EEL 6814
HMW # 3

Model 1-Single Convolution layer

The architecture consists of 1 Convolution Layers with 64 filters of size 3x3 followed by maxpooling layers and flattened and followed by one dense hidden layer with 1024 PE's with relu activation and a output layer with 10 PE's with sigmoid activation. The loss function is categorical crossentropy used with adam optimizer.

Hyperparameter- lr-0.001, batch size=150, weight initialization=random with std dev of 0.01, Stopping Criteria-Generalization on unseen data (validation dataset)

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 28, 28, 64)	640
max_pooling2d_12 (MaxPooling	(None, 14, 14, 64)	0
flatten_6 (Flatten)	(None, 12544)	0
dense_12 (Dense)	(None, 1024)	12846080
dropout_6 (Dropout)	(None, 1024)	0
dense_13 (Dense)	(None, 10)	10250

Total params: 12,856,970
Trainable params: 12,856,970
Non-trainable params: 0

Fig 1

```
Test loss: 0.040245024617027954
Test accuracy: 98.72000217437744
Test error rate: 1.2799978256225586
Train loss: 0.011857842646213248
Train accuracy: 99.63399767875671
Train error rate: 0.36600232124328613
```

Fig 2

```
array([[ 969,    1,    1,    0,    1,    1,    4,    2,    1,    0],
       [   0, 1132,    2,    0,    0,    0,    0,    1,    0,    0],
       [   3,    5, 1012,    0,    1,    0,    3,    5,    3,    0],
       [   0,    0,    1,  998,    0,    1,    0,    6,    4,    0],
       [   0,    0,    0,    0,  971,    0,    1,    0,    0,   10],
       [   2,    0,    0,    7,    0,  879,    2,    1,    1,    0],
       [   2,    2,    0,    0,    1,    3,  950,    0,    0,    0],
       [   0,    1,    7,    0,    0,    0,    0, 1017,    2,    1],
       [   4,    0,    1,    0,    3,    1,    0,    4,  957,    4],
       [   1,    1,    0,    0,    8,    2,    0,    7,    3,  987]],
      dtype=int64)
```

Fig 3

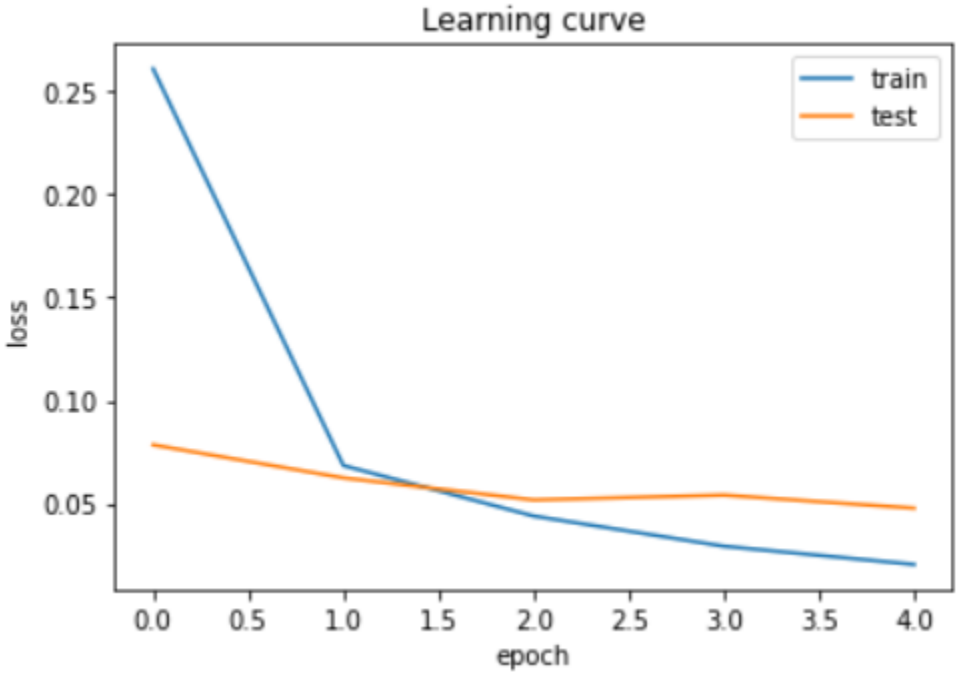

Fig 4

The architecture consists of 1 Convolution Layers with 64 filters of size 5x5 followed by maxpooling layers and flattened and followed by one dense hidden layer with 1024 PE's with relu activation and an output layer with 10 PE's with sigmoid activation. The loss function is categorical crossentropy used with adam optimizer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 28, 28, 64)        1664
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_14 (Dense)             (None, 1024)              12846080
_________________________________________________________________
dropout_7 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                10250
=================================================================
Total params: 12,857,994
Trainable params: 12,857,994
Non-trainable params: 0
_________________________________________________________________
```

Fig 5

```
Test loss: 0.030324110887246205
Test accuracy: 99.08000230789185
Test error rate: 0.9199976921081543
Train loss: 0.011489726623981 26
Train accuracy: 99.64200258255005
Train error rate: 0.35799741744995117
```

Fig 6

