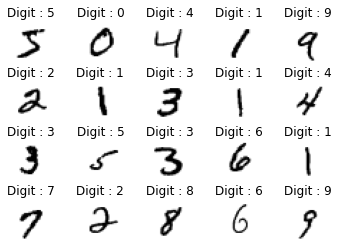
The Keras library already contains some datasets such as CIFAR10, CIFAR100, Boston Housing price regression dataset, IMDB movie review sentiment classification dataset etc



The MNIST dataset is also part of it. So, we imported it from keras.datasets and loaded it into variable “objects”. The objects.load\_data() method returns us the training data(train\_img), its labels(train\_lab) and also the testing data(test\_img) and its labels(test\_lab). Out of the 70,000 images provided in the dataset, 60,000 are given for training and 10,000 are given for testing.

Before preprocessing the data, we first displayed the first 20 images of the training set with the help of for loop.





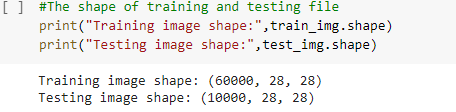
subplot() is used to add a subplot or grid-like structure to the current figure. The first argument is for “no. of rows”, second for “no. of columns” and third for position index in the grid.

imshow() is used to display data as an image i.e. training image (train\_img[i])

Cmap stands for colour map. Basically, if the image is in the array of shape (M, N), then the cmap controls the colour map used to display the values. cmap=‘gray’ will display image as grayscale while cmap=‘gray\_r’ is used to display image as inverse grayscale.

title() sets title for each image. We have set “Digit: train\_lab[i]” as the title for each image in the subplot.

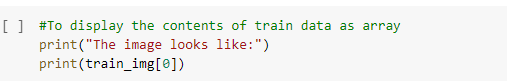
subplots\_adjust() is used for tuning subplot layout. In order to change the space provided between two rows, we have used hspace. If you want to change space between two columns then you can use wspace.

After that, we displayed the shape of training and testing section

60000,28,28) means there are 60,000 images in the training set and each image is of size 28x28 pixels. Similarly, there are 10,000 images of the same size in the testing set.

So each image is of size 28x28 i.e. 784 features, and each feature represents the intensity of each pixel from 0 to 255.

We have used print(train\_img[0]) to print the first training set image in the matrix form of 28x28.

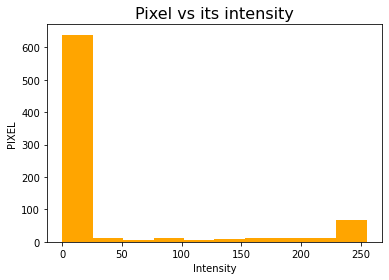


**PRE PROCESSING THE DATA**

We plotted the first training image on a histogram.

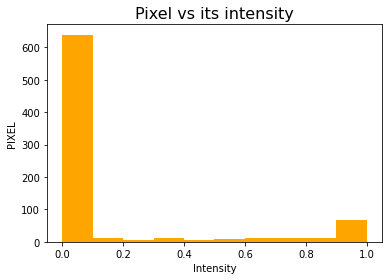
Before feeding the data to the network, we will normalize it. Normalizing the input data helps to speed up the training. Also, it reduces the chance of getting stuck in local optima, since we’re using stochastic gradient descent to find the optimal weights for the network.

Before normalization



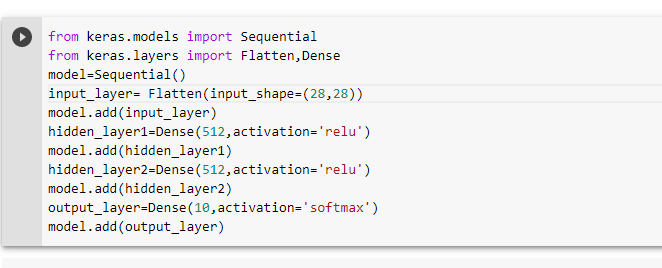
hist() is used to plot the histogram for the first training image i.e. train\_img[0]. The image has been reshaped into a 1-D array of size 784. facecolor is an optional parameter which specifies the colour of the histogram.

After normalization



**CREATING THE MODEL**

Here, we have used the Sequential model. This model has one input layer, one output layer and two hidden layers. Sequential() is used to create a layer of the network in sequence



.add() is used here to add the layer into the model.

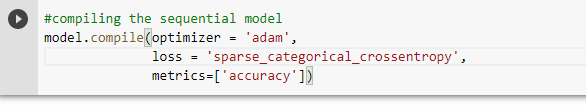
In the first layer(input layer), we feed image as the input. Since each image is of size 28x28, hence we have used Flatten() to compress the input.

We have used Dense() in the other layers. It ensures that each neuron in the previous layer is connected to every neurone in the next layer

The model is a simple neural network with two hidden layers with 512 neurons. A rectifier linear unit activation (ReLU) function is used for the neurons in the hidden layers. The nicest thing about it is that its gradient is always equal to 1, this way we can pass the maximum amount of the error through the network during back-propagation.

