Name: Raveena Rai
ED21S006

Programming Assignment 1: MNIST Classification using MLP From scratch

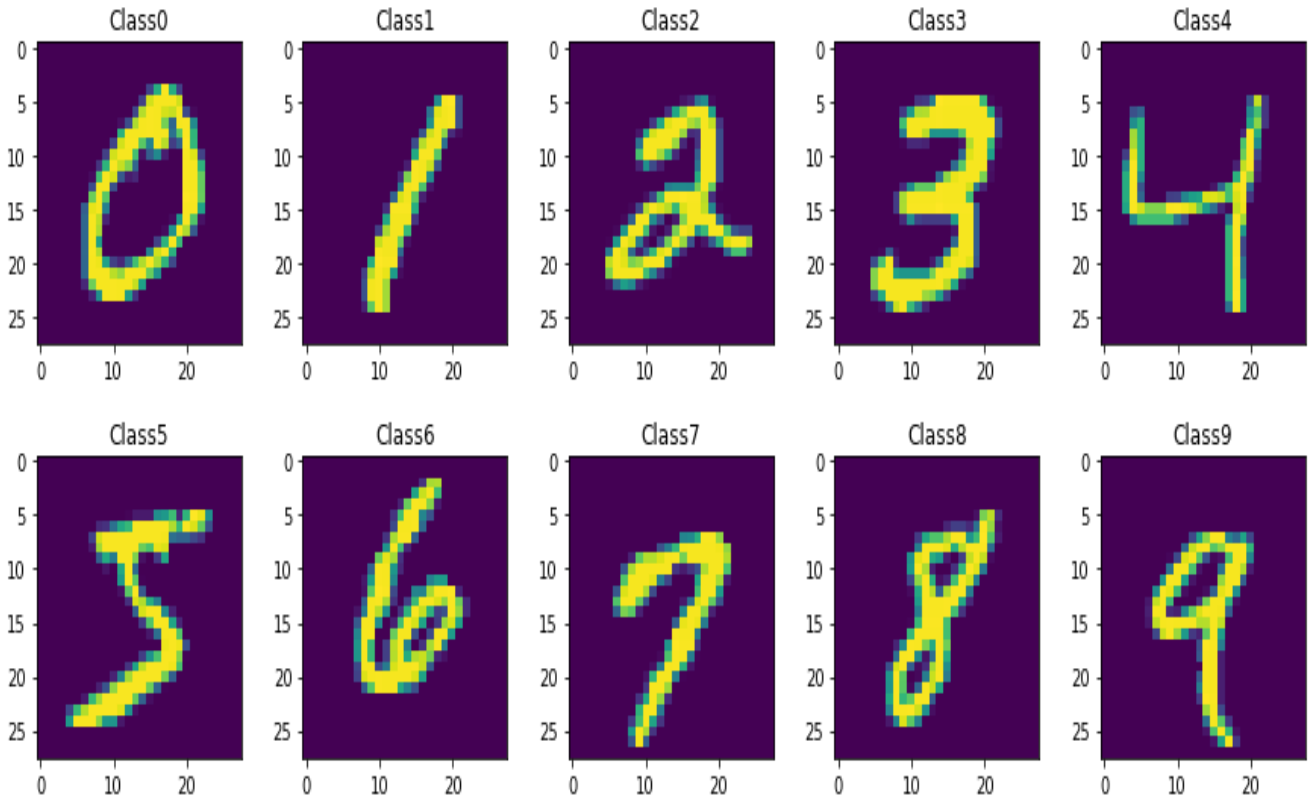
(note: references cited in the code in each section and this report is intended to be just the mention of the important points and results summary of the assignment)

Resources used:

1. Google Colab
2. MNIST Dataset of handwritten digits
3. Tutorials provided in class of course ‘Deep Learning for Imaging’
4. Online references.

About the dataset:

1. The dataset used in the algorithm is MNIST Digits Recognition dataset.
2. It comprises of handwritten digits which are pre-processed to ensure that the digits are centered and size normalized.
3. Train set consist of 60,000 images and test set consists of 10,000 images.



About the algorithm:

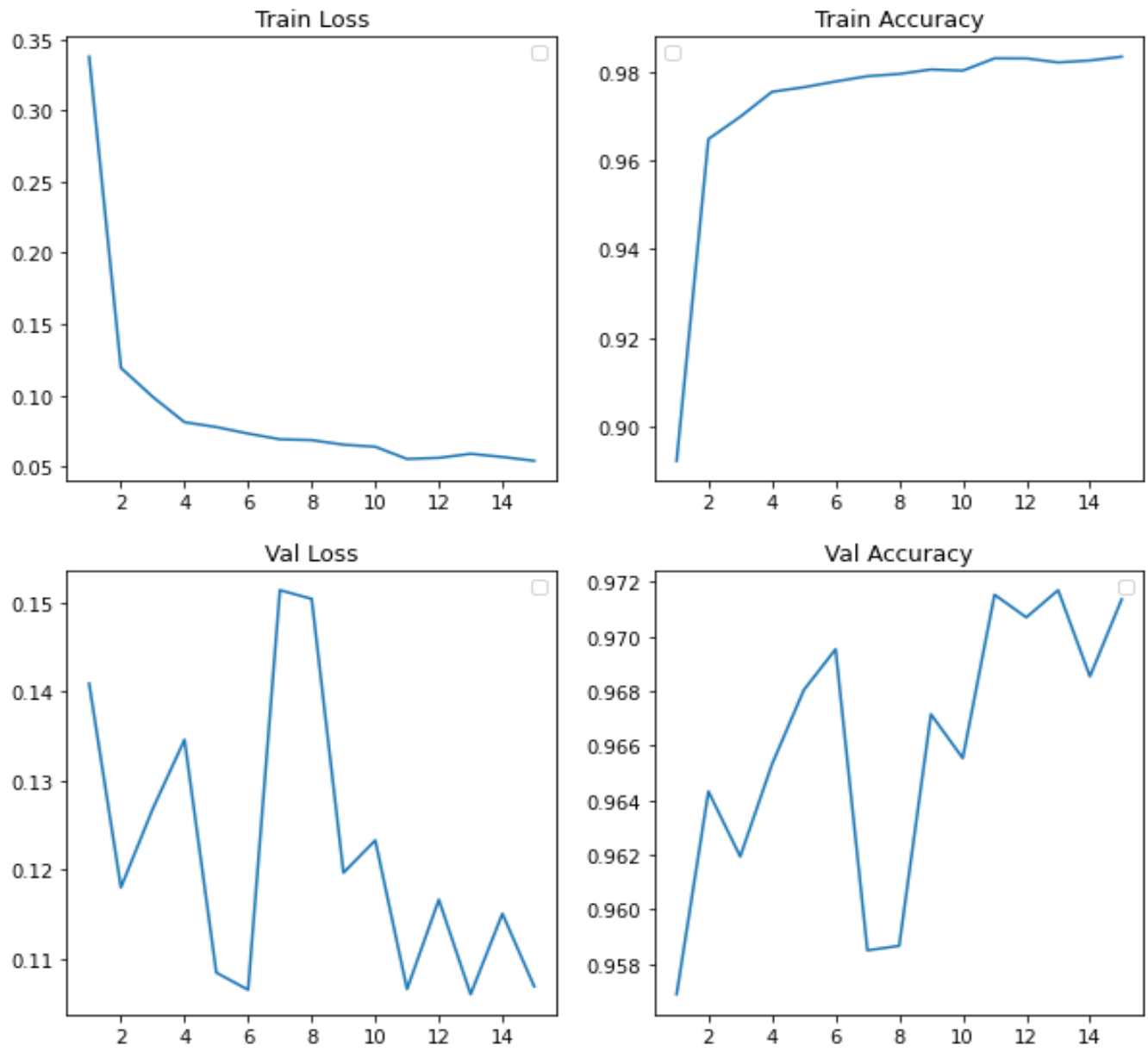
1. The baseline model has the following architecture :
   I/P → h1(500) → h2(250) → h3(100) → O/P.
2. Thus it has 3 hidden layers of sizes [500,250,100] with input layer size as 784 and output layer size as 10 which is the number of classes in the dataset.
3. The algorithm is build with using only numpy and matplotlib library for its functions.
4. Tensorflow.keras is used to import the data and sklearn library is used to split the training data into train and validation data.
5. Weights are initialized using Glorot initialization.
6. Sigmoid activation function used as activation for hidden layers in basemodel and softmax activation function to predict the output.
7. Cross- entropy loss used and also mean squared error tried out in the algorithm.
8. Gradient descent algorithm is used to improve the performance of the feedforward network.
9. Batch size used = 64 and learning rate used is 0.01 with epoch=15.