```
array([[ 968,    0,    3,    0,    1,    1,    5,    1,    1,    0],
       [   0, 1123,    4,    1,    0,    1,    4,    0,    2,    0],
       [   0,    0, 1024,    0,    1,    0,    0,    5,    2,    0],
       [   0,    0,    1, 1005,    0,    2,    0,    0,    1,    1],
       [   0,    0,    0,    0,  977,    0,    1,    0,    1,    3],
       [   0,    0,    0,    4,    0,  886,    2,    0,    0,    0],
       [   2,    2,    0,    0,    2,    3,  948,    0,    1,    0],
       [   1,    0,    5,    2,    0,    0,    0, 1018,    1,    1],
       [   2,    0,    2,    1,    0,    2,    1,    1,  962,    3],
       [   0,    0,    0,    0,    6,    3,    0,    2,    1,  997]],
      dtype=int64)
```

Fig 7

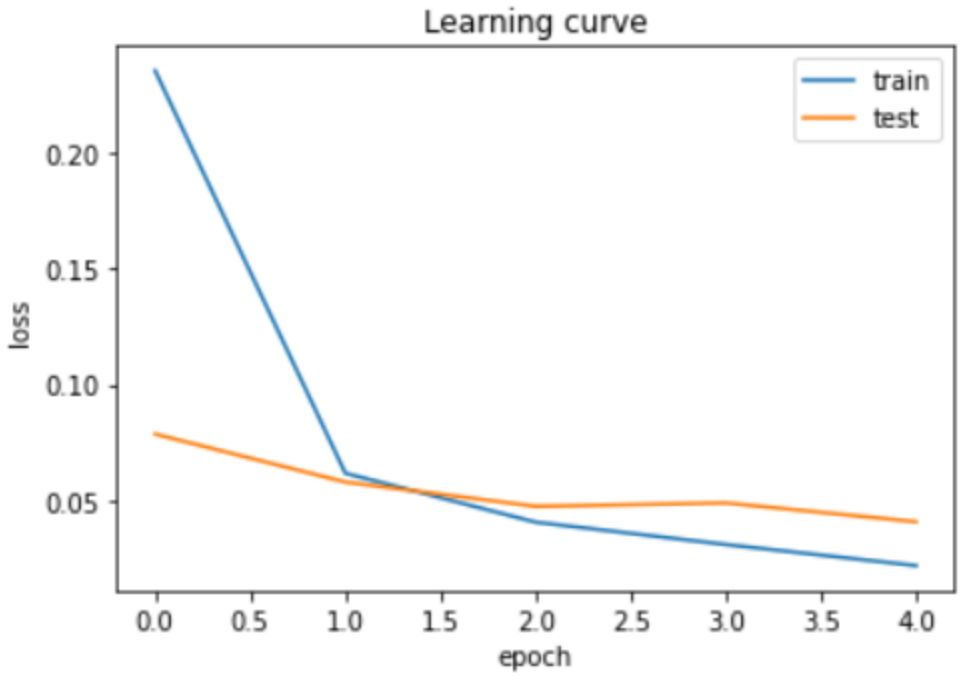

Fig 8

For the single convolutional layer architecture, a series of test with different numbers of filters and filter sizes and also different number of PE's in the dense layer were carried out and the best resulting outcomes have been depicted above. The best error rate captured by a single layer convolution layer network is 0.91%.

Model 2- Two Convolution layers

To improve the classification accuracy and the error rate I introduced a second convolution layer in the network and a series of test runs were carried out, but the number of PE's in the hidden layer were fixed to 1024 because when changed in the previous architecture did not produce any significant change in the outcome and also 1024 and 2048 produce good results when compared to other number of PE's tried (trial and error).

The architecture consists of 2 Convolution Layers with 32 filters of size 7x7 and 64 filters of size 5x5 respectively, followed by maxpooling layers after each convolution layer and finally flattened and followed by one dense hidden layer with 1024 PE's with relu activation and a output layer with 10 PE's with sigmoid activation. The loss function is categorical crossentropy used with adam optimizer.

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 28, 28, 32)	1600
max_pooling2d_14 (MaxPooling	(None, 14, 14, 32)	0
conv2d_17 (Conv2D)	(None, 14, 14, 64)	51264
max_pooling2d_15 (MaxPooling	(None, 7, 7, 64)	0
flatten_8 (Flatten)	(None, 3136)	0
dense_16 (Dense)	(None, 1024)	3212288
dropout_8 (Dropout)	(None, 1024)	0
dense_17 (Dense)	(None, 10)	10250

Total params: 3,275,402
Trainable params: 3,275,402
Non-trainable params: 0

Fig 9

```
Test loss: 0.025914058169879718
Test accuracy: 99.16999936103821
Test error rate: 0.830000638961792
Train loss: 0.015613190341126173
Train accuracy: 99.50199723243713
Train error rate: 0.4980027675628662
```

Fig 10

