INDIAN INSTITUTE OF TECHNOLOGY DELHI

ELL 784
INTRODUCTION TO MACHINE LEARNING
Assignment 3

Name: Tamal Majumder

Entry No: 2023PHS7226

Submission date: 20-04-2024

THE MLP MODEL

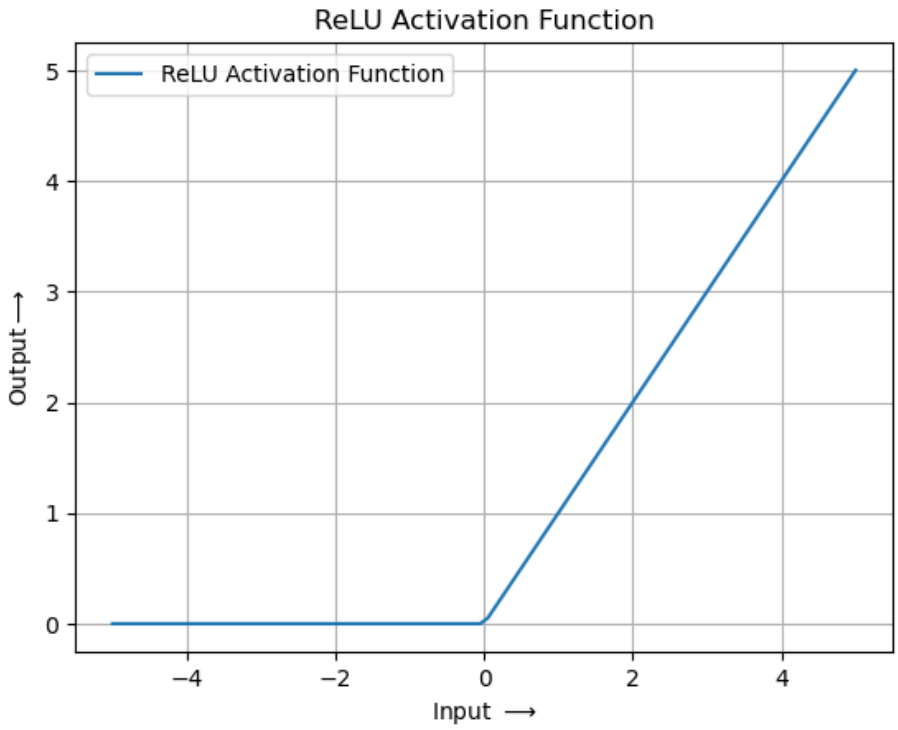
STEPS TAKEN :

- First we imported the MNIST dataset.
- There were 60000 images for training and 10000 for testing
- Then we one hot encoded the data which turned the data into :

  [0 0 0 0 1 0 0 0 0 0] -> represents 4

- After that I defined the Multi Layer Perceptron function .
  - In $f(x)=w_i * x_i + b_i$ in this format
  - And then applied a ReLU activation function
  - After that passed the values onto the next layer
- In this step I added a SoftMax function to the outputs
- Then after setting up the learning rate and using Adam Optimizer for the back propagation we proceed to next step where I set the Epoch / No of iterations
- Finally We run the code and get the results of loss and Accuracy score.
- We then proceed to plot the loss and accuracy also the classified images, misclassified images , confusion matrix for correlation etc. for more in depth Insights .
- Test 16 Gives the best result **

ReLU activation function:



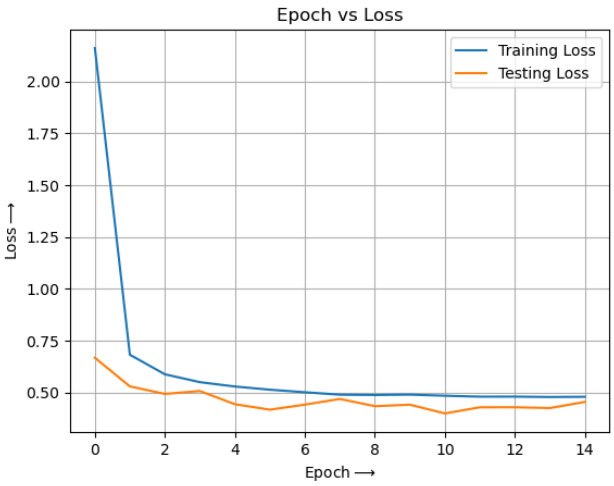
TEST RESULTS AND OPTIMIZATION

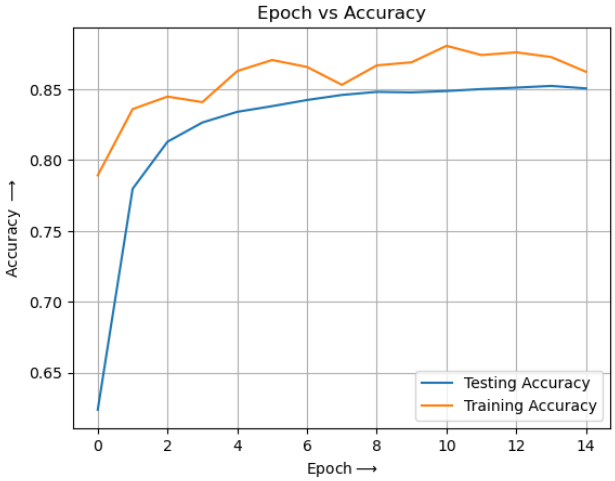
Different no of Layers

Test 01 :

- Learning rate: 0.01
- No of hidden layers : 2
- NN Model: 784 -> 256 -> 128 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.4666783809661865
Test Accuracy: 0.8551999926567078

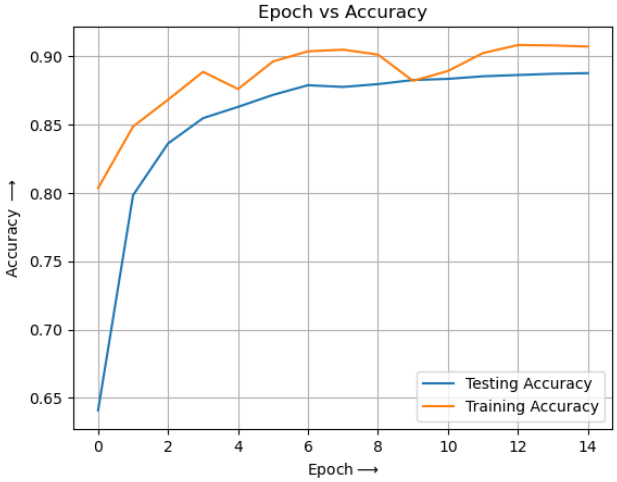


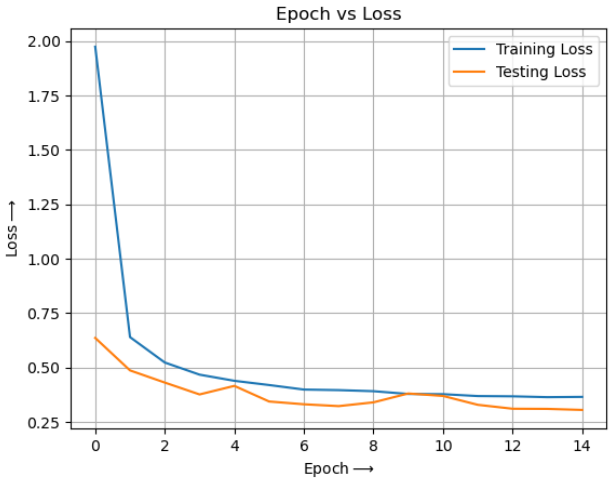


Test 02 :

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model: 784 -> 512->256 -> 128 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.3187595009803772
Test Accuracy: 0.9060999751091003



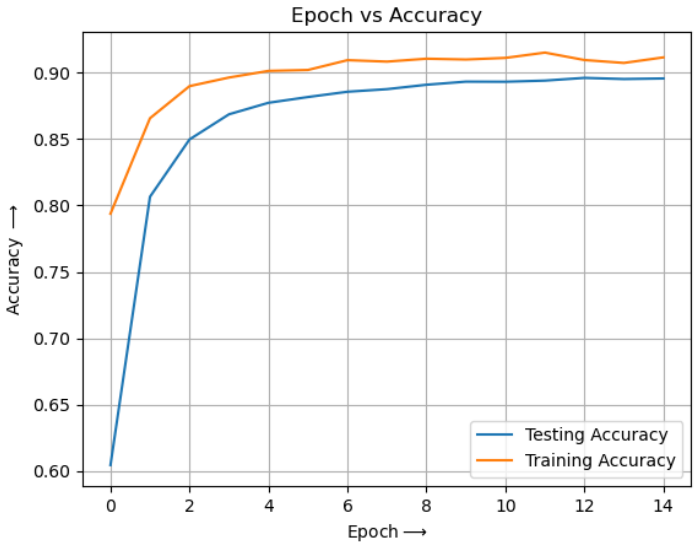


Test 03 :

