CSE: 575 PROJECT 3 REPORT

-Harshith Chittajallu, ASU ID- 1218707243

April 29, 2020

INTRODUCTION

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. [1]

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area. [1]

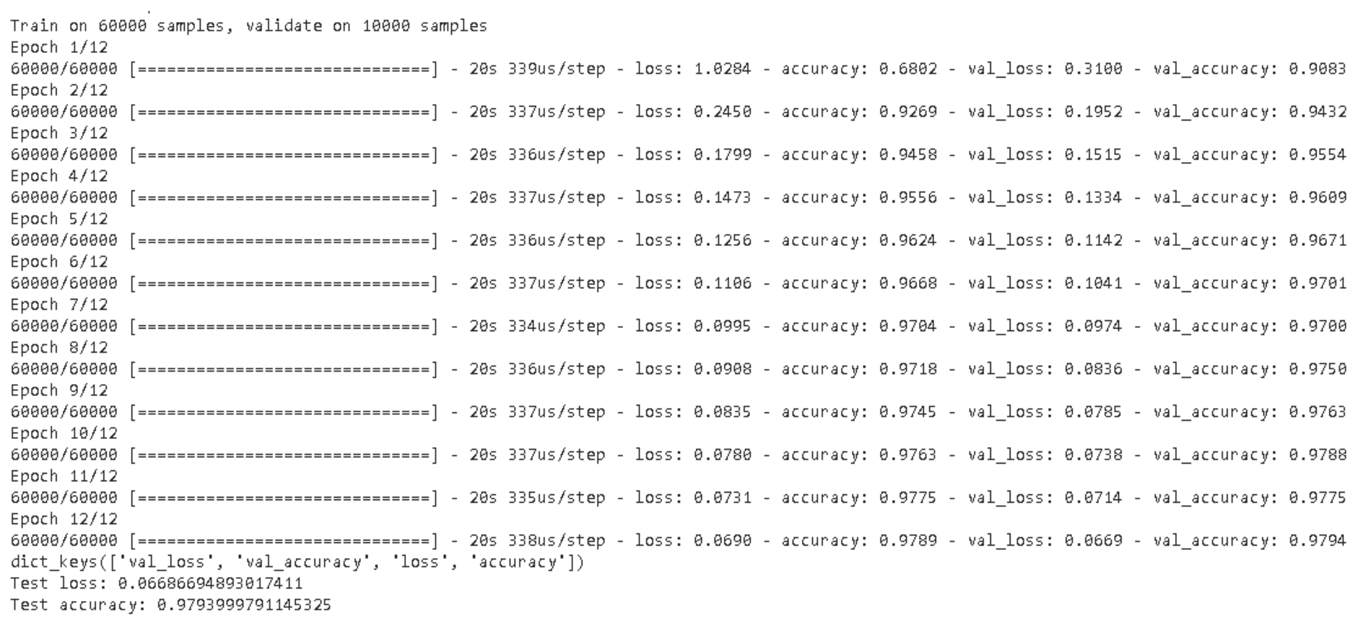
In this project, we use the convolutional neural network architecture to predict the complete MNIST dataset, imported from Keras in Python. We initially have a baseline code, in which there are 2 convolutional layers having initial hyper-parameters. We now experiment with the different parameter settings and check the testing error along with the epoch and report the accuracy and loss results. The format of the code is in Python 3 and it is executed in the Google collab workspace for additional computational power.

Initial Results:

The initially set hyper parameters were as follows:

Layers	Feature-maps	Kernel Size	Optimizer used	Learning rate	Pool size	Batch size
2 layered network Of CNN's followed by a Max-pooling layer each: CNN-P-CNN-P	CNN1: 6 CNN2: 16	CNN1: 3x3 CNN2: 3x3	Adadelta	0.1	Maxpooling1: (2,2) Maxpooling1: (2,2)	128

After running the code, these were the following observations:



```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 20s 339us/step - loss: 1.0284 - accuracy: 0.6802 - val_loss: 0.3100 - val_accuracy: 0.9083
Epoch 2/12
60000/60000 [==============================] - 20s 337us/step - loss: 0.2450 - accuracy: 0.9269 - val_loss: 0.1952 - val_accuracy: 0.9432
Epoch 3/12
60000/60000 [==============================] - 20s 336us/step - loss: 0.1799 - accuracy: 0.9458 - val_loss: 0.1515 - val_accuracy: 0.9554
Epoch 4/12
60000/60000 [==============================] - 20s 337us/step - loss: 0.1473 - accuracy: 0.9556 - val_loss: 0.1334 - val_accuracy: 0.9609
Epoch 5/12
60000/60000 [==============================] - 20s 336us/step - loss: 0.1256 - accuracy: 0.9624 - val_loss: 0.1142 - val_accuracy: 0.9671
Epoch 6/12
60000/60000 [==============================] - 20s 337us/step - loss: 0.1106 - accuracy: 0.9668 - val_loss: 0.1041 - val_accuracy: 0.9701
Epoch 7/12
60000/60000 [==============================] - 20s 334us/step - loss: 0.0995 - accuracy: 0.9704 - val_loss: 0.0974 - val_accuracy: 0.9700
Epoch 8/12
60000/60000 [==============================] - 20s 336us/step - loss: 0.0908 - accuracy: 0.9718 - val_loss: 0.0836 - val_accuracy: 0.9750
Epoch 9/12
60000/60000 [==============================] - 20s 337us/step - loss: 0.0835 - accuracy: 0.9745 - val_loss: 0.0785 - val_accuracy: 0.9763
Epoch 10/12
60000/60000 [==============================] - 20s 337us/step - loss: 0.0780 - accuracy: 0.9763 - val_loss: 0.0738 - val_accuracy: 0.9788
Epoch 11/12
60000/60000 [==============================] - 20s 335us/step - loss: 0.0731 - accuracy: 0.9775 - val_loss: 0.0714 - val_accuracy: 0.9775
Epoch 12/12
60000/60000 [==============================] - 20s 338us/step - loss: 0.0690 - accuracy: 0.9789 - val_loss: 0.0669 - val_accuracy: 0.9794
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Test loss: 0.06686694893017411
Test accuracy: 0.9793999791145325
```

