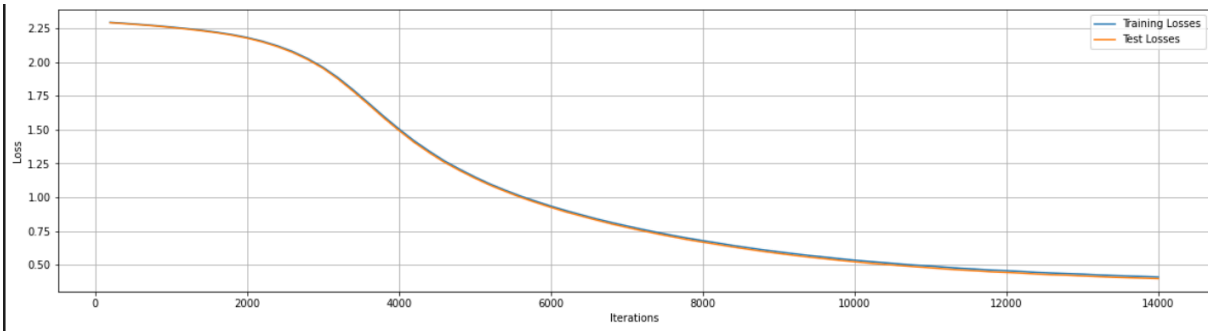
EE5179:Deep Learning for Imaging Programming
Assignment 1: MNIST Classification using MLP
Anirud N CE21B014

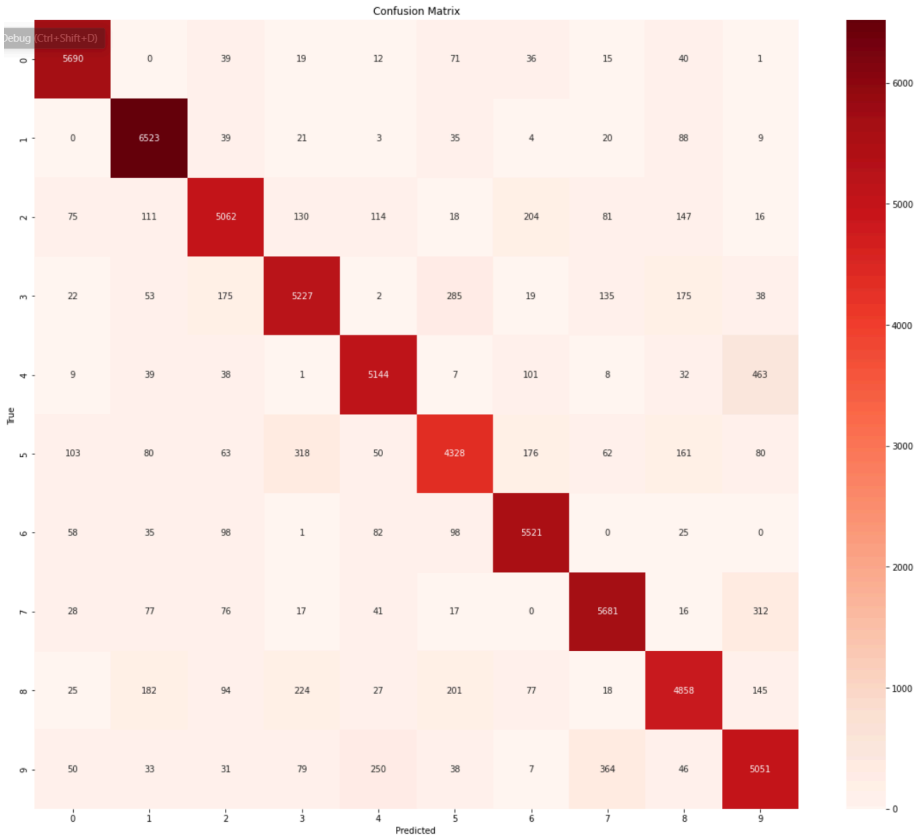
Baseline Performance

Implemented the baseline model - with Sigmoid activation function for the proposed learning rate and architecture. Built from scratch without using the existing libraries.

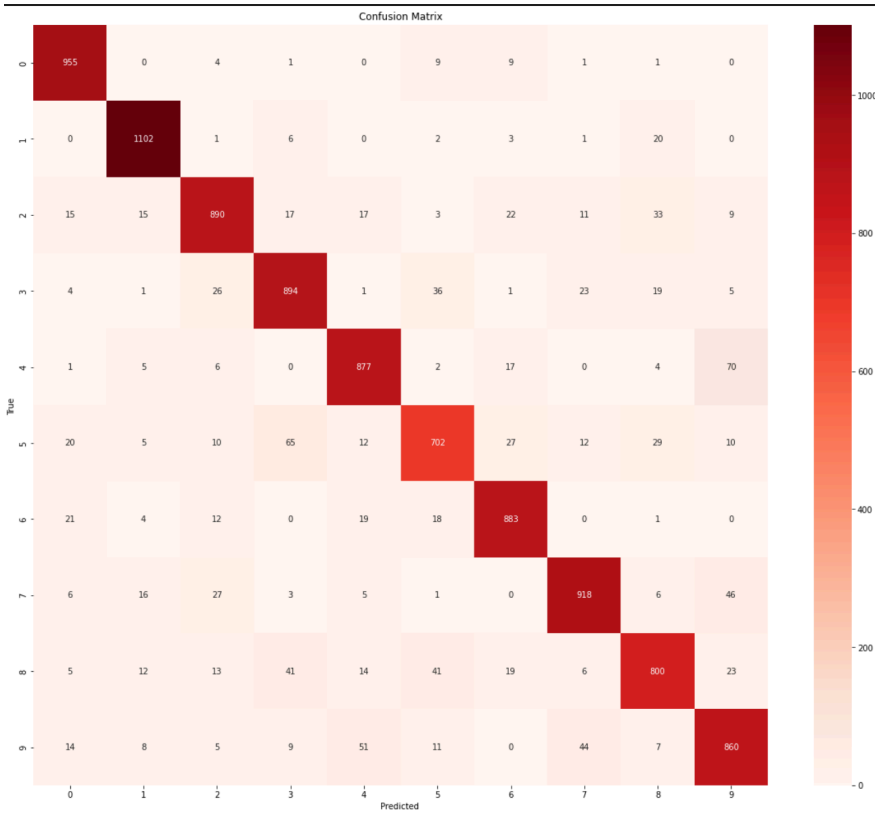
Plot for Iteration vs Loss for both training and testing:



Confusion matrix for better understanding of the mistakes made:
Train:



Test:



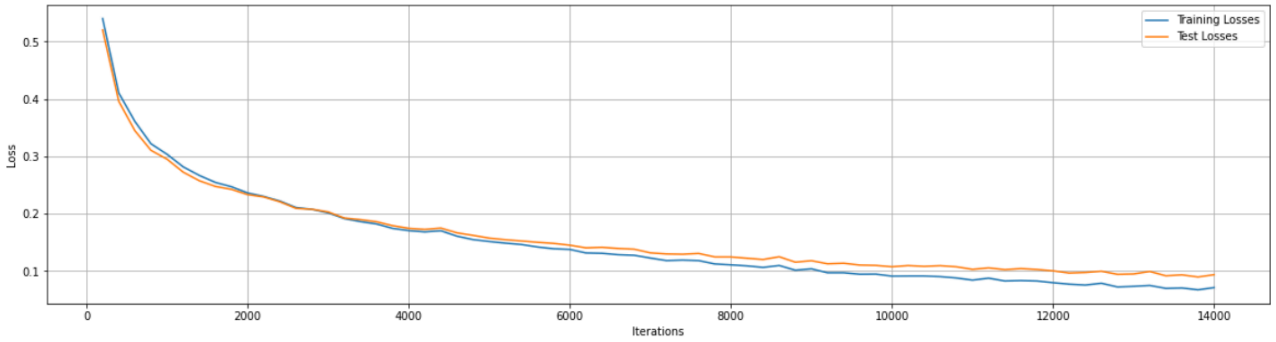
Accuracy achieved on the test and train dataset:
Test Accuracy: 88.81%

Train Accuracy : 88.48%
Surprisingly, train accuracy is lesser than test accuracy
From the confusion matrix we can see that at times 9 is confused as 4 and 7, 3 is confused as 5, 2 is confused as 6, and 4 is confused as 9 etc.
Hyperparameter tuning
In order to allow for hyperparameter tuning using already existing packages, such as wand (weights and biases), the function was made to have many arguments as hyperparameters. This function could then be passed to find those set of hyperparams that gives the most optimal solution i.e. minimise the error.

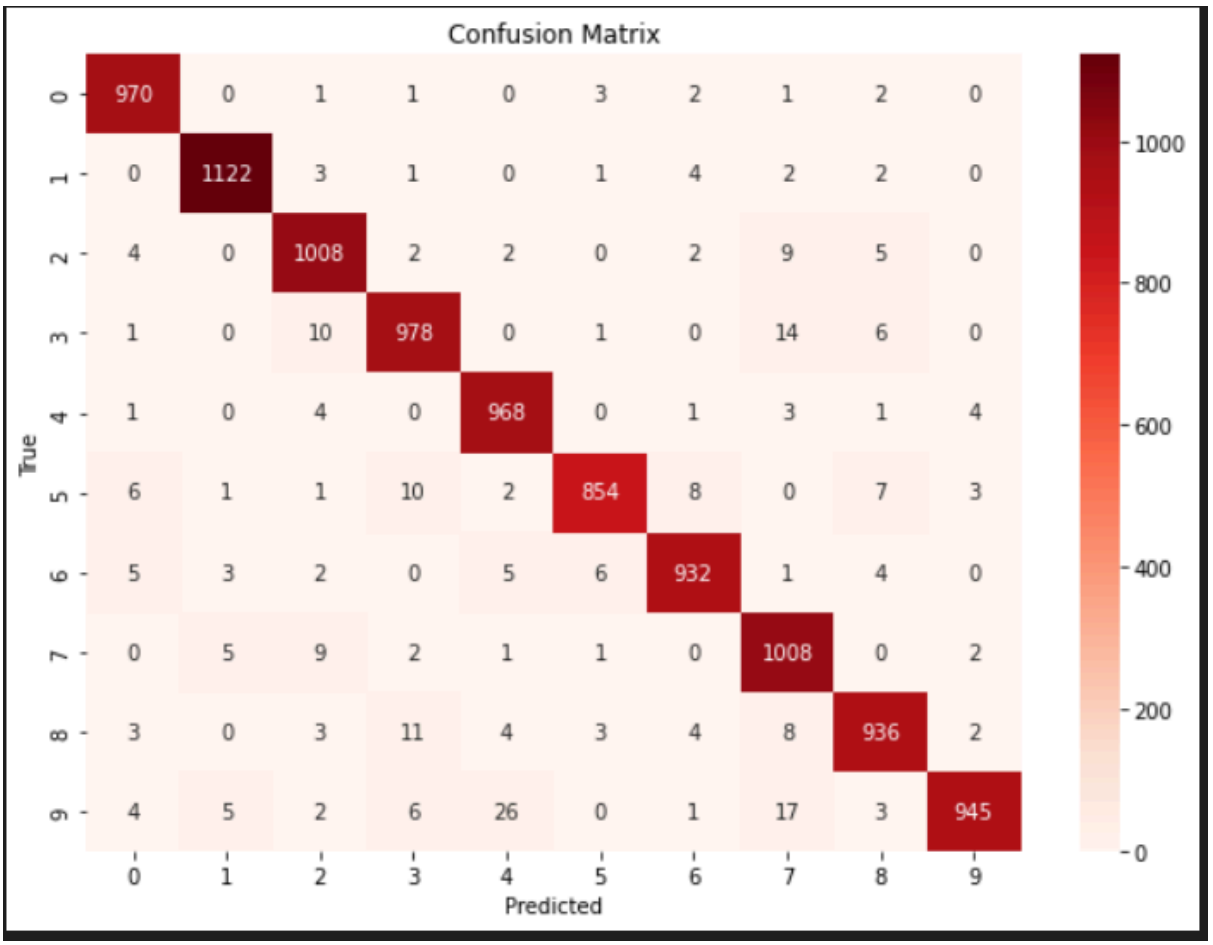
Activation Function Changed :

1) TanH activation function :