```
array([[ 979,    0,    0,    0,    0,    0,    0,    0,    0,    1],
       [   0, 1133,    1,    0,    0,    0,    0,    1,    0,    0],
       [   4,    0, 1024,    0,    0,    0,    0,    1,    3,    0],
       [   1,    0,    5,  998,    0,    2,    0,    1,    3,    0],
       [   0,    1,    1,    0,  976,    0,    0,    1,    0,    3],
       [   2,    0,    1,    3,    0,  884,    1,    0,    0,    1],
       [   3,    2,    1,    0,    4,    2,  946,    0,    0,    0],
       [   0,    2,    5,    0,    0,    0,    0, 1015,    1,    5],
       [   3,    0,    1,    0,    1,    1,    0,    1,  966,    1],
       [   0,    0,    0,    0,    8,    3,    0,    1,    1,  996]],
      dtype=int64)
```

Fig 11

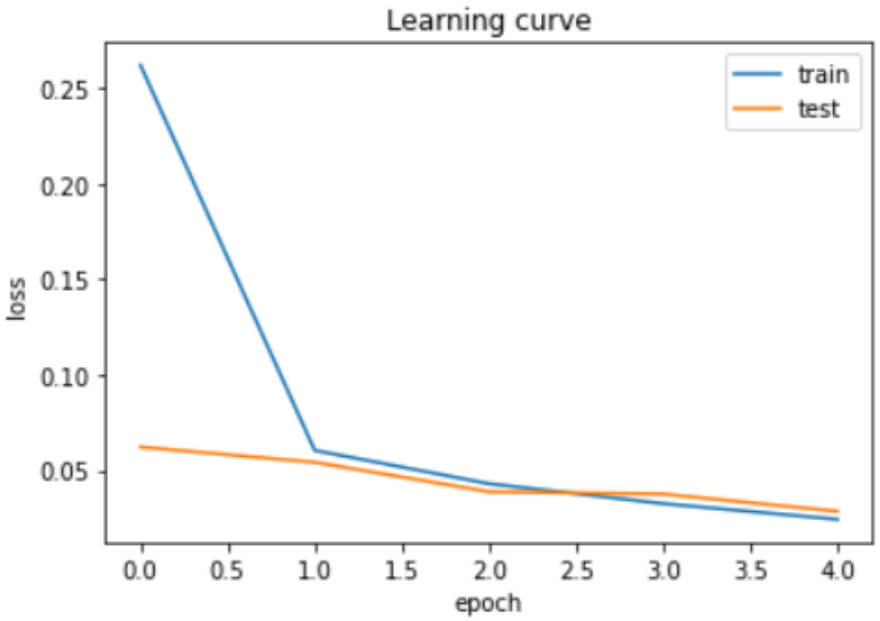

Fig 12

The architecture consists of 2 Convolution Layers with 32 filters of size 7x7 and 64 filters of size 3x3 respectively, followed by maxpooling layers after each convolution layer and finally flattened and followed by one dense hidden layer with 1024 PE's with relu activation and a output layer with 10 PE's with sigmoid activation. The loss function is categorical crossentropy used with adam optimizer.

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 28, 28, 32)	1600
max_pooling2d_16 (MaxPooling	(None, 14, 14, 32)	0
conv2d_19 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_17 (MaxPooling	(None, 7, 7, 64)	0
flatten_9 (Flatten)	(None, 3136)	0
dense_18 (Dense)	(None, 1024)	3212288
dropout_9 (Dropout)	(None, 1024)	0
dense_19 (Dense)	(None, 10)	10250

Total params: 3,242,634
Trainable params: 3,242,634
Non-trainable params: 0

Fig 13

```
Test loss: 0.03186189399957657
Test accuracy: 99.01000261306763
Test error rate: 0.989997386932373
Train loss: 0.020531072150440886
Train accuracy: 99.33800101280212
Train error rate: 0.661998987197876
```

Fig 14

```
array([[ 971,    0,    0,    1,    0,    0,    3,    0,    3,    2],
       [   0, 1127,    1,    3,    0,    1,    1,    1,    1,    0],
       [   0,    0, 1029,    0,    0,    0,    0,    1,    2,    0],
       [   0,    0,    0, 1007,    0,    3,    0,    0,    0,    0],
       [   0,    1,    1,    0,  970,    0,    0,    0,    0,   10],
       [   0,    0,    0,    7,    0,  883,    1,    0,    1,    0],
       [   1,    2,    0,    0,    1,    6,  945,    0,    3,    0],
       [   0,    1,    7,    2,    0,    1,    0, 1007,    1,    9],
       [   0,    0,    1,    3,    0,    3,    0,    0,  962,    5],
       [   0,    0,    0,    3,    2,    4,    0,    0,    0, 1000]],
      dtype=int64)
```

Fig 15

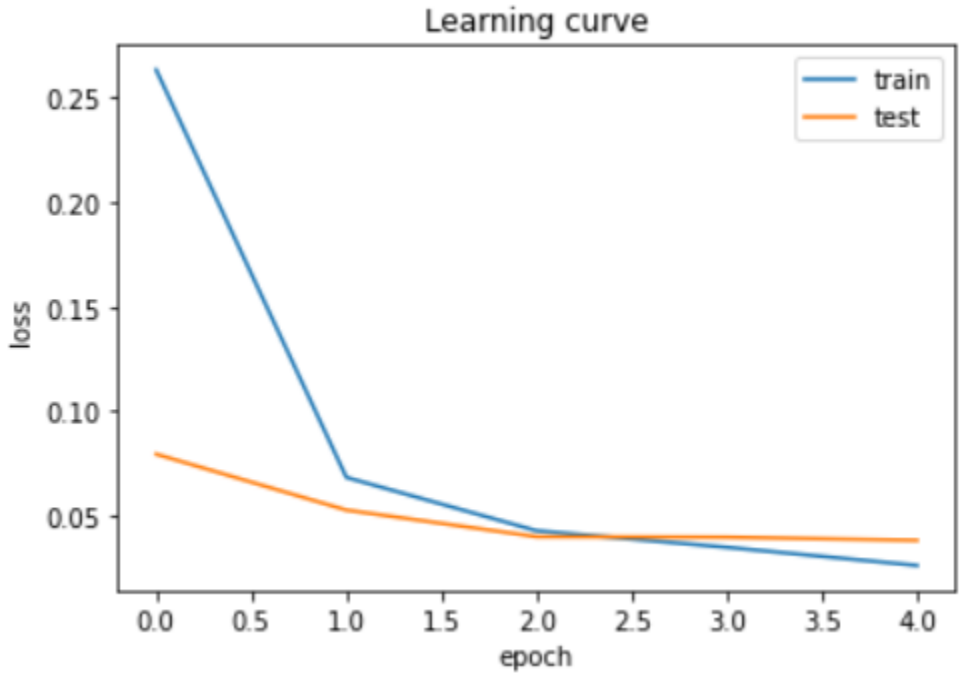

Fig 16

The architecture consists of 2 Convolution Layers with 32 filters of size 5x5 and 64 filters of size 5x5 respectively, followed by maxpooling layers after each convolution layer and finally flattened and followed by one dense hidden layer with 1024 PE's with relu activation and a output layer with 10 PE's with sigmoid activation. The loss function is categorical crossentropy used with adam optimizer.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_5 (Conv2D)	(None, 14, 14, 64)	51264
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 64)	0
flatten_2 (Flatten)	(None, 3136)	0
dense_4 (Dense)	(None, 1024)	3212288
dropout_2 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 10)	10250

Total params: 3,274,634
Trainable params: 3,274,634
Non-trainable params: 0

Fig 17