- Learning rate: 0.01
- No of hidden layers: 4
- NN Model: 784 -> 512->256 -> 128->64 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.3178767263889313

Test Accuracy: 0.901199996471405



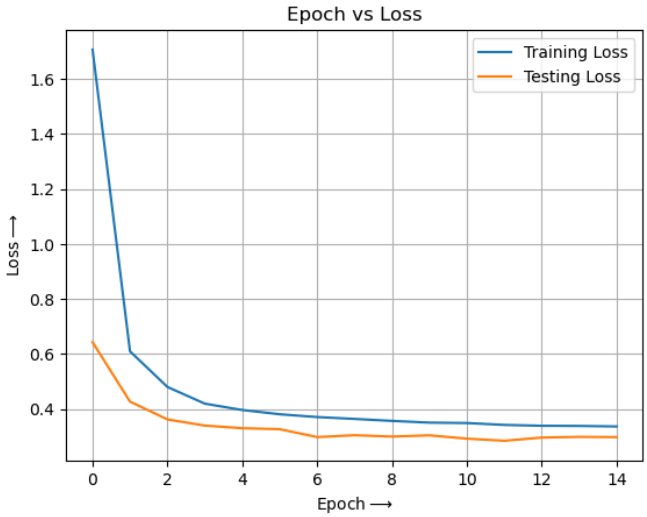


Table for Choice of No of layers

No of layers	Test Loss	Test Accuracy
2	0.467	0.856
3	0.319	0.906
4	0.318	0.901

Discussion:

- Here we can see that the accuracy increased significantly when we went from 2 layers to 3 layers
- The second thing we can see is the accuracy when we go from 3 layers to 4 layers is almost same. So, we can choose 3 Layer perceptron as the optimum as it is less computationally expensive than 4 layer perceptron
- With minimum loss

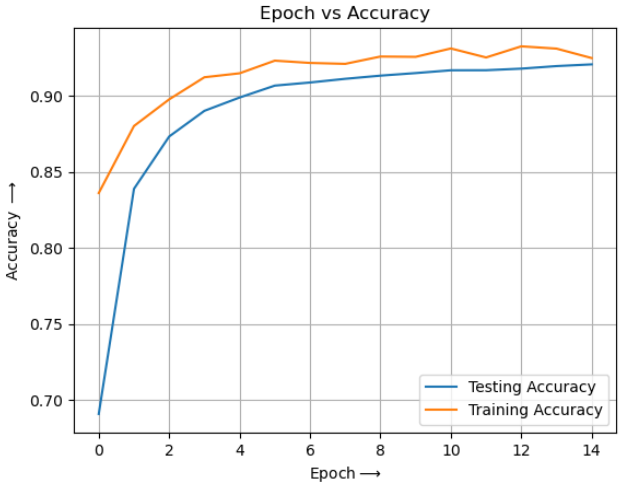
So from this step we choose 3 Layer Perceptron as Optimum.

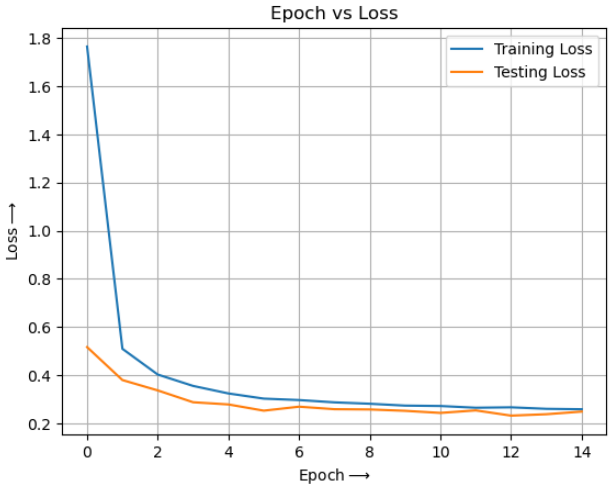
Different no of neurons

Test 04 :

- Learning rate: 0.01
- No of hidden layers: 3
- NN Model: 784->1024 -> 512 -> 128 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.24006494879722595
Test Accuracy: 0.9261000156402588

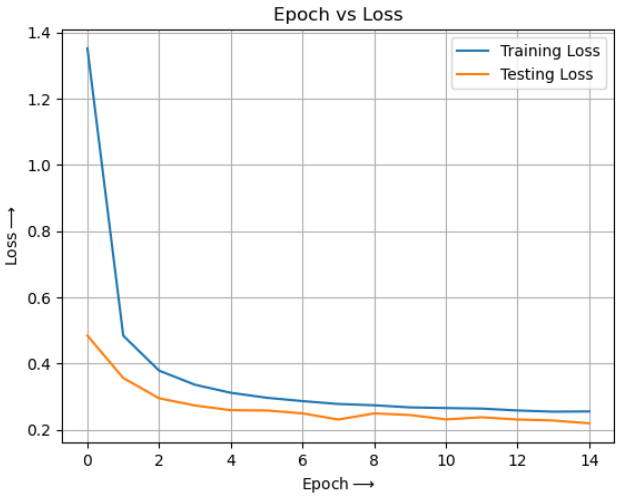


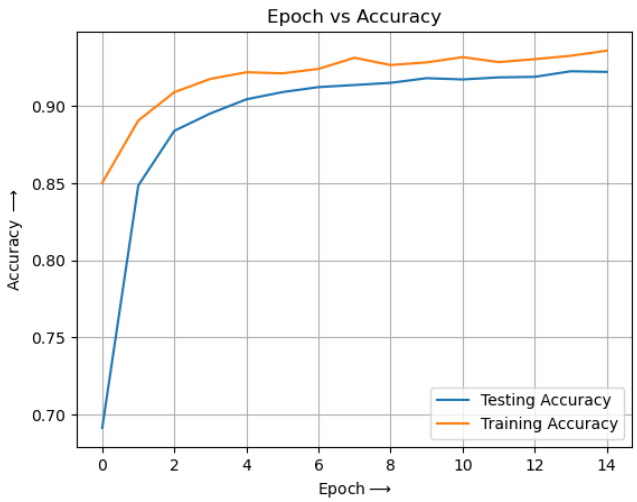


Test 05 :

- Learning rate: 0.01
- No of hidden layers: 3
- NN Model: 784->1024 -> 512 -> 64 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.2207665592432022
Test Accuracy: 0.9340000152587891

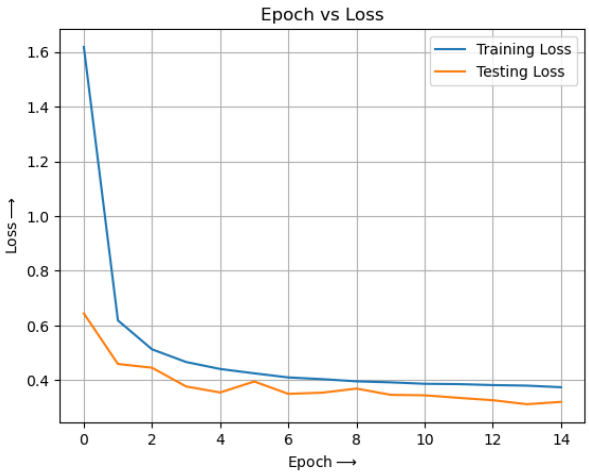


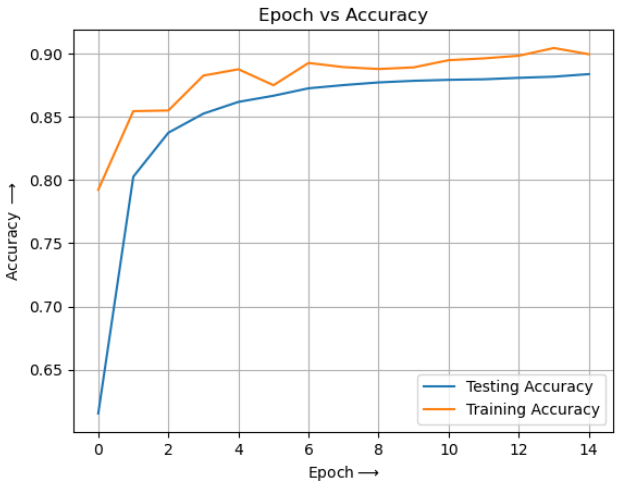


Test 06 :

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model:784-> 512 ->264-> 64 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.32002076506614685
Test Accuracy: 0.8981999754905701

