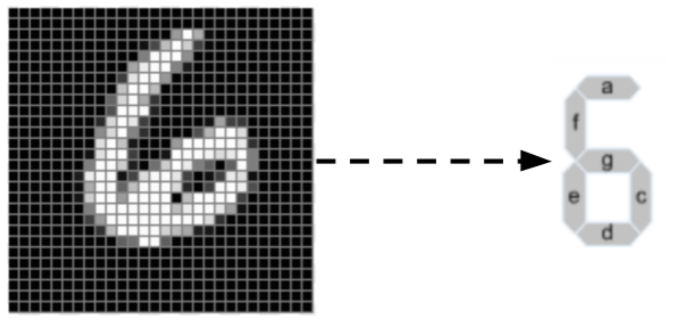
Project:
Transform MNIST images into their corresponding seven segment display representation

Goal

To train a feedforward neural network that transforms a 28 x 28 grayscale image of a hand-written digit into its seven-segment display representation.



Understanding of Neural Network and Feed Forward method

Neural networks and deep learning at present provide the best solutions to a wide range of problems in image recognition, speech recognition, time series analysis and natural language processing. Neural network algorithms are stimulated by the working of a human brain to perform a specific task or function by executing calculations through a learning process. It is a group of connected input / output elements called nodes in which each link has an associated weight and tasks are performed in parallel.

One of the types of neural network used in this assignment is the feedforward neural network where the data travels throughout one direction only, forward, from the input nodes, through the hidden nodes and to the output nodes. This type of network has no loops or cycles as shown in Fig.1. Deep learning includes multiple hidden layers used for learning in neural networks. The model works by learning from the input provided, i.e. it trains itself from the raw data that has a known result and optimizes its weights for better predictability with undefined results.