```
Test loss: 0.024818787165054527
Test accuracy: 99.22000169754028
Test error rate: 0.7799983024597168
Train loss: 0.009098877657623962
Train accuracy: 99.68400001525879
Test error rate: 0.31599998474121094
```

Fig 18

```
array([[ 976,    0,    0,    1,    0,    0,    2,    0,    1,    0],
       [   0, 1132,    0,    0,    0,    0,    1,    1,    1,    0],
       [   0,    2, 1026,    0,    0,    0,    0,    2,    2,    0],
       [   0,    0,    2, 1005,    0,    2,    0,    0,    1,    0],
       [   0,    0,    0,    0,  978,    0,    1,    1,    0,    2],
       [   0,    0,    0,    5,    0,  884,    1,    0,    0,    2],
       [   1,    1,    0,    1,    1,    6,  947,    0,    1,    0],
       [   0,    1,    4,    1,    0,    0,    0, 1019,    1,    2],
       [   3,    0,    2,    2,    1,    1,    1,    1,  959,    4],
       [   1,    0,    0,    0,    5,    3,    0,    3,    1,  996]],
      dtype=int64)
```

Fig 19

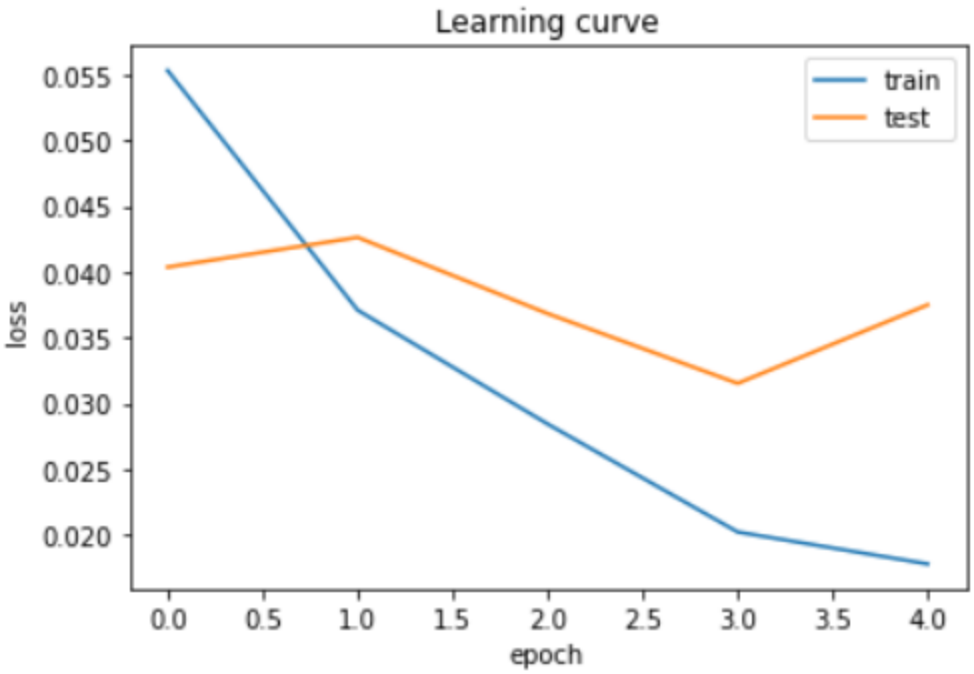

Fig 20

For the two convolutional layer architecture, a series of test with different numbers of filters and filter sizes were carried out and the best resulting outcomes have been depicted above. The best error rate captured by a two layer convolution layer network is 0.77% where in the 1st convolution layer the number of filters are 32 and filter size is 5x5 and in the second convolution layer the number of filters are 64 and the filter size is 5x5. Further adding layers did not improve the classification accuracy more than the current results and hence I have chosen 2 layers in my model for classification of the MNSIT dataset. I further run different test cases on this model to depict the learning with different hyperparameter settings.

Experiments run on the Two convolution layer network

The architecture consists of 2 Convolution Layers with 32 filters of size 5x5 and 64 filters of size 5x5 respectively, followed by maxpooling layers after each convolution layer and finally flattened and followed by one dense hidden layer with 1024 PE's with relu activation and a output layer with 10 PE's with sigmoid activation. The loss function is categorical crossentropy used with adam optimizer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              3212288
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                10250
=================================================================
Total params: 3,274,634
Trainable params: 3,274,634
Non-trainable params: 0
```