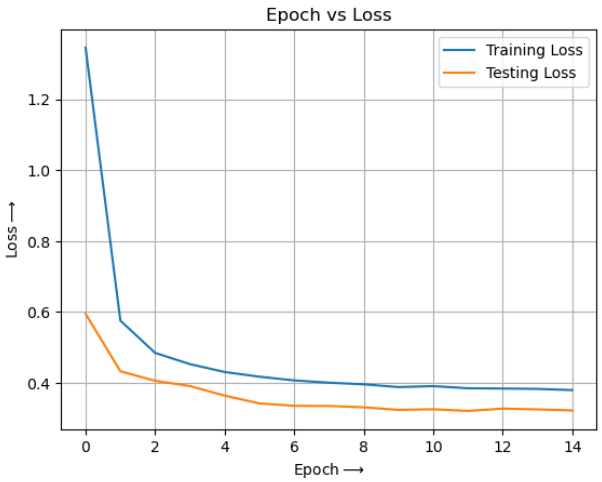


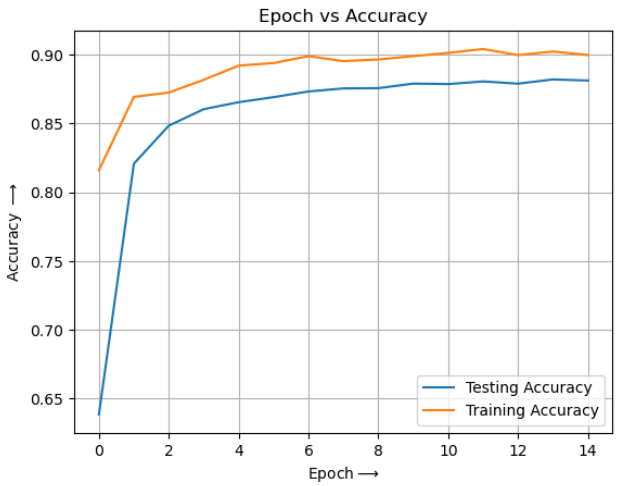


Test 07 :

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model: 512 ->264-> 32 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.3509306311607361
Test Accuracy: 0.8924999833106995

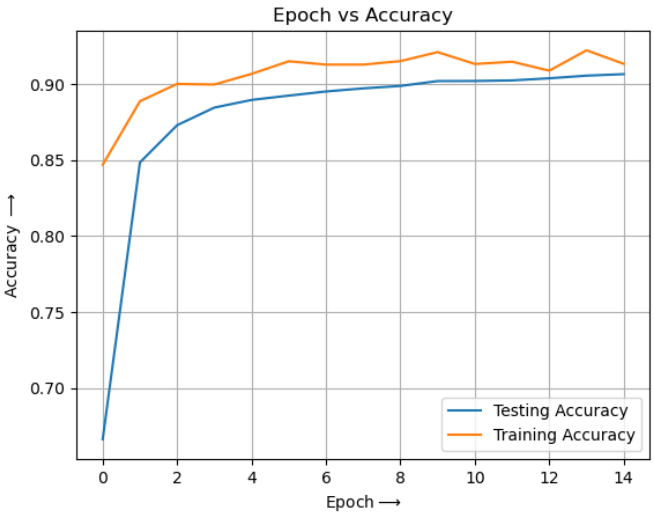


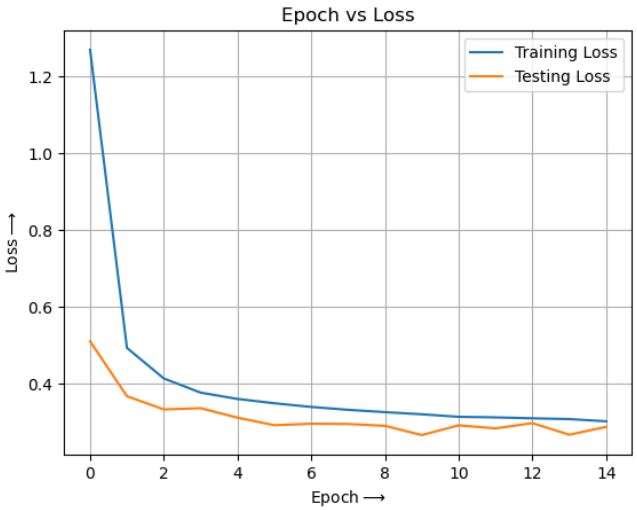


Test 08 :

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model: 784->1024 ->264-> 32 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.3055843412876129
Test Accuracy: 0.9025999903678894

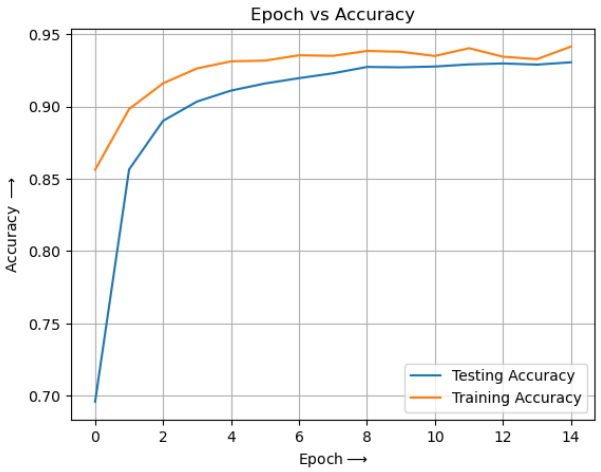


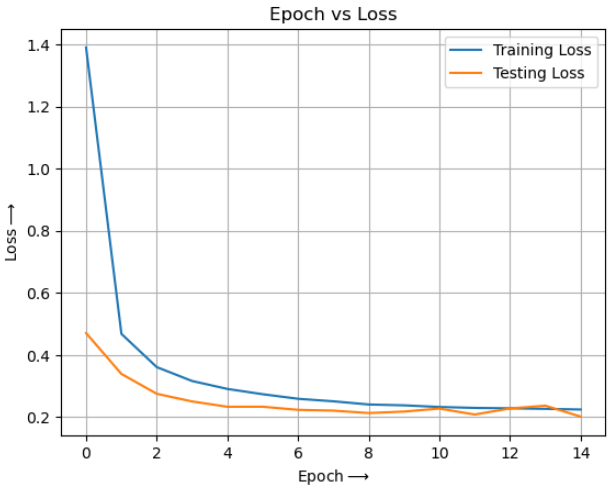


Test 09 :

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model: 2048 ->264-> 64 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.19705885648727417
Test Accuracy: 0.9416000247001648

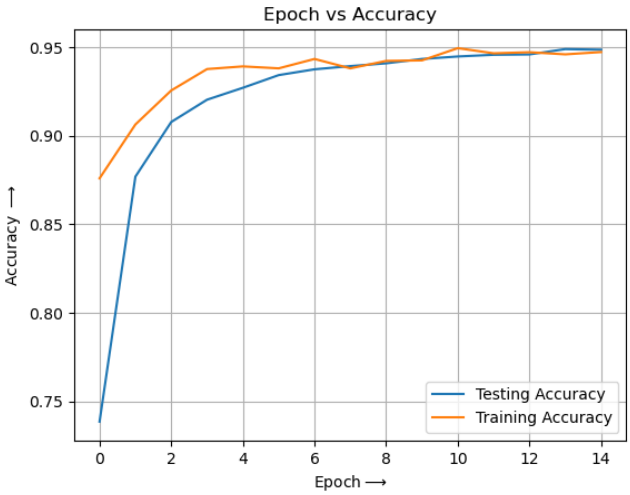


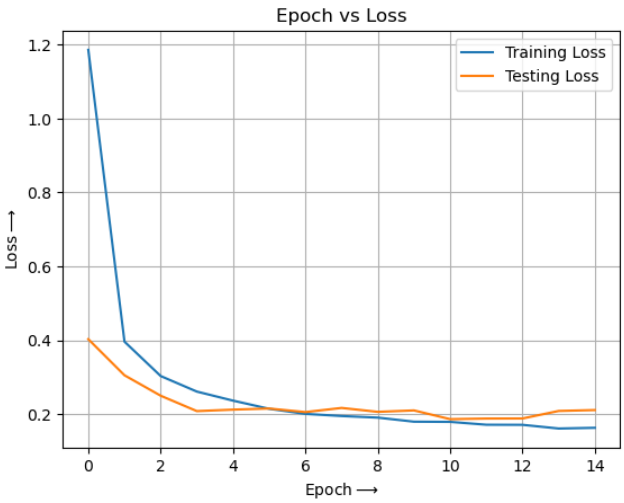


Test 10 :

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model: 4096 ->264-> 64 ->10
- Epochs/ No of iterations: 15