Baseline model:

With sigmoid as the activation function for hidden layers:

Note: the details about the activation function are mentioned in the code pdf.

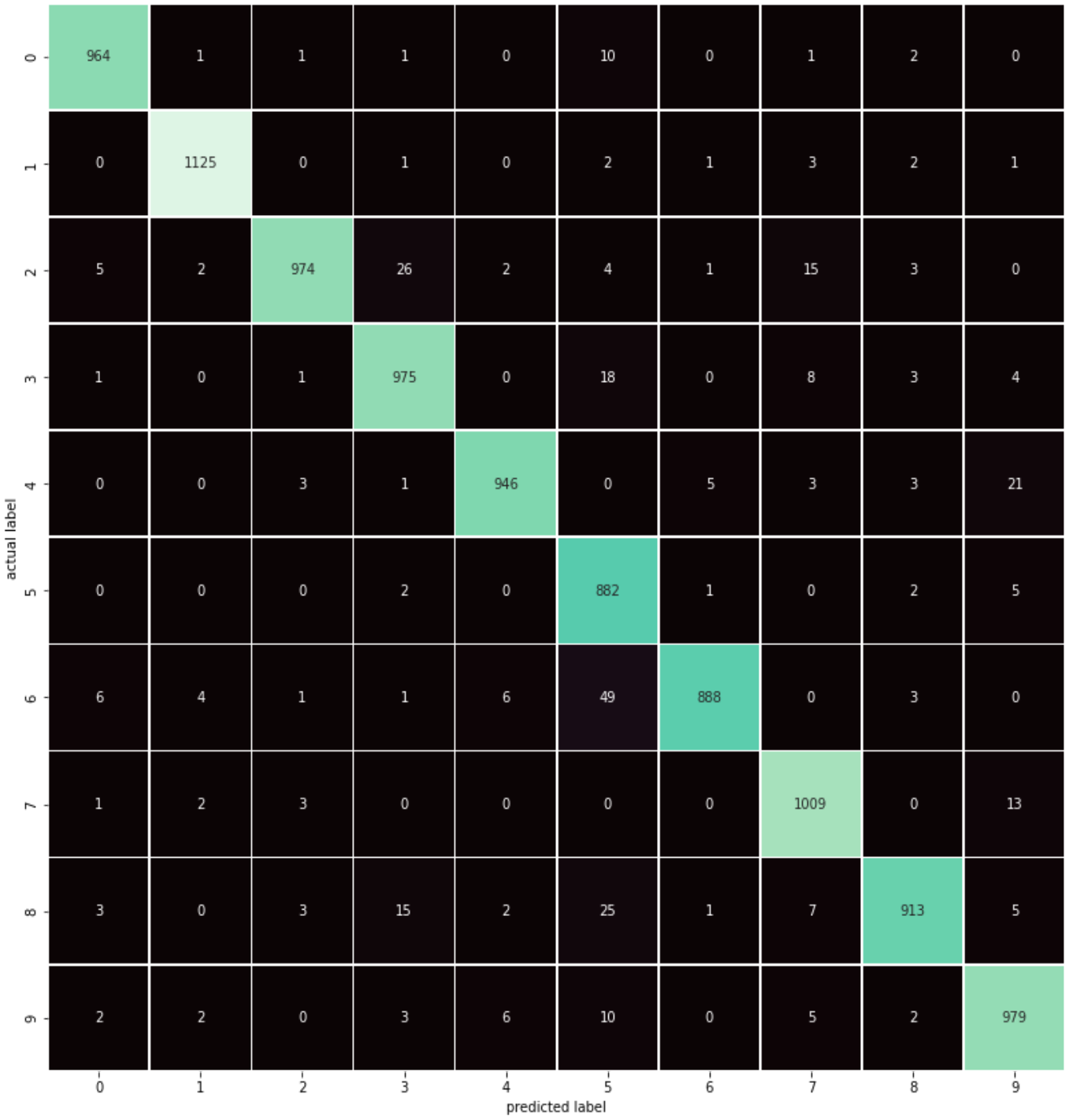
1. After 15 epoch the train loss was 0.053988 and train accuracy was 98.32% whereas test accuracy was 97.01% with test loss as 0.09784
2. The graphs are mentioned below:



Confusion matrix:

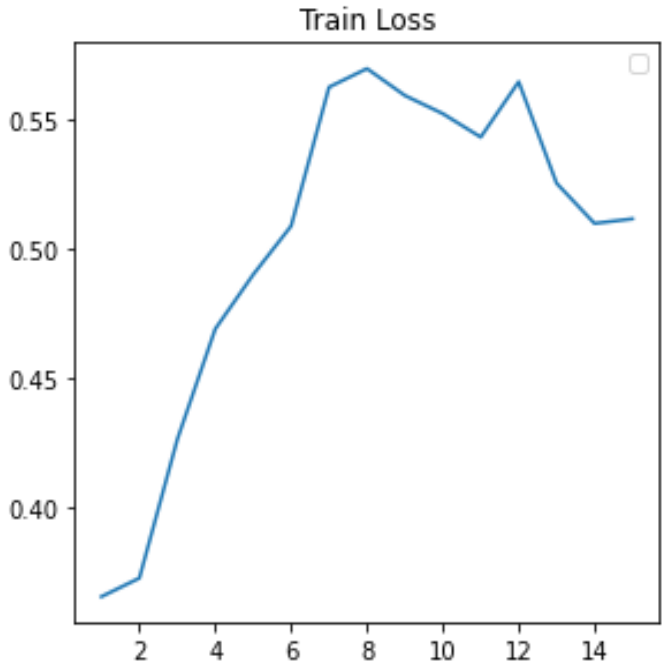
```
Model trained successfully!
[[ 964    1    1    1    0   10    0    1    2    0]
 [   0 1125    0    1    0    2    1    3    2    1]
 [   5    2  974   26    2    4    1   15    3    0]
 [   1    0    1  975    0   18    0    8    3    4]
 [   0    0    3    1  946    0    5    3    3   21]
 [   0    0    0    2    0  882    1    0    2    5]
 [   6    4    1    1    6   49  888    0    3    0]
 [   1    2    3    0    0    0    0 1009    0   13]
 [   3    0    3   15    2   25    1    7  913    5]
 [   2    2    0    3    6   10    0    5    2  979]]
```