Fig 21

Test Run:1

Hyperparameter- lr-0.001, batch size=100, weight initialization=random with std dev of 0.01, Stopping Criteria-Generalization on unseen data (validation dataset)

```
Test loss: 0.024818787165054527
Test accuracy: 99.22000169754028
Test error rate: 0.7799983024597168
Train loss: 0.009098877657623962
Train accuracy: 99.68400001525879
Test error rate: 0.31599998474121094
```

Fig 22

```
array([[ 976,    0,    0,    1,    0,    0,    2,    0,    1,    0],
       [   0, 1132,    0,    0,    0,    0,    1,    1,    1,    0],
       [   0,    2, 1026,    0,    0,    0,    0,    2,    2,    0],
       [   0,    0,    2, 1005,    0,    2,    0,    0,    1,    0],
       [   0,    0,    0,    0,  978,    0,    1,    1,    0,    2],
       [   0,    0,    0,    5,    0,  884,    1,    0,    0,    2],
       [   1,    1,    0,    1,    1,    6,  947,    0,    1,    0],
       [   0,    1,    4,    1,    0,    0,    0, 1019,    1,    2],
       [   3,    0,    2,    2,    1,    1,    1,    1,  959,    4],
       [   1,    0,    0,    0,    5,    3,    0,    3,    1,  996]],
      dtype=int64)
```

Fig 23

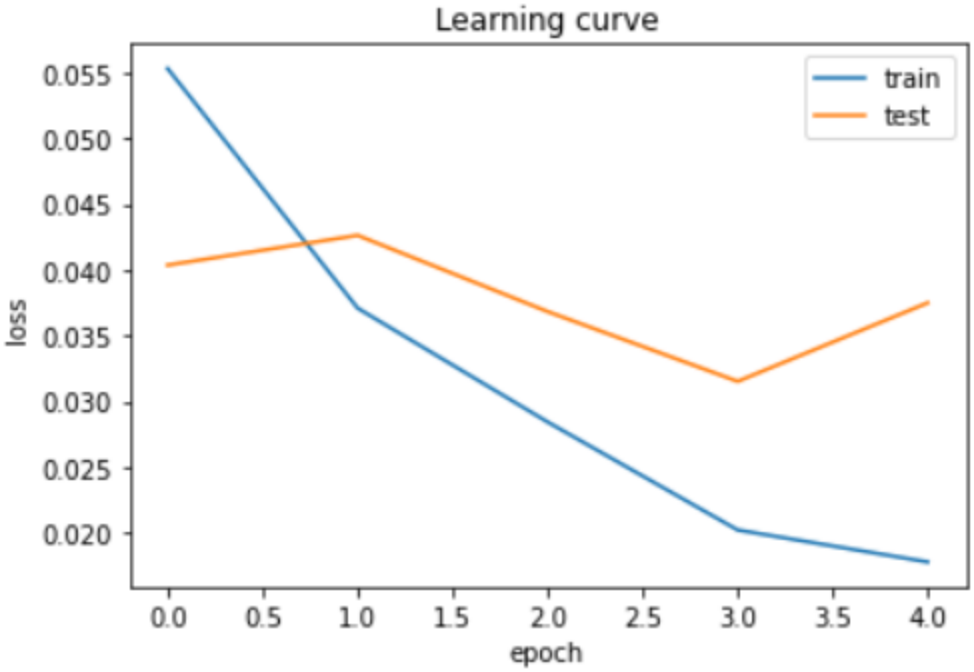

Fig 24

For the test case above, after the training the network for 5 epochs the learning curve was observed and the best generalization on the validation dataset was observed for the weights after epoch 3 and hence the results are produced for the saved weights of epoch 3.

Test Run:2

Hyperparameter- lr-0.01, batch size=100, weight initialization=random with std dev of 0.01, Stopping Criteria-Generalization on unseen data (validation dataset)

```
Test loss: 0.0770605988492258
Test accuracy: 97.83999919891357
Test error rate: 2.160000801086426
Train loss: 0.07114984861165286
Train accuracy: 97.86400198936462
Test error rate: 2.135998010635376
```

Fig 25

```
array([[ 974,    0,    3,    0,    0,    0,    1,    1,    1,    0],
       [   0, 1124,    4,    1,    0,    1,    0,    4,    0,    1],
       [   0,    0, 1023,    0,    1,    0,    0,    7,    1,    0],
       [   0,    0,    9,  989,    0,    1,    0,    6,    2,    3],
       [   0,    2,    2,    0,  938,    0,    4,    4,    2,   30],
       [   4,    0,    2,   15,    0,  856,    3,    2,    3,    7],
       [   8,    2,    3,    0,    2,    1,  941,    0,    1,    0],
       [   0,    4,    8,    0,    1,    0,    0, 1001,    1,   13],
       [   8,    0,    8,    2,    0,    1,    0,    2,  942,   11],
       [   3,    0,    1,    1,    2,    2,    0,    2,    2,  996]],
      dtype=int64)
```

