CS671: Deep Learning and Applications
Programming Assignment 2
Group 16

Anmol Bishnoi (B19069)
Rachita (B19258)
Anshul (B19150)

Evaluating the performance of various different optimisers

We've been given images from the mnist dataset for 5 different integers (1, 5, 6, 7, 9) to train into a classifier model. The data has been pre-split into train, val and test. The data is also shuffled in the import step itself to ensure all samples belonging to the same class do not show up together.

The target labels were created as a one hot notation for the neural network to train on since we were required to run the cross-entropy loss function.

We will be testing five different gradient descent approaches for their performance with the criteria under consideration being the number of epochs taken for convergence, the time taken for the same on an RTX 3060 GPU running CUDA_11.2

All 5 different gradient descent algorithms will be run on three different architectures to test all the different possibilities.

Finally the models with the highest validation accuracy and the smallest time for convergence will be chosen as the best model.

1. Stochastic Gradient Descent Algorithm (batch_size = 1)

Architecture (Hidden Layer1 - Hidden Layer2 - Hidden Layer3)	Epochs	Loss at convergence	Time taken for convergence
32-15-3	3	1.6098	96.42s
78-39-20	12	0.3345	393.25s
60-20-40	4	0.8381	124.58s

Table 1

As evident from Table 1, SGD takes a large amount of time even for running very few epochs taking approximately 31 seconds per epoch. This is because we are being forced to run every sample one by one which prevents us from utilising the matrix computation optimisations inbuilt into CUDA and CuDNN.

Plots for avg. training error vs epochs

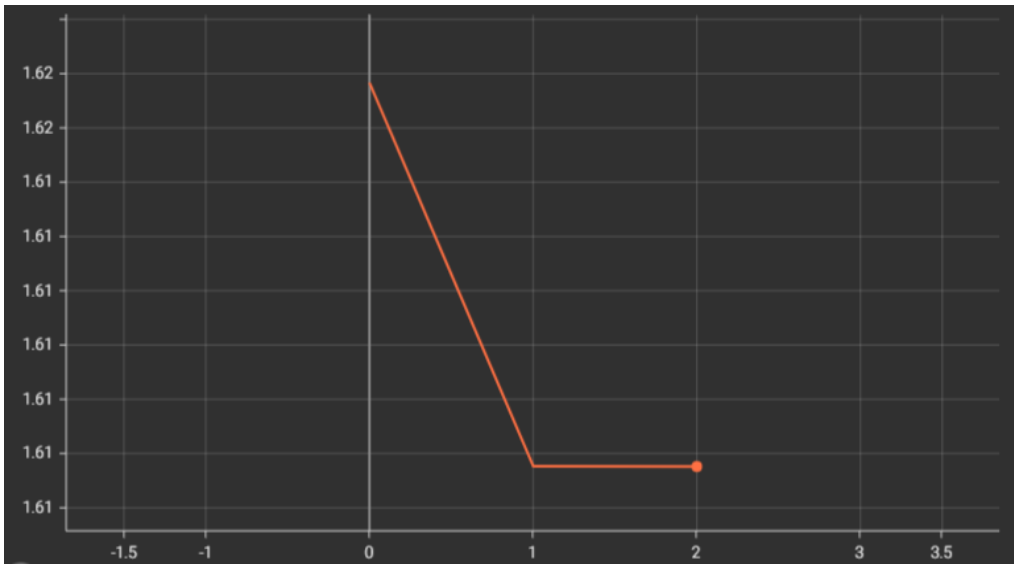


Fig. 1 Avg Training Error vs Epochs for SGD (32-15-3)

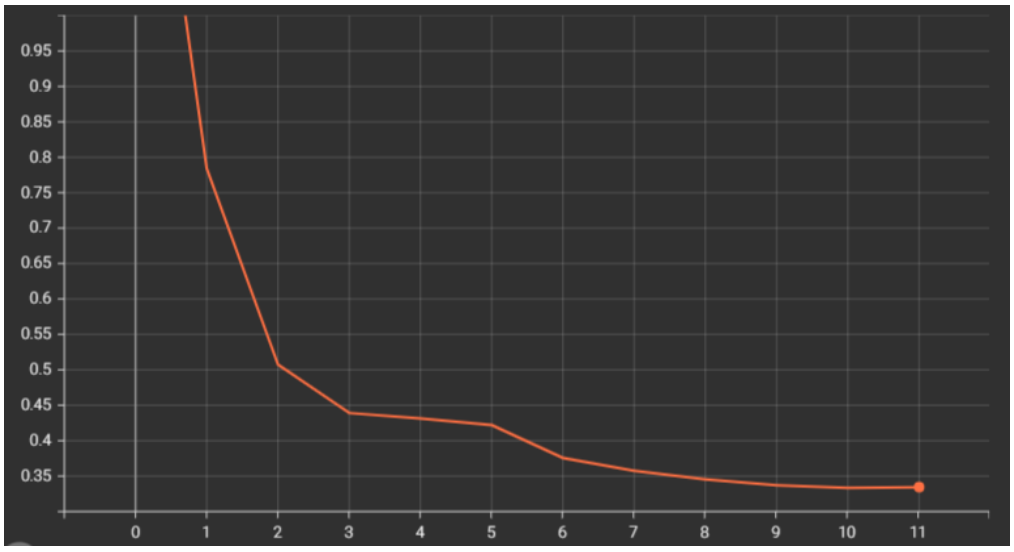


Fig. 2 Avg Training Error vs Epochs for SGD (78-39-20)

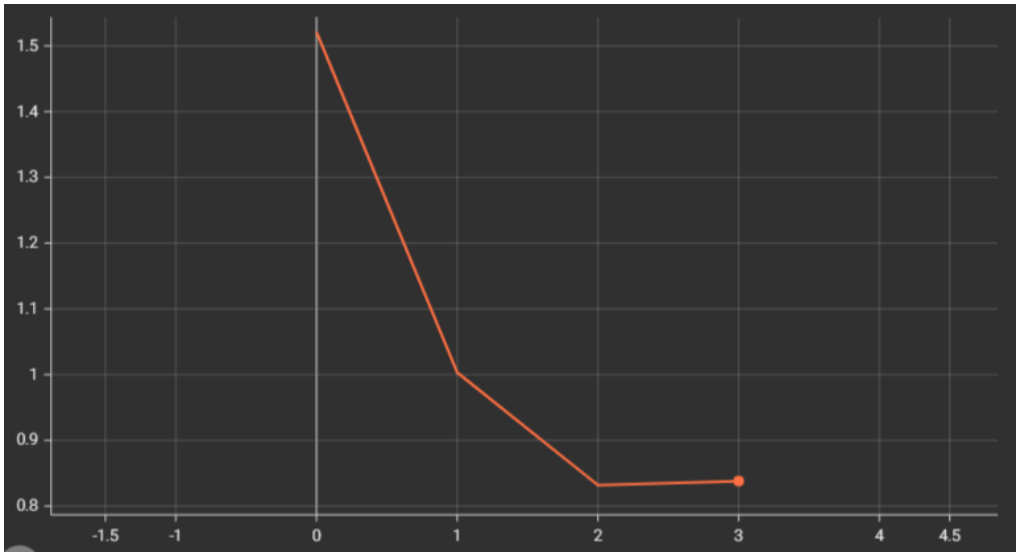


Fig. 3 Avg Training Error vs Epochs for SGD (60-20-40)

Architecture	Training Accuracy	Validation Accuracy
32-15-3	0.1979	0.2000
78-39-20	0.7961	0.7958
60-20-40	0.6329	0.9094

Table 2

The second architecture seems to have done better than both the first and third one in terms of both final validation accuracy and train loss as evident from table 2 and the graphs.
It seems that the first architecture has gotten stuck in a local minima at a training loss of about 1.6 and validation accuracy of 0.2. Training the model over and over again (thereby changing the initial weights) still led to similar results

2. Batch Gradient Descent Algorithm (batch_size = 11385)

Architecture (Hidden Layer1 - Hidden Layer2 - Hidden Layer3)	Epochs	Loss at convergence	Time taken for convergence
32-15-3	642	1.3157	84.46s
78-39-20	945	0.1725	129.56s
60-20-40	961	0.1895	105.23s

Table 3

Epochs with Batch Gradient Descent ran much faster than SGD with every epoch taking about 120ms. Comparing Table 3 with Table 1, we can see that for architecture 1, we were able to run 642 epochs of Vanilla Gradient Descent faster than just 3 epochs of SGD.