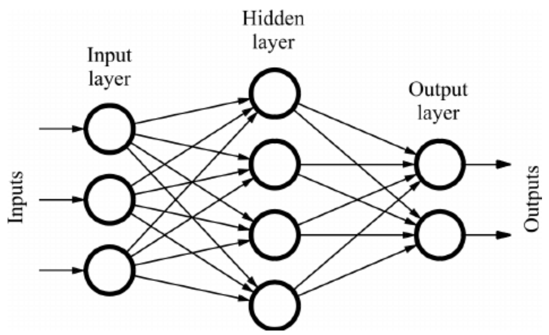


Fig.1 Feed forward neural network with 1 hidden layer

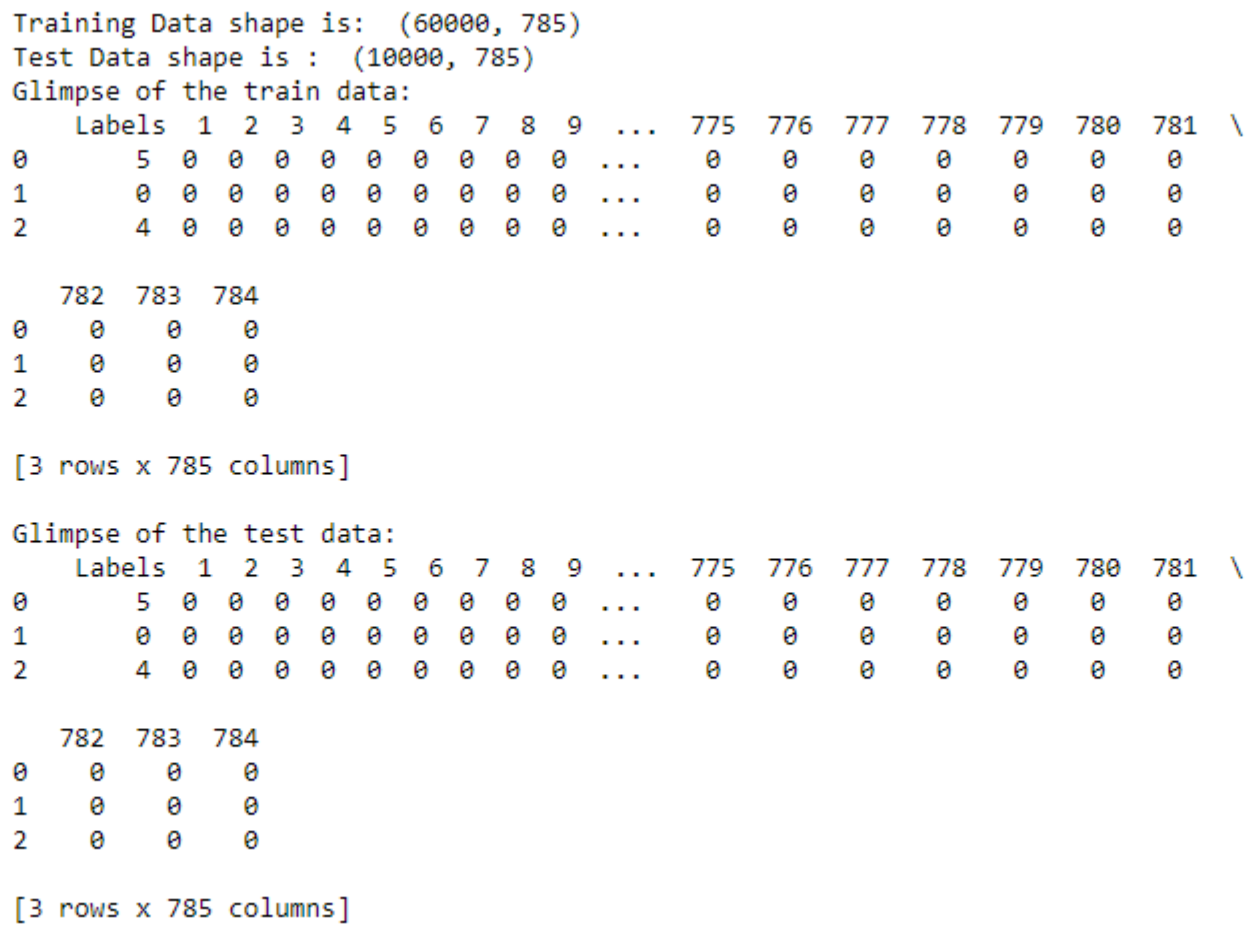
Analysis – Implementation using Python

Implementation of feed forward deep neural network for handwritten digit recognition is performed using Keras library, an API developed in Python focusing on faster and productive experimentation. Here, we focus on identifying the digital handwritten images by classifying the images as pattern that can represent LEDs when added to a seven-segment display representation. These images are represented as a matrix with values mentioned as pixels.

Data Exploration - Dataset files mnist_train.csv and mnist_test.csv consists of gray-scale images of handwritten digits, from 0 - 9. Each image is 28 x 28 pixels and represented as a row of 784-pixel values between 0 and 255 indicating the brightness and darkness (intensities) of each pixel. The label associated with each image is determined as an integer value between 0 and 9 telling which digit it is.

Train the Neural Network -

1. Import all the required packages and libraries to predict the feedforward NN model.
2. Load and Read the dataset – Both the train and test csv files are loaded and are ready for the analysis.

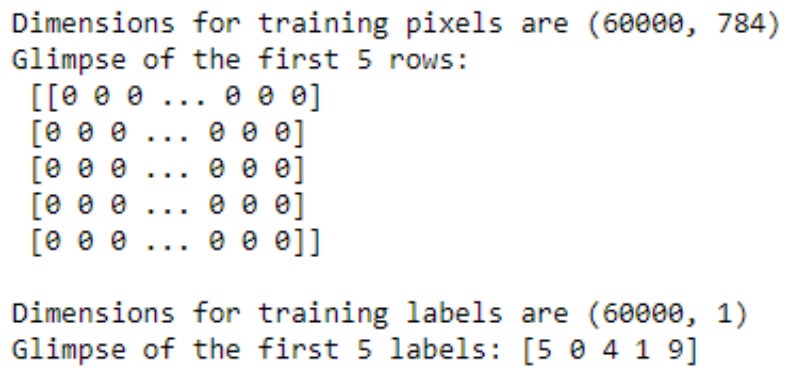


```
Training Data shape is:  (60000, 785)
Test Data shape is :  (10000, 785)
Glimpse of the train data:
    Labels  1  2  3  4  5  6  7  8  9  ...  775  776  777  778  779  780  781  \
0        5  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0
1        0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0
2        4  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0

   782  783  784
0    0    0    0
1    0    0    0
2    0    0    0

[3 rows x 785 columns]

Glimpse of the test data:
    Labels  1  2  3  4  5  6  7  8  9  ...  775  776  777  778  779  780  781  \
0        5  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0
1        0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0
2        4  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0

   782  783  784
0    0    0    0
1    0    0    0
2    0    0    0

[3 rows x 785 columns]
```