Fig 26

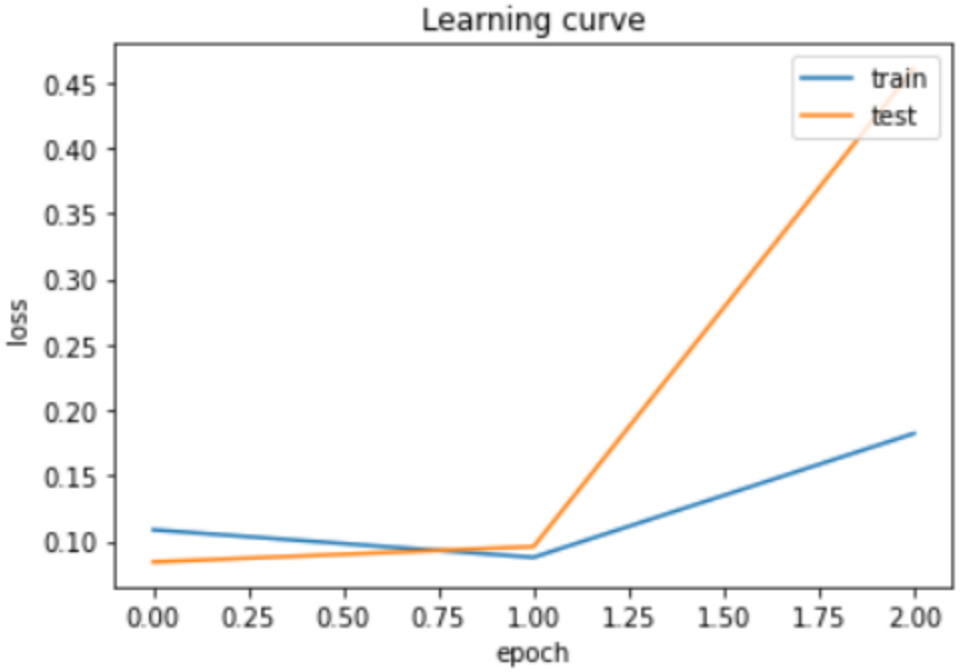

Fig 27

For the test case above, after the training the network for 5 epochs the learning curve was observed and the best generalization on the validation dataset was observed for the weights after epoch 1 and hence the results are produced for the saved weights of epoch 1. Also, from the fig 7 it can be noticed that the validation error increases drastically denoting a dip in the graph which is the effect of a very large learning rate.

From the above analysis I came to a conclusion that the learning rate cannot be further increased and hence keeping the learning rate fixed to 0.001, which gave the best results until now and further experimentation was carried out varying the batch sizes to improve classification.

Test Run:3

Hyperparameter- lr-0.001, batch size=50, weight initialization=random with std dev of 0.01, Stopping Criteria-Generalization on unseen data (validation dataset)

```
Test loss: 0.03750843368843198
Test accuracy: 98.90999794006348
Test error rate: 1.0900020599365234
Train loss: 0.014281417755584698
Train accuracy: 99.5419979095459
Train error rate: 0.45800209045410156
```

Fig 28

```
array([[ 979,    0,    0,    0,    0,    0,    0,    1,    0,    0],
       [   1, 1125,    0,    2,    0,    1,    2,    1,    3,    0],
       [   3,    1, 1018,    5,    0,    0,    1,    3,    1,    0],
       [   0,    0,    0, 1006,    0,    3,    0,    0,    1,    0],
       [   0,    0,    0,    0,  977,    0,    1,    0,    1,    3],
       [   2,    0,    0,    6,    0,  881,    1,    0,    0,    2],
       [   8,    2,    0,    1,    2,    4,  939,    0,    2,    0],
       [   0,    3,    4,    4,    0,    1,    0, 1008,    3,    5],
       [   5,    0,    0,    1,    0,    1,    0,    0,  966,    1],
       [   1,    1,    0,    0,    7,    3,    0,    1,    4,  992]],
      dtype=int64)
```

Fig 29

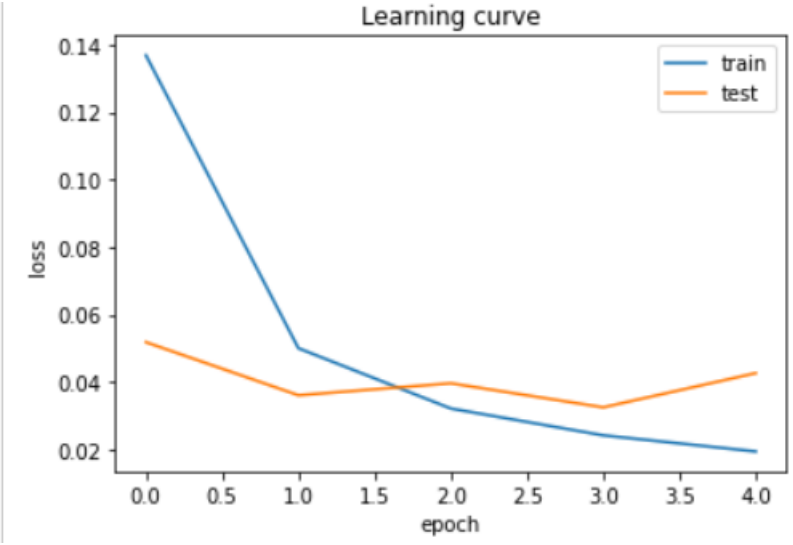

Fig 30

For the test case above, after the training the network for 5 epochs the learning curve was observed and the best generalization on the validation dataset was observed for the weights after epoch 3 and hence the results are produced for the saved weights of epoch 3.