Test Loss: 0.18259774148464203
Test Accuracy: 0.9466999769210815



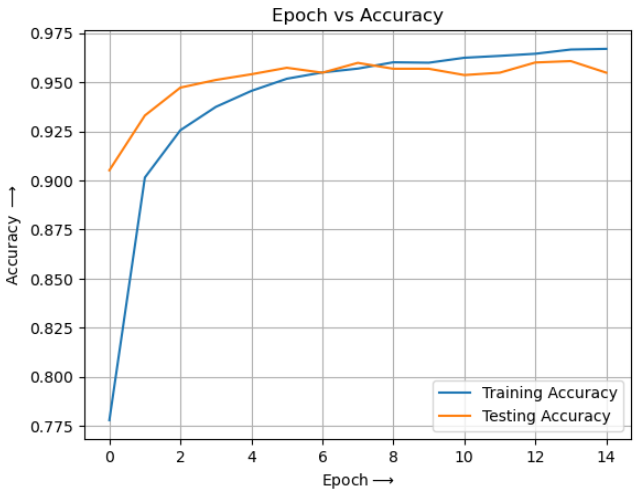


Test 11 : (best upto test 11)

- Learning rate: 0.01
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10

Epochs/ No of iterations: 15

Test Loss: 0.16477909684181213
Test Accuracy: 0.9557999968528748



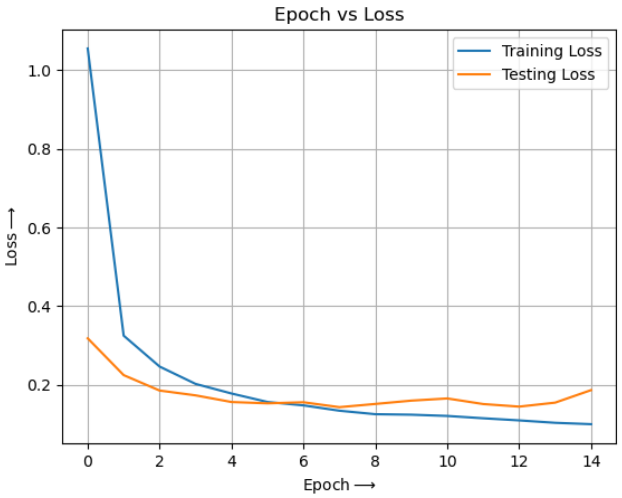


Table for Choice of Different No of neurons in each layer

NO of Neurons	Test Loss	Test Accuracy
1024 -> 512 -> 128	0.240	0.926
1024 -> 512 -> 64	0.221	0.934
512 ->264-> 64	0.320	0.898
512 ->264-> 32	0.351	0.892
1024 ->264-> 32	0.306	0.903
2048 ->264-> 64	0.197	0.942
4096 ->264-> 64	0.183	0.947
8192 ->512-> 64	0.165	0.956

Discussion:

- Here we see that as we increase in general the Number of Neurons in each layer the accuracy score increases.
- 64 is the optimal number of Neuron in the last layer of the perceptron.
- 264 or 512 is the optimal in the middle layer . Here 512 is optimal.
- And for the first layer we see that as the number of Neurons increase , the accuracy score also increases. The best accuracy we are getting is at 8192 Neurons in the 1st layer .
- The best accuracy score: 95.6%
- The loss is also minimum at 1.65%

So, **8192 ->512-> 64**

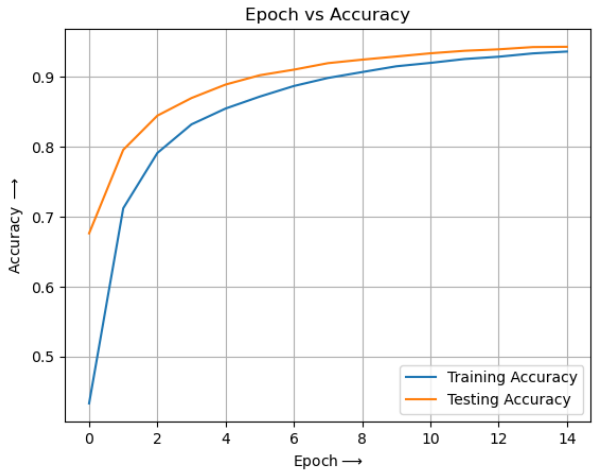
This is the optimal no of Neuron choice for each layer which gives best accuracy.

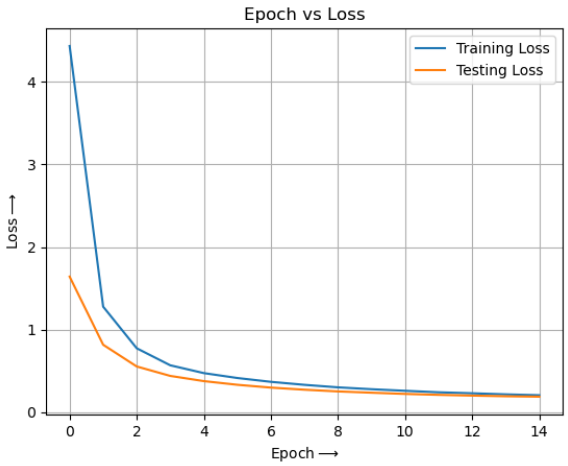
Cahnging the learning rate

Test 12 :

- Learning rate: 0.001
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10

Epochs/ No of iterations: 15
Test Loss: 0.1997593641281128
Test Accuracy: 0.9401000142097473

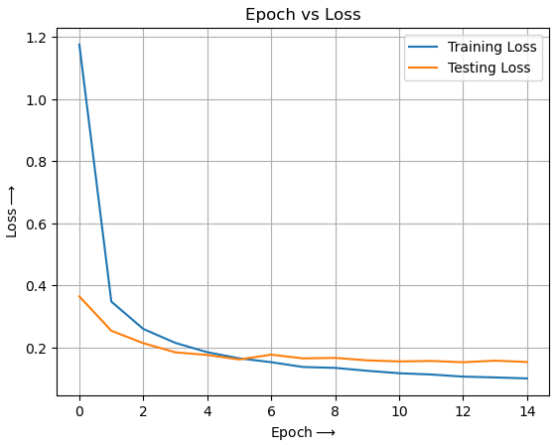


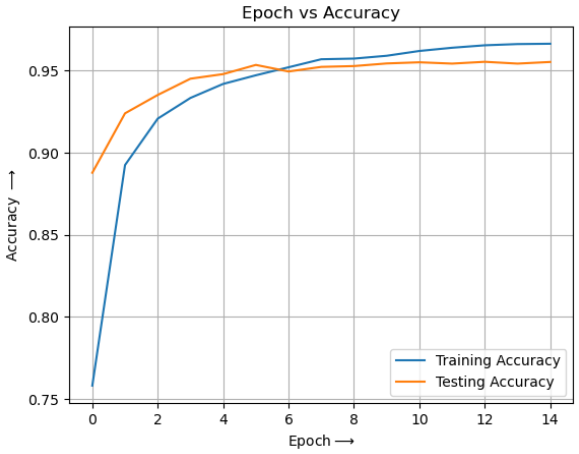


Test 13 : (best upto test 13)

- Learning rate: 0.007
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10

Epochs/ No of iterations: 15
Test Loss: 0.1436964327096939
Test Accuracy: 0.9590999870300293

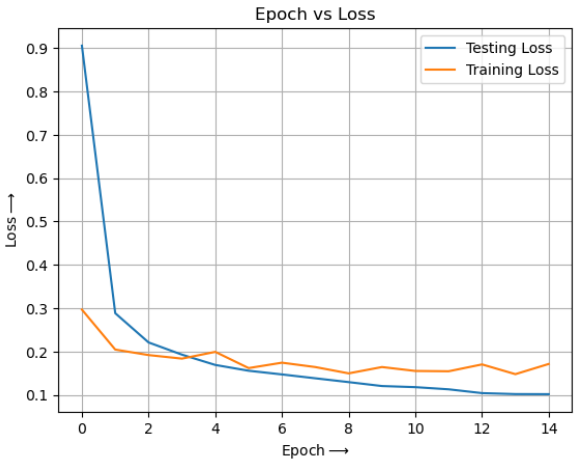




Test 14 :

- Learning rate: 0.012
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10

Epochs/ No of iterations: 15
Test Loss: 0.1814928650856018
Test Accuracy: 0.9531999826431274



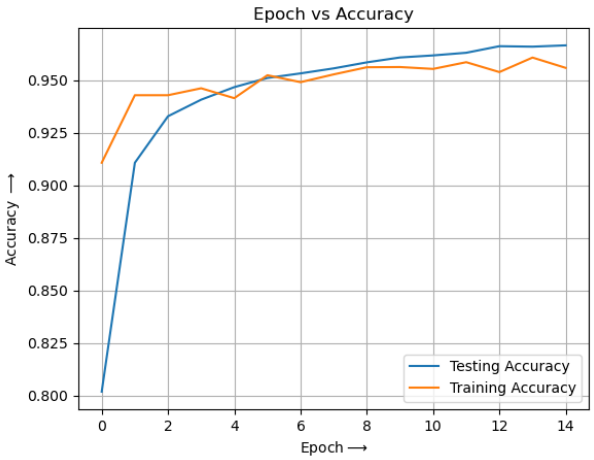


Table for Changing Learning Rate

Learning Rate	Test Loss	Test Accuracy
0.1	0.240	0.926
0.001	0.200	0.940
0.007	0.141	0.959
0.012	0.181	0.953