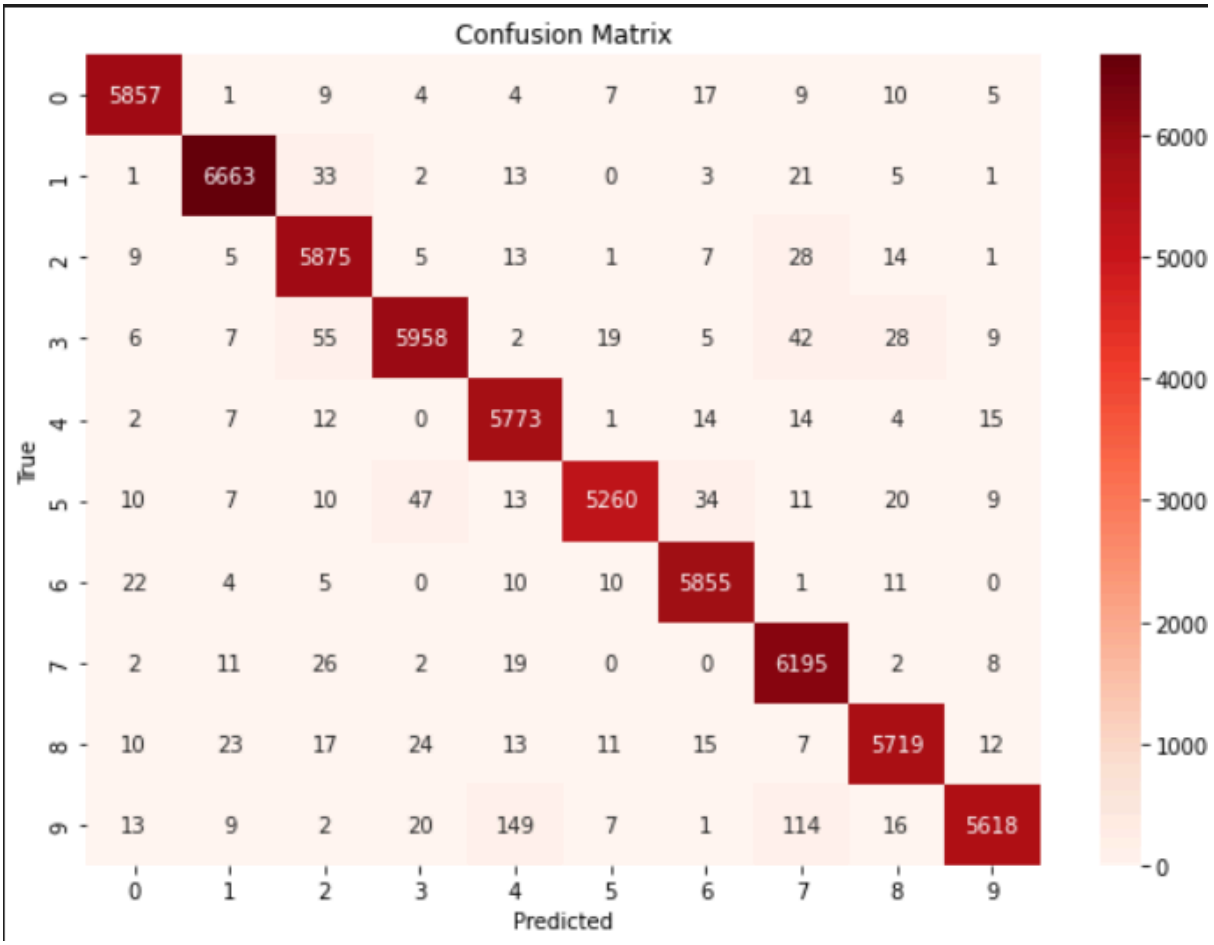
Plot for Iteration vs Loss for both training and testing:



Confusion matrix for better understanding of the mistakes made:
Test :


Confusion Matrix
True
Predicted
0 970 0 1 1 0 3 2 1 2 0
1 0 1122 3 1 0 1 4 2 2 0
2 4 0 1008 2 2 0 2 9 5 0
3 1 0 10 978 0 1 0 14 6 0
4 1 0 4 0 968 0 1 3 1 4
5 6 1 1 10 2 854 8 0 7 3
6 5 3 2 0 5 6 932 1 4 0
7 0 5 9 2 1 1 0 1008 0 2
8 3 0 3 11 4 3 4 8 936 2
9 4 5 2 6 26 0 1 17 3 945
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

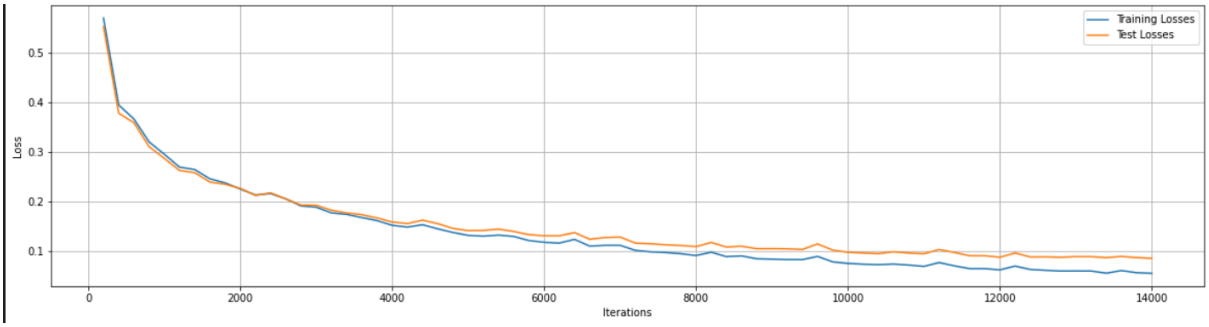
Train



Accuracy achieved on the test and train dataset:
Test Accuracy : 97.21 %
Train Accuracy : 97.95 %
Comparison:
Issues with 9 predicted as 4, still persist, but 4 is not that often being predicted as 9. This shows improvement and this could be even seen in the accuracy values that using a tanh activation function gives a very high accuracy and increase of over 10% from the baseline model. This could be attributed to **quicker learning** which can be inferred from the plots for learning curves above. The tanh losses start from <1 while training, showing better and quicker learning.

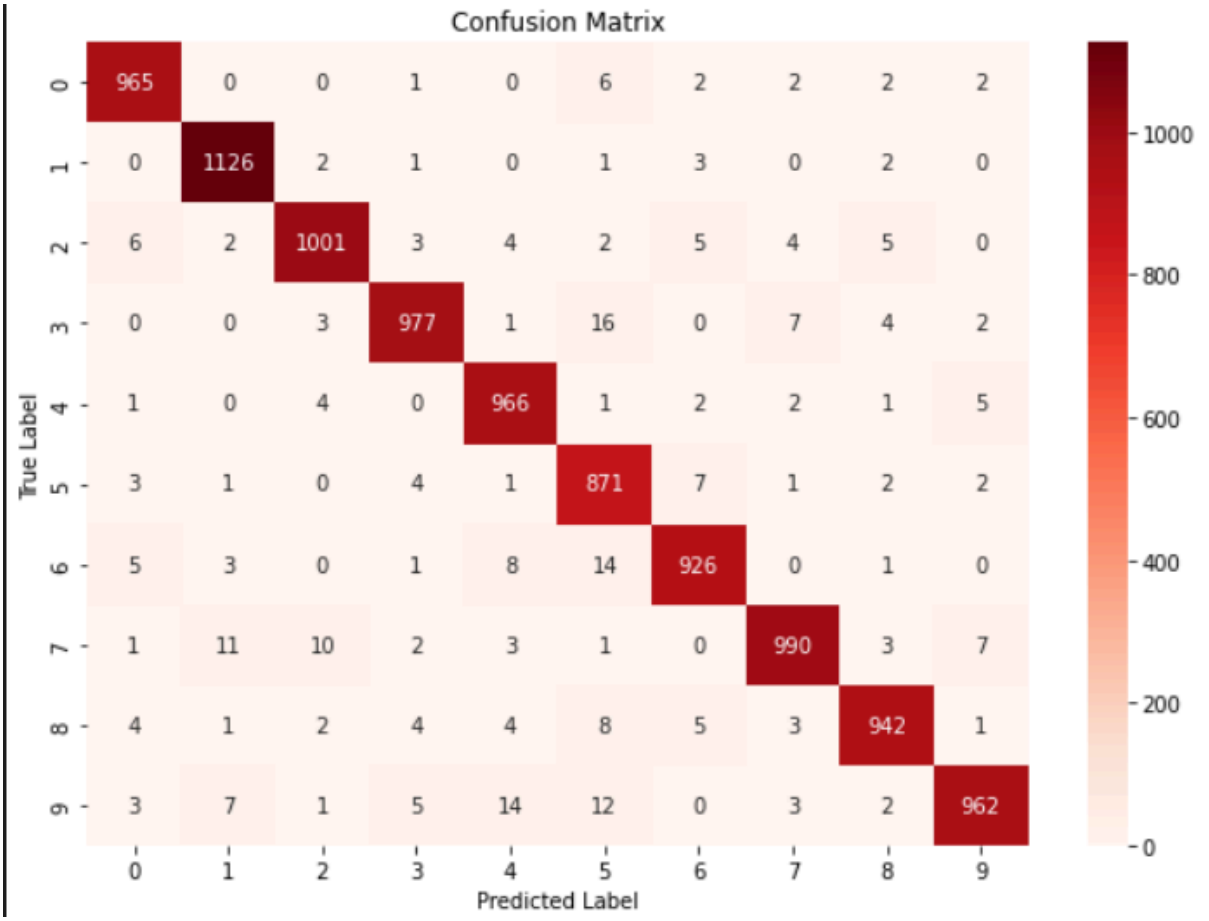
2) ReLu Activation function :