Plots for avg. training error vs epochs

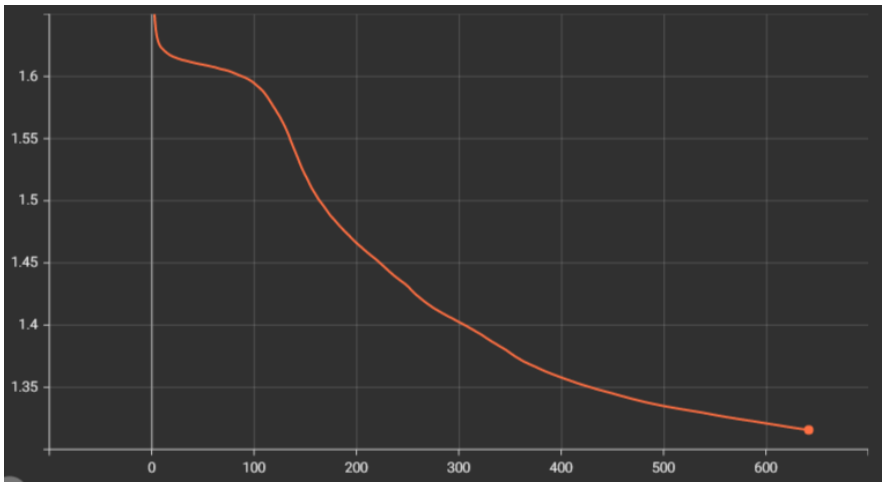


Fig. 4 Avg Training Error vs Epochs for Batch (32-15-3)

From Fig. 4, we can notice that architecture 1 yet again seemed to have struggled at the local minima at 1.6 loss but the difference over successive epochs never got low enough for it to early stop and the model eventually overcame it to get closer to the global minima.

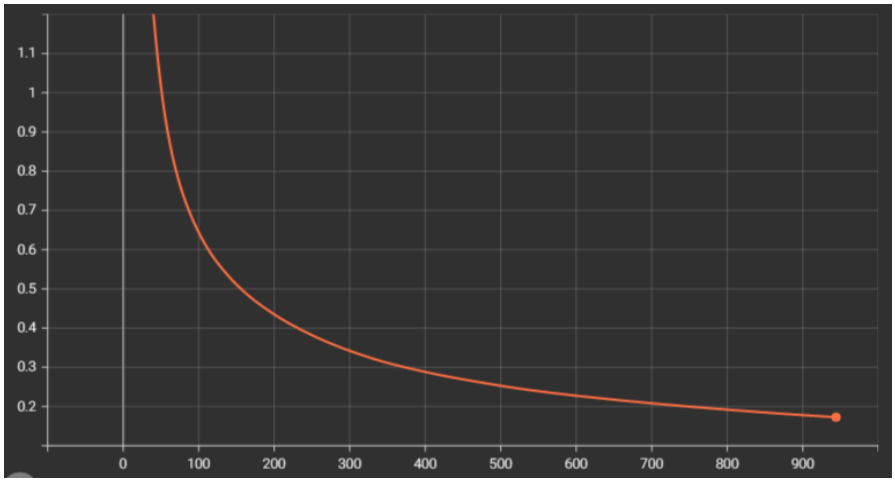


Fig. 5 Avg Training Error vs Epochs for Batch (78-39-20)

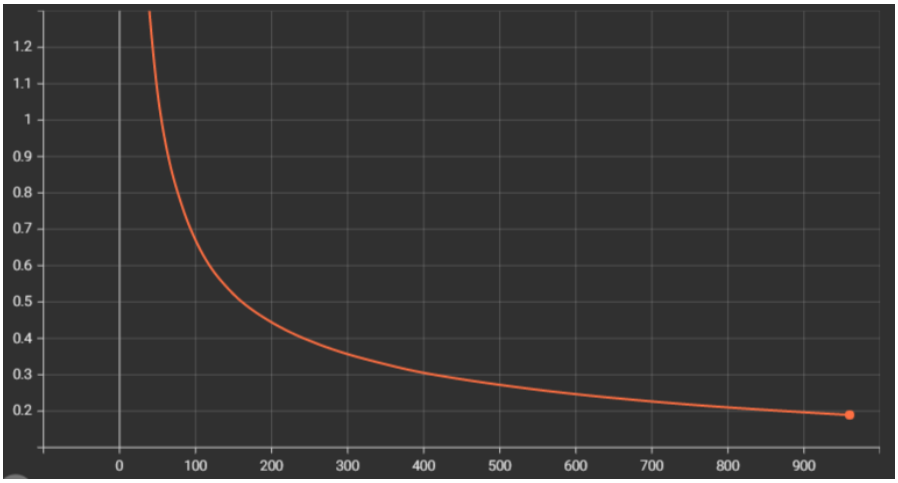


Fig. 6 Avg Training Error vs Epochs for Batch (60-20-40)

Architecture	Training Accuracy	Validation Accuracy
32-15-3	0.3871	0.3855
78-39-20	0.9421	0.9138
60-20-40	0.9400	0.9149

Table 4

Architecture 1 seemed to have performed the worst while architectures 2 and 3 had comparable performance as understood from Table 4.

3. Nesterov Accelerated Gradient Descent Algorithm (batch_size = 32)

Architecture (Hidden Layer1 - Hidden Layer2 - Hidden Layer3)	Epochs	Loss at convergence	Time taken for convergence
32-15-3	3	1.6094	4.65s
78-39-20	3	1.6094	4.58s
60-20-40	4	1.6094	6.02s

Table 5

All architectures with NAG converged at the local minima at 1.6094. Even after running the model different times (thereby changing the initial weight values), we still got the exact same results.
We also tried shuffling the data in different ways but the way our parameters for momentum, batch_size and early stopping criteria are set, we got no difference in results.

Plots for avg. training error vs epochs

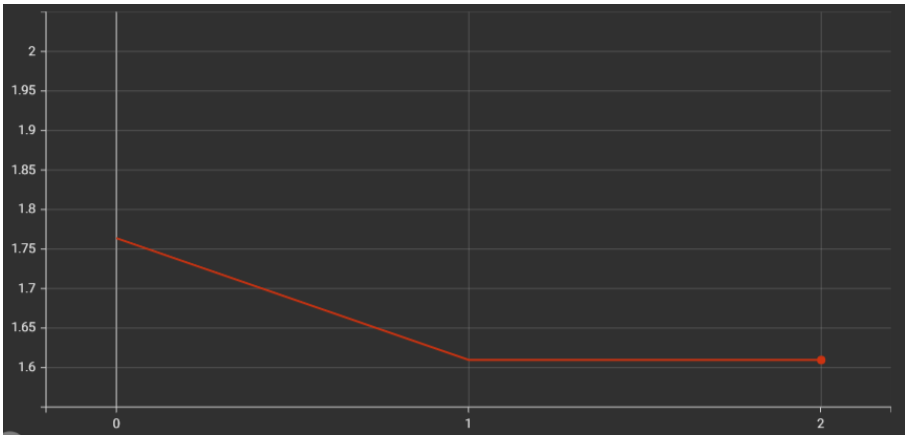


Fig. 7 Avg Training Error vs Epochs for NAG (32-15-3)

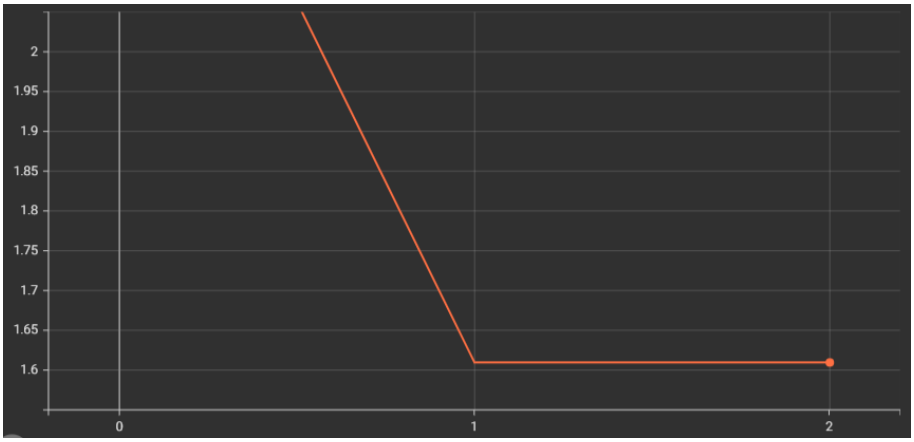


Fig. 8 Avg Training Error vs Epochs for NAG (78-39-20)

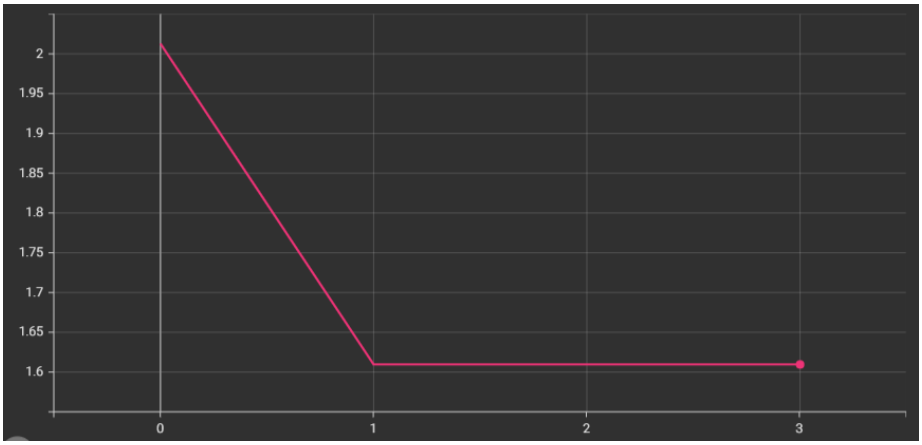


Fig. 9 Avg Training Error vs Epochs for NAG (60-20-40)

Architecture	Training Accuracy	Validation Accuracy
32-15-3	0.1931	0.2000
78-39-20	0.1953	0.2000
60-20-40	0.1921	0.2000

Table 6

All of the architectures seem to have been plagued by the problem of local minima and always converged at the point of validation accuracy being 0.2

4. RMSProp Gradient Descent Algorithm (batch_size = 1)

Architecture (Hidden Layer1 - Hidden Layer2 - Hidden Layer3)	Epochs	Loss at convergence	Time taken for convergence
32-15-3	5	1.6103	7.25s
78-39-20	3	1.5847	4.73s
60-20-40	3	1.6159	4.77s

Table 7

RMSprop seems to have the same problem of converging at the local minima. The one comparison that we can make from comparing Table 5 and Table 7 is that RMSprop takes slightly longer per epoch than NAG which can be accounted for by the additional computation of square roots, etc. being performed as a part of RMSprop.