Discussion:

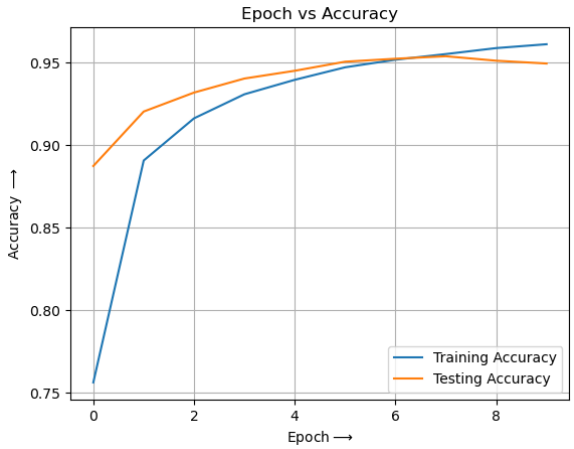
- Here we see that as we decrease the learning rate drom 0.1 to 0.001 the accuracy increases and after a value it stabilizes and the change in accuracy is not that much so we can choose a value .
- In this case the Optimum value is found to be 0.007
- Also the loss is minimized to be 1.41%

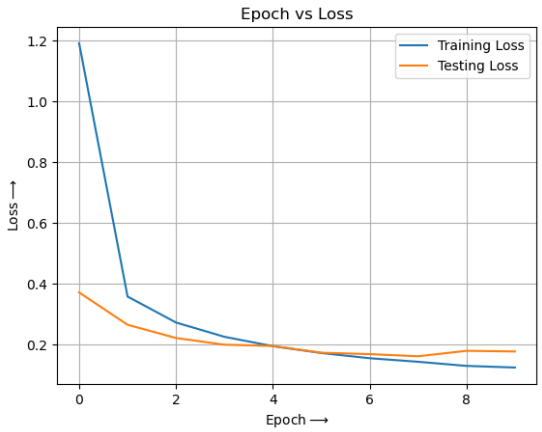
CHANGING EPOCH

Test 15 :

- Learning rate: 0.007
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10
- Epochs/ No of iterations: 10

Test Loss: 0.16511093080043793
Test Accuracy: 0.947700023651123

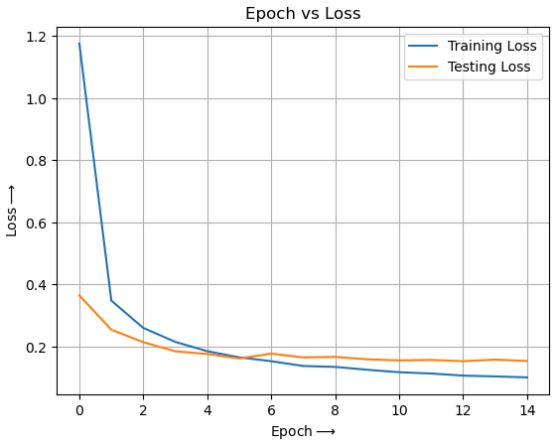


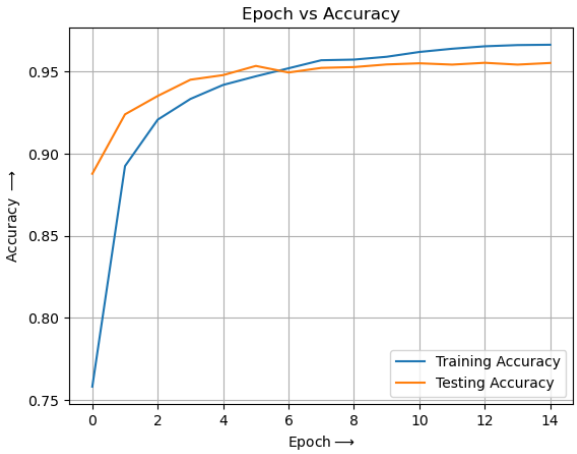


Test 16 : (best of all)**

- Learning rate: 0.007
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10

Epochs/ No of iterations: 15
Test Loss: 0.1436964327096939
Test Accuracy: 0.9590999870300293

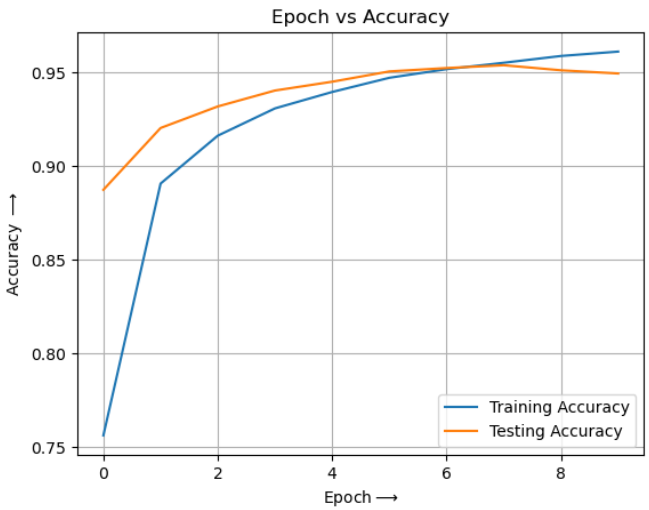




Test 17 :

- Learning rate: 0.007
- No of hidden layers : 3
- NN Model: 8192 ->512-> 64 ->10
- Epochs/ No of iterations: 20

Test Loss: 0.14140480756759644
Test Accuracy: 0.958899974822998



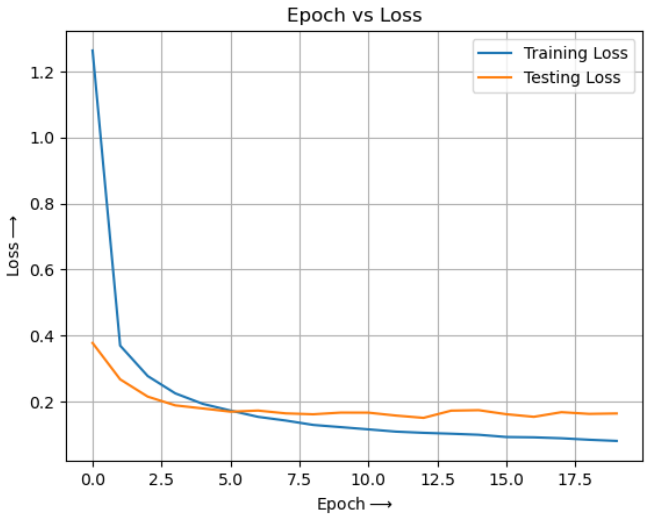


Table for Changing Learning Rate

Epoch	Test Loss	Test Accuracy
10	0.165	0.948
15	0.143	0.959
20	0.141	0.959

Discussion:

- Here we see that as we increase the no of epoch the accuracy increases and converges around 12-15 and stays relatively constant from there.
- So the optimum choice for the Epoch / No of iteration is 15
- It also has relatively minimum loss .

SUMMARY

Table for Choice of No of layers

No of layers	Test Loss	Test Accuracy
2	0.467	0.856
3	0.319	0.906
4	0.318	0.901

Table for Choice of Different No of neurons in each layer

NO of Neurons	Test Loss	Test Accuracy
1024 -> 512 -> 128	0.240	0.926
1024 -> 512 -> 64	0.221	0.934
512 ->264-> 64	0.320	0.898
512 ->264-> 32	0.351	0.892
1024 ->264-> 32	0.306	0.903
2048 ->264-> 64	0.197	0.942
4096 ->264-> 64	0.183	0.947
8192 ->512-> 64	0.165	0.956

Table for Changing Learning Rate

Learning Rate	Test Loss	Test Accuracy
0.1	0.240	0.926
0.001	0.200	0.940
0.007	0.141	0.959
0.012	0.181	0.953