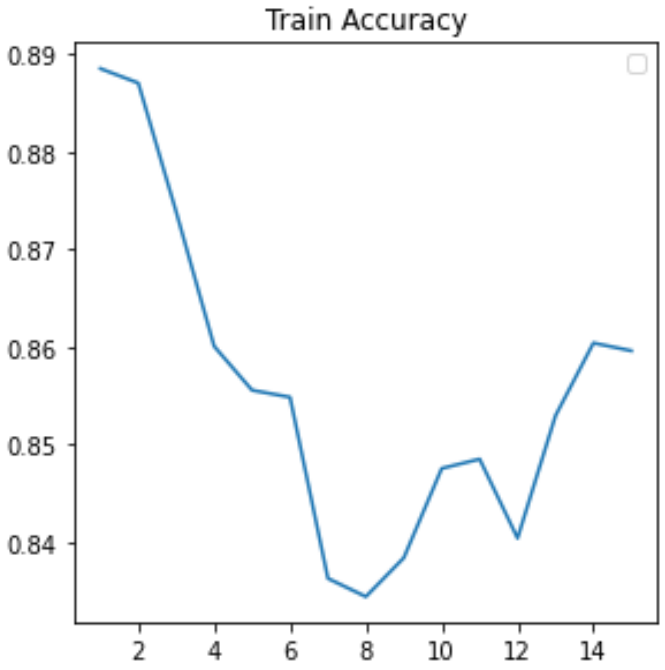
Using seaborn to display the information better: (Sigmoid)

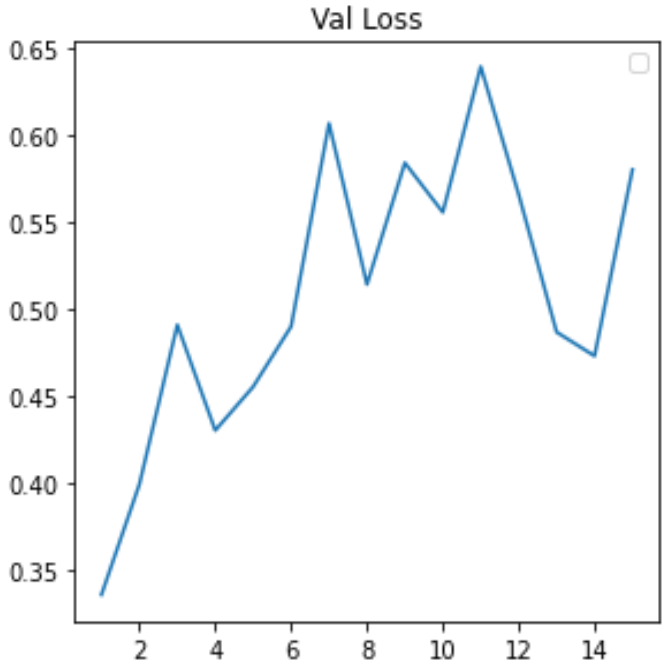


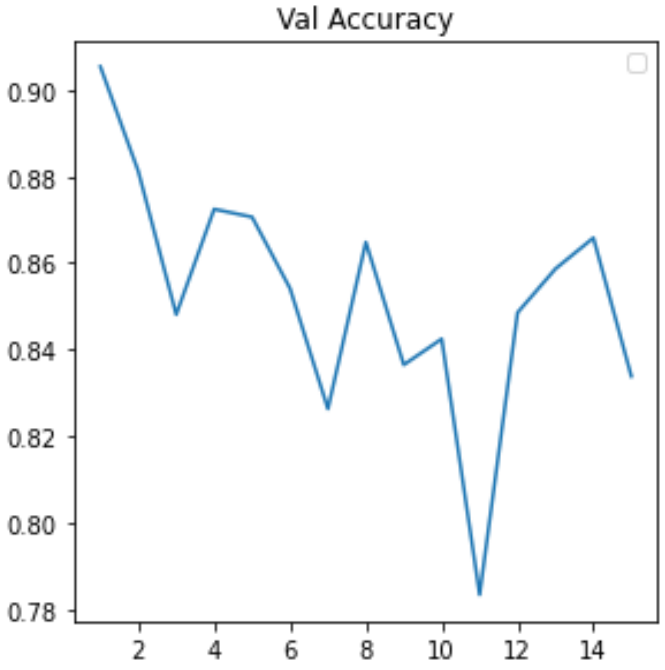
For tanh activation function:

1. The train loss obtained was 0.5118 and train accuracy was 85.96% whereas for test data loss was 0.5333 and accuracy was 84.42%
2. The graphs are mentioned below:





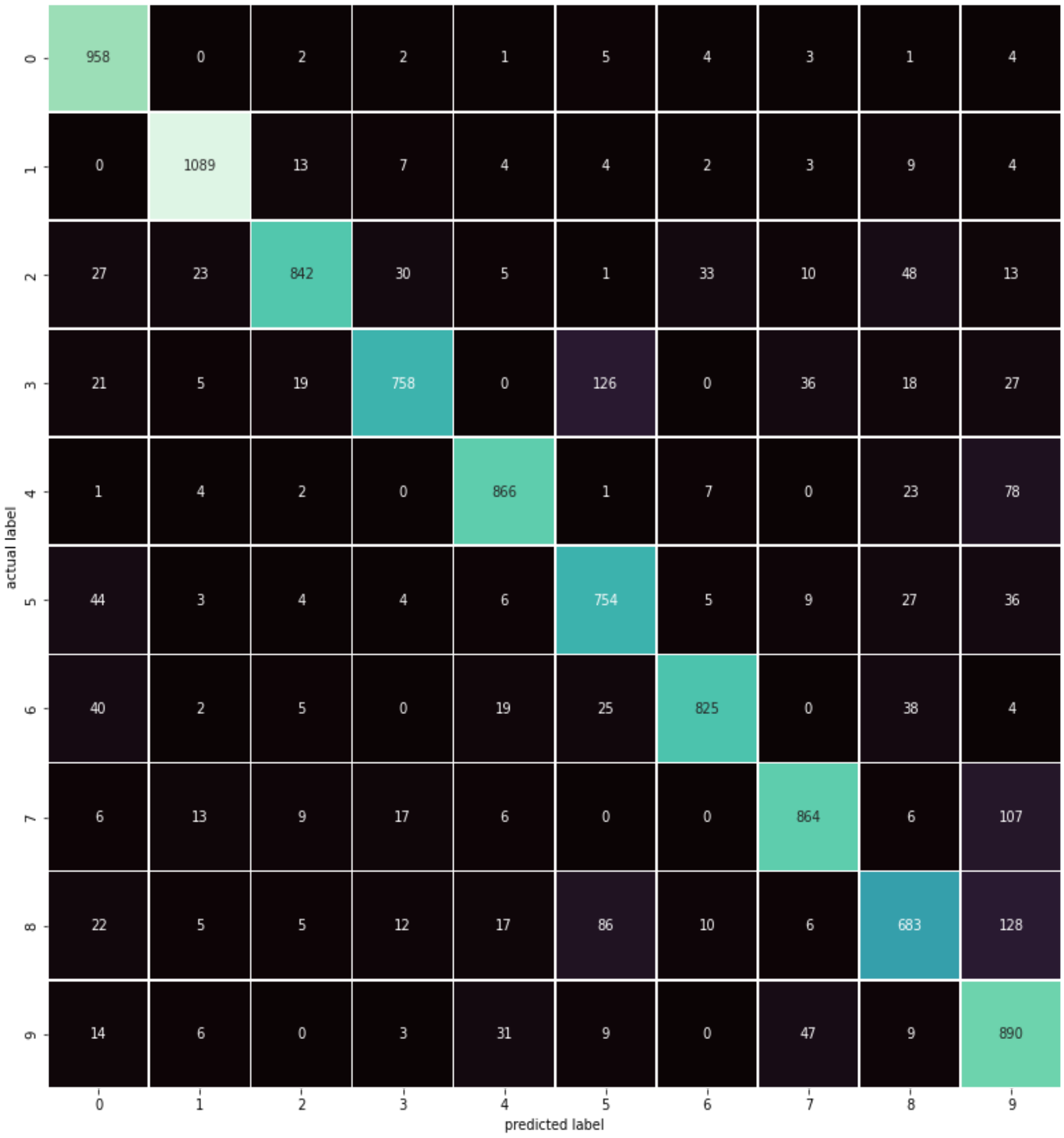




Confusion matrix:

```
Model trained successfully!
[[ 958    0    2    2    1    5    4    3    1    4]
 [   0 1089   13    7    4    4    2    3    9    4]
 [  27   23  842   30    5    1   33   10   48   13]
 [  21    5   19  758    0  126    0   36   18   27]
 [   1    4    2    0  866    1    7    0   23   78]
 [  44    3    4    4    6  754    5    9   27   36]
 [  40    2    5    0   19   25  825    0   38    4]
 [   6   13    9   17    6    0    0  864    6  107]
 [  22    5    5   12   17   86   10    6  683  128]
 [  14    6    0    3   31    9    0   47    9  890]]
```

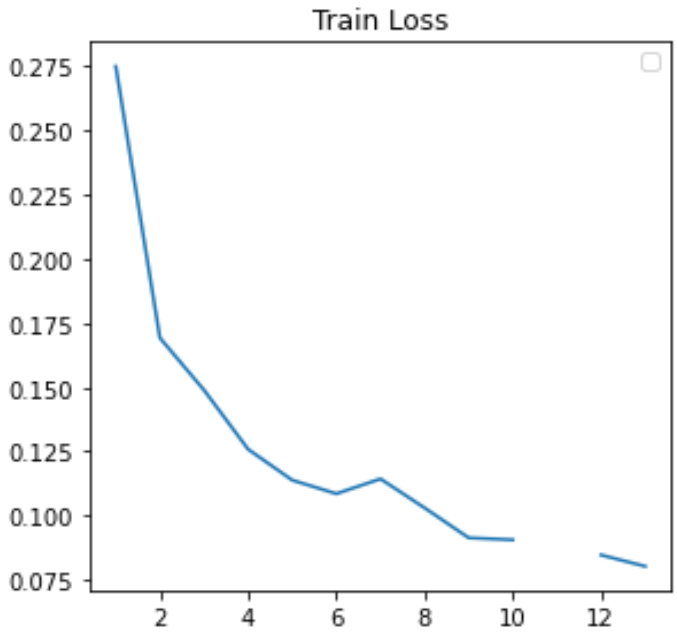
Using seaborn to display the information better: (TanH)

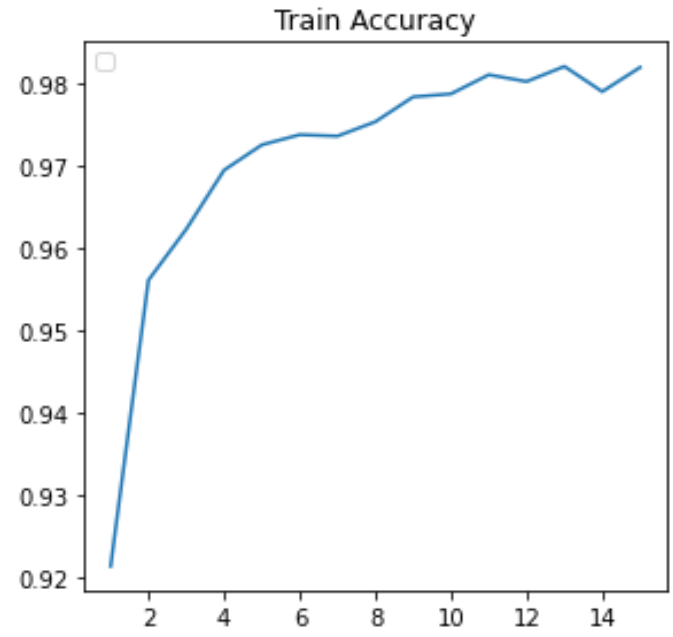


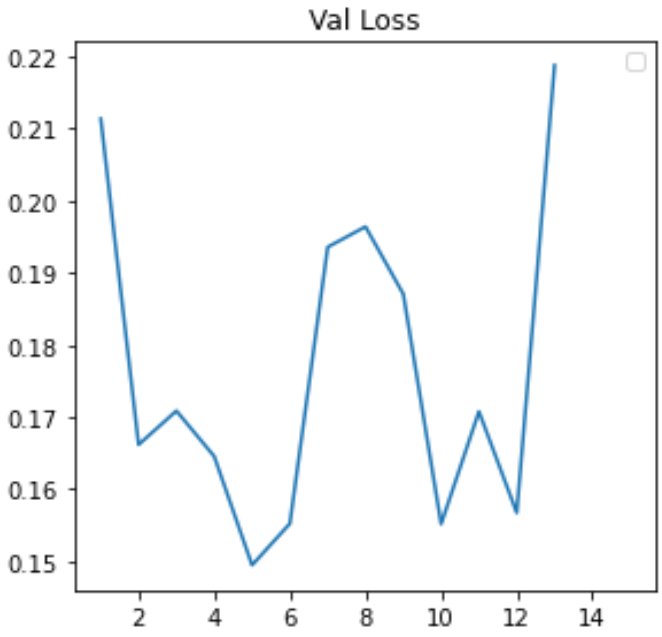
For ReLU activation function:

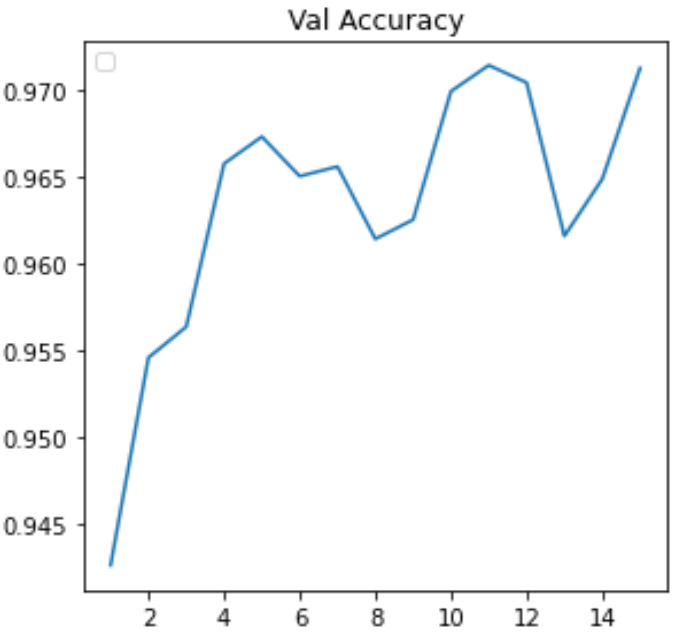
1. The train loss obtained was 0.9820 and train accuracy was 98.20% whereas for test data loss was 0.17688 and accuracy was 97.24%

2. The graphs are mentioned below:





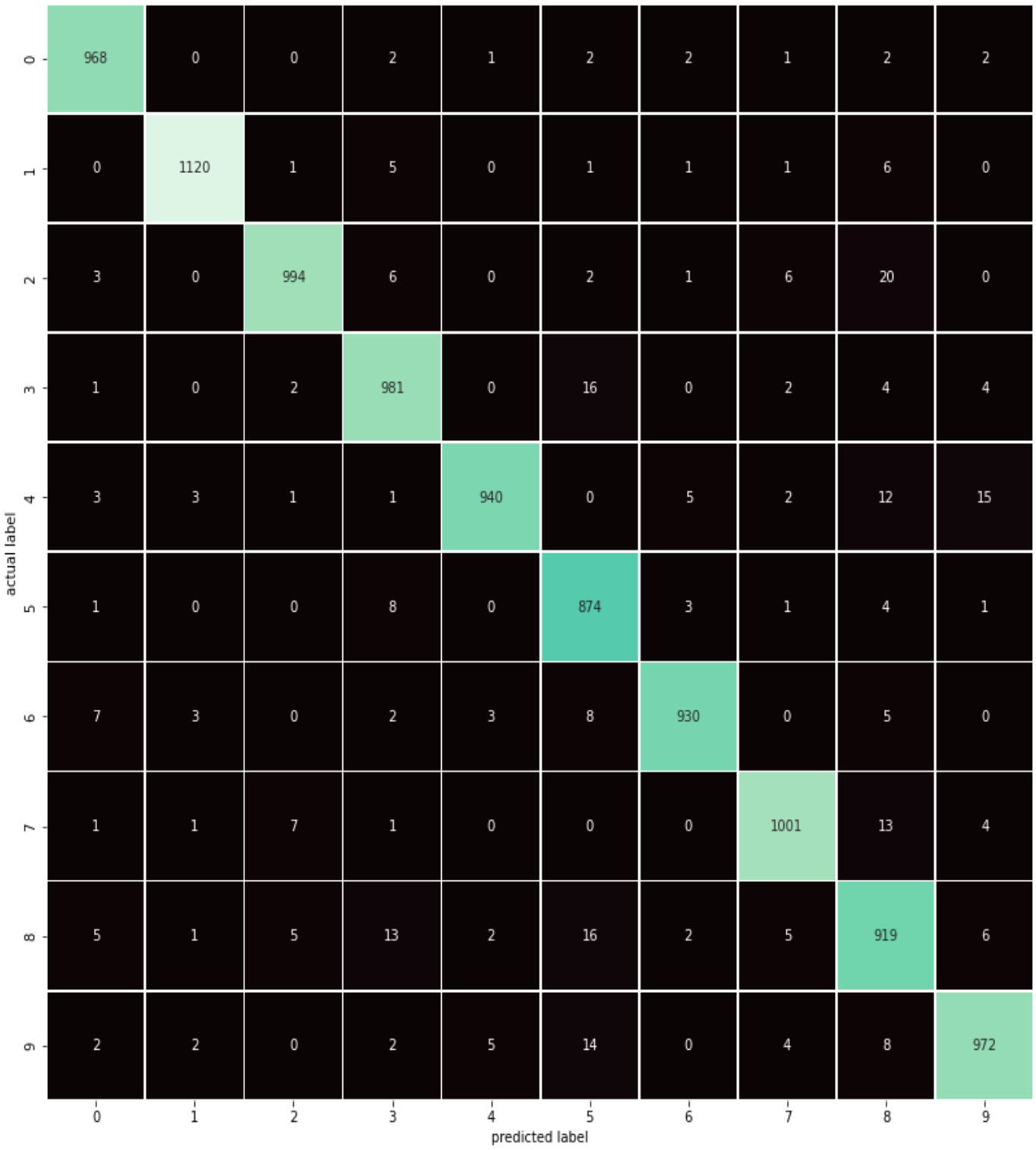




Confusion matrix:

```
Model trained successfully!
[[ 968    0    0    2    1    2    2    1    2    2]
 [   0 1120    1    5    0    1    1    1    6    0]
 [   3    0  994    6    0    2    1    6   20    0]
 [   1    0    2  981    0   16    0    2    4    4]
 [   3    3    1    1  940    0    5    2   12   15]
 [   1    0    0    8    0  874    3    1    4    1]
 [   7    3    0    2    3    8  930    0    5    0]
 [   1    1    7    1    0    0    0 1001   13    4]
 [   5    1    5   13    2   16    2    5  919    6]
 [   2    2    0    2    5   14    0    4    8  972]]
```

Using seaborn to display the information better: (ReLU)


actual label
predicted label
0 1 2 3 4 5 6 7 8 9
968 0 0 2 1 2 2 1 2 2
0 1120 1 5 0 1 1 1 6 0
3 0 994 6 0 2 1 6 20 0
1 0 2 981 0 16 0 2 4 4
3 3 1 1 940 0 5 2 12 15
1 0 0 8 0 874 3 1 4 1
7 3 0 2 3 8 930 0 5 0
1 1 7 1 0 0 0 1001 13 4
5 1 5 13 2 16 2 5 919 6
2 2 0 2 5 14 0 4 8 972

Activation function for all hidden layers	Train accuracy(%)	Test accuracy (%)
sigmoid	98.32	97.01
tanh	85.96	84.42
ReLU	98.20	97.24

