Project: 3
Classification Using Neural Networks and Deep Learning

Name- Kunal Kumar
ASU ID- 1217167445

Introduction:

In this project we are using MNIST dataset to perform classification using Convolution Neural Networks. The basic requirements of the project are below:

i. The input size is the size of the image (28x28).
ii. The first hidden layer is a convolution layer, with 6 feature maps. The convolution kernels are of 3x3 in size. Use stride 1 for convolution.
iii. The convolution layer is followed by a max pooling layer. The pooling is 2x2 with stride 1.
iv. After max pooling, the layer is connected to the next convolution layer, with 16 feature maps. The convolution kernels are of 3x3 in size. Use stride 1 for convolution.
v. The second convolution layer is followed by a max pooling layer. The pooling is 2x2 with stride 1.
vi. After max pooling, the layer is fully connected to the next hidden layer with 120 nodes and relu as the activation function.
vii. The fully connected layer is followed by another fully connected layer with 84 nodes and relu as the activation function, then connected to a softmax layer with 10 output nodes (corresponding to the 10 classes).

Resources:

- Google Colaboratory - allows you to write and execute Python in your browser, with Zero configuration required, Free access to GPUs, and Easy sharing.
- MNIST Dataset- The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples.
- Keras-Library- Keras is an open-source neural-network library written in python. It is capable of running on top of TensorFlow, R, Theano or PlaidML.

Approach:

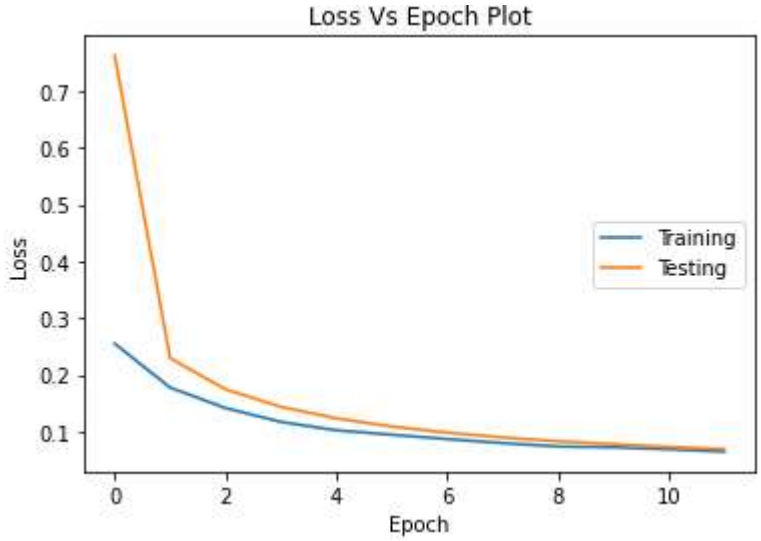
The approach is to follow and modify the base line code as per below requirements:

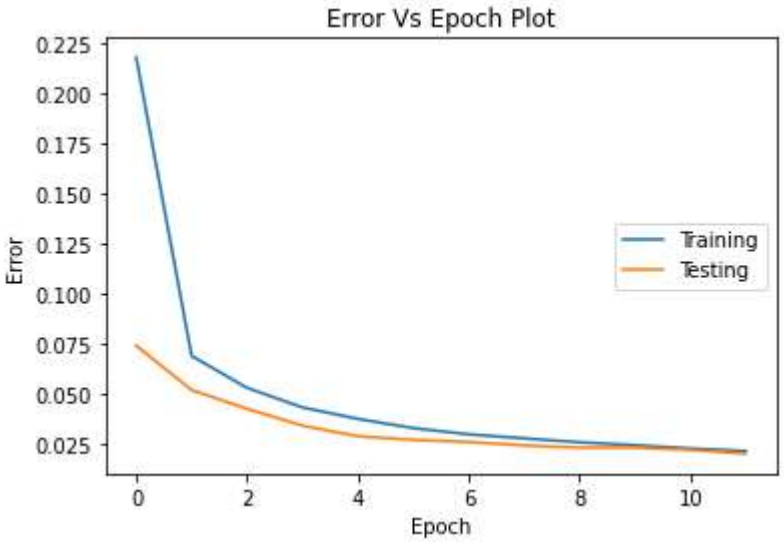
i. Run the baseline code and report accuracy.
ii. Change the kernel size to 5*5, redo the experiment, plot the learning errors along with the epoch, and report the testing error and accuracy on the test set.