The output layer has 10 neurons i.e. for each class from 0 to 9. A softmax activation function is used on the output layer to turn the outputs into probability-like values.

Next, we need to compile our model. Compiling the model takes three parameters: optimizer, loss and metrics.



We will use ‘Sparse\_Categorical\_Crossentropy’ for our loss function because it saves time in memory as well as computation since it simply uses a single integer for a class, rather than a whole vector. A lower score indicates that the model is performing better.

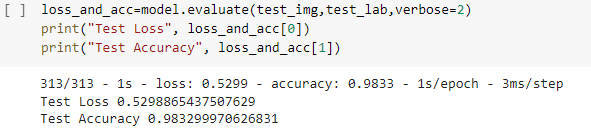
**TRAINING THE MODEL**

In order to determine the accuracy, we will use the ‘accuracy’ metric to see the accuracy score on the validation set when we train the model.

We will train the model with the help of fit() function. It will have parameters as training data (train\_img), training labels (train\_lab) and the number of epochs. The number of epochs is the number of times the model will cycle through the data. The more epochs we run, the more the model will improve, up to a certain point. After that point, the model will stop improving during each epoch.We will save the model as project.h5

**EVALUATING THE MODEL**

The accuracy is computed on the 10,000 testing examples using the network weights given by the saved model.

****

model.evaluate() method computes the loss and any metric defined when compiling the model. So in our case, the accuracy is computed on the 10,000 testing examples using the network weights given by the saved model.

Verbose can be either 0,1, or 2. By default verbose is 1.

verbose = 0, means silent.

verbose = 1, which includes both progress bar and one line per epoch.

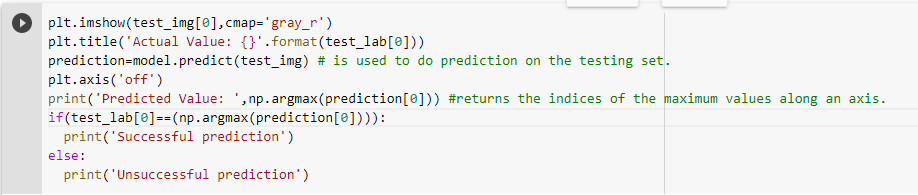
verbose = 2, one line per epoch i.e. epoch no./total no. of epochs.

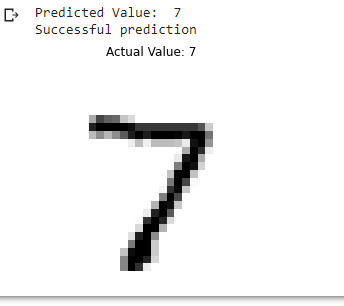
**TESTING THE MODEL**

After evaluating the model, we will now check the model for the testing section.

model.predict() is used to do prediction on the testing set.

np.argmax() returns the indices of the maximum values along an axis

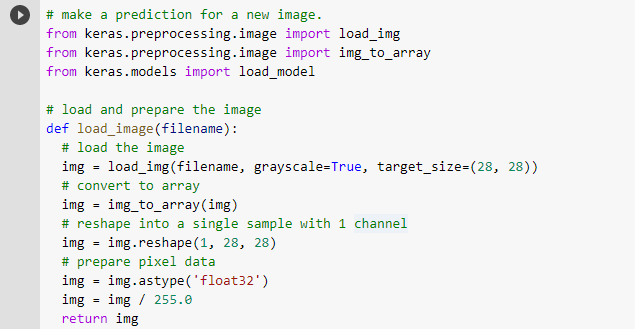




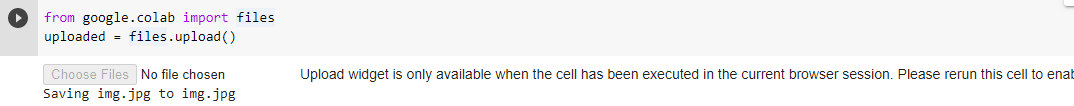
THIS SHOWS US THAT THE MODEL IS PREDICTING IT ACCURATELY

**TESTING THE MODEL WITHOUT USING THIS DATASET**

Now, in order to make a prediction for a new image that is not part of MNIST dataset. We will first create a function named “load\_image”.



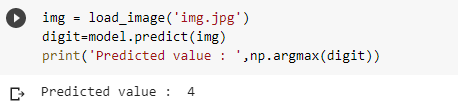
Above function converts the image into an array of pixels which is fed to the model as an input.



It will lead you to select a file. Click on “Choose Files” then select and upload the file and wait for the file to be uploaded 100%. You will see the name of the file once Colab has uploaded it.

In order to display image file, we used the code:





As you can see we have successfully predicted the value as 4.

Now, if we want to run the model after a few days then, we will have to run the whole code again, which is time-consuming.

In that case, you can use the saved model i.e. project.h5

So, before closing the colab notebook, you can download the model from the folder symbol. NOTEBOOK – [https://colab.research.google.com/drive/1W5CrlK6i3nDtqejztnQtOQgjxaag24H8#scrollTo=\_jtdzHuH\_NTG](../Downloads/Major_project.ipynb)