Plot for Iteration vs Loss for both training and testing:

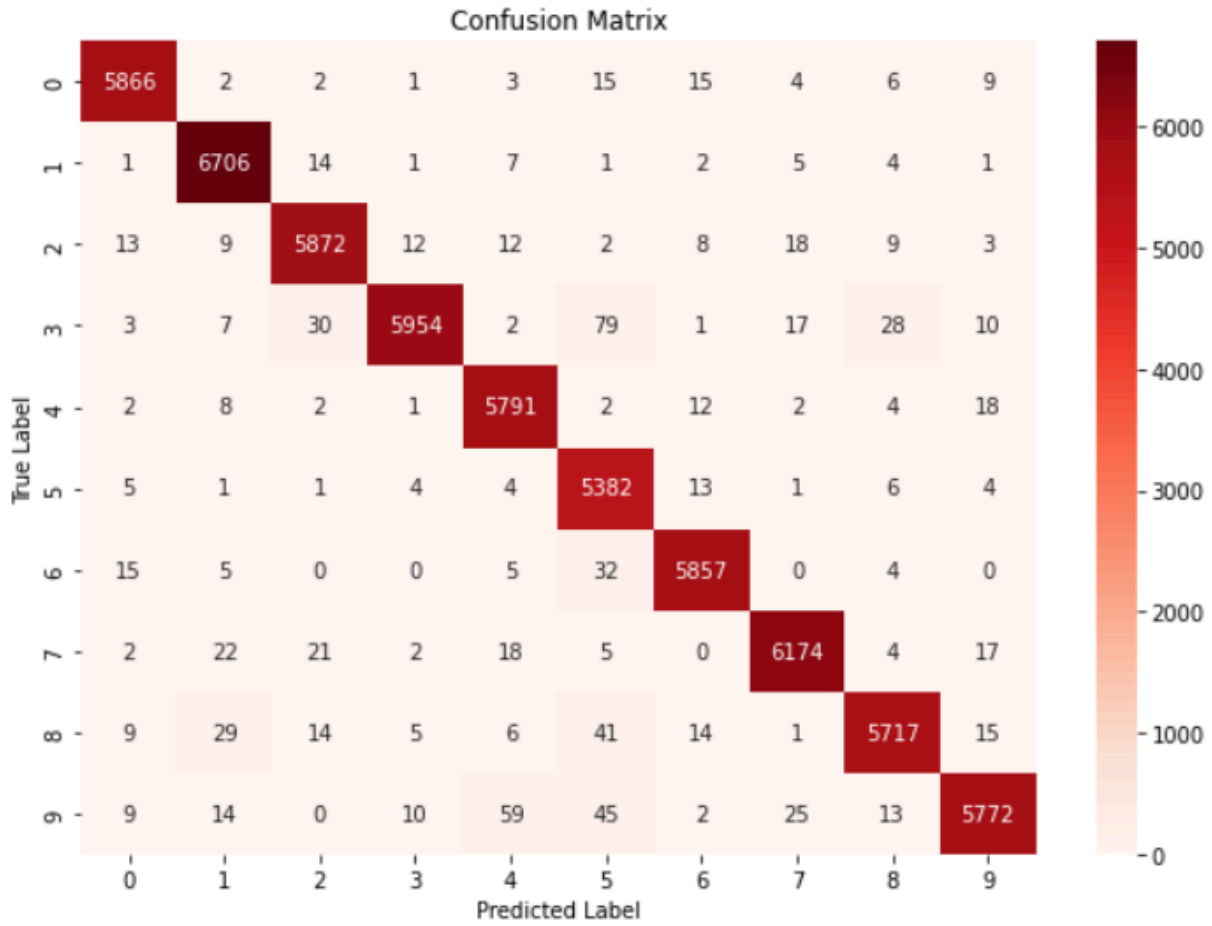


Confusion matrix for better understanding of the mistakes made:

Test :