Table for Changing Epoch

Epoch	Test Loss	Test Accuracy
10	0.165	0.948
15	0.143	0.959
20	0.141	0.959

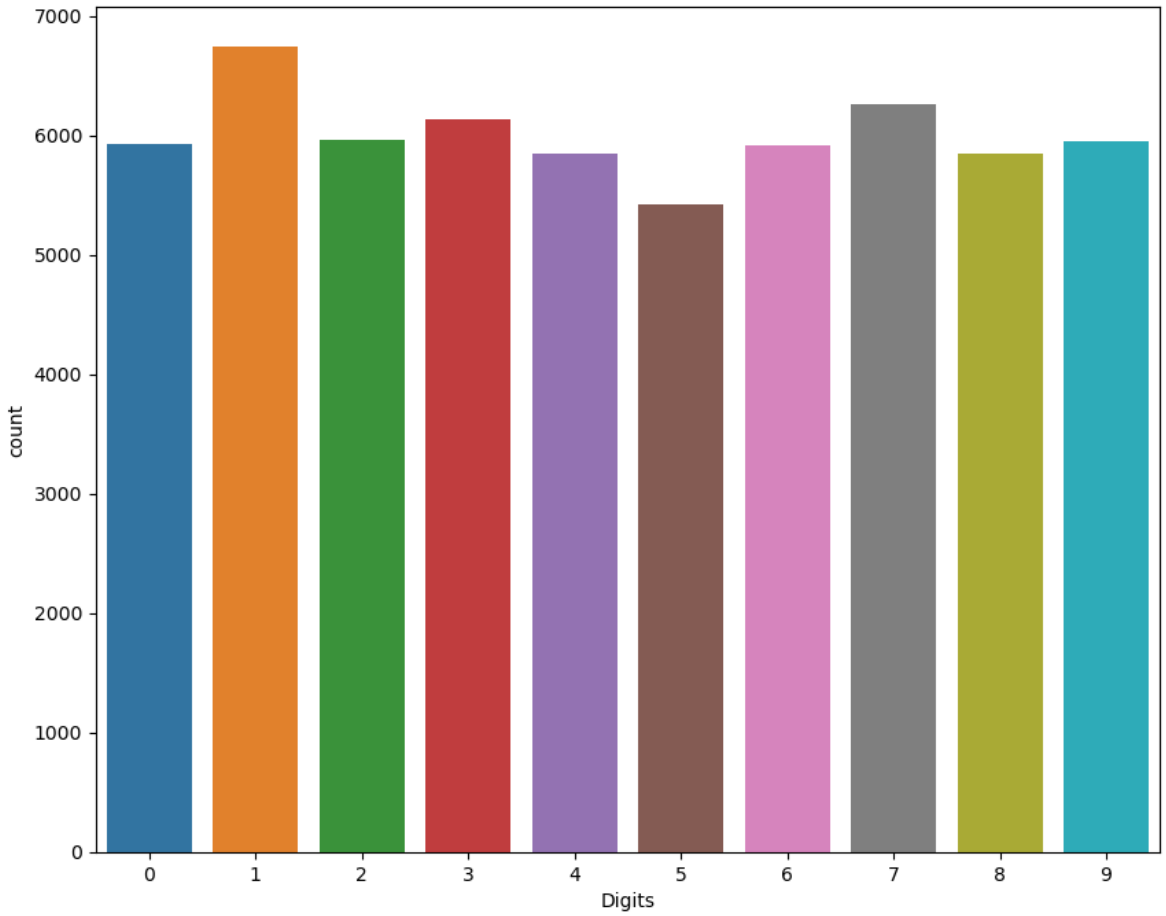
Red Marked Ones are the optimum Choices

DISCUSSION ON THE BEST MODEL

- The best model parameters are:

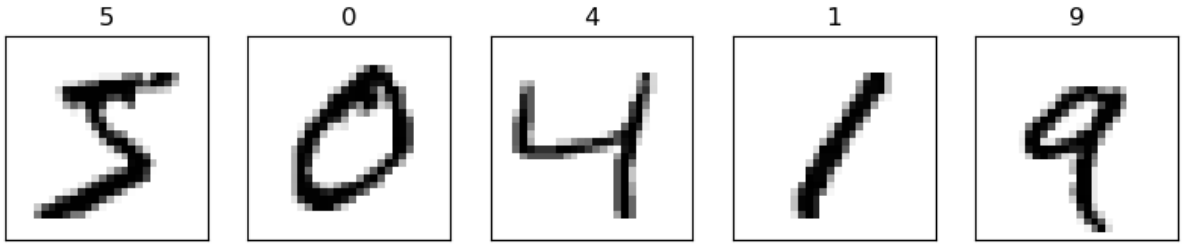
No of layers	NO of Neurons	Learning Rate	Epoch	Test Loss	Test Accuracy
3	8192->512-> 64	0.007	15	0.143	0.959

Number count in the dataset:

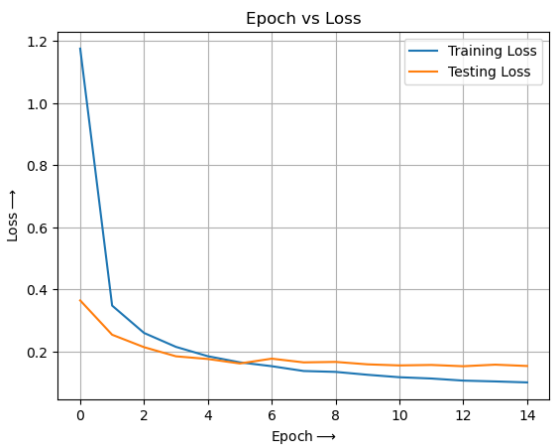


- The dataset has sufficient data to train the model.

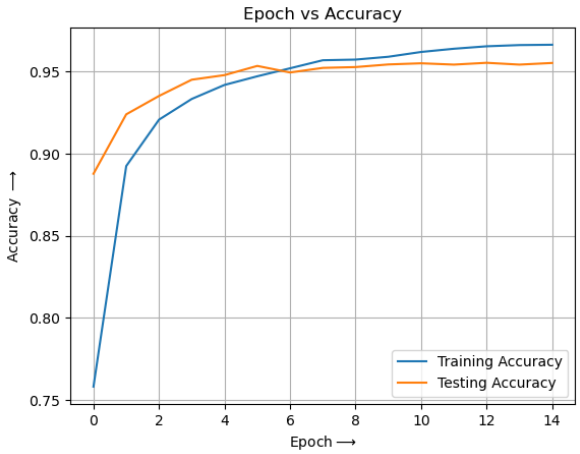
Randomly printed numbers to visualize the dataset:



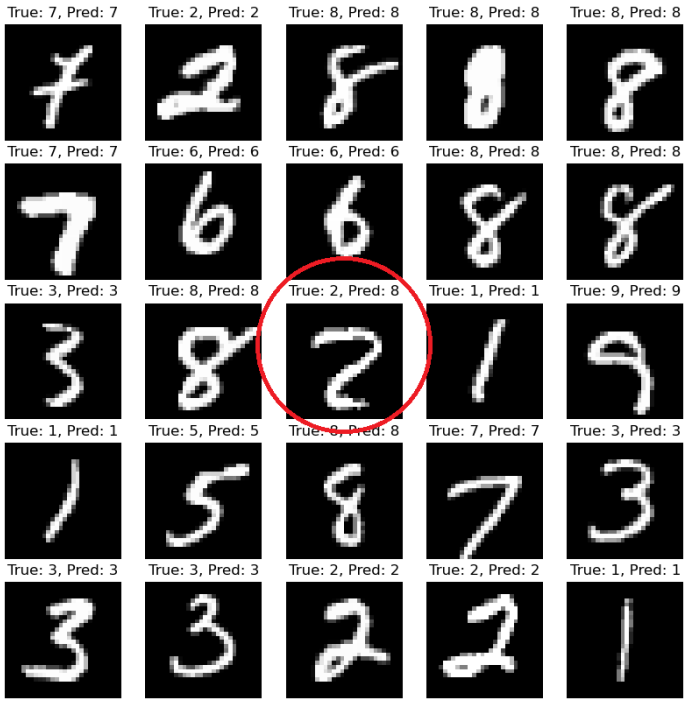
: : Loss : :



: : Accuracy : :