(note: epoch= 15, batch size=64 for all the models above)

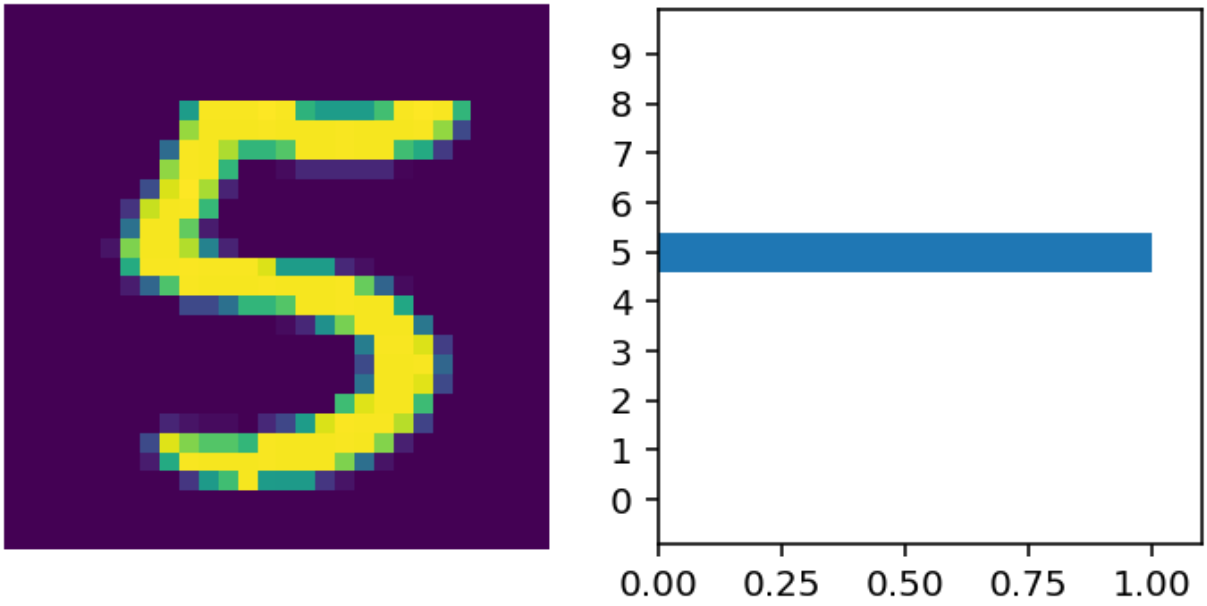
2. USING PYTORCH:

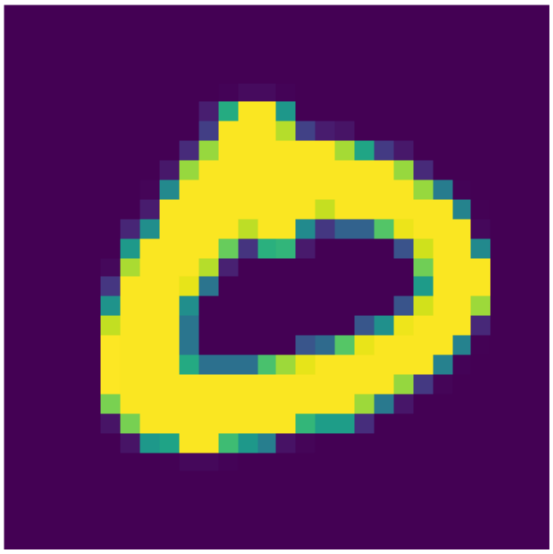
MODEL:

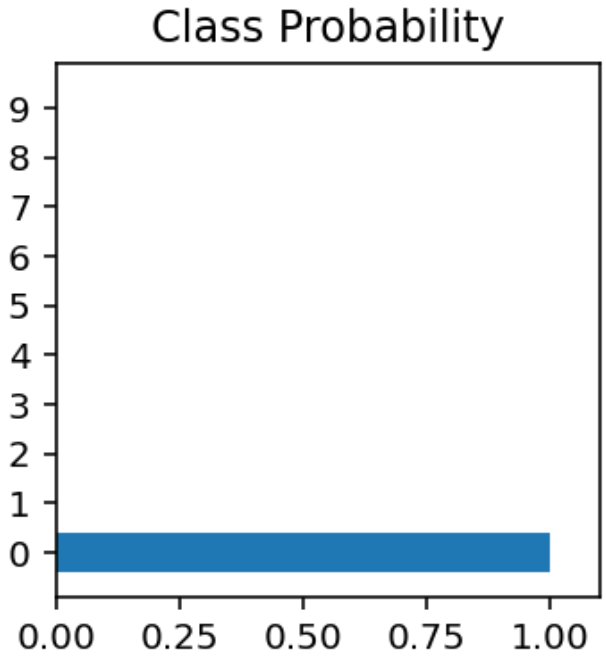
Layers size: [784,500,250,100,10]
Activation function for hidden layers = ReLU
Activation function for output layer: softmax

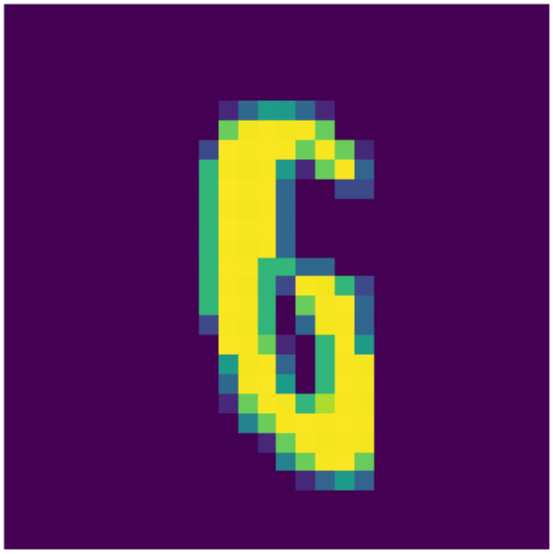
Model accuracy obtained is 97.81%

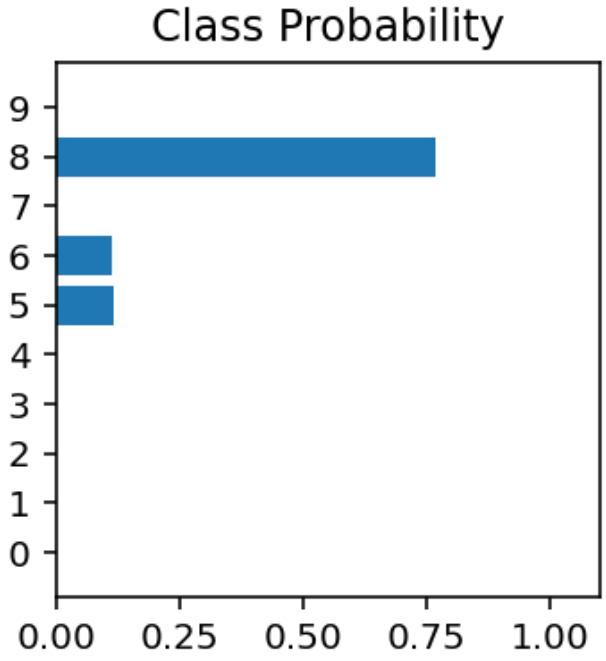
Some predictions are shown below:

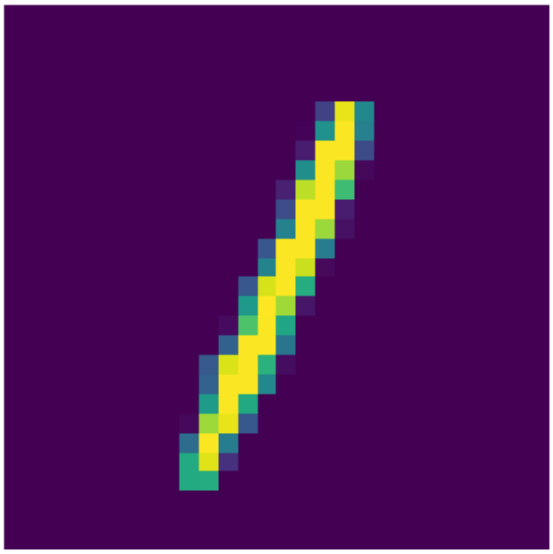


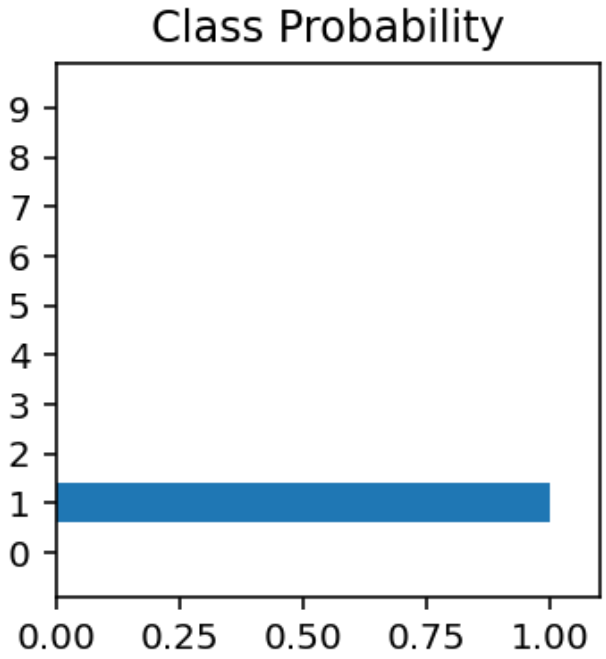


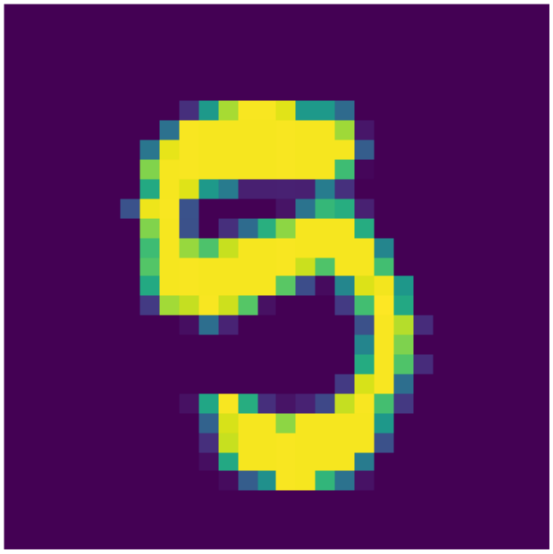

Class Probability
9
8
7
6
5
4
3
2
1
0
0.00
0.25
0.50
0.75
1.00

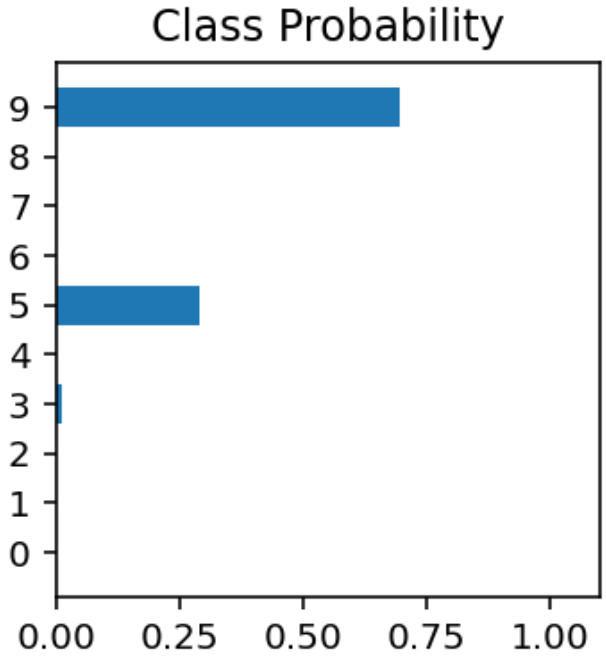



Class Probability
9
8
7
6
5
4
3
2
1
0
0.00
0.25
0.50
0.75
1.00







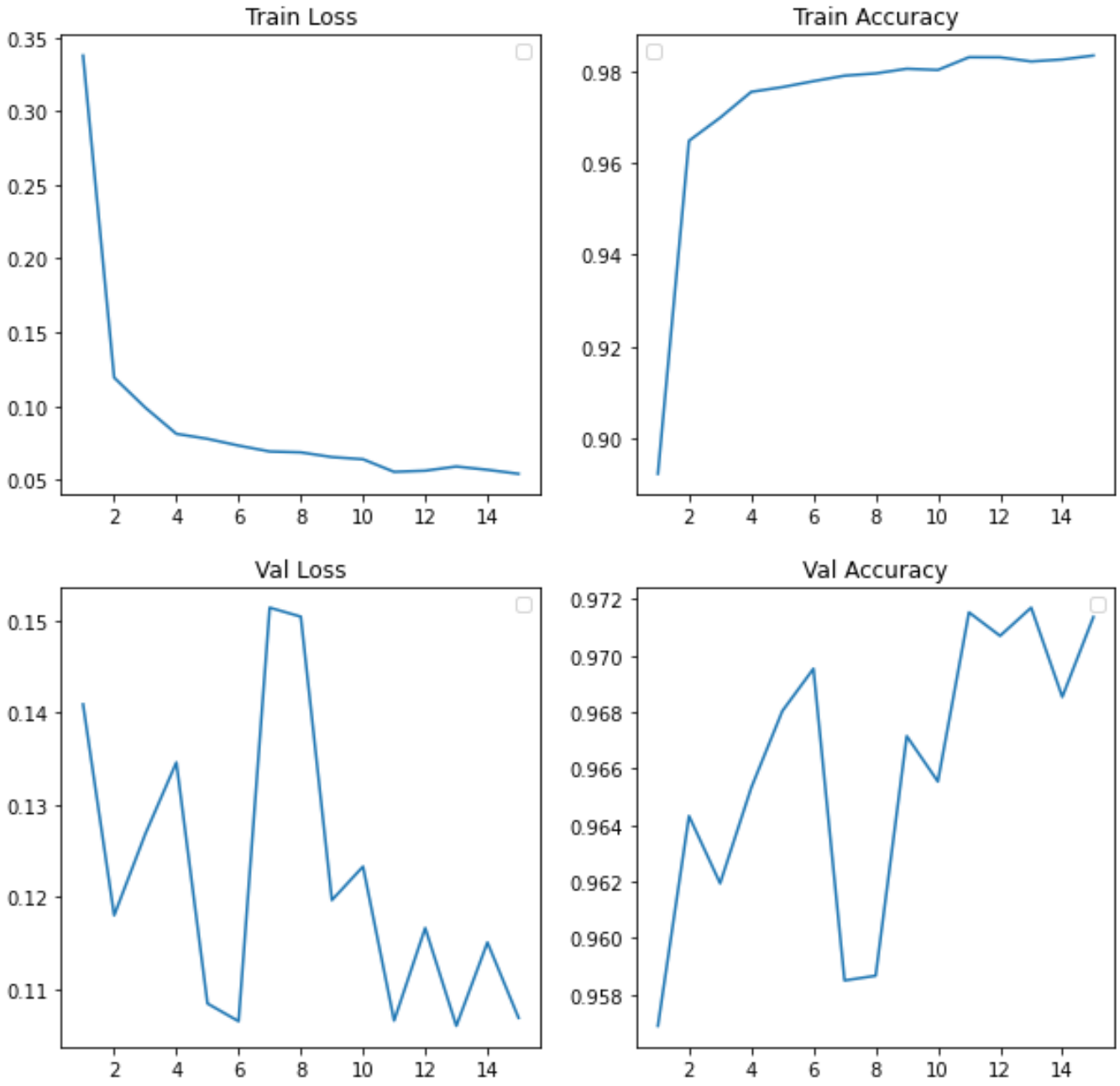


ADAM optimizer was used in this part.

Comparison with the network coded from scratch:

Keeping all the parameters same, the accuracy is mentioned below: (for relu)

	Test accuracy (%)
Network from scratch	**97.24**
Pytorch neural network	**97.81**



L2 regularization:

1. L2 regularization was implemented using sklearn model function Ridge.
2. This model was given a run with different alpha values.