Fig.1 Representation of the given training and testing data

Train data has 60000 and Test data has 10000 handwritten digit images with 785 pixels attributes.

3. After producing the data, the data is split into Pixels (x) and Labels (y) for both the train and test datasets as x_train, y_train, x_test, y_test and are reshaped into an array individually.
   This dataset will be taken as input for the model. It is represented as shown in Fig.2.



```
Dimensions for training pixels are (60000, 784)
Glimpse of the first 5 rows:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]

Dimensions for training labels are (60000, 1)
Glimpse of the first 5 labels: [5 0 4 1 9]
```

Fig.2 Pixels and Labels Array for Train and Test Data

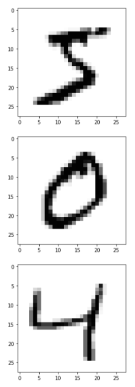
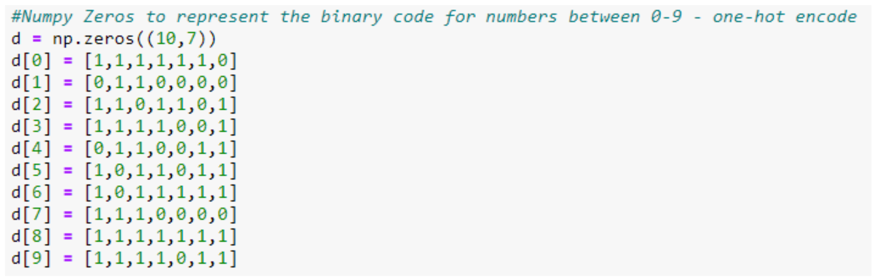


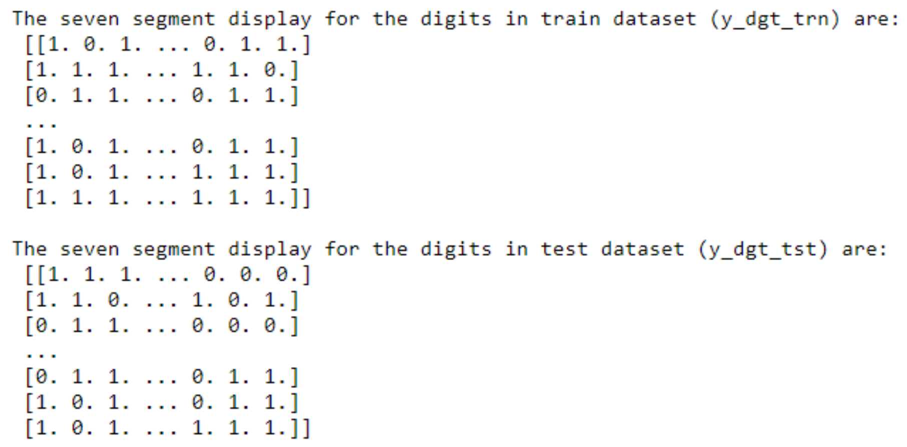
Fig.3 Hand-written Image Plots

Now to represent the digit into a pattern that can be represented in seven -segment display. We use one-hot encoded vector where we replace all the labels(digits) with it's equivalent one-hot code. For ex:



```
#Numpy Zeros to represent the binary code for numbers between 0-9 - one-hot encode
d = np.zeros((10,7))
d[0] = [1,1,1,1,1,1,0]
d[1] = [0,1,1,0,0,0,0]
d[2] = [1,1,0,1,1,0,1]
d[3] = [1,1,1,1,0,0,1]
d[4] = [0,1,1,0,0,1,1]
d[5] = [1,0,1,1,0,1,1]
d[6] = [1,0,1,1,1,1,1]
d[7] = [1,1,1,0,0,0,0]
d[8] = [1,1,1,1,1,1,1]
d[9] = [1,1,1,1,0,1,1]
```