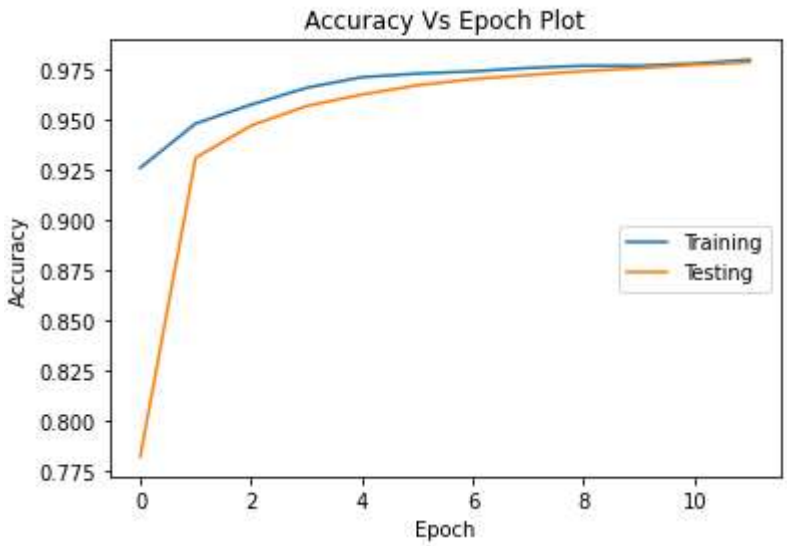
iii. Change the number of the feature maps in the first and second convolution layers, redo the experiment, plot the learning errors along with the epoch, and report the testing error and accuracy on the test set.

Results:

Approach1:

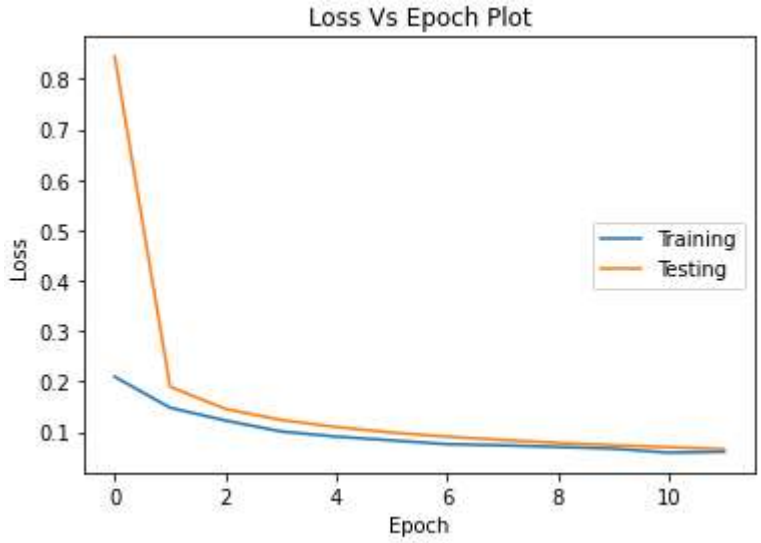


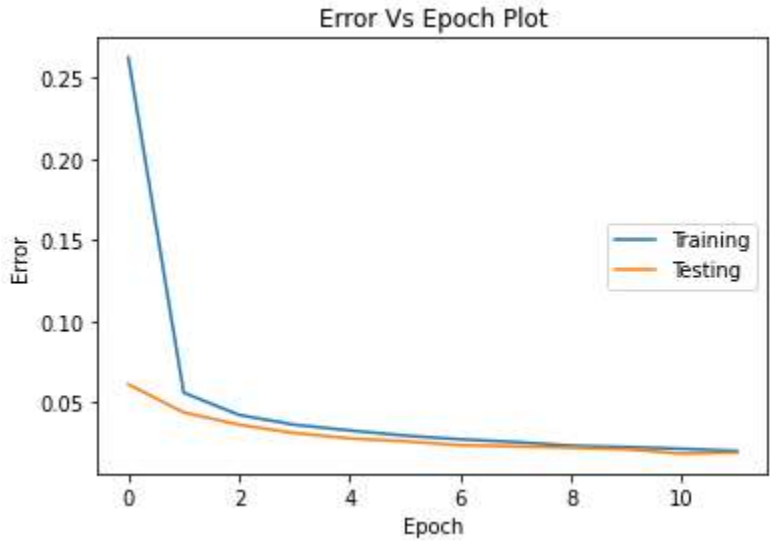


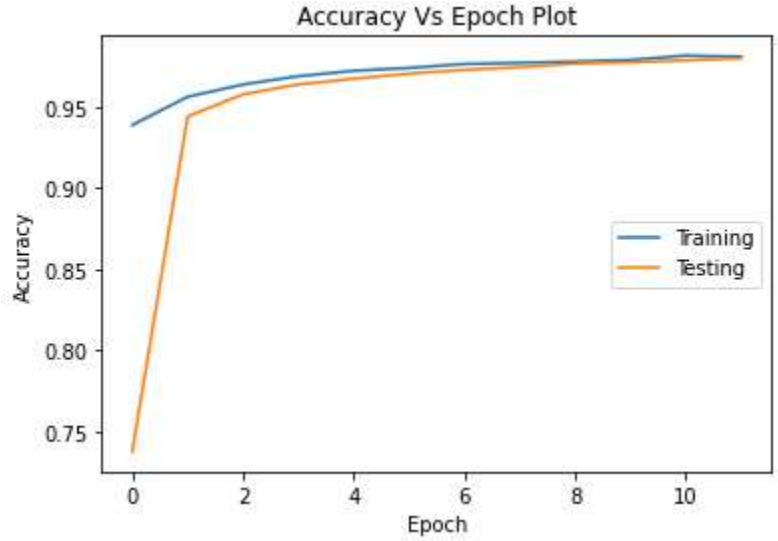



```
Test Error: 0.019800007343292236
Test Loss: 0.06463430673712864
Test Accuracy: 0.9801999926567078
```