3. The score obtained did not differ much with a lot of change in alpha.
4. The details of the results are in the code pdf.

Trying out different combinations of hyperparameters with best performing activation function ReLU:

MODEL0:

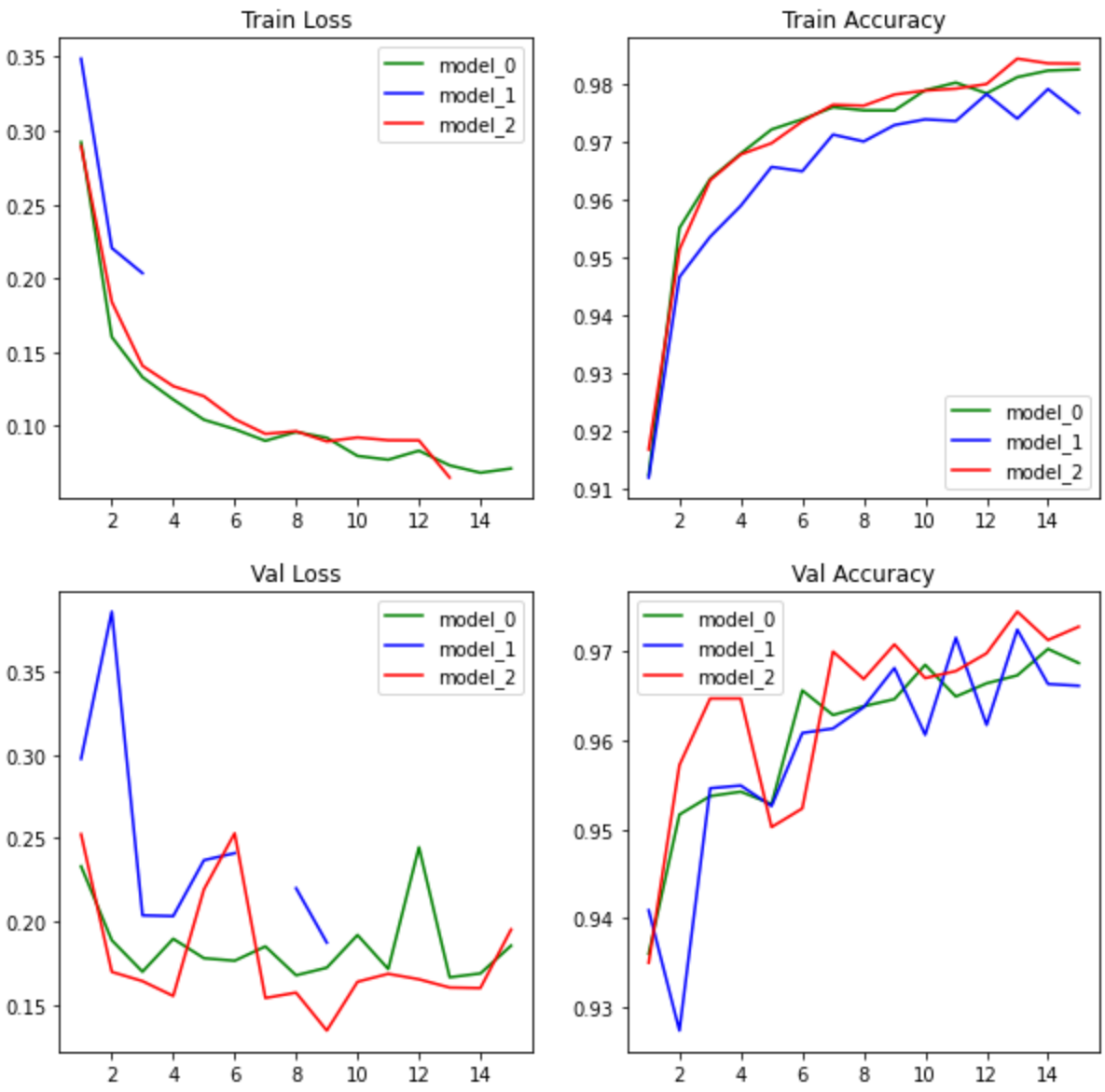
ReLU activation function
Layer sizes= [784,128,250,100,10]
Epoch = 15
Batch size= 128

MODEL1:

ReLU activation function
Layer sizes= [784,750,128,100,10]
Epoch = 15
Batch size= 25

MODEL2:

ReLU activation function
Layer sizes= [784,500,50,250,10]
Epoch = 15
Batch size= 64


Train Loss
model_0
model_1
model_2
Train Accuracy
model_0
model_1
model_2
Val Loss
model_0
model_1
model_2
Val Accuracy
model_0
model_1
model_2