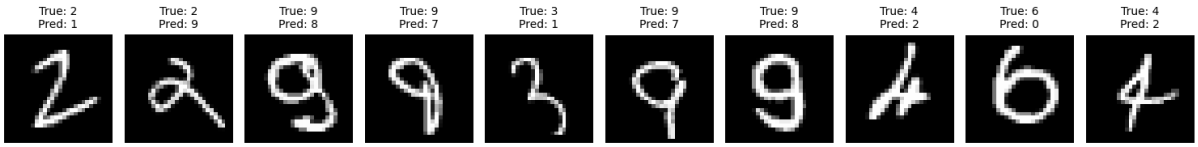


Sample of Test Results

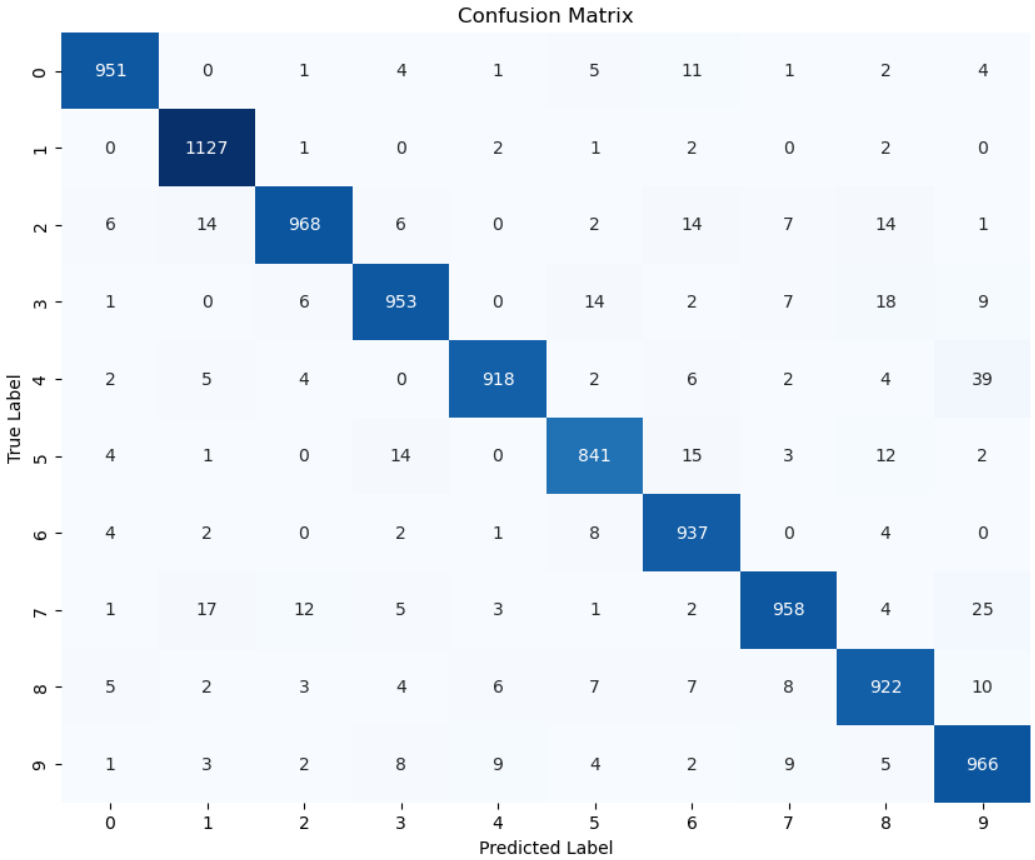


In this image we can clearly see the misclassified number among mostly correct classified numbers.

Misclassified Examples:



The confusion matrix:



From the confusion matrix of the Mnist dataset, we can draw several insights:

1. Here the diagonal values represent the number of correctly classified instances for each class. Higher values on the diagonal indicate better classification performance for those classes.

2. The offdiagonal values represent misclassifications By analyzing the misclassifications, we can identify patterns and areas where the classifier is struggling. This can help us improve the model by focusing on those specific classes or features.

CONCLUSION

- Maintaining the pyramid structure gives the best result
- 2 or 3 hidden layers is sufficient to give good results the deeper we go the better the result gets
- The no of neurons in layers if increased gives better accuracy
- There is also an issue of overfitting which can be resolved using batch normalization up to an extent.
- 15 epoch is sufficient although 10 is also a good no of epoch.
- Analyzing the misclassified results, we can see the numbers which are very closely related sometimes gave rise to error, like in the case of '9' if the tail extends upwards, it kind of seems like an '8', for '6' if the head is small, it resembles a '0'. One way we can improve on this is by looking at looking at local and global features separately and then together taking conclusion. So, It will be a twostep matching process which will give more accurate result.
- The loss function here signifies the convergence of the functions to its local/global minima. Our main goal is to minimize the loss function. As the loss function decreases, we get more and more accurate results. In our series of tests we were able to see the decrease of the loss functions which is also visible in the plots. And experimenting with the parameters we were able to significantly decrease the loss also

Finally, we can say that doing a series of test and trails we were able to reach the approximate optimum of the model. By changing the depth of the layers, changing number of neurons in each layer, fine tuning the Learning rate, adjusting the No of epochs we were able to hugely improve the accuracy of the model from a 80% accuracy score to 95.9% accuracy score, A huge improvement of 15% was achieved by fine tuning the parameters of the model.

Part 2

IMPROVING THE ACCURACY SCORE

My first approach was , I took the Fourier transform and applied a low pass filter so the we get the refined edges only ,and then applied the MLP on it to improve the accuracy score , but it did not work and gave result with less accuracy around 90-91% with same model parameters.

Then on further investigation on how to improve the accuracy looking at the local and global feature I came across LSTM (Long short term memory) model .

Using LSTM Model:

Here's the enhanced algorithm with the incorporation of local and global details:

1. Import the MNIST dataset.

2. The dataset comprises 60,000 images for training and 10,000 for testing.

3. Perform one-hot encoding on the labels to represent each digit as a vector, for example:
   - [0 0 0 0 1 0 0 0 0 0] represents the digit 4.

4. Define the LSTM (Long Short-Term Memory) model architecture.
   - Incorporate local details by adding convolutional layers at the beginning of the network to capture low-level features.
   - Follow the convolutional layers with LSTM layers to capture temporal dependencies in the sequence of image data.
   - Use the Dense layer for the output.
   - Apply ReLU activation function after each convolutional and LSTM layer to introduce non-linearity.
   - Add a Softmax activation function to the output layer to get probabilities for each class.

5. Set up the learning rate and choose the Adam Optimizer for backpropagation to efficiently update model parameters.

6. Specify the number of epochs/iterations for training to allow the model to learn from the data over multiple passes.
7. Train the model using the training data.
8. Evaluate the model on the test data to calculate loss and accuracy scores.

9. Plot the loss and accuracy curves to visualize model performance over epochs and identify trends.

10. Incorporating global details by applying batch normalization during training to improve generalization of the model.

11. Testing different hyperparameters such as - changing the number of LSTM units, adjusting learning rate, Changing Number of epochs

Based on experimentation, we see that the configuration with the best performance is Test 4 with 99.1% accuracy and minimum loss.

This algorithm not only captures the temporal dependencies in the sequence of image data through LSTM layers but also incorporates local details through convolutional layers and global details through techniques like batch normalization.

Mathematical Model:

1. Input Initialization:

- - Initialize input sequences x1, x2, ..., xT and initial cell state c0 and hidden state h0.

2. Forward Pass:

- For each time step t from 1 to T:
  - Compute the forget gate ft to determine which information to forget.
  - Compute the input gate it to determine which new information to store.
  - Compute the candidate cell state c̃t to update the cell state.
  - Update the cell state ct by combining the forget gate and input gate.
  - Compute the output gate ot to determine which information to output.
  - Update the hidden state ht based on the output gate and the updated cell state.
  - Output the prediction yt based on the hidden state ht.

3. Backward Pass :

- - Compute gradients of loss with respect to model parameters using backpropagation through time
- - Update model parameters using gradient descent optimization algorithms we used Adam optimizer.

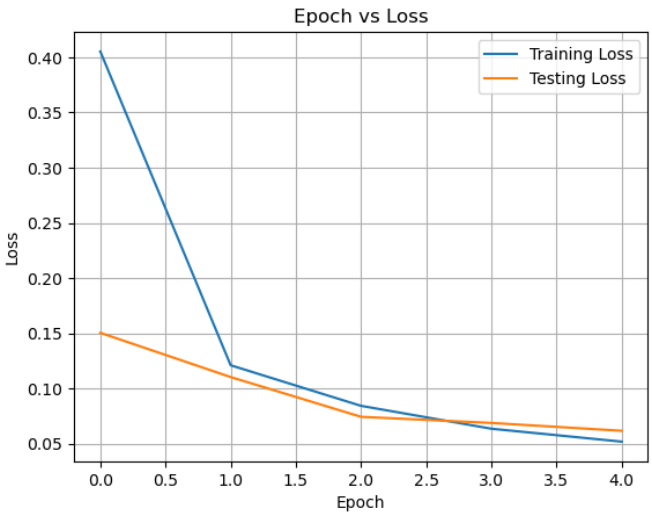
TEST RESULTS

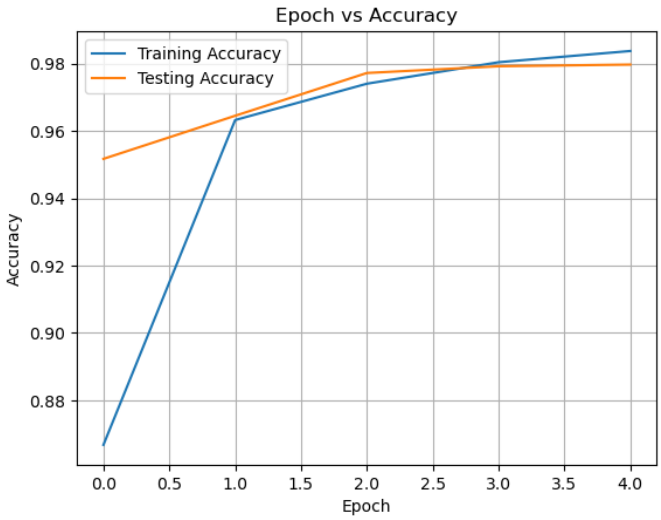
Test 01:

Epochs: 5
Layers 128
Learning rate: 0.001

```
Test Loss: 0.061741407960653305
Test Accuracy: 0.9797000288963318
```

- A 1.5% increase in the accuracy using the LSTM Model



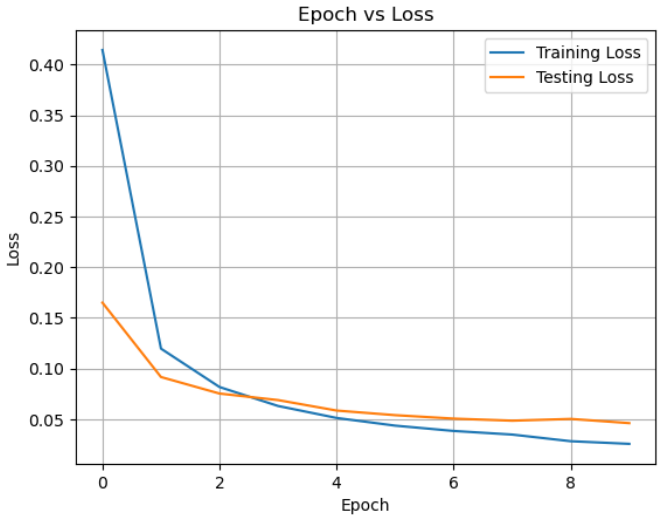


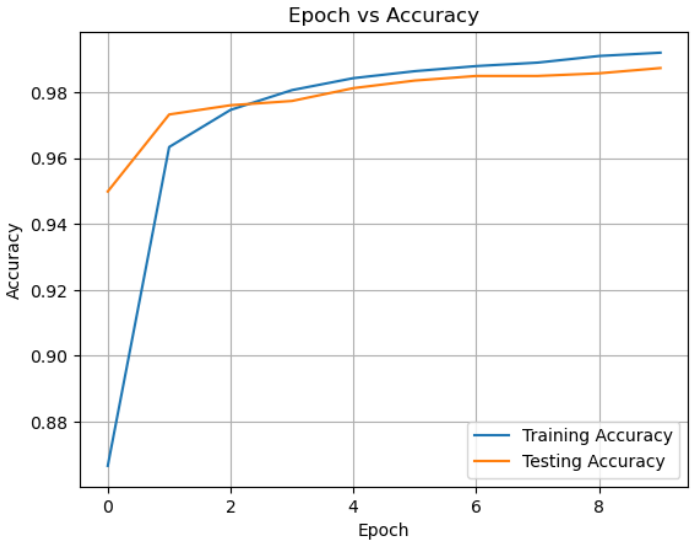
Test 02:

Epochs: 10
Layers 128
Learning rate: 0.001

```
Test Loss: 0.0460471548140049
Test Accuracy: 0.9873999953269958
```

- A 2.8 % increase in the accuracy using the LSTM Model





Test 03:

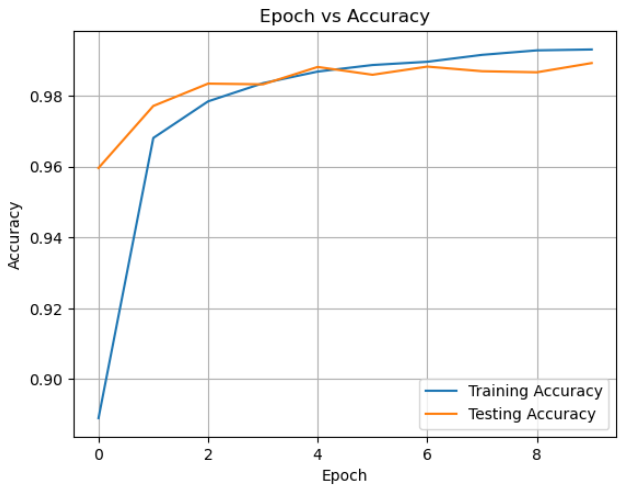
Epochs: 10

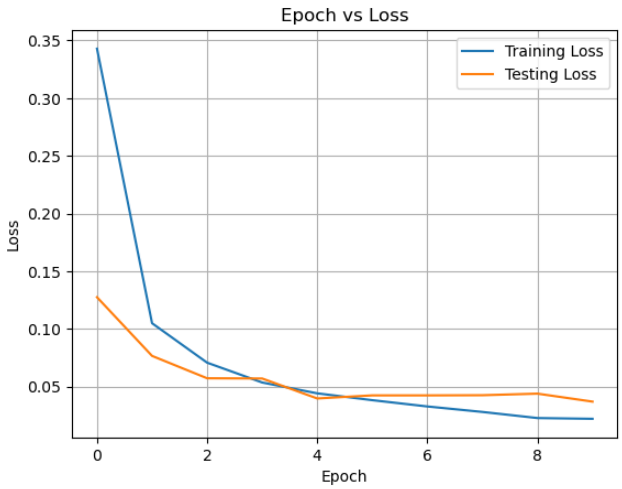
Layers 256

Learning rate: 0.001

```
Test Loss: 0.03692658245563507
Test Accuracy: 0.9891999959945679
```

- 3.0 % increase in the accuracy using the LSTM Model (not that different from 128 layers)





Test 04:

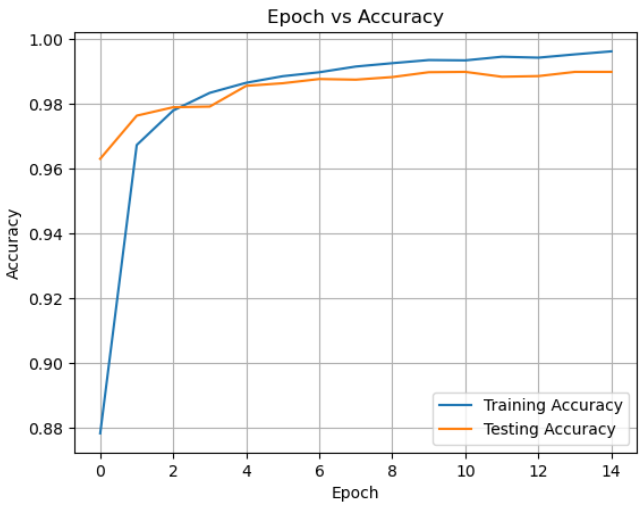
Epochs: 15

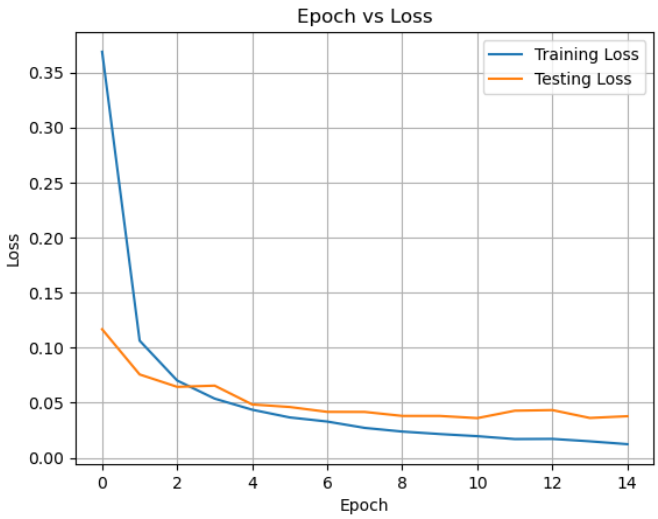
Layers 256

Learning rate: 0.001

```
Test Loss: 0.037714771926403046
Test Accuracy: 0.9909899992942811
```

- 3.2% increase in the accuracy using the LSTM Model





CONCLUSION

So, using the MLP based model we were able to get from 80% accuracy to 95.9% accuracy by changing the parameters.

To improve on the accuracy score we took a LSTM based approach and were able to acquire a 99.1% accuracy score . which is a big improvement of about 3.2%

- Accuracy Scores

Unmodified MLP	Properly tuned MLP	Properly tuned LSTM Model
80%	95.6%	99.1%

In the case of different recognition scenario like postal address recognition , cheque authentication, handwritten equation recognition having a higher percentage of accuracy and low loss is very much expected . So, through this project we were able to experiment with the models hands on and were able to improve on them.