Test Loss: 0.0668

Test Accuracy: 0.9793 = 97.93%

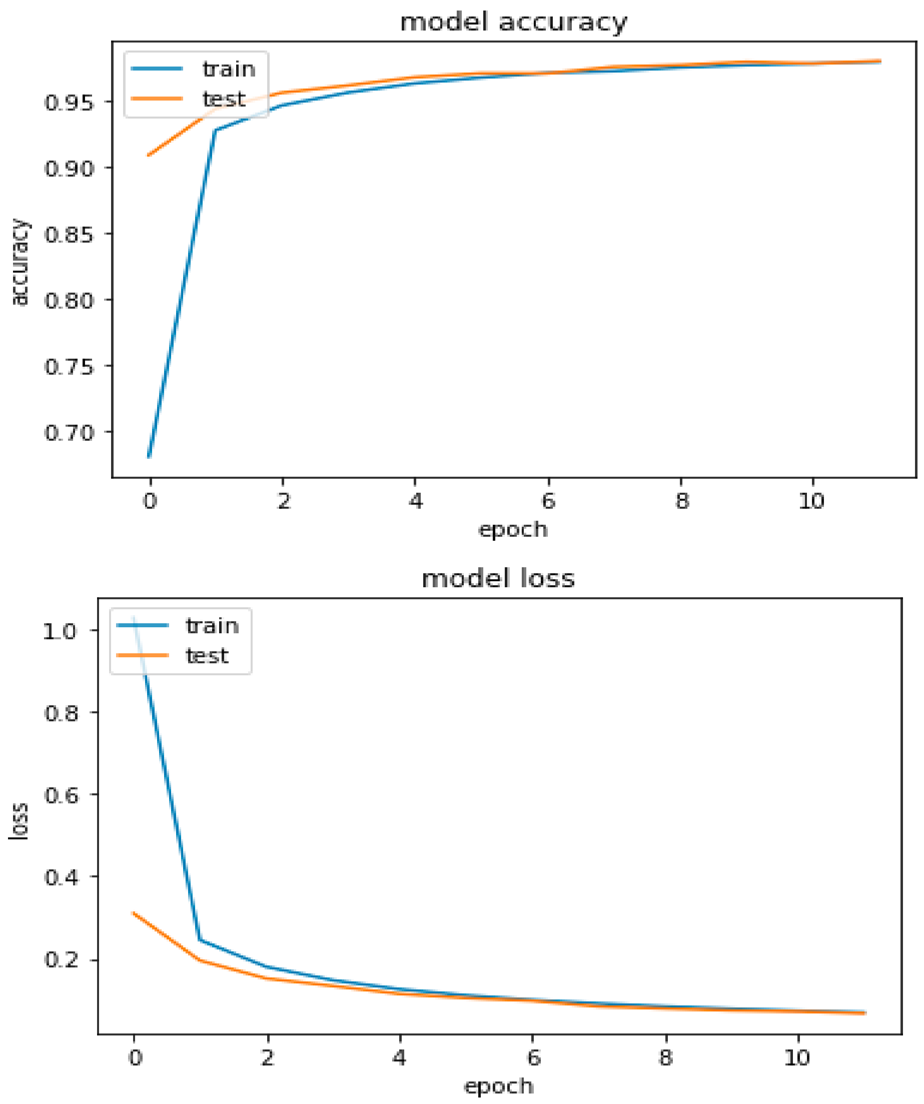
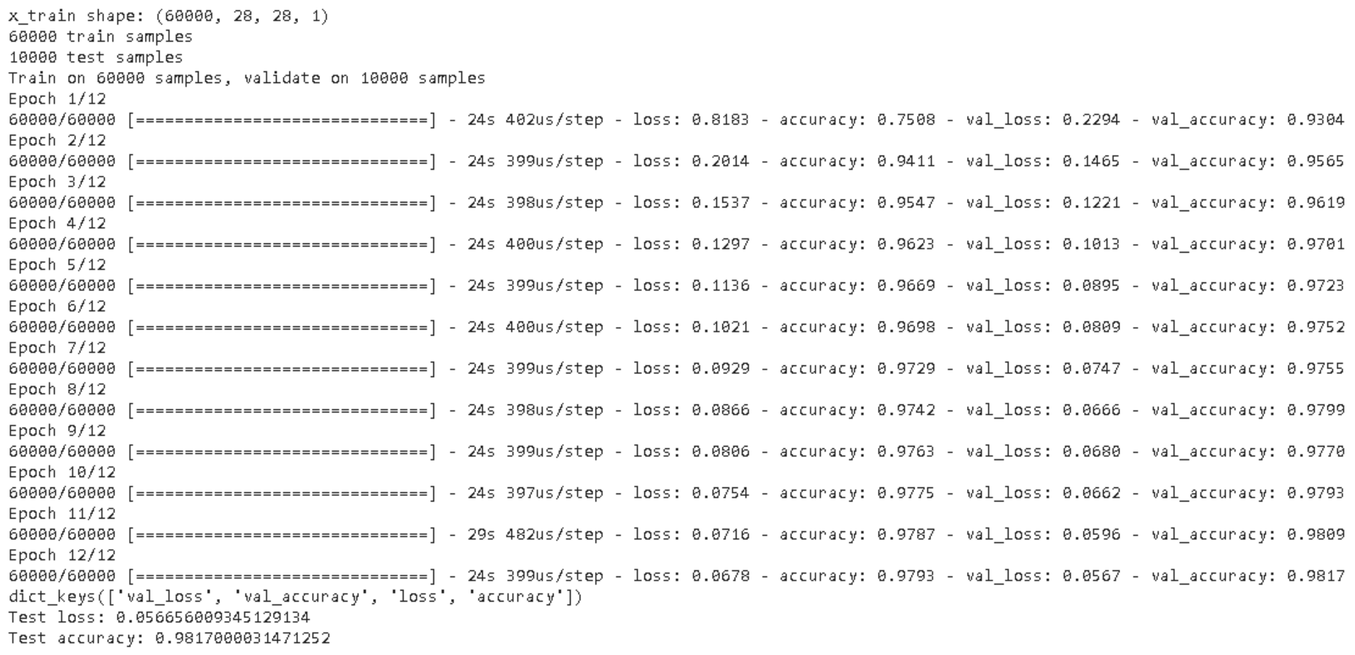


Fig1: Plot of Accuracy vs epoch and Loss vs epoch, base code.