I have used **NumPy np.zeros** to give a new array of the shape and type mentioned, with zeros. Then, we pass the values to the function's "digits_display" which matches the label value and the pattern After which, we pass y_train and y_test to the functions as label data to replace it with the binary values array. This results as shown below:



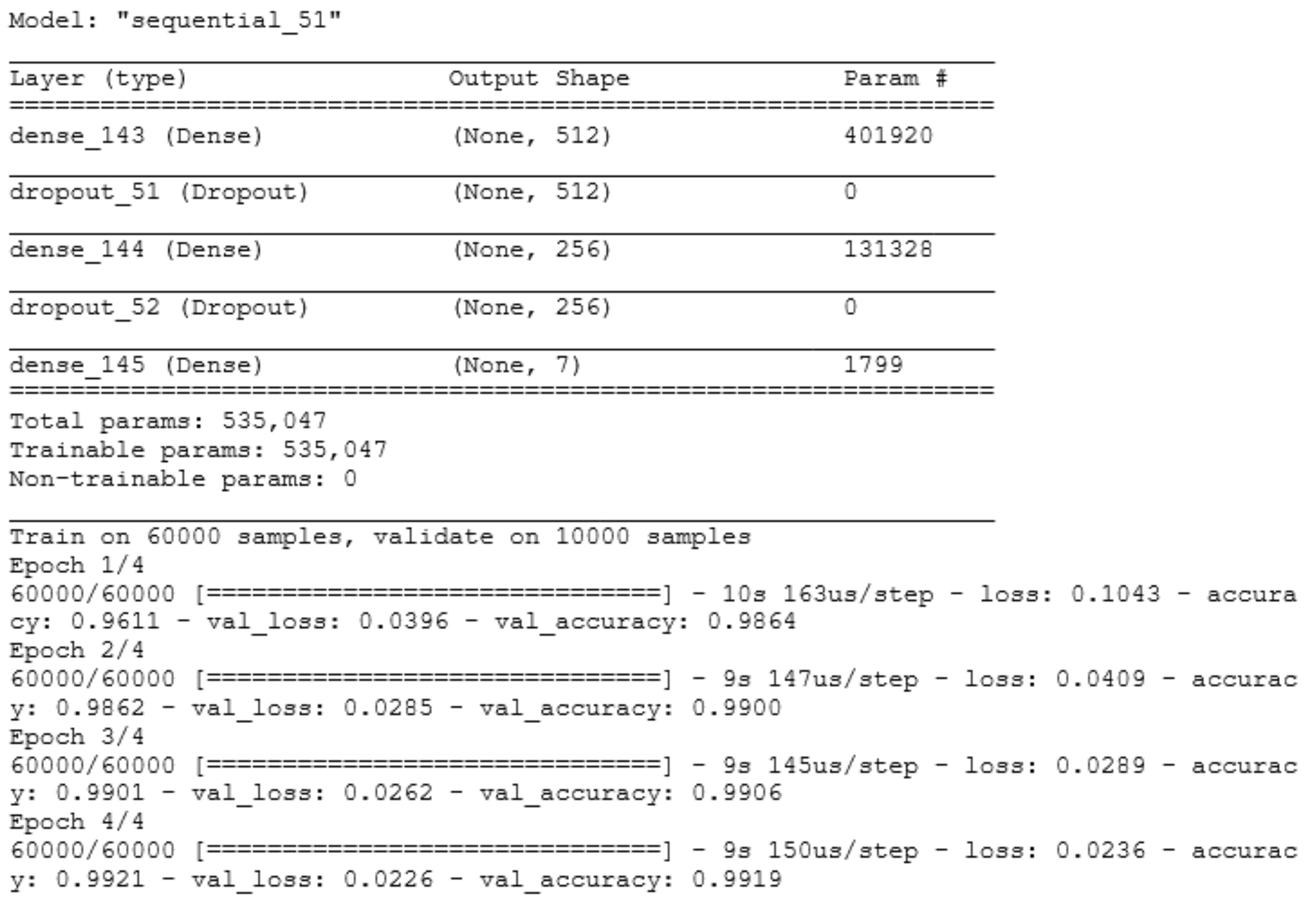
```
The seven segment display for the digits in train dataset (y_dgt_trn) are:
 [[1. 0. 1. ... 0. 1. 1.]
 [1. 1. 1. ... 1. 1. 0.]
 [0. 1. 1. ... 0. 1. 1.]
 ...
 [1. 0. 1. ... 0. 1. 1.]
 [1. 0. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]]

The seven segment display for the digits in test dataset (y_dgt_tst) are:
 [[1. 1. 1. ... 0. 0. 0.]
 [1. 1. 0. ... 1. 0. 1.]
 [0. 1. 1. ... 0. 0. 0.]
 ...
 [0. 1. 1. ... 0. 1. 1.]
 [1. 0. 1. ... 0. 1. 1.]
 [1. 0. 1. ... 1. 1. 1.]]
```