Test Run:4
Hyperparameter- lr-0.001, batch size=150, weight initialization=random with std dev of 0.01, Stopping Criteria-Generalization on unseen data (validation dataset)

```
Test loss: 0.030488684059240042
Test accuracy: 98.989999294281
Test error rate: 1.0100007057189941
Train loss: 0.017643283761539497
Train accuracy: 99.46200251579285
Train error rate: 0.5379974842071533
```

Fig 31

```
array([[ 977,    0,    0,    0,    0,    0,    2,    1,    0,    0],
       [   0, 1132,    1,    0,    0,    0,    1,    1,    0,    0],
       [   3,    0, 1024,    0,    0,    0,    1,    4,    0,    0],
       [   0,    0,    0, 1009,    0,    0,    0,    1,    0,    0],
       [   1,    3,    1,    0,  970,    0,    1,    1,    0,    5],
       [   2,    0,    0,    9,    0,  875,    4,    1,    0,    1],
       [   5,    2,    0,    0,    1,    2,  948,    0,    0,    0],
       [   0,    2,    2,    1,    0,    0,    0, 1021,    1,    1],
       [   3,    0,    2,    1,    0,    2,    3,    2,  959,    2],
       [   2,    6,    0,    2,    3,    4,    0,    6,    2,  984]],
      dtype=int64)
```

Fig 32

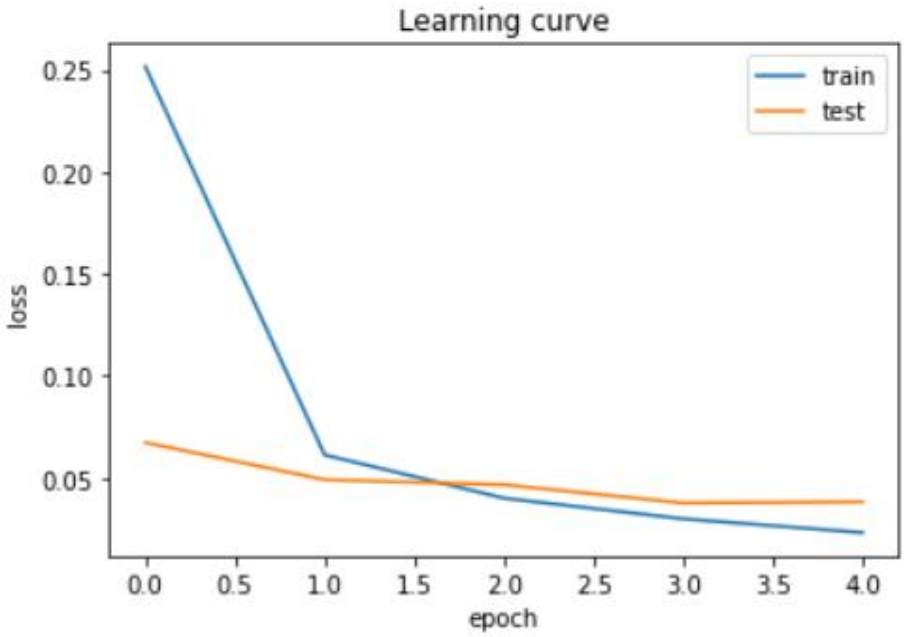

Fig 33

For the test case above, after the training the network for 5 epochs the learning curve was observed and the best generalization on the validation dataset was observed for the weights after epoch 4 and hence the results are produced for the saved weights of epoch 4. It was also noticed that as the batch size is increased the fluctuation in the validation test is reduced.

Comparison with other Architectures

The results of my model are presented below
Hyperparameter- lr-0.001, batch size=100, weight initialization=random with std dev of 0.01, Stopping Criteria-Generalization on unseen data (validation dataset)

```
array([[ 976,    0,    0,    1,    0,    0,    2,    0,    1,    0],
       [   0, 1132,    0,    0,    0,    0,    1,    1,    1,    0],
       [   0,    2, 1026,    0,    0,    0,    0,    2,    2,    0],
       [   0,    0,    2, 1005,    0,    2,    0,    0,    1,    0],
       [   0,    0,    0,    0,  978,    0,    1,    1,    0,    2],
       [   0,    0,    0,    5,    0,  884,    1,    0,    0,    2],
       [   1,    1,    0,    1,    1,    6,  947,    0,    1,    0],
       [   0,    1,    4,    1,    0,    0,    0, 1019,    1,    2],
       [   3,    0,    2,    2,    1,    1,    1,    1,  959,    4],
       [   1,    0,    0,    0,    5,    3,    0,    3,    1,  996]],
      dtype=int64)
```

Fig I- Confusion Matrix

```
Test loss: 0.024818787165054527
Test accuracy: 99.22000169754028
Test error rate: 0.7799983024597168
Train loss: 0.009098877657623962
Train accuracy: 99.68400001525879
Test error rate: 0.31599998474121094
```

Fig II- Accuracy and Loss Metrics

From the above shown metrics it can be seen that the model performs well on the test dataset but still lags when compared to the state-of-the-art. By observing the confusion matrix we can come to an conclusion that the misclassified samples are mostly the ones that visually look similar
0 is misclassified as 6 and 8 which maybe a result of the writing style differences.
7 is misclassified as 2 and 9
8 is classified as 0,3,9
9 is classified as 4
Some of the misclassification could be because of the wrong labels as well.