EXPERIMENT & RESULTS

1) To start off the experiment, initially both the kernel sizes were changed to 5x5. The new result was as follows:



```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 24s 402us/step - loss: 0.8183 - accuracy: 0.7508 - val_loss: 0.2294 - val_accuracy: 0.9304
Epoch 2/12
60000/60000 [==============================] - 24s 399us/step - loss: 0.2014 - accuracy: 0.9411 - val_loss: 0.1465 - val_accuracy: 0.9565
Epoch 3/12
60000/60000 [==============================] - 24s 398us/step - loss: 0.1537 - accuracy: 0.9547 - val_loss: 0.1221 - val_accuracy: 0.9619
Epoch 4/12
60000/60000 [==============================] - 24s 400us/step - loss: 0.1297 - accuracy: 0.9623 - val_loss: 0.1013 - val_accuracy: 0.9701
Epoch 5/12
60000/60000 [==============================] - 24s 399us/step - loss: 0.1136 - accuracy: 0.9669 - val_loss: 0.0895 - val_accuracy: 0.9723
Epoch 6/12
60000/60000 [==============================] - 24s 400us/step - loss: 0.1021 - accuracy: 0.9698 - val_loss: 0.0809 - val_accuracy: 0.9752
Epoch 7/12
60000/60000 [==============================] - 24s 399us/step - loss: 0.0929 - accuracy: 0.9729 - val_loss: 0.0747 - val_accuracy: 0.9755
Epoch 8/12
60000/60000 [==============================] - 24s 398us/step - loss: 0.0866 - accuracy: 0.9742 - val_loss: 0.0666 - val_accuracy: 0.9799
Epoch 9/12
60000/60000 [==============================] - 24s 399us/step - loss: 0.0806 - accuracy: 0.9763 - val_loss: 0.0680 - val_accuracy: 0.9770
Epoch 10/12
60000/60000 [==============================] - 24s 397us/step - loss: 0.0754 - accuracy: 0.9775 - val_loss: 0.0662 - val_accuracy: 0.9793
Epoch 11/12
60000/60000 [==============================] - 29s 482us/step - loss: 0.0716 - accuracy: 0.9787 - val_loss: 0.0596 - val_accuracy: 0.9809
Epoch 12/12
60000/60000 [==============================] - 24s 399us/step - loss: 0.0678 - accuracy: 0.9793 - val_loss: 0.0567 - val_accuracy: 0.9817
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Test loss: 0.056656009345129134
Test accuracy: 0.9817000031471252
```