Train



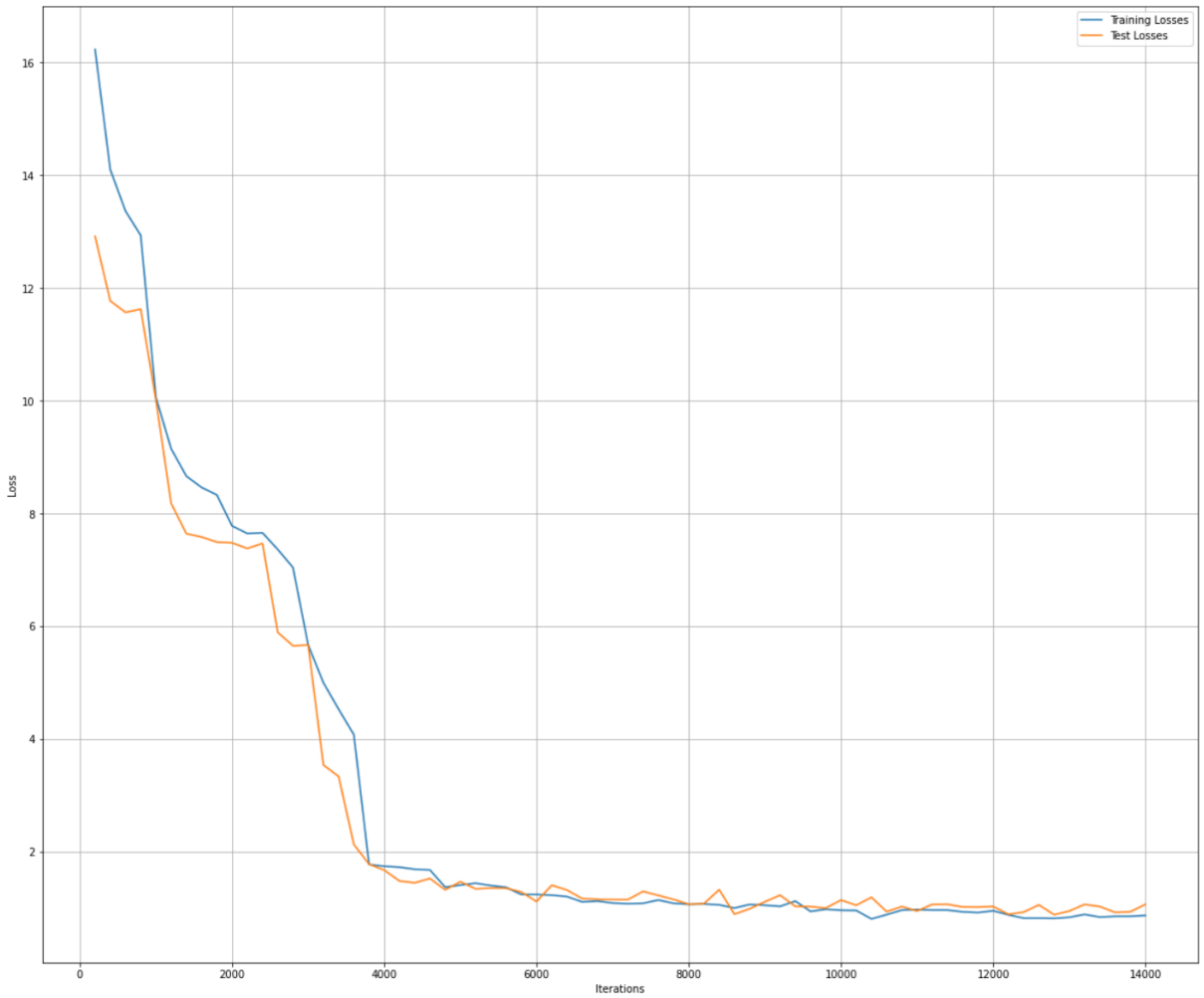
Accuracy achieved on the test and train dataset:
Test Accuracy : 97.28%
Train Accuracy : 98.48%

Comparison
This gives an increased performance in Train data. Relu gives better accuracy in 15 epochs. But a new problem is identified: **From the learning plots, it is now evident that as the number of epochs increase the gap between the train and test errors are also increasing. ReLU in our case, is more prone to overfitting for higher number of epochs if this continues.**

Pytorch Implementation:

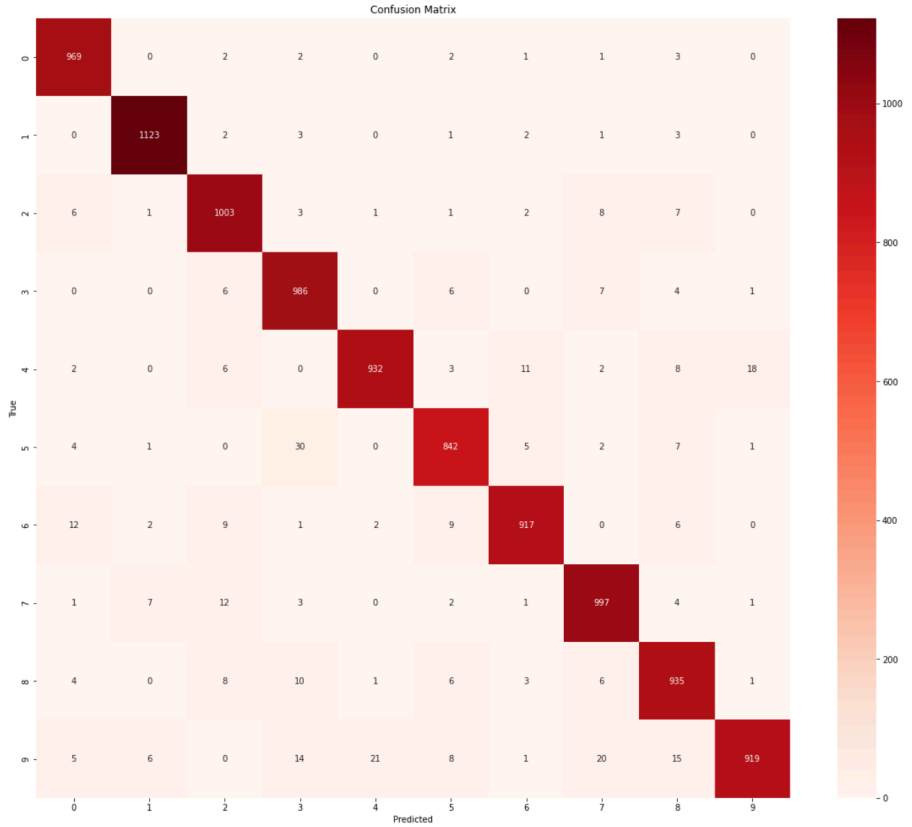
1) **Baseline Pytorch Implementation with Sigmoid activation**

Plot for Iteration vs Loss for both training and testing:

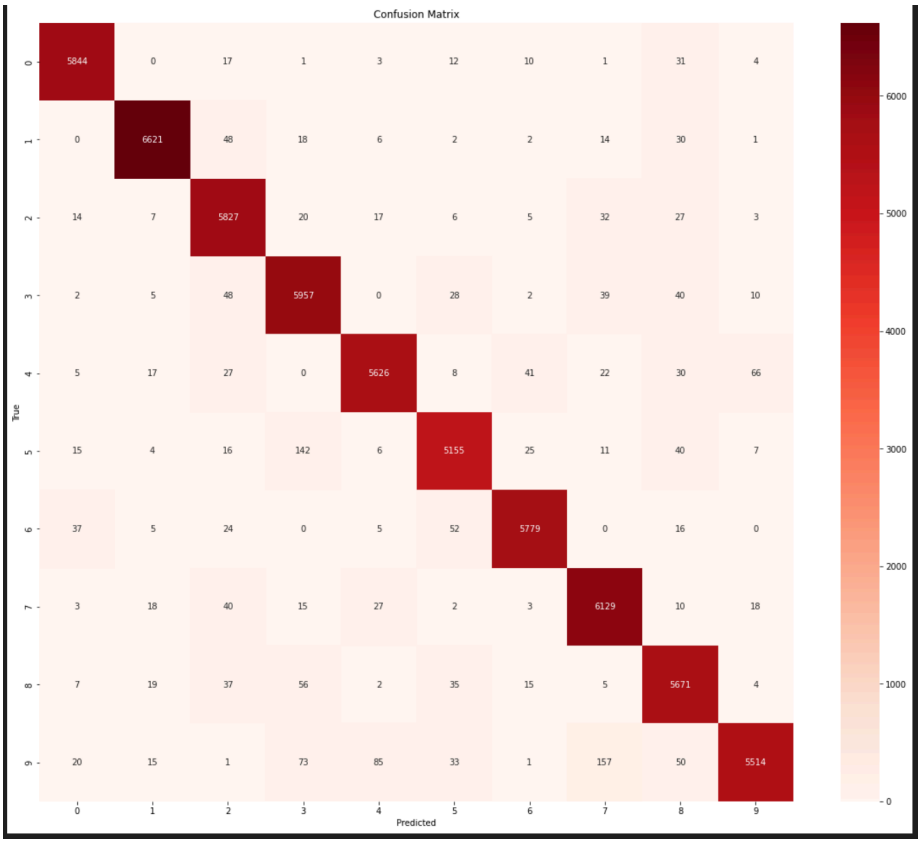


Confusion matrix for better understanding of the mistakes made:

Test:



Train

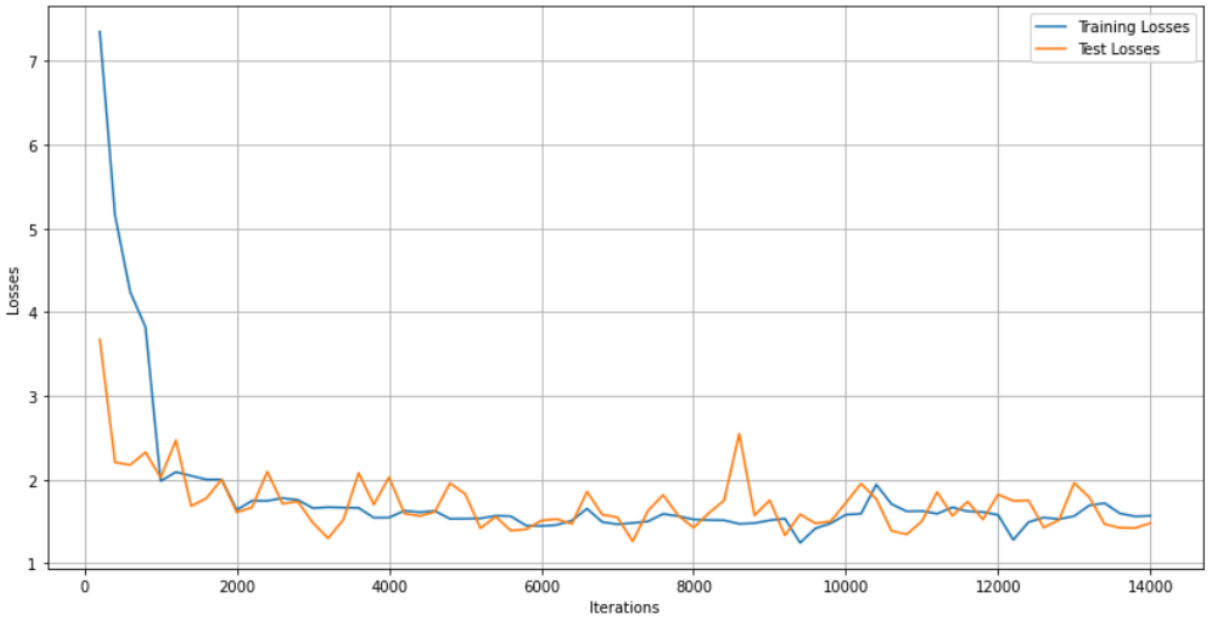


Accuracy achieved on the test and train dataset:
Test accuracy : 96.23%
Train Accuracy : 96.87%

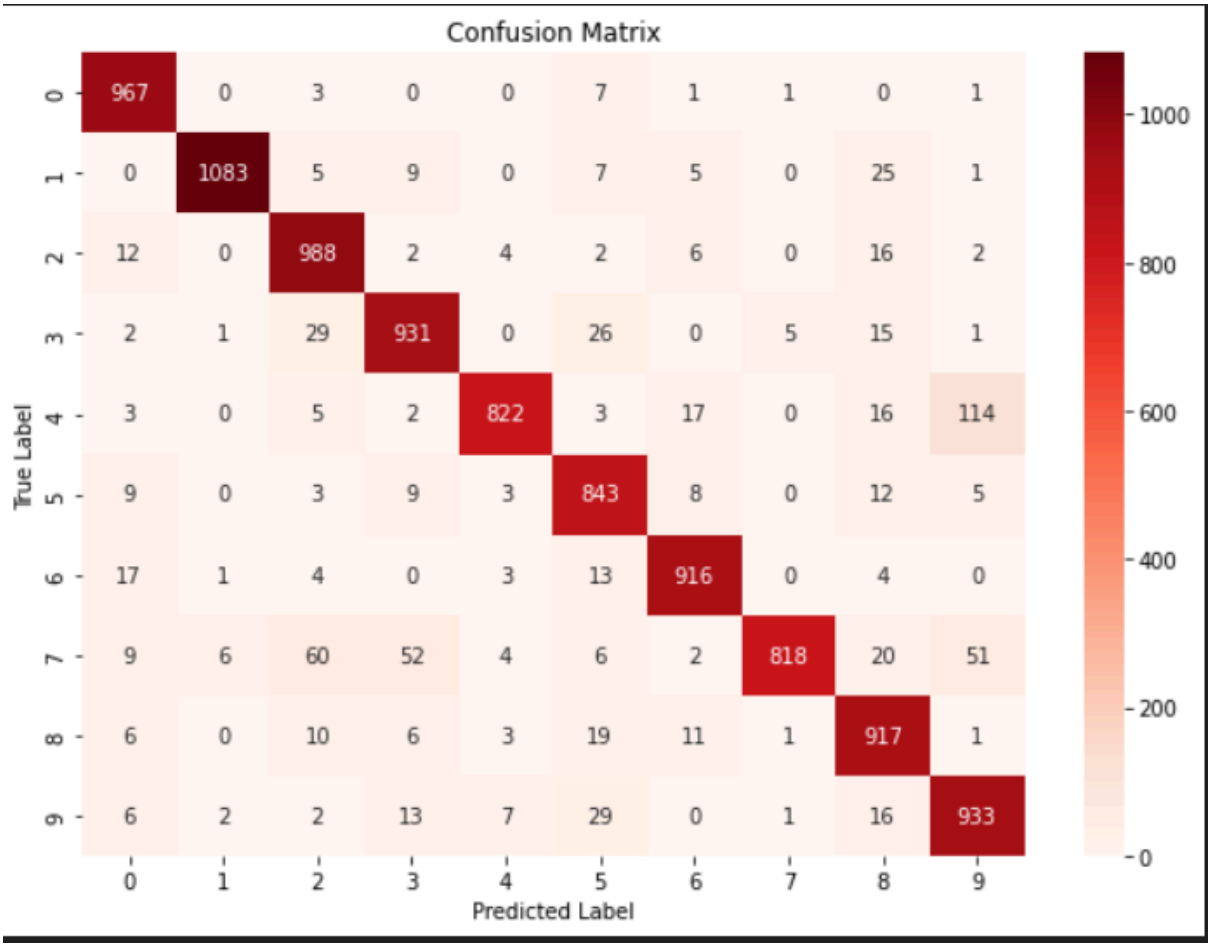
Comparison:
This gives better results than the code I had made so far from scratch for the MLP. Predicting class 9 faces issues as it is confused with many other numbers such as 3,4 etc