Plots for avg. training error vs epochs

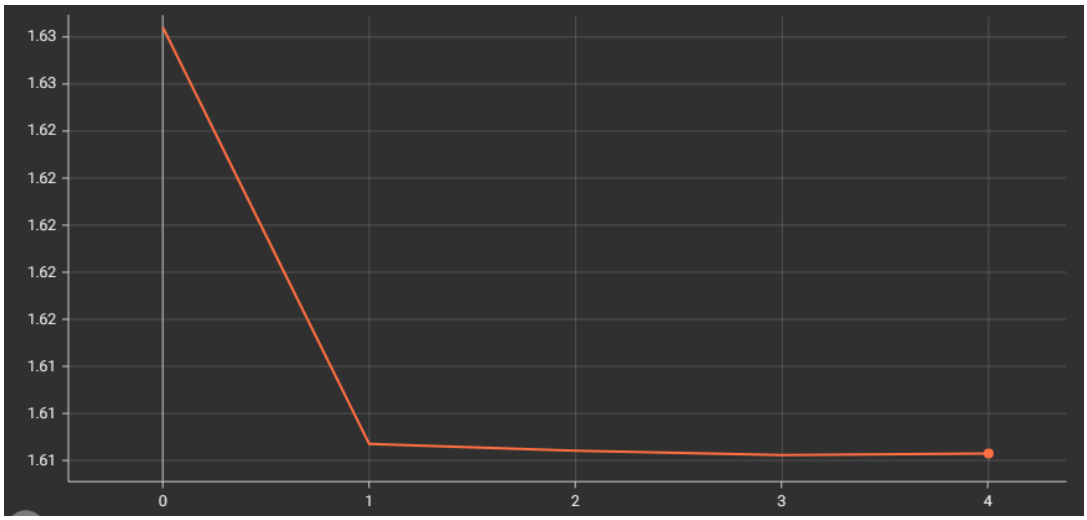


Fig. 10 Avg Training Error vs Epochs for RMSprop (32-15-3)

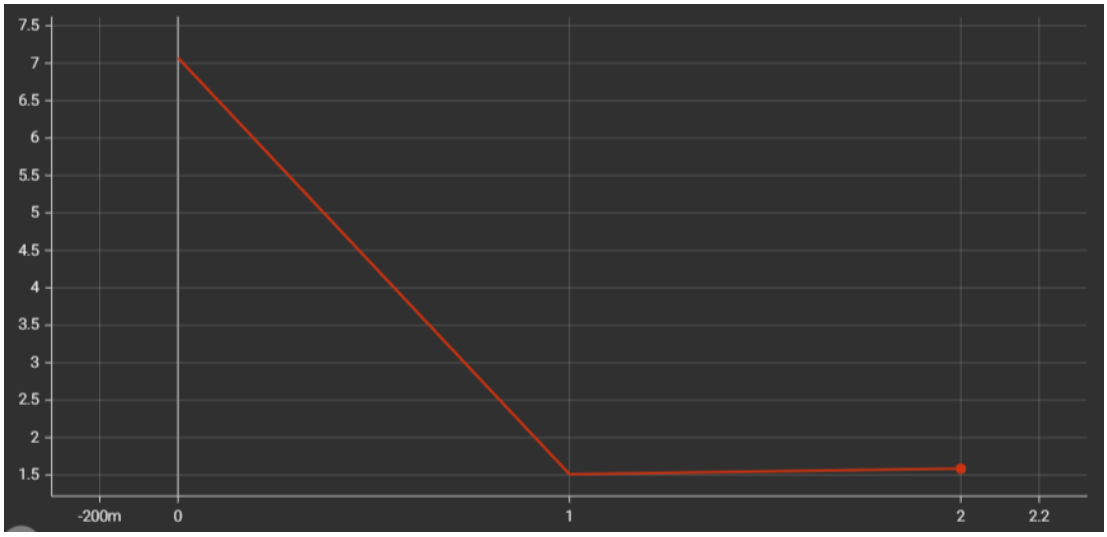


Fig. 11 Avg Training Error vs Epochs for RMSprop (78-39-20)

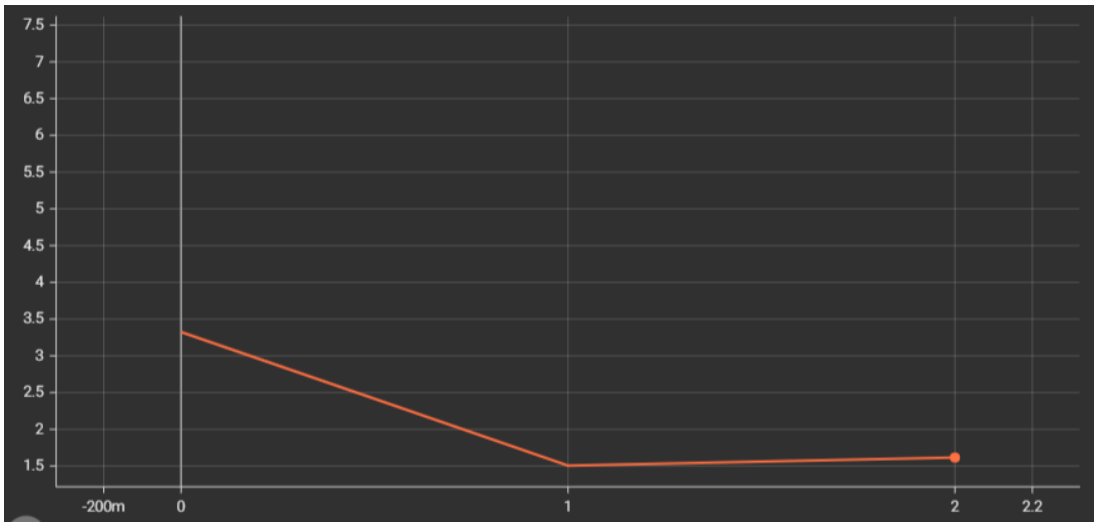


Fig. 12 Avg Training Error vs Epochs for RMSprop (60-20-40)

Architecture	Training Accuracy	Validation Accuracy
32-15-3	0.1998	0.2000
78-39-20	0.2170	0.2000
60-20-40	0.2793	0.3078

Table 8

5. Adam Gradient Descent Algorithm (batch_size = 1)

Architecture (Hidden Layer1 - Hidden Layer2 - Hidden Layer3)	Epochs	Loss at convergence	Time taken for convergence
32-15-3	28	0.1490	37.26s
78-39-20	9	0.0453	13.01s
60-20-40	6	0.0884	8.54s

Table 9

As evident from Table 9, not only did Adam successfully overcome the local minima problem in our dataset, it also was the fastest converging algorithm with an extremely small time per epoch.

Plots for avg. training error vs epochs

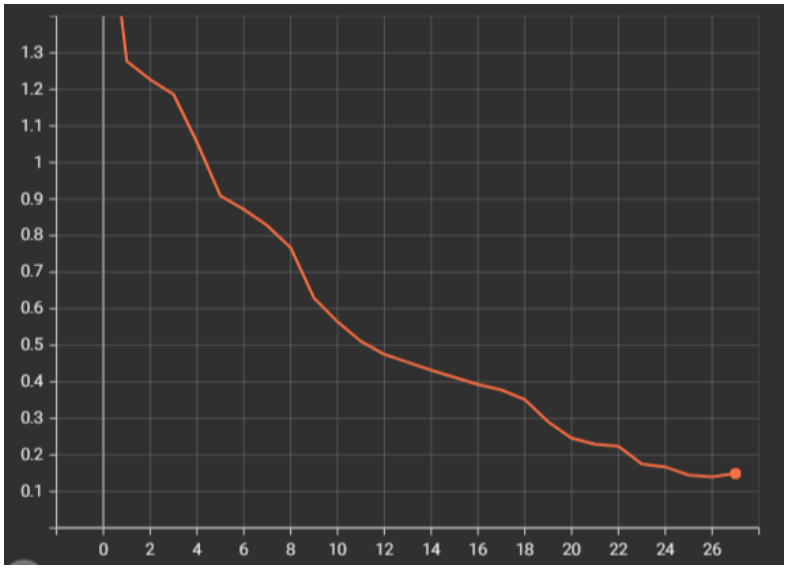


Fig. 13 Avg Training Error vs Epochs for Adam (32-15-3)