Test loss: 0.05665

Test Accuracy: 0.9817 = 98.17%

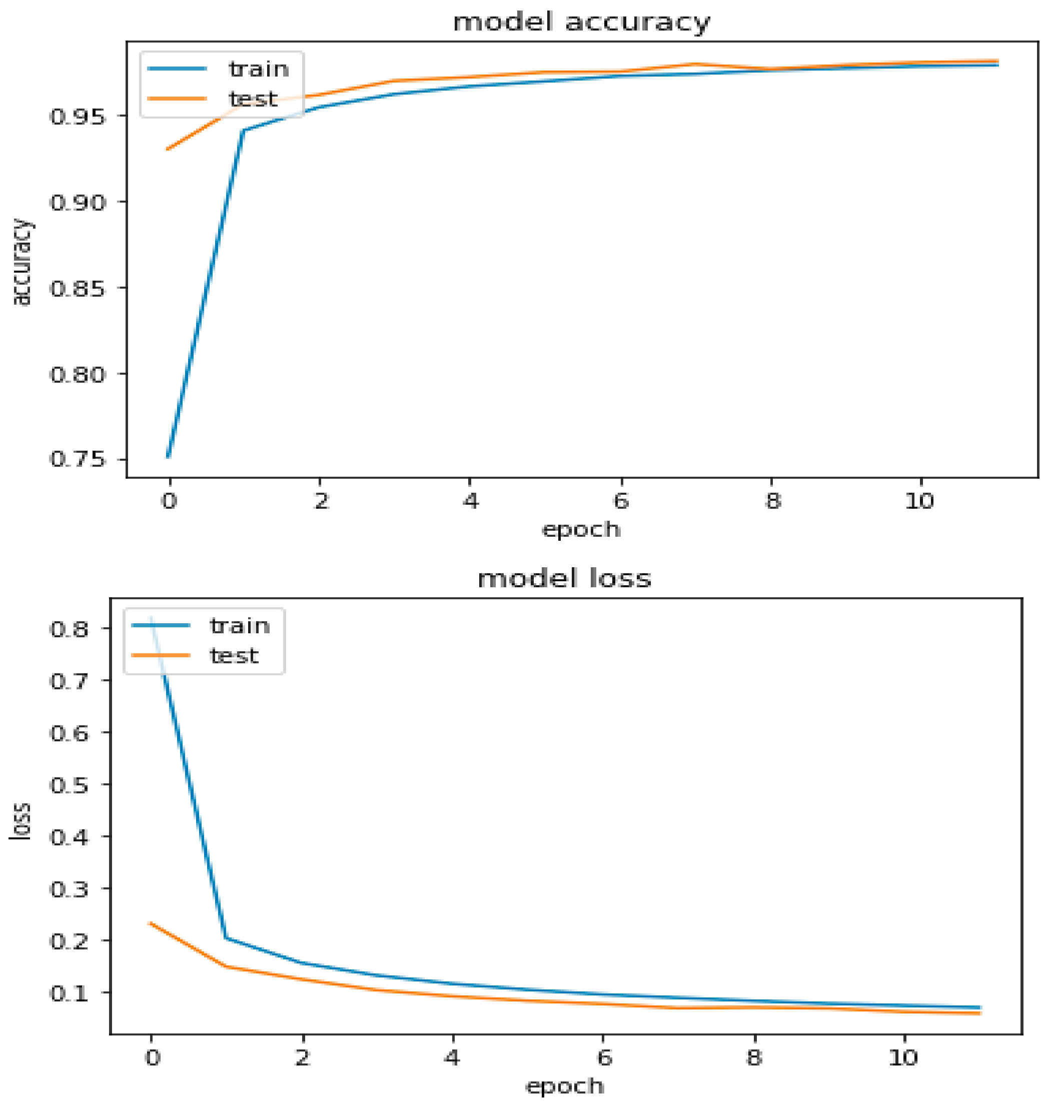
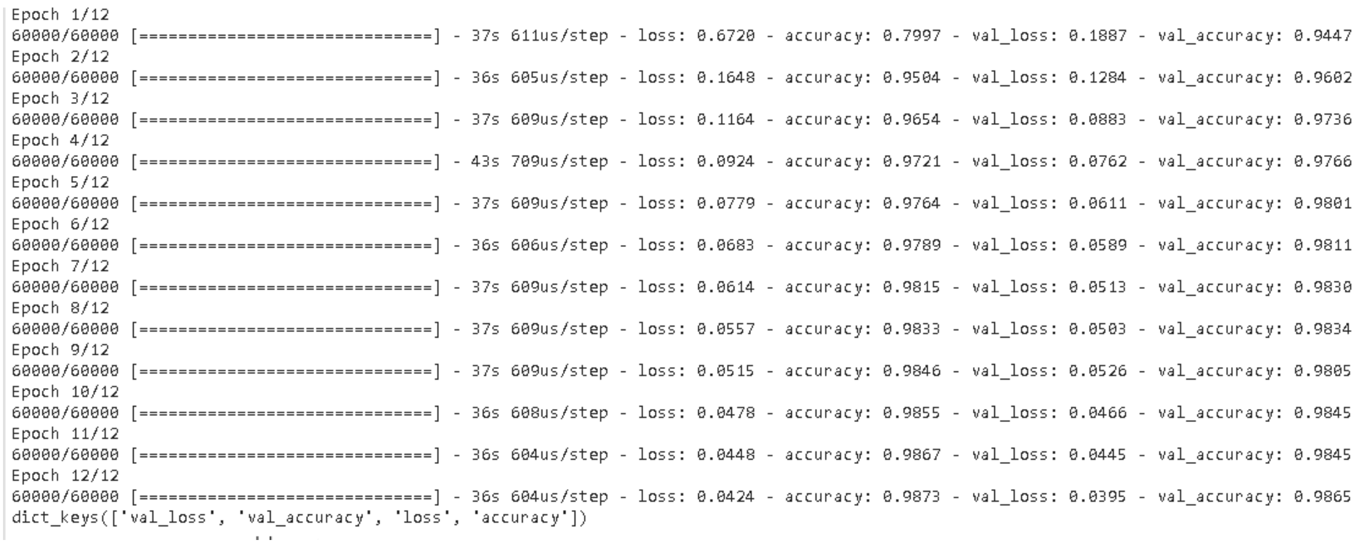