2) Pytorch + Tanh activation

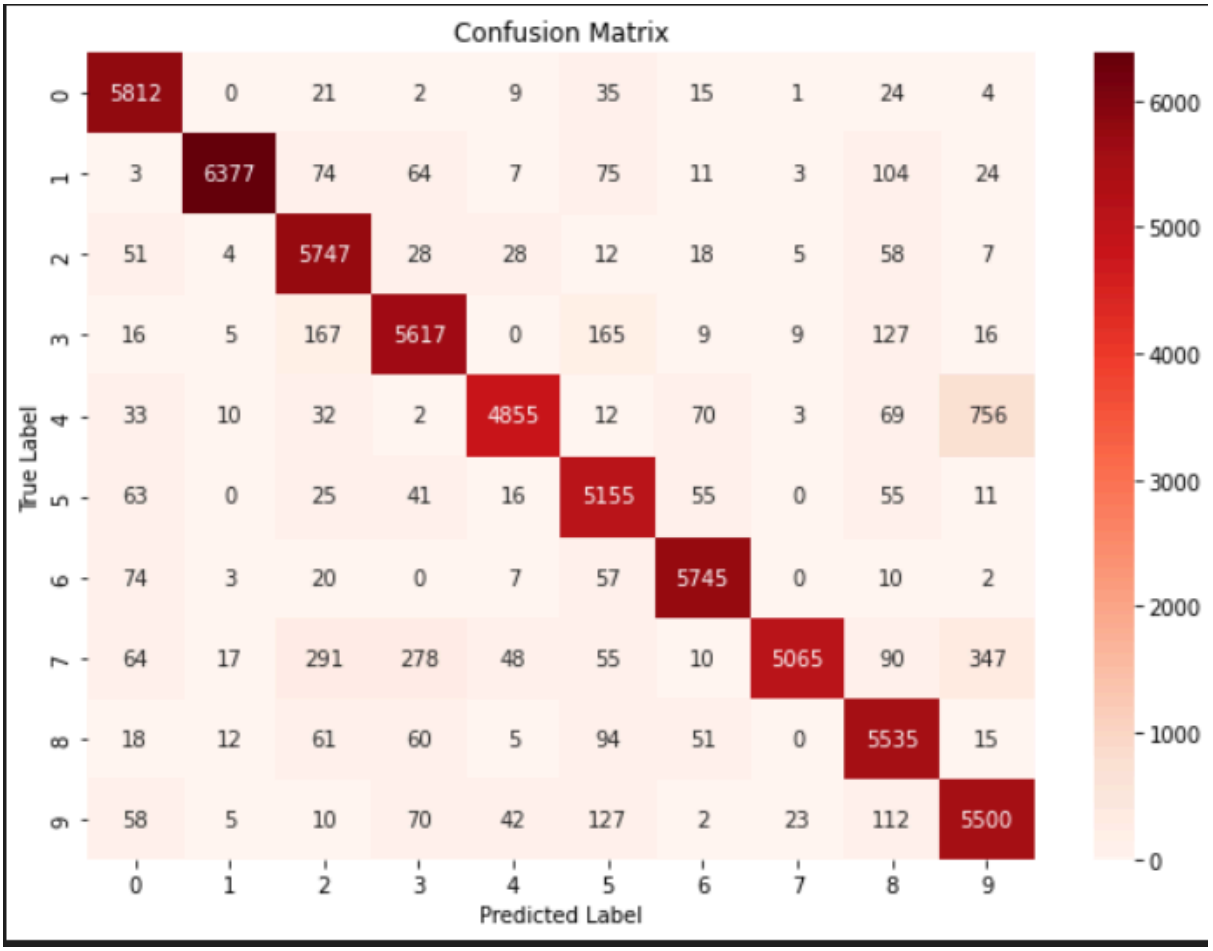
Plot for Iteration vs Loss for both training and testing:



Confusion matrix for better understanding of the mistakes made:
Test :


Confusion Matrix
True Label
Predicted Label
0 967 0 3 0 0 7 1 1 0 1
1 0 1083 5 9 0 7 5 0 25 1
2 12 0 988 2 4 2 6 0 16 2
3 2 1 29 931 0 26 0 5 15 1
4 3 0 5 2 822 3 17 0 16 114
5 9 0 3 9 3 843 8 0 12 5
6 17 1 4 0 3 13 916 0 4 0
7 9 6 60 52 4 6 2 818 20 51
8 6 0 10 6 3 19 11 1 917 1
9 6 2 2 13 7 29 0 1 16 933
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

Train



Accuracy achieved on the test and train dataset:

Test Accuracy : 92.18%

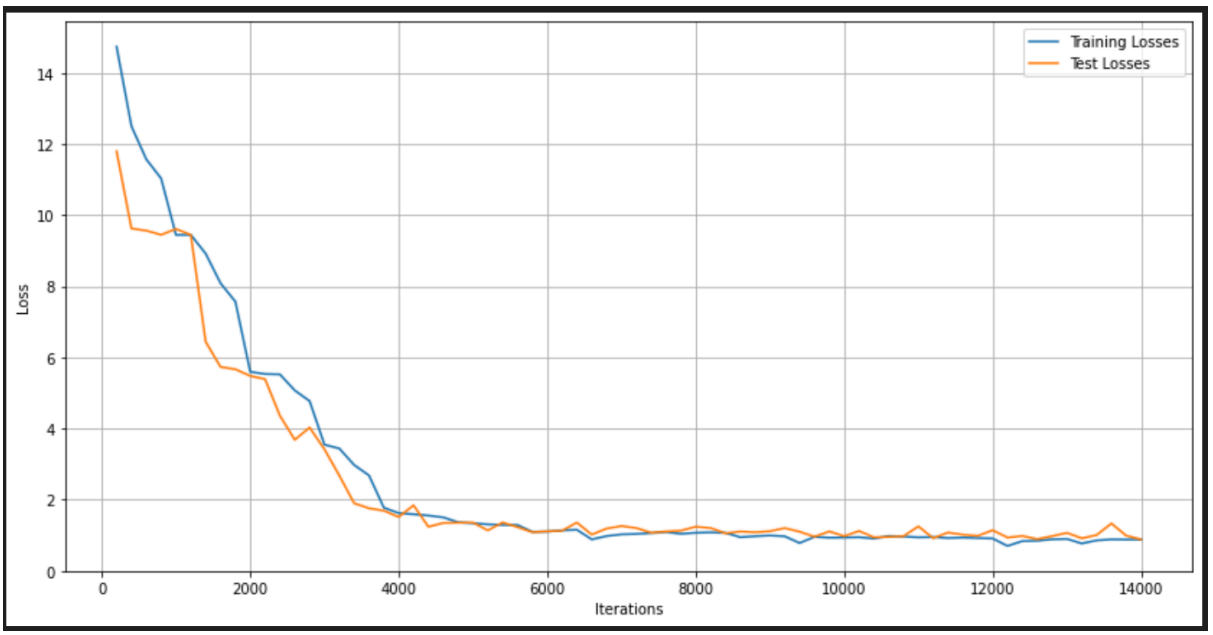
Train Accuracy : 92.35%

Comparison

In this case, when using a pytorch model, the Accuracy Achieved is lesser than the sigmoid activation function. So we can go with sigmoid activation that gives a better accuracy for 15 epochs. But it can be observed that for Tanh, The learning starts from fewer errors than when compared to the Sigmoid activation function which has higher epochs in the initial epochs. This might mean that in this case tanh learnt faster but it hit a local optimal solution within 15 epochs.

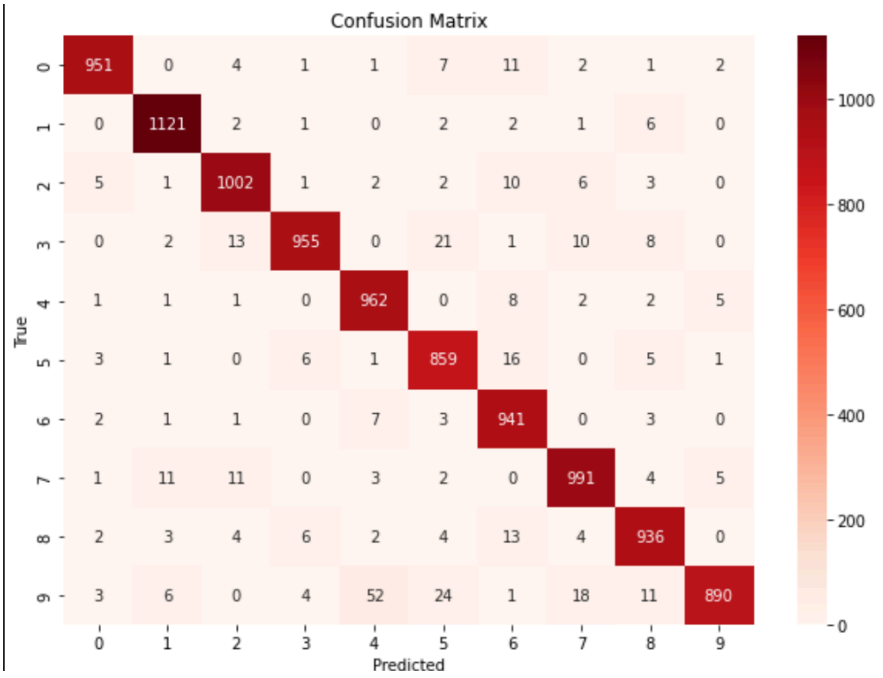
3) **Pytorch + sigmoid activation + l2 norm regularisation**
   After various iterations the weight for regularisation was set to 0.0001.

Plot for Iteration vs Loss for both training and testing:

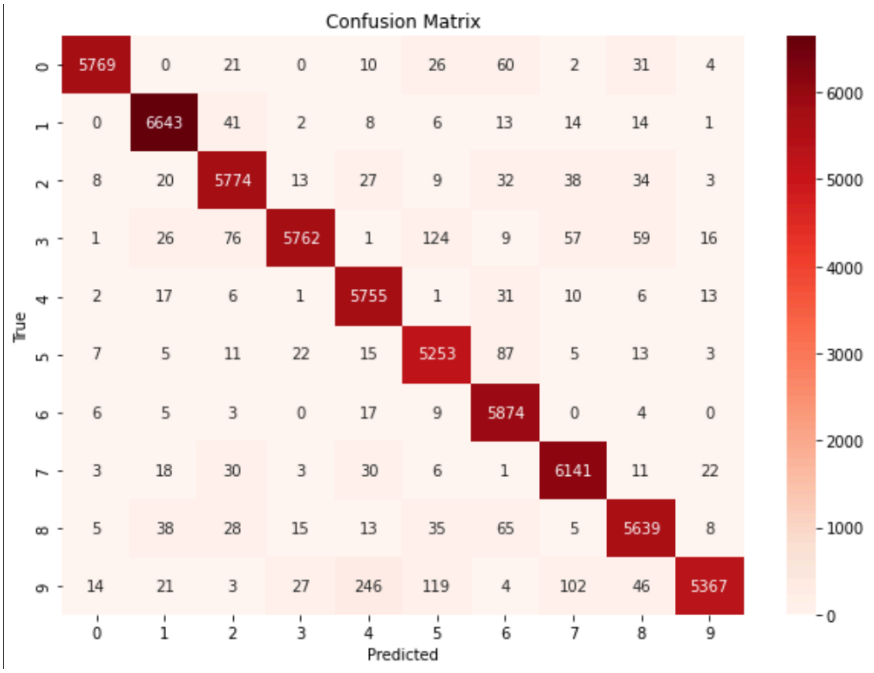


Confusion matrix for better understanding of the mistakes made:

Test :



Train:



Accuracy achieved on the test and train dataset:
Test Accuracy : 96.08%
Train Accuracy : 96.93%
Classes 9 are at times wrongly predicted as class 4,5, and 7. The model struggles in class 9. Similarly class 3 is wrongly predicted as class 5 in a few cases.

This shows that the model did not overfit so much and it is a better model than others so far.

In general it could be observed that the PYtorch models start from higher losses initially in the initial epochs but they learn faster - accelerated learning is observed in the pytorch than my scratch implementation.

Final Conclusions:

We find that the baseline implementation performed least when compared to the other models. The Tanh function on the scratch code performed the best. The Relu action also gave better accuracies but it was found that it is prone to overfitting with increasing the number of epochs. The replication of baseline using Pytorch gave results, probably because of the ADAMS optimiser. Tanh activation

using pytorch implementation did not give a very high performance accuracy. Using a l2 regularisation could be considered as a constrained optimisation problem with a limit on the magnitude of weights - this may not always give the best results but if the hyperparameter - the weight for the regularisation term is chosen wisely, it might give accelerated and faster learning at times- as this constrains the model to search within the region where the optimal weights are more likely to be.

References

https://medium.com/