I) Stochastic Optimization of Plain Convolutional Neural Networks with Simple methods

In the above-mentioned research paper the architecture of the CNN comprises of 4 convolution layer each followed by Max pool layers every 2 convolution layers with a homogenous design of 3x3 and 2x2 kernel size respectively. Before each maxpooling relu activation was applied followed by dropout and further followed by 2 hidden MLP followed by a softmax layer with 10 outputs.
This architecture was trained with a learning rate=0.01 and batch size=256.
The models error rate is 0.17% with an accuracy of 99.79% .
The accuracy of my model is 99.22% and error rate=0.77% which is greater by a factor 4.5.

The results I have obtained with my model is comparatively less and this can be reasoned on different notes as follows:

- My model was trained with 50000 samples where as in training the model in the above said paper data augmentation such as rotation, shearing, shifting up and down, zooming rescale, cutouts and filling is applied which multiplies the data samples by 7 times. This results in a large dataset with all orientations and scales.
- In turn because of the large dataset additional layers can be added to increase the accuracy without the fear of overfitting.
- Also, in the model presented in the paper careful monitoring of the dropout rate and units in the dense layer was carried out and 2048 units with 0.8 dropouts produced the result.
- From the confusion matrix shown above it can be seen that a lot of the digits which appear visually similar such as [4,9], [0,6],[5,6] are misinterpreted by the classifier. This could be avoided by data augmentation such as rotation and scaling.

II) Convolutional Neural Network Committees For Handwritten Character Classification

In the above-mentioned research paper the architecture of the CNN comprises of an input layer with 29x29 neuron followed by convolution layer with 20 maps of 26x26 with filters of size 4x4 followed by a maxpooling layer with kernel size 4x4 which is followed by a convolution layer with 40 maps of 9x9 neuron followed by maxpooling layer with kernel size 3x3 followed by a fully connected layer with 150 PE's and a output layer with 10 outputs.

This architecture was trained with full online mode with a decaying learning rate(initially 0.001).

The models error rate is 0.27+0.02% or 0.27-0.02%with an accuracy of 99.73% .

The error rate of my model with comparison to the learning rate of the model in the paper is greater by a factor 2.85.

The results I have obtained with my model is comparatively less and this can be reasoned follows:

- The MNSIT dataset is replicated as 6 different dataset in which the width of the characters are normalized as 10,12,14,16,18 and 20 pixels. A committee of 5 CNNs with different initialization is trained with each of the datasets and the average of the output is considered as the final result.
- Also, the number of data samples increases in each dataset because of the elastic deformation and scaling by 15% at the beginning of every epoch. This results in extra data samples to train on.
- The above model is an ensemble model where the output is a combination of outputs from the 5 CNN committee and hence unless a sample is misclassified by all the CNNs the average of the outputs result in the correct output.
- In my model some of the samples which are misclassified could be corrected by data augmentation techniques.

III) Multi-column Deep Neural Networks for Image Classification

In the above-mentioned research paper the architecture of the DNN comprises of a convolution layer with 100 maps and 5x5 kernel size followed by max-pooling layer over the non overlapping region with kernel size 2x2 followed by a convolution layer with 100 maps with kernel size 4x4 followed by 2 hidden layer MLP with 300 PE's and 100 Pe's respectively with hyperbolic tangent activation, followed by an output layer with 10 PE's with softmax.
This architecture was trained with full online mode with a decaying learning rate(initially 0.001) till it reaches 0.00003.
The models error rate is 0.23 with an accuracy of 99.77% .
The error rate of my model with comparison to the learning rate of the model in the paper is greater by a factor 3.34
The results I have obtained with my model is comparatively less and this can be reasoned follows:

- The dataset is duplicated in 6 more datasets with normalizing the digit widths to 10,12,14,16,18 and 20. And hence data with different aspect ratios are addressed by pre-processing the images.
- The DNN have a very deep structure when compared to my architecture and hence it is potentially more prone to learn additional parameters.
- 5 columns off DNN for each dataset is trained and hence a total of 35 columns are trained and hence the average of the models is presented as the result.
- In training the model, fully online mode is adapted and the images are rotated, translated and scaled but tested only on the original images. This results is extra data samples to train on.
- In comparison to my model, this model is deeper and hence have more trainable parameters and also has extra data samples to train on and since it presents the result as the average of 5 networks it less likely for all the models to predict a particular sample wrong.

References

[1] Cireşan, D., Meier, U., Masci, J. and Schmidhuber, J., 2012. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32, pp.333-338.
[2] D. C. Ciresan, U. Meier, L. M. Gambardella and J. Schmidhuber, "Convolutional Neural Network Committees for Handwritten Character Classification," 2011 International Conference on Document Analysis and Recognition, Beijing, 2011, pp. 1135-1139, doi: 10.1109/ICDAR.2011.229.
[3] Assiri, Yahia. (2020). Stochastic Optimization of Plain Convolutional Neural Networks with Simple methods.
[4] GitHub. 2020. *Amritk10/MNIST-CNN*. [online] Available at: <https://github.com/AmritK10/MNIST-CNN> [Accessed 21 October 2020].