Fig2: Plot of Accuracy vs epoch and Loss vs epoch, both Kernels changed to 5x5

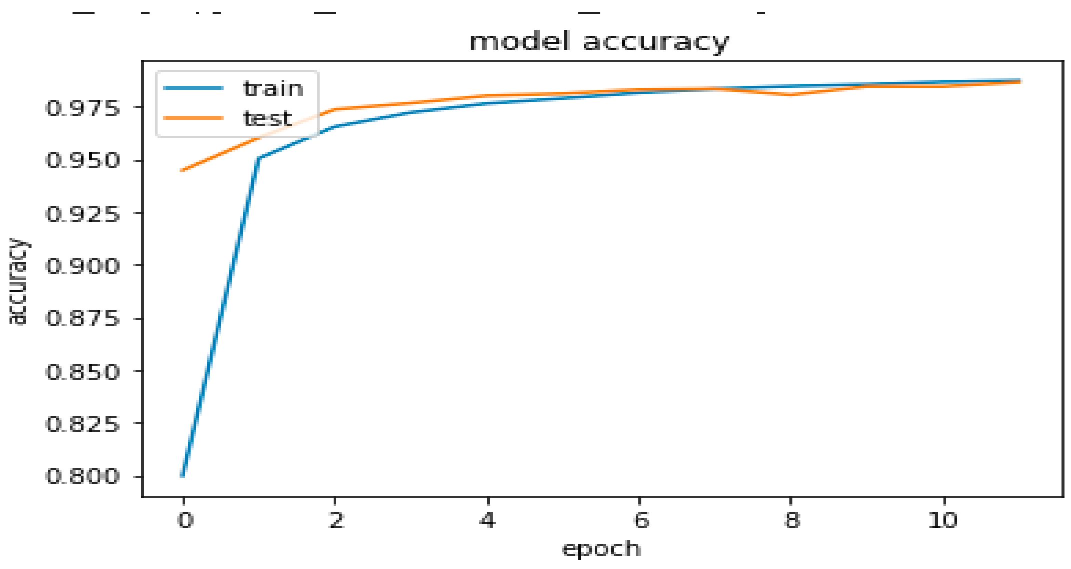
2) Then, The number of feature maps was increased: For the first CNN, the feature maps are now set to 12, and for the second CNN, the feature maps were set to 32
   Essentially, we doubled the number of feature maps.
   The changed result was as follows:



```
Epoch 1/12
60000/60000 [==============================] - 37s 611us/step - loss: 0.6720 - accuracy: 0.7997 - val_loss: 0.1887 - val_accuracy: 0.9447
Epoch 2/12
60000/60000 [==============================] - 36s 605us/step - loss: 0.1648 - accuracy: 0.9504 - val_loss: 0.1284 - val_accuracy: 0.9602
Epoch 3/12
60000/60000 [==============================] - 37s 609us/step - loss: 0.1164 - accuracy: 0.9654 - val_loss: 0.0883 - val_accuracy: 0.9736
Epoch 4/12
60000/60000 [==============================] - 43s 709us/step - loss: 0.0924 - accuracy: 0.9721 - val_loss: 0.0762 - val_accuracy: 0.9766
Epoch 5/12
60000/60000 [==============================] - 37s 609us/step - loss: 0.0779 - accuracy: 0.9764 - val_loss: 0.0611 - val_accuracy: 0.9801
Epoch 6/12
60000/60000 [==============================] - 36s 606us/step - loss: 0.0683 - accuracy: 0.9789 - val_loss: 0.0589 - val_accuracy: 0.9811
Epoch 7/12
60000/60000 [==============================] - 37s 609us/step - loss: 0.0614 - accuracy: 0.9815 - val_loss: 0.0513 - val_accuracy: 0.9830