Similarly, we normalize the pixel values (x_trn and x_tst) from 0-255 to 0-1.

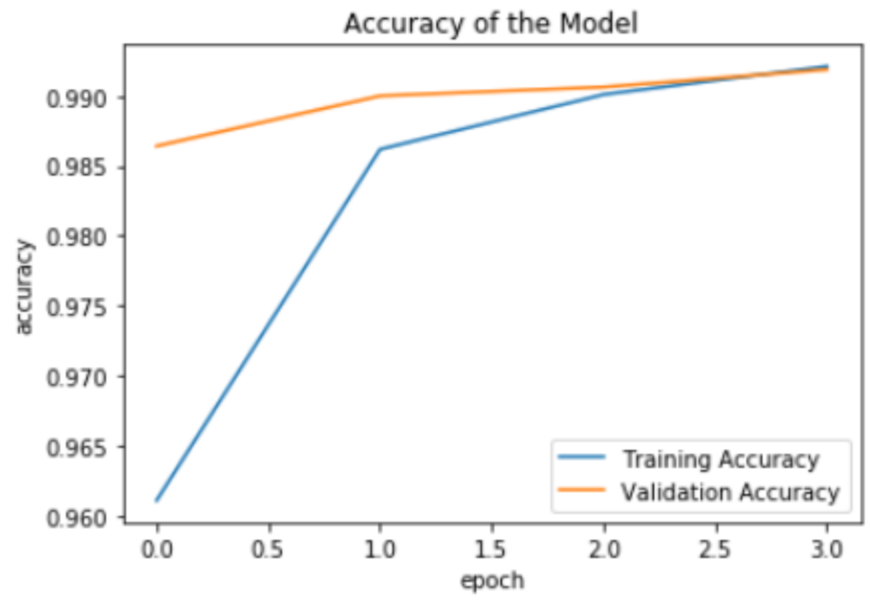
4. Now, we build the neural network model using Keras Sequential Model. Here we linearly stack the layers specifying the input, hidden and output layers shape. In our case, we train the array of input data and labels using fit function (). The input layer shape is 784-pixel values and output are 7 digits (one-hot encoded) pattern.
   For our model, with 2 classes (binary classification) we use the **special activation function as "sigmoid"** in the output layer and "relu" for the input and the hidden layers.
   Before training the model, we configure the learning process using compile method and mention the optimizer, loss function and list of metrics. Here, we mention **binary_crossentropy** as the loss function.