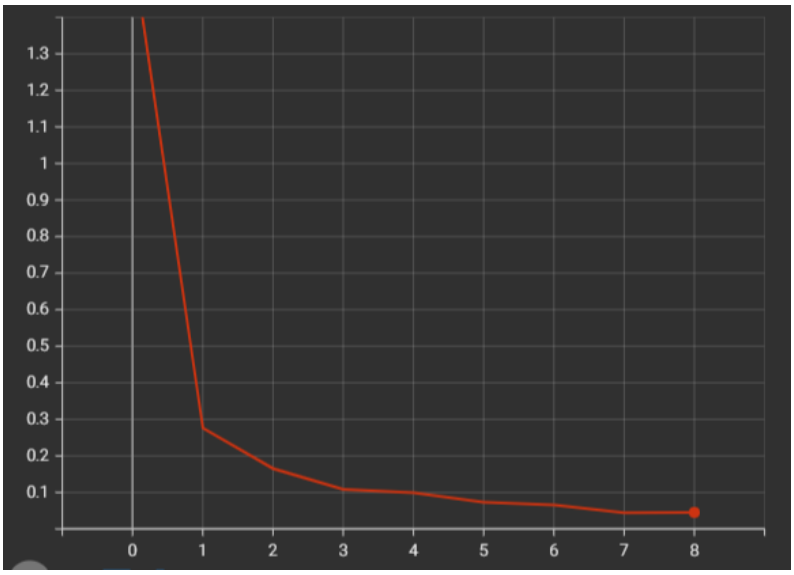


Fig. 14 Avg Training Error vs Epochs for Adam (78-39-20)

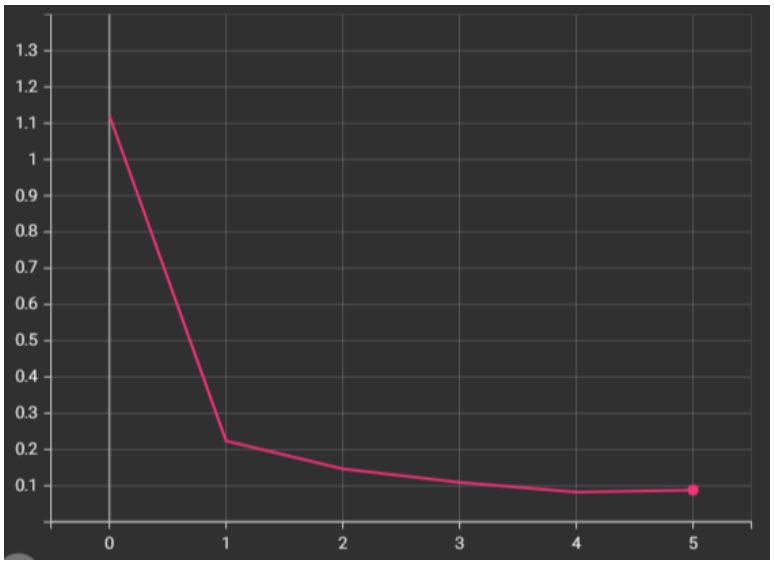


Fig. 15 Avg Training Error vs Epochs for Adam (60-20-40)

Architecture	Training Accuracy	Validation Accuracy
32-15-3	0.9745	0.9655
78-39-20	0.9875	0.9755
60-20-40	0.9744	0.9652

Table 10

Best Model

Model	Validation Accuracy	Epochs (Time) for convergence
SGD 78-39-20	0.7958	12 (393.25s)
Batch 60-20-40	0.9149	961 (105.23s)
NAG 78-39-20	0.2000	3 (4.58s)
RMSprop 60-20-40	0.3078	3 (4.77s)
Adam 78-39-20	0.9755	9 (13.01s)

Table 11

Therefore, the best of the best architecture after considering both the validation accuracy and the epochs (time) taken for convergence, Adam 78-39-20 wins by a small margin over Batch 60-20-40 whereas the rest mostly got stuck at the point of local minima.

For the best model, statistics are as follows:

Train Accuracy - *0.991*
Test Accuracy - *0.972*

Confusion Matrix for Train:

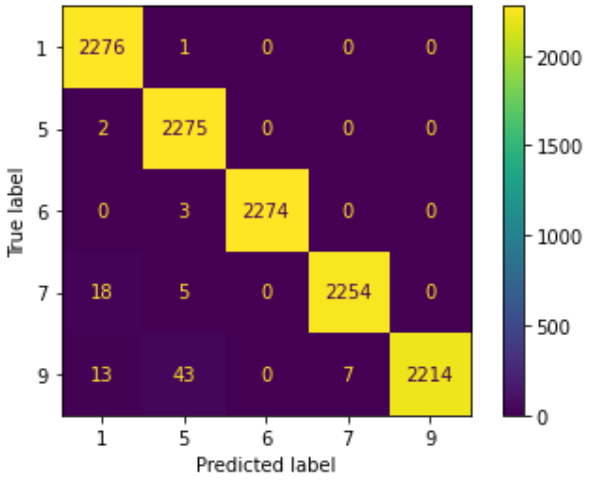


Fig. 16

Confusion Matrix for Test:

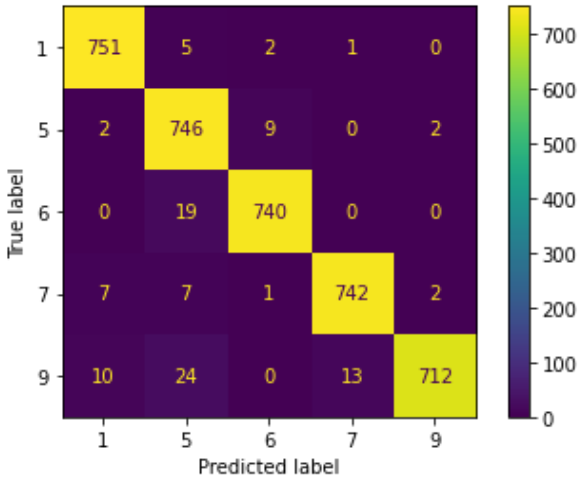


Fig. 17

Tasks based on Autoencoders

Here, we are supposed to test the performance of autoencoders with 1 or 3 hidden layers by also varying the number of nodes in each hidden layer. The dataset being encoded here is the same subset of mnist dataset as used in the previous question.

Choosing the best model simply based on the reconstruction error won't be a good metric since, as we increase the number of nodes in hidden representation, the amount of information being conveyed before decoding will also increase thus reducing reconstruction error as we increase the number of nodes in hidden representation.

To overcome this problem, we will be fitting the hidden representation from each of the different experiments into the best model as obtained from question 1 (Adam 78-39-20) and evaluate the classification accuracy on the validation set.

All data before giving into the autoencoder is normalized to 0-1 using min-max normalization.

One Hidden Layer

No. of hidden nodes in autoencoder	Training Reconstruction Error (mse)	Validation Reconstruction Error (mse)	No. of hidden nodes in Adam Classifier	Classification Accuracy on Validation Set (Crossentropy)
157	0.0037	0.0038	32-15-3	0.9673
			78-39-20	0.9697
			60-20-40	0.9657
78	0.0071	0.0073	32-15-3	0.9634
			78-39-20	0.9700
			60-20-40	0.9750
32	0.0145	0.0147	32-15-3	0.9650
			78-39-20	0.9797
			60-20-40	0.9684
16	0.0223	0.0227	32-15-3	0.7784
			78-39-20	0.9710
			60-20-40	0.9684

Table 12

From Table 12, we can further confirm that the Adam 78-39-20 classifier performs the best where it does better than the other two alternatives run for the same autoencoder architecture.

Among the different autoencoder architectures, the best average and overall classification accuracy comes for the single layer autoencoder with 32 dimensions in the hidden representation.