Epoch 8/12
60000/60000 [==============================] - 37s 609us/step - loss: 0.0557 - accuracy: 0.9833 - val_loss: 0.0503 - val_accuracy: 0.9834
Epoch 9/12
60000/60000 [==============================] - 37s 609us/step - loss: 0.0515 - accuracy: 0.9846 - val_loss: 0.0526 - val_accuracy: 0.9805
Epoch 10/12
60000/60000 [==============================] - 36s 608us/step - loss: 0.0478 - accuracy: 0.9855 - val_loss: 0.0466 - val_accuracy: 0.9845
Epoch 11/12
60000/60000 [==============================] - 36s 604us/step - loss: 0.0448 - accuracy: 0.9867 - val_loss: 0.0445 - val_accuracy: 0.9845
Epoch 12/12
60000/60000 [==============================] - 36s 604us/step - loss: 0.0424 - accuracy: 0.9873 - val_loss: 0.0395 - val_accuracy: 0.9865
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

Test loss: 0.03955

Test Accuracy: 0.9817 = <u>98.65%</u>



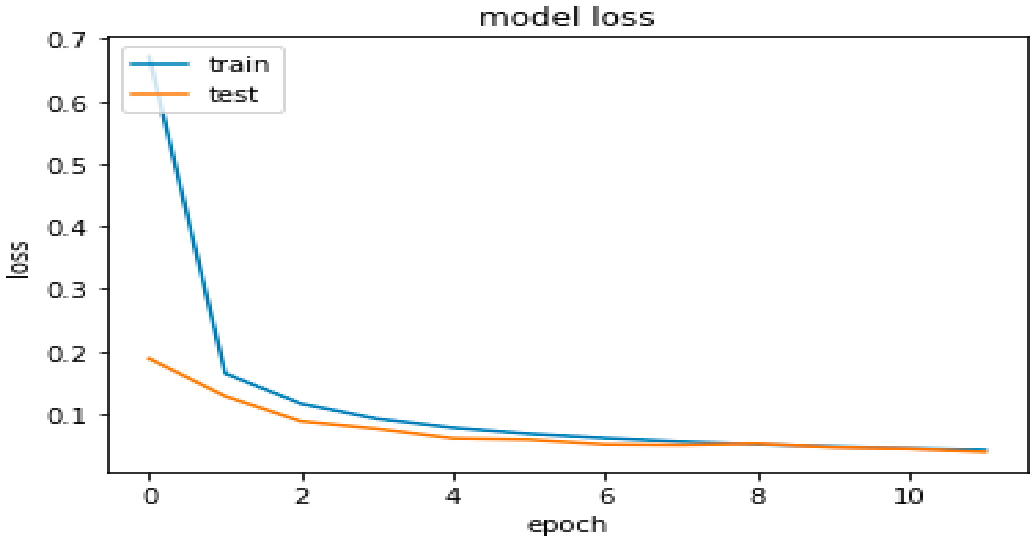
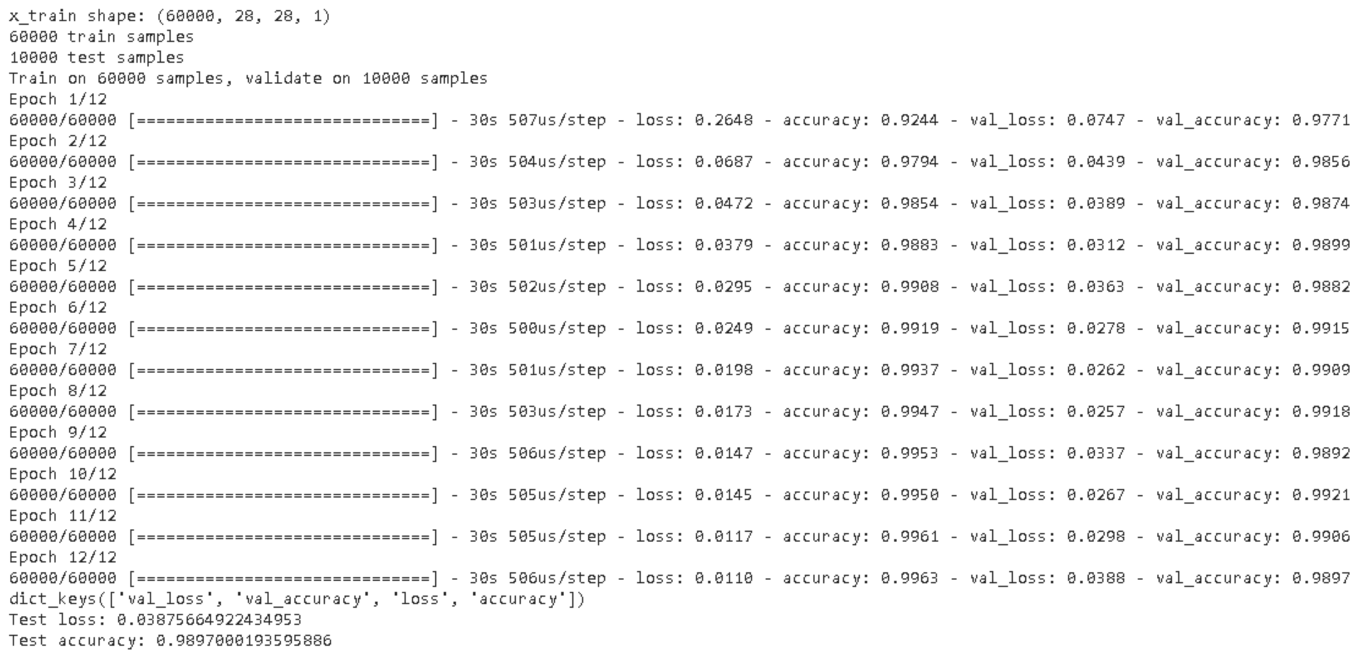


Fig3: Plot of Accuracy vs epoch and Loss vs epoch, both Kernels changed to 5x5, number of feature maps doubled in size

3) Now, we use a different optimizer and a different learning rate. The optimizer used is ‘Adam’ and its parameters are set to (beta_1=0.9, beta_2=0.999, amsgrad=False)
   The learning rate is now lowered to 0.001.
   After execution, the results were as follows:



```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 30s 507us/step - loss: 0.2648 - accuracy: 0.9244 - val_loss: 0.0747 - val_accuracy: 0.9771
Epoch 2/12
60000/60000 [==============================] - 30s 504us/step - loss: 0.0687 - accuracy: 0.9794 - val_loss: 0.0439 - val_accuracy: 0.9856
Epoch 3/12
60000/60000 [==============================] - 30s 503us/step - loss: 0.0472 - accuracy: 0.9854 - val_loss: 0.0389 - val_accuracy: 0.9874
Epoch 4/12
60000/60000 [==============================] - 30s 501us/step - loss: 0.0379 - accuracy: 0.9883 - val_loss: 0.0312 - val_accuracy: 0.9899
Epoch 5/12
60000/60000 [==============================] - 30s 502us/step - loss: 0.0295 - accuracy: 0.9908 - val_loss: 0.0363 - val_accuracy: 0.9882
Epoch 6/12
60000/60000 [==============================] - 30s 500us/step - loss: 0.0249 - accuracy: 0.9919 - val_loss: 0.0278 - val_accuracy: 0.9915
Epoch 7/12
60000/60000 [==============================] - 30s 501us/step - loss: 0.0198 - accuracy: 0.9937 - val_loss: 0.0262 - val_accuracy: 0.9909
Epoch 8/12
60000/60000 [==============================] - 30s 503us/step - loss: 0.0173 - accuracy: 0.9947 - val_loss: 0.0257 - val_accuracy: 0.9918
Epoch 9/12
60000/60000 [==============================] - 30s 506us/step - loss: 0.0147 - accuracy: 0.9953 - val_loss: 0.0337 - val_accuracy: 0.9892
Epoch 10/12
60000/60000 [==============================] - 30s 505us/step - loss: 0.0145 - accuracy: 0.9950 - val_loss: 0.0267 - val_accuracy: 0.9921
Epoch 11/12
60000/60000 [==============================] - 30s 505us/step - loss: 0.0117 - accuracy: 0.9961 - val_loss: 0.0298 - val_accuracy: 0.9906
Epoch 12/12
60000/60000 [==============================] - 30s 506us/step - loss: 0.0110 - accuracy: 0.9963 - val_loss: 0.0388 - val_accuracy: 0.9897
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Test loss: 0.03875664922434953
Test accuracy: 0.9897000193595886
```

Test loss: 0.03875

Test Accuracy: 0.9897 = 98.97%

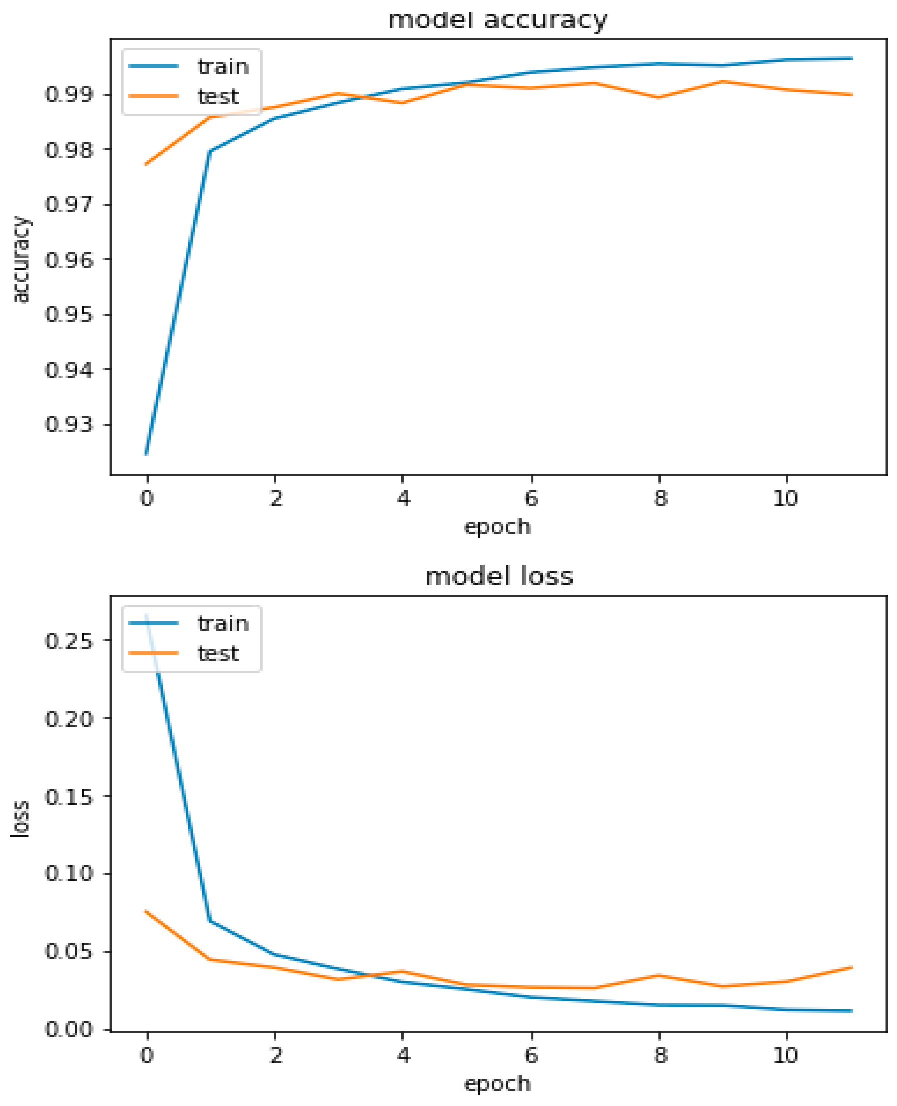
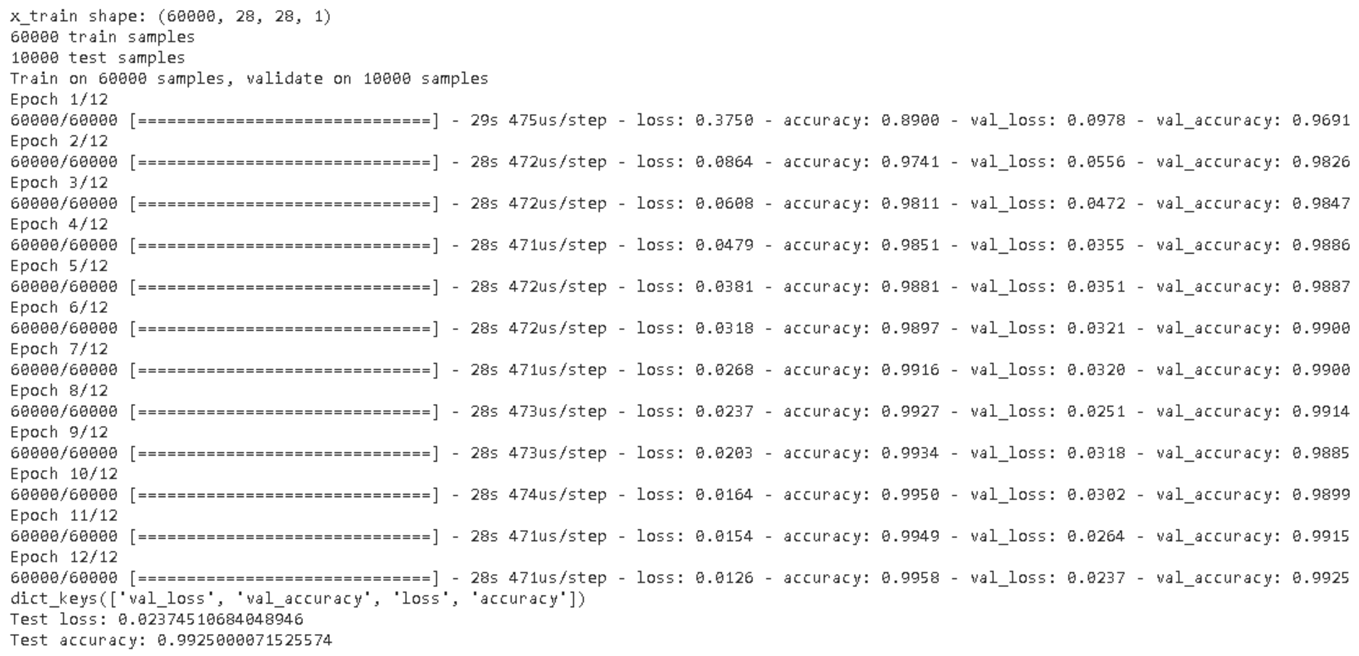


Fig3: Plot of Accuracy vs epoch and Loss vs epoch, both Kernels changed to 5x5, number of feature maps doubled in size and a new optimizer with a learning rate of 0.001 used

4) Finally, we change the batch size of 128 to 256, essentially doubling the batch size. Keeping the previous changes, the following given below is the result:



```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 29s 475us/step - loss: 0.3750 - accuracy: 0.8900 - val_loss: 0.0978 - val_accuracy: 0.9691
Epoch 2/12
60000/60000 [==============================] - 28s 472us/step - loss: 0.0864 - accuracy: 0.9741 - val_loss: 0.0556 - val_accuracy: 0.9826
Epoch 3/12
60000/60000 [==============================] - 28s 472us/step - loss: 0.0608 - accuracy: 0.9811 - val_loss: 0.0472 - val_accuracy: 0.9847
Epoch 4/12
60000/60000 [==============================] - 28s 471us/step - loss: 0.0479 - accuracy: 0.9851 - val_loss: 0.0355 - val_accuracy: 0.9886
Epoch 5/12
60000/60000 [==============================] - 28s 472us/step - loss: 0.0381 - accuracy: 0.9881 - val_loss: 0.0351 - val_accuracy: 0.9887
Epoch 6/12
60000/60000 [==============================] - 28s 472us/step - loss: 0.0318 - accuracy: 0.9897 - val_loss: 0.0321 - val_accuracy: 0.9900
Epoch 7/12
60000/60000 [==============================] - 28s 471us/step - loss: 0.0268 - accuracy: 0.9916 - val_loss: 0.0320 - val_accuracy: 0.9900
Epoch 8/12
60000/60000 [==============================] - 28s 473us/step - loss: 0.0237 - accuracy: 0.9927 - val_loss: 0.0251 - val_accuracy: 0.9914
Epoch 9/12
60000/60000 [==============================] - 28s 473us/step - loss: 0.0203 - accuracy: 0.9934 - val_loss: 0.0318 - val_accuracy: 0.9885
Epoch 10/12
60000/60000 [==============================] - 28s 474us/step - loss: 0.0164 - accuracy: 0.9950 - val_loss: 0.0302 - val_accuracy: 0.9899
Epoch 11/12
60000/60000 [==============================] - 28s 471us/step - loss: 0.0154 - accuracy: 0.9949 - val_loss: 0.0264 - val_accuracy: 0.9915
Epoch 12/12
60000/60000 [==============================] - 28s 471us/step - loss: 0.0126 - accuracy: 0.9958 - val_loss: 0.0237 - val_accuracy: 0.9925
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Test loss: 0.02374510684048946
Test accuracy: 0.9925000071525574
```

Test loss: 0.023745

Test Accuracy: 0.9925 = 99.25%

Changing the parameters further at this point did not improve the accuracy anymore and the only way to further improve the accuracy even further was to change the architecture of the neural net.

Regardless, the highest possible accuracy is currently 99.81% on the MNIST dataset, which is proof of our model's hyper-parameter selection being good enough.

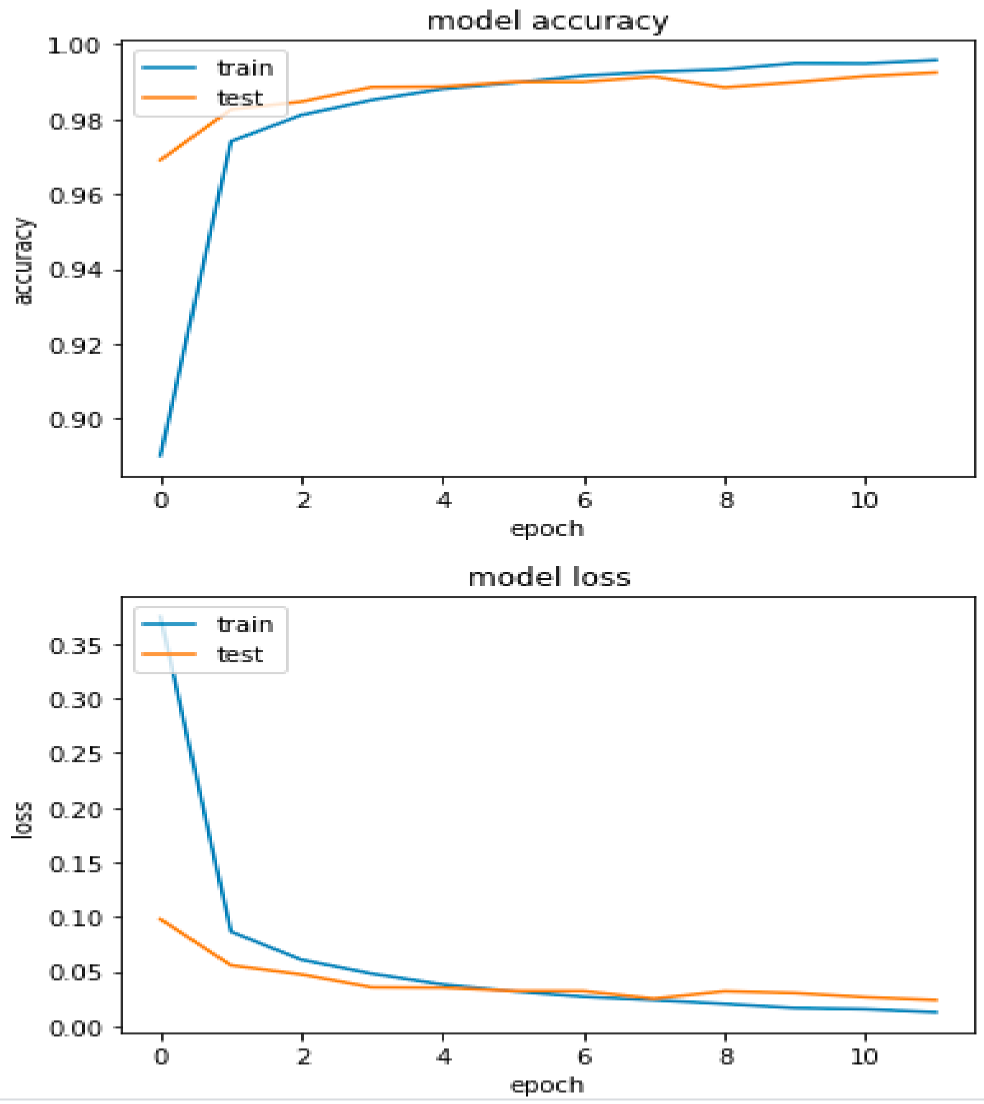


Fig4: Plot of Accuracy vs epoch and Loss vs epoch, both Kernels changed to 5x5, number of feature maps doubled in size, new optimizer with a learning rate of 0.001 used and the batch size is doubled.

Conclusion

The experiment has highlighted the effect of different hyper-parameters on the loss and the accuracy of the CNN model Each data set will have an optimal criteria of parameters, whose results need to be shown via brute force method to determine which parameters give consistently the highest accuracy and lowest loss per epoch. They may not necessarily be the highest or lowest of the values allowed and ultimately depend on the nature of the given dataset.

The final parameters used to achieve the final classification accuracy of 99.25% are given below:

Layers	Feature-maps	Kernel Size	Optimizer used	Learning rate	Pool size	Batch size
2 layered network Of CNN's followed by a Max-pooling layer each: CNN-P-CNN-P	CNN1: 12 CNN2: 32	CNN1: 5x5 CNN2: 5x5	Adam(beta_1=0.9, beta_2=0.999, amsgrad=False)	0.001	Maxpooling1: (2,2) Maxpooling1: (2,2)	256

REFERENCES:

[1] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, by Sumit Saha