Approach2:

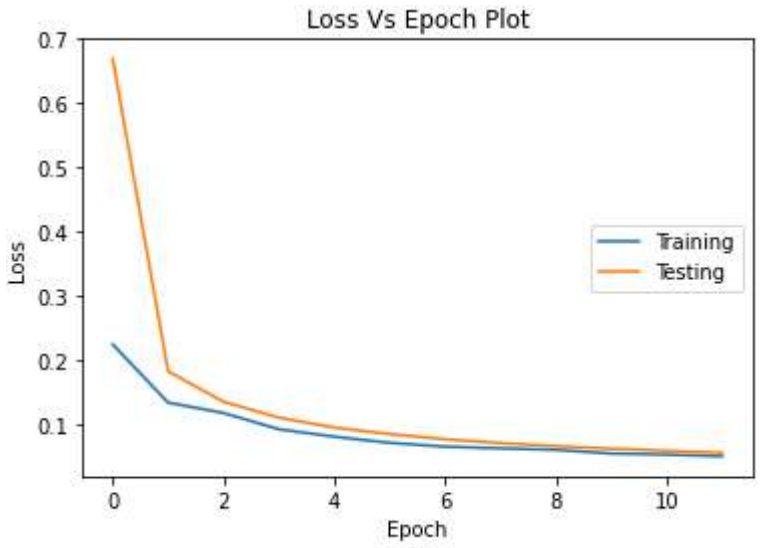


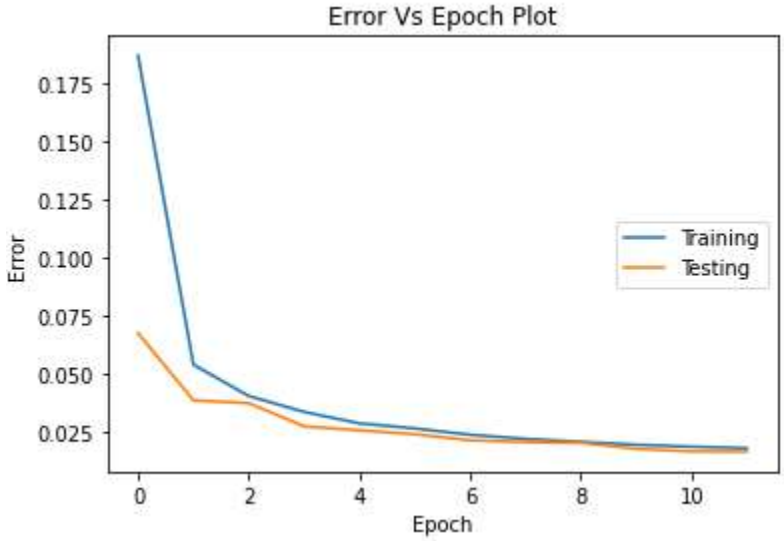


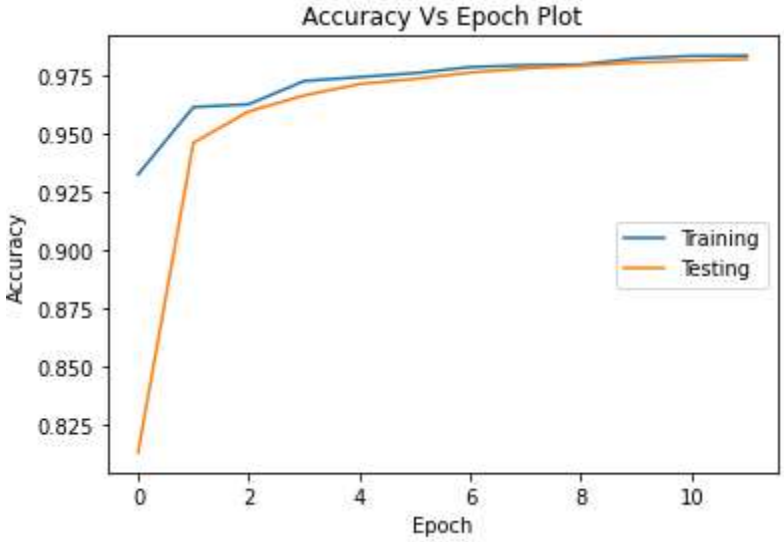



```
Test Error: 0.01910001039505005
Test Loss: 0.06056100460686721
Test Accuracy: 0.98089998960495
```

Approach3:


Loss Vs Epoch Plot
Loss
Epoch
Training
Testing
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10


Error Vs Epoch Plot
Error
Epoch
Training
Testing
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
2
4
6
8
10


Accuracy Vs Epoch Plot
Accuracy
Epoch
Training
Testing
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0
2
4
6
8
10

```
Test Error: 0.016200006008148193
Test Loss: 0.05119375395392999
Test Accuracy: 0.9837999939918518
```

Conclusion:

It can be concluded from the above results that if we want a powerful model, we need to apply more and big filters.