Actual Images vs Reconstructed Images

Fig. 18 Autoencoder with 157 hidden nodes

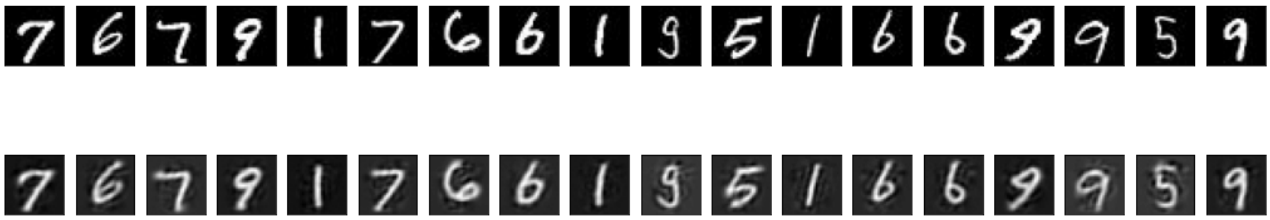


Fig. 19 Autoencoder with 78 hidden nodes

Fig. 20 Autoencoder with 32 hidden nodes

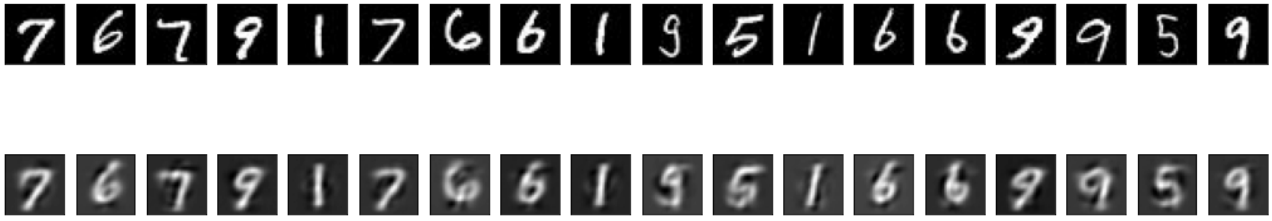


Fig. 21 Autoencoder with 16 hidden nodes

Three Hidden Layers

No. of hidden nodes in autoencoder	Training Reconstruction Error (mse)	Validation Reconstruction Error (mse)	No. of hidden nodes in Adam Classifier	Classification Accuracy on Validation Set (Crossentropy)
32-16-32			32-15-3	0.9592
	0.0194	0.0199	78-39-20	0.9623
			60-20-40	0.9679
256-16-256			32-15-3	0.9547
	0.0135	0.0142	78-39-20	0.9781
			60-20-40	0.9744
128-16-128			32-15-3	0.9549
	0.0148	0.0154	78-39-20	0.9705
			60-20-40	0.9755
128-32-128			32-15-3	0.9605
	0.0118	0.0122	78-39-20	0.9763
			60-20-40	0.9766
256-32-256			32-15-3	0.9621
	0.0102	0.0107	78-39-20	0.9742
			60-20-40	0.9744
256-128-256			32-15-3	0.9642
	0.0065	0.0068	78-39-20	0.9789
			60-20-40	0.9747
128-64-128			32-15-3	0.9634
	0.0089	0.0093	78-39-20	0.9681

Table 13

From Table 13, we can further confirm that the Adam 78-39-20 classifier performs the best where it does better than the other two alternatives run for the same autoencoder architecture.

Among the different autoencoder architectures, the best average and overall classification accuracy comes for the three layer autoencoder with 256-128-256 dimensions in the hidden representation.

Actual Images vs Reconstructed Images

Fig. 22 Autoencoder with 32-16-32 hidden nodes

Fig. 23 Autoencoder with 256-16-256 hidden nodes

Fig. 24 Autoencoder with 128-16-128 hidden nodes

Fig. 25 Autoencoder with 128-32-128 hidden nodes

Fig. 26 Autoencoder with 256-32-256 hidden nodes

Fig. 27 Autoencoder with 256-128-256 hidden nodes

Fig. 28 Autoencoder with 128-64-128 hidden nodes

Best Architecture Results

One Hidden Layer with 32 hidden nodes (+ Adam 78-39-20)

Test Reconstruction Error (mse) = 0.0148

Average Training Reconstruction Error vs Epochs

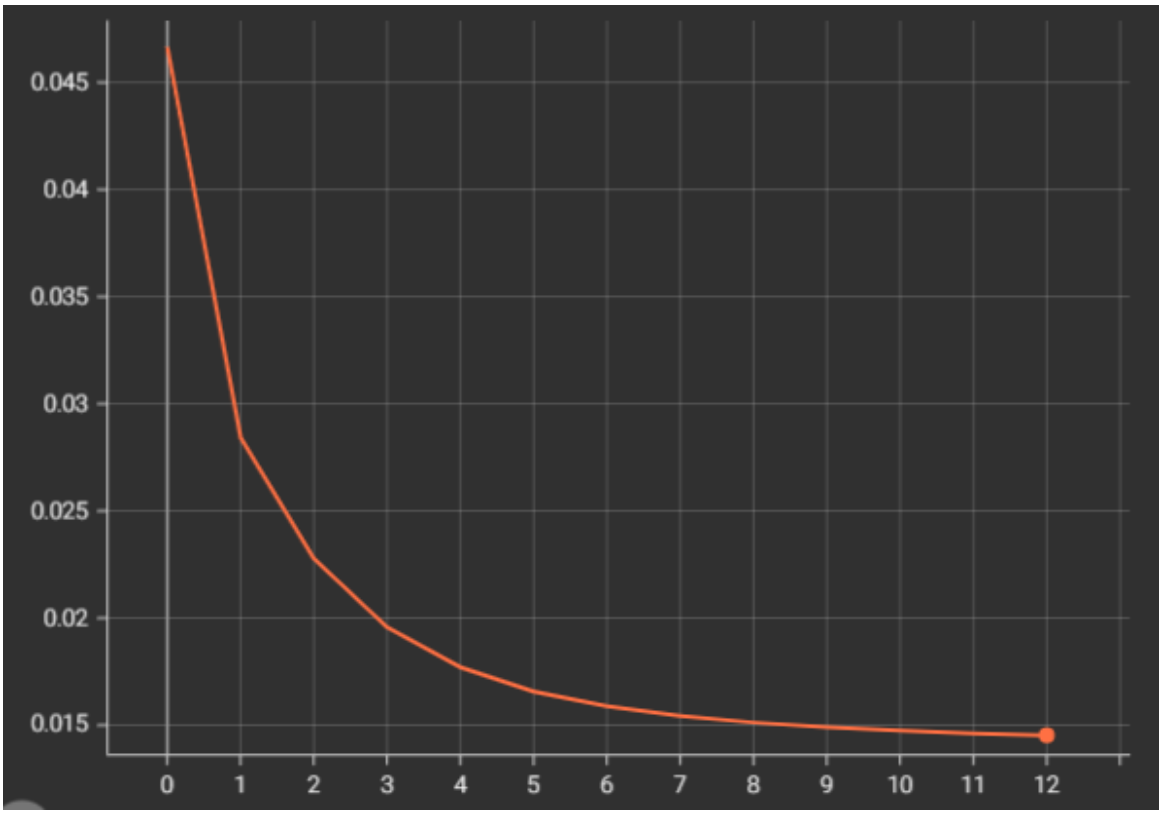


Fig. 26 mse vs epochs for 32 node Autoencoder

Accuracy Scores and Confusion Matrices for Best FCNN (Adam 78-39-20)

Train Accuracy - *0.987*
Val Accuracy - *0.975*
Test Accuracy - *0.973*

Confusion Matrix for Train:

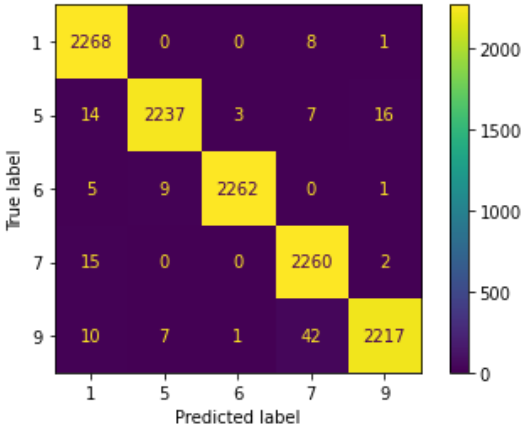


Fig.27

Confusion Matrix for Val:

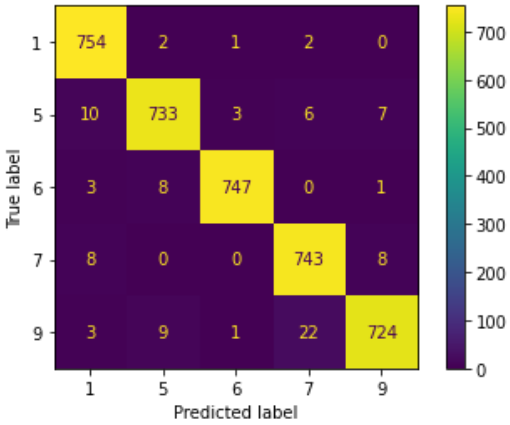


Fig.28

Confusion Matrix for Test:

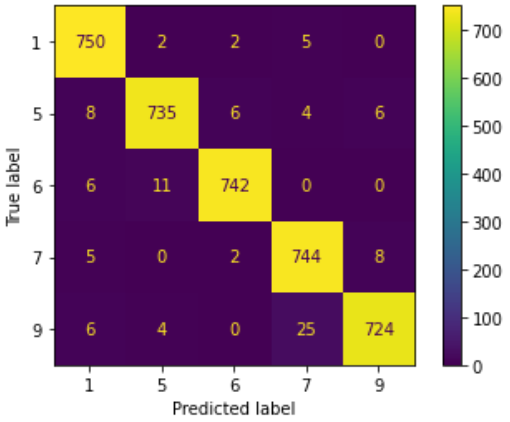
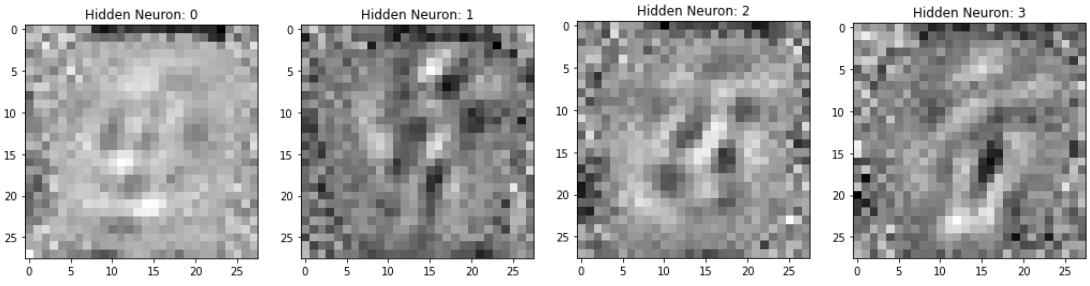
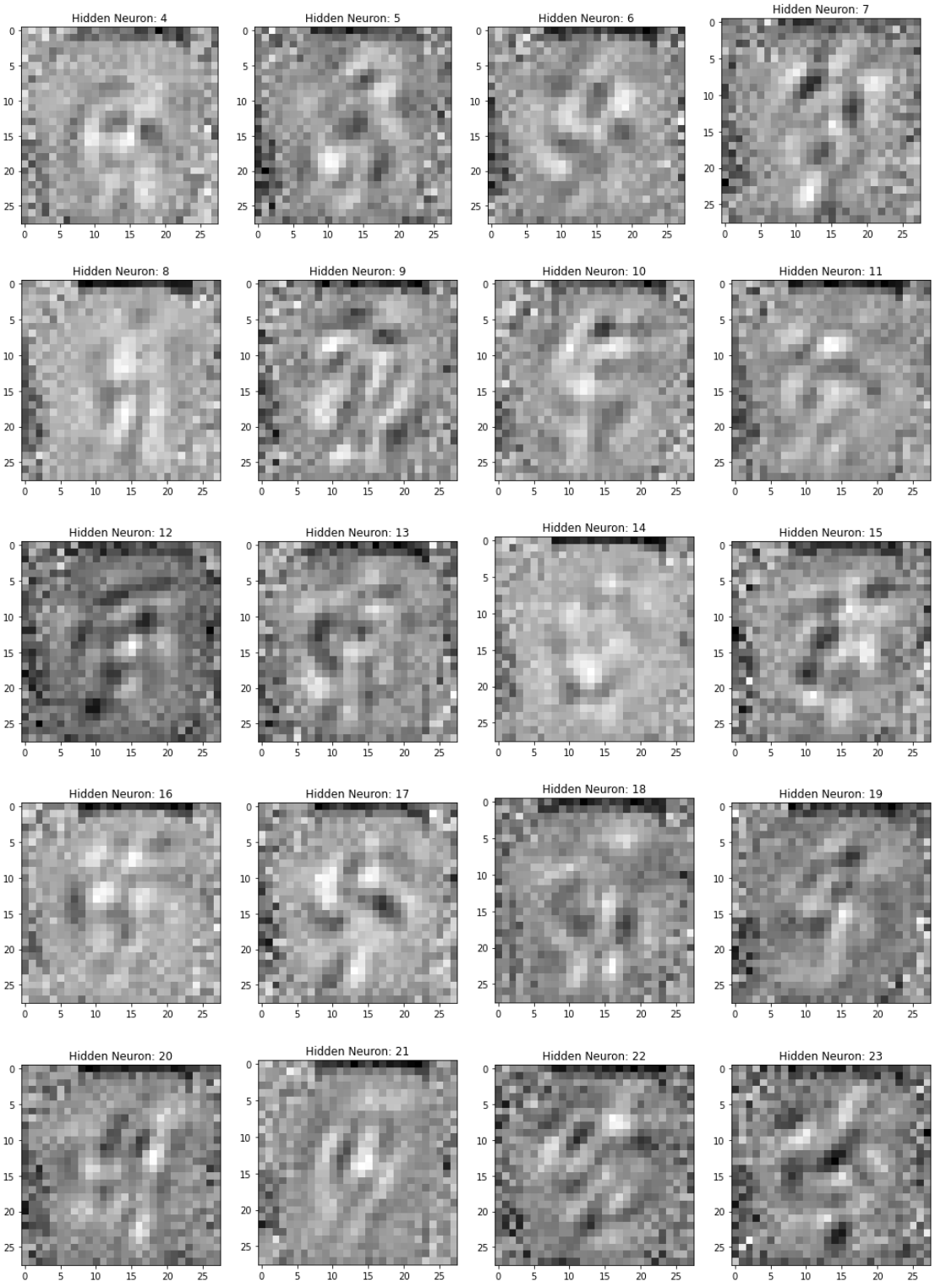
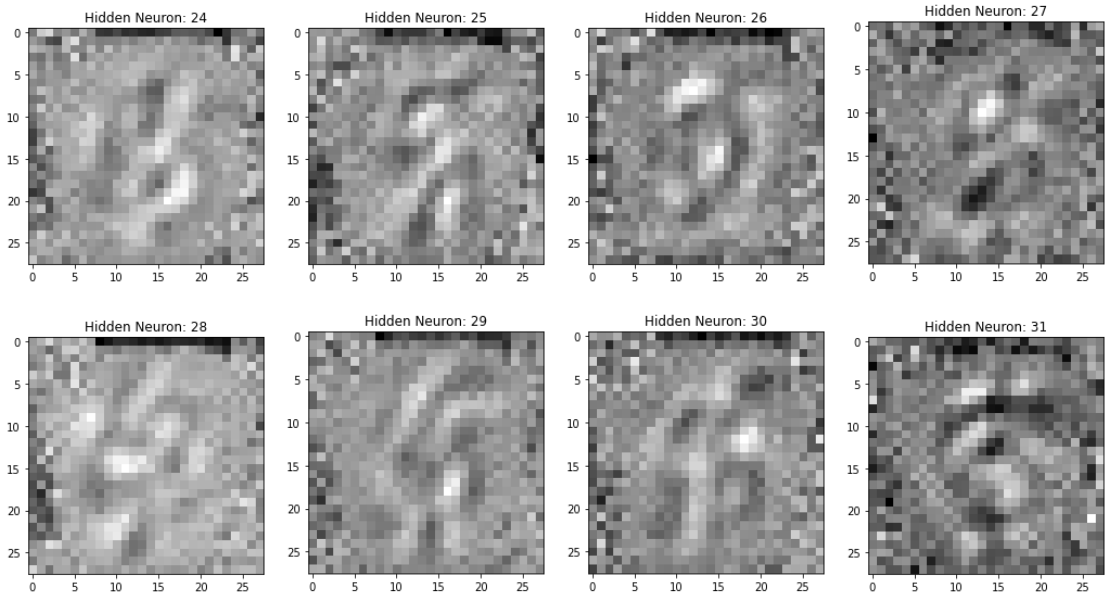


Fig.29

Weight Visualisation




Hidden Neuron: 4
Hidden Neuron: 5
Hidden Neuron: 6
Hidden Neuron: 7
Hidden Neuron: 8
Hidden Neuron: 9
Hidden Neuron: 10
Hidden Neuron: 11
Hidden Neuron: 12
Hidden Neuron: 13
Hidden Neuron: 14
Hidden Neuron: 15
Hidden Neuron: 16
Hidden Neuron: 17
Hidden Neuron: 18
Hidden Neuron: 19
Hidden Neuron: 20
Hidden Neuron: 21
Hidden Neuron: 22
Hidden Neuron: 23



Three hidden layers 256-128-256 (+ Adam 78-39-20)

Test Reconstruction Error (mse) = 0.0065

Input Images vs Reconstructed Images

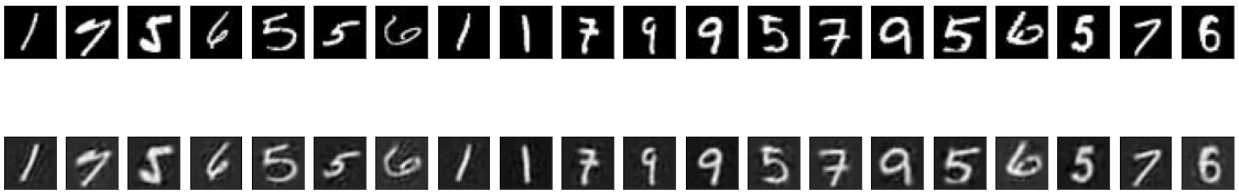


Fig. 30

Average Training Reconstruction Error vs Epochs

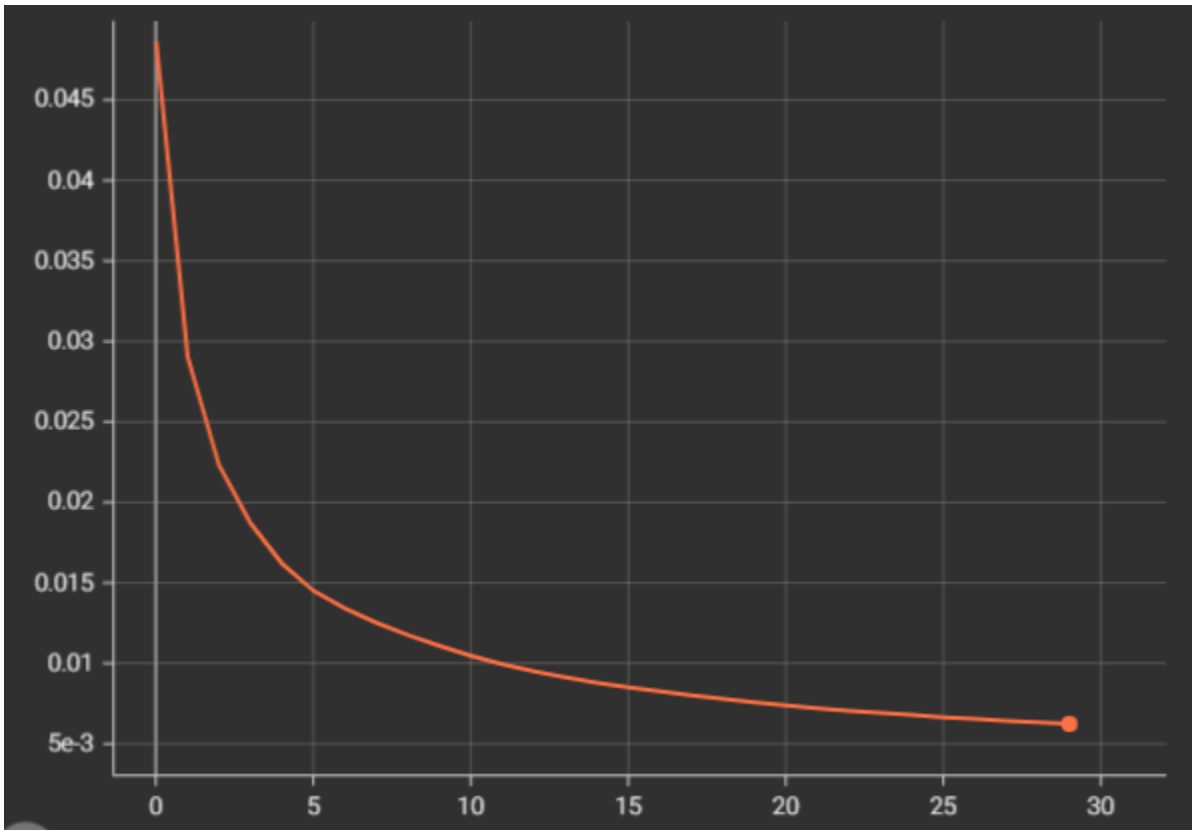


Fig. 31 mse vs epochs for 256-128-256 node Autoencoder

Accuracy Scores for Best architecture:

Train Accuracy - *0.978*
Val Accuracy - *0.972*
Test Accuracy - *0.967*

Confusion Matrix for Train Data:

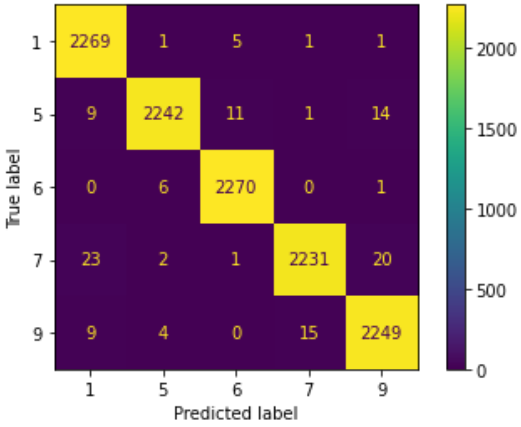


Fig.32

Confusion Matrix for Validation Data

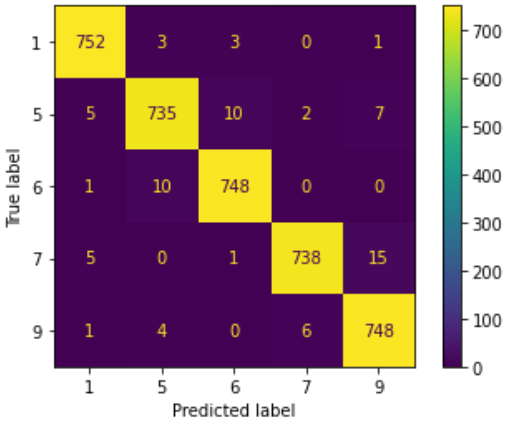


Fig.33

Confusion Matrix for Test Data

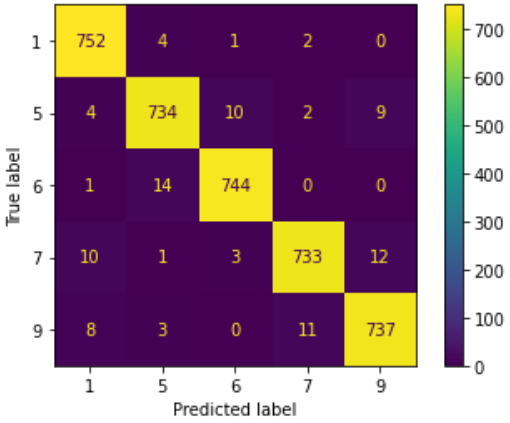
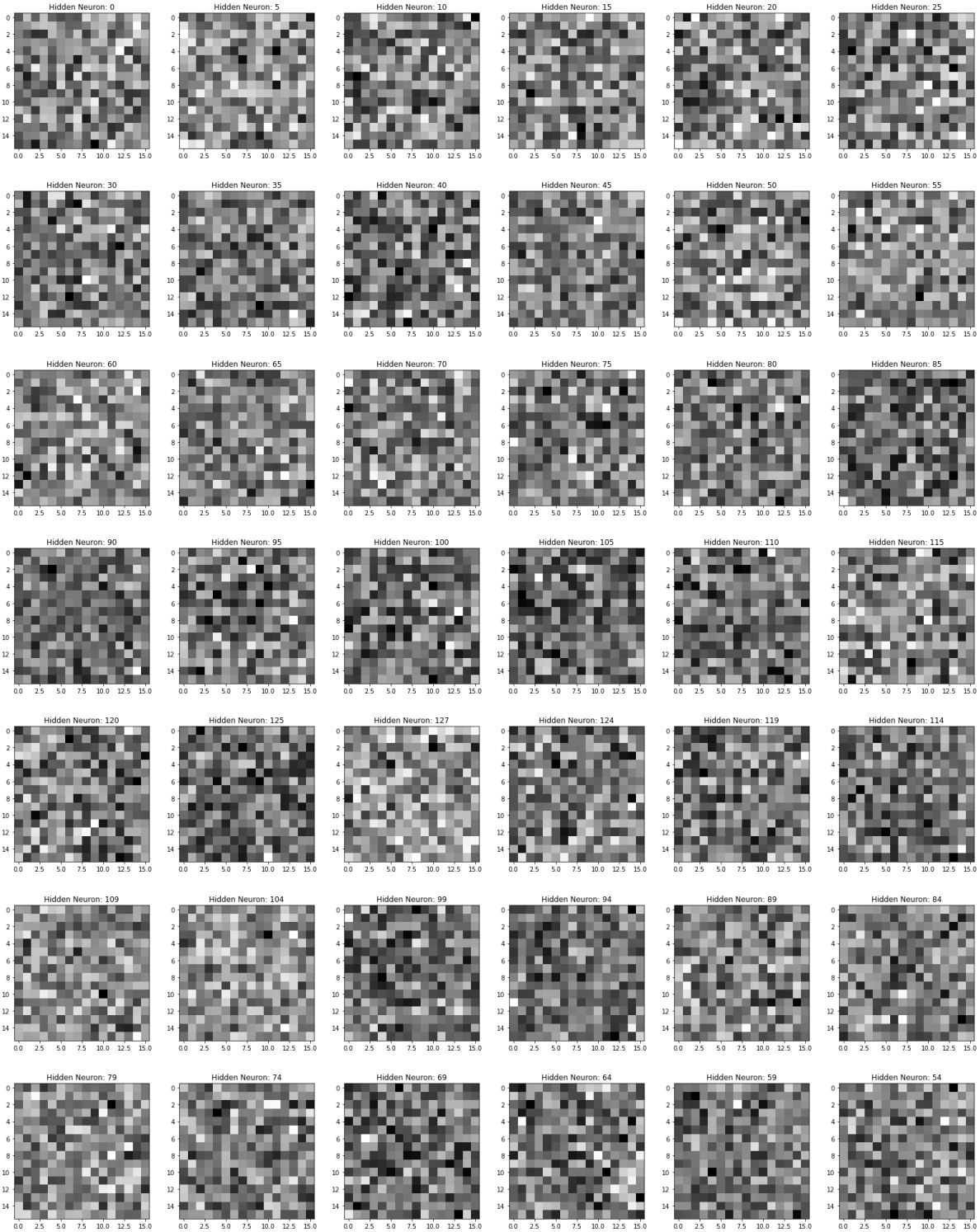


Fig.34

Weight Visualisation

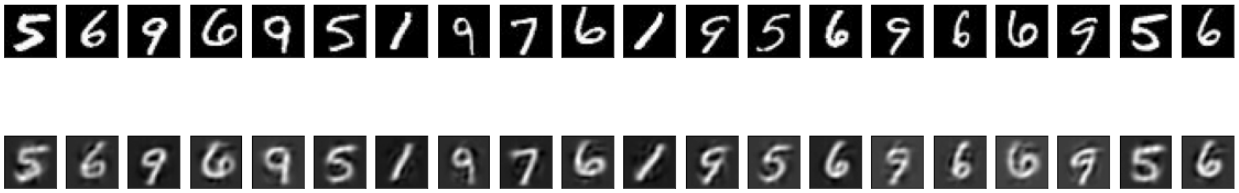


Denoising Encoder

Noise is randomly added to the examples during training with a probability of 20%/40%. This also means that an example that was made noisy in a certain epoch might not be noisy in the next or the previous epoch.

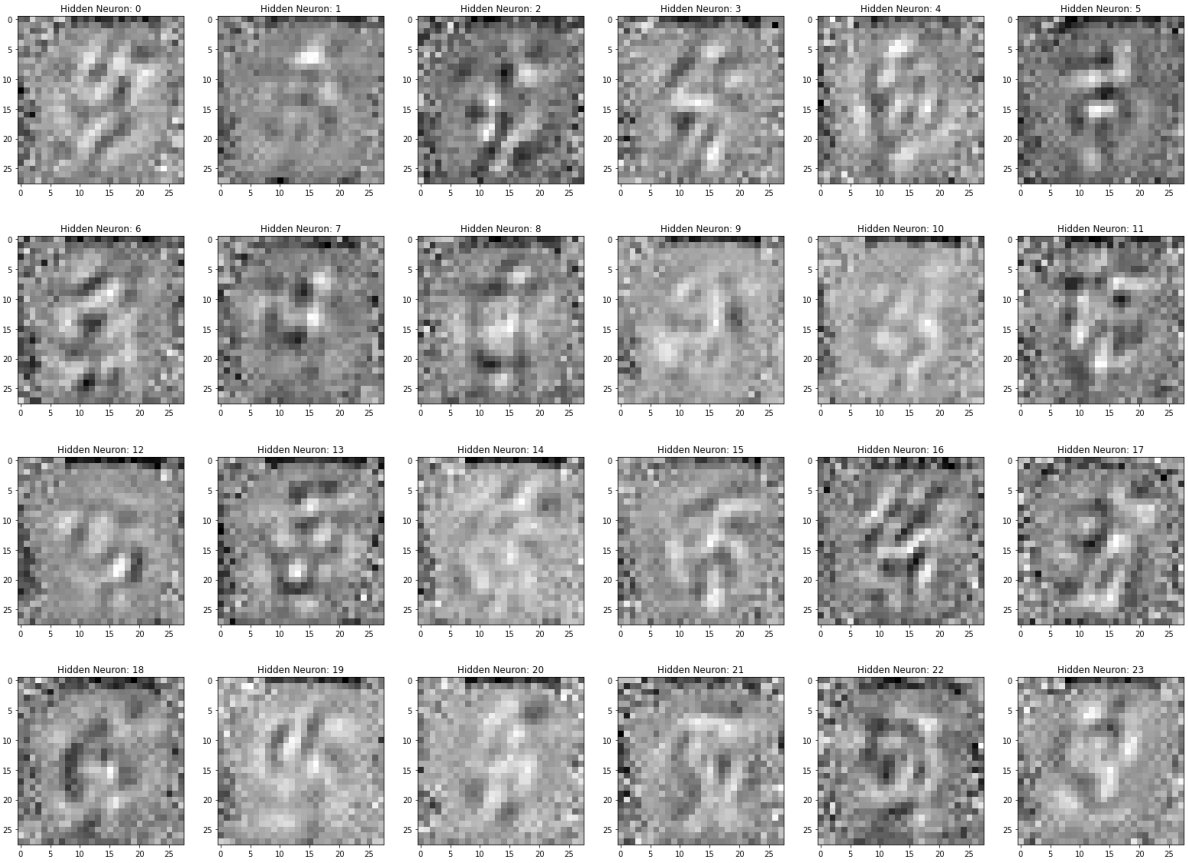
20% Noise

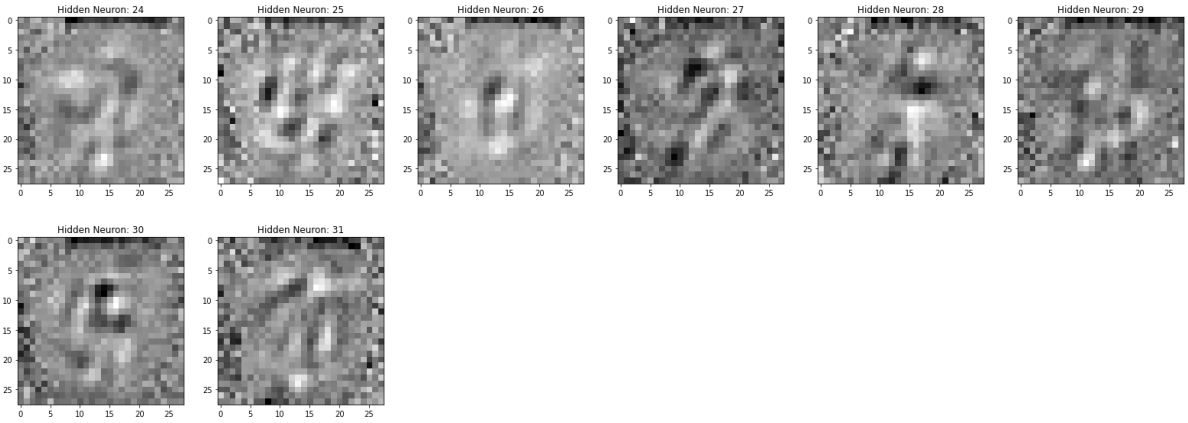
Input vs Reconstructed



Classification Accuracy = 0.9804

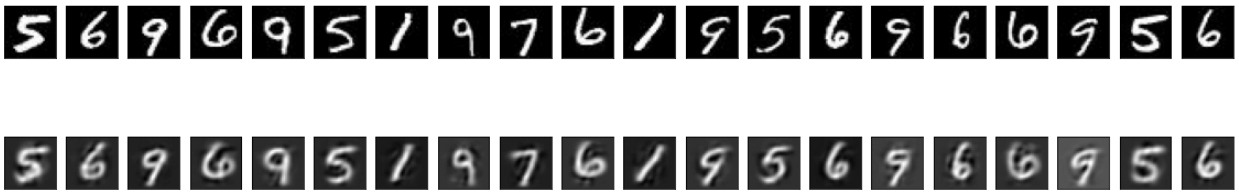
Weight Visualisation





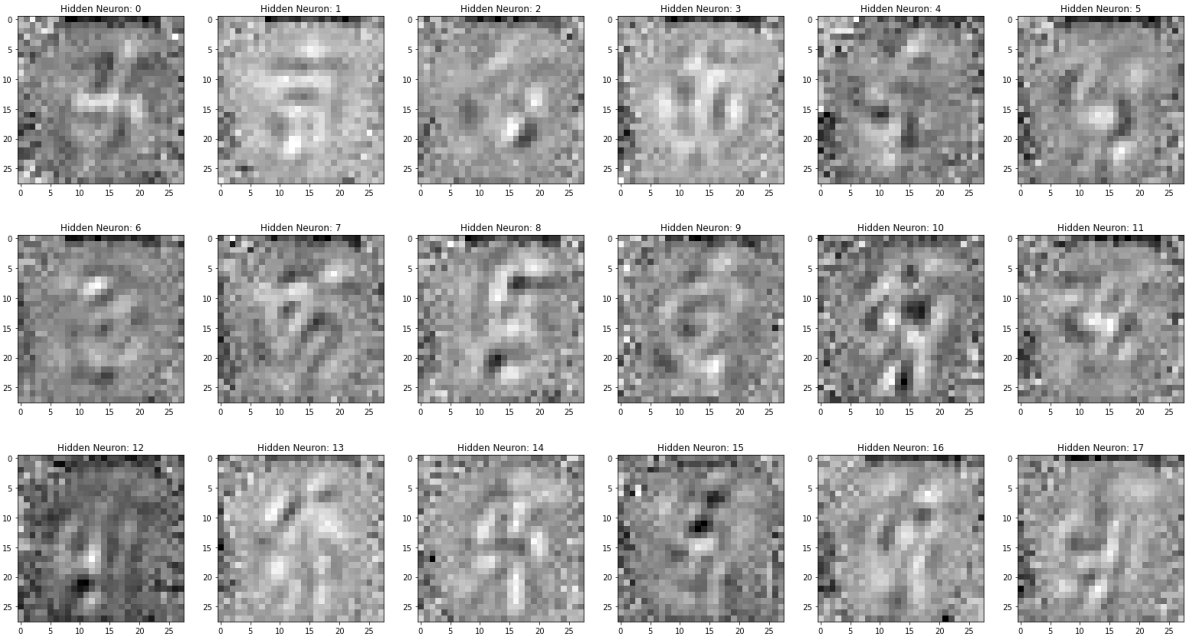
40% Noise