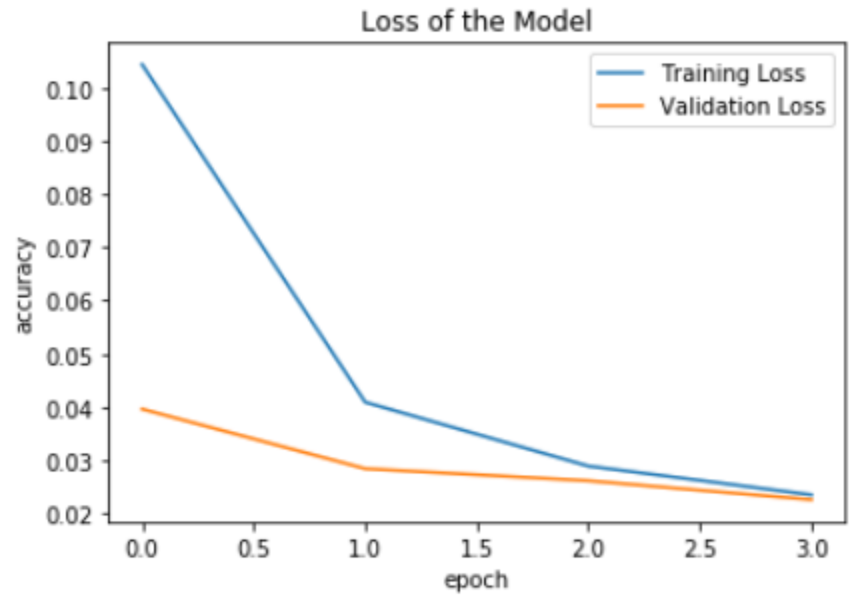
When we use Keras's binary_crossentropy, with the inputs from the result of the sigmoid activation function, the model does not lead to over fitting or underfitting.

We train the model with fixed number of epochs – iterations on the train data. Following is the summary for the model:



```
Model: "sequential_51"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_143 (Dense)            (None, 512)               401920
_________________________________________________________________
dropout_51 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_144 (Dense)            (None, 256)               131328
_________________________________________________________________
dropout_52 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_145 (Dense)            (None, 7)                 1799
=================================================================
Total params: 535,047
Trainable params: 535,047
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/4
60000/60000 [==============================] - 10s 163us/step - loss: 0.1043 - accura
cy: 0.9611 - val_loss: 0.0396 - val_accuracy: 0.9864
Epoch 2/4
60000/60000 [==============================] - 9s 147us/step - loss: 0.0409 - accurac
y: 0.9862 - val_loss: 0.0285 - val_accuracy: 0.9900
Epoch 3/4
60000/60000 [==============================] - 9s 145us/step - loss: 0.0289 - accurac
y: 0.9901 - val_loss: 0.0262 - val_accuracy: 0.9906
Epoch 4/4
60000/60000 [==============================] - 9s 150us/step - loss: 0.0236 - accurac
y: 0.9921 - val_loss: 0.0226 - val_accuracy: 0.9919
```





```
Final loss: 0.023, Final accuracy: 0.992
```

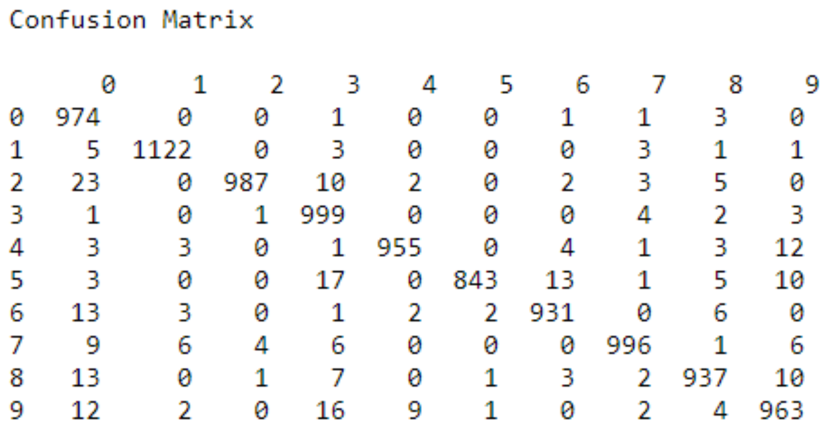
Fig.4 Accuracy and Loss Plot for the model

Here we observe the loss is close to 2 % and **accuracy is 99.2%** for the model. The performance of the Model is really good with only 4 iterations. With proper training the results are far better.

Now to see the results between the predicted and the test data, we plot a confusion matrix.

5. Now, before we plot the confusion matrix, the label values need to be remapped with the labels. So now we convert the binary code to the digits using remap_label() function. From this, we generate the predicted labels and test data labels as digit format.

6. Confusion Matrix for the predicted and test data labels is generated as shown below:



```
Confusion Matrix

      0     1    2    3    4    5    6    7    8    9
0   974     0    0    1    0    0    1    1    3    0
1     5  1122    0    3    0    0    0    3    1    1
2    23     0  987   10    2    0    2    3    5    0
3     1     0    1  999    0    0    0    4    2    3
4     3     3    0    1  955    0    4    1    3   12
5     3     0    0   17    0  843   13    1    5   10
6    13     3    0    1    2    2  931    0    6    0
7     9     6    4    6    0    0    0  996    1    6
8    13     0    1    7    0    1    3    2  937   10
9    12     2    0   16    9    1    0    2    4  963
```

Fig.5 Confusion Matrix for the predicted and test data labels

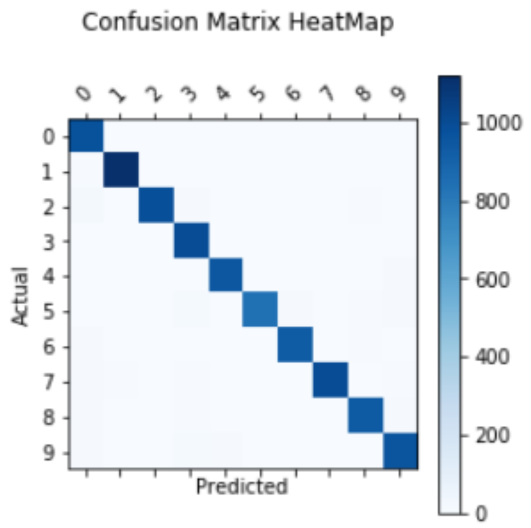


Fig.6 Heatmap for Confusion Matrix

From Fig.5 we can observe that except label 5, all other digits are recognized properly and with accuracy. Moreover, highest mis prediction is - 23 digits with label 2 were predicted as 0. Apart from this the digit recognition neural network model performed extremely good with Accuracy of 99.2%.

Conclusion

From all the observations that the model resulted, we can say that this neural network model with Keras is successfull to recognize the labels (digits) based on the seven-segment display. Some mismatching was detected using binary class with few labels, but it can be corrected by tweaking few parameters like the layers, dropout values, learning rates, iterations etc. All trial and error can result in better accuracy and performance. Here, with 4 epochs with 2 hidden layers gave us 99.2% of accuracy in identifying the digits by mapping them to binary one-hot coding for seven-segment LED display. We can conclude that the model is fast to train the data and predict the output.

References

Hastie, Tibshirani & Friedman. (2008). The Elements of Statistical learning. Stanford: Springer.

Guide to the Sequential model - Keras Documentation. (2019). Keras.io. Retrieved from https://keras.io/getting-started/sequential-model-guide/

(2019). Databricks-prod-cloudfront.cloud.databricks.com. Retrieved from https://databricks-prodcloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2961012104553482/4462572393058030/1806228006848429/latest.html

Hentschke,H. Sigmoid Activation and Binary Crossentropy —A Less Than Perfect Match? (2019). Medium. Retrieved from https://towardsdatascience.com/sigmoid-activation-and-binary-crossentropy-a-less-than-perfect-match-b801e130e31

Bettilyon,T. (2018). How to classify MNIST digits with different neural network architectures. Medium. Retrieved from https://medium.com/tebs-lab/how-to-classify-mnist-digits-with-different-neural-network-architectures-39c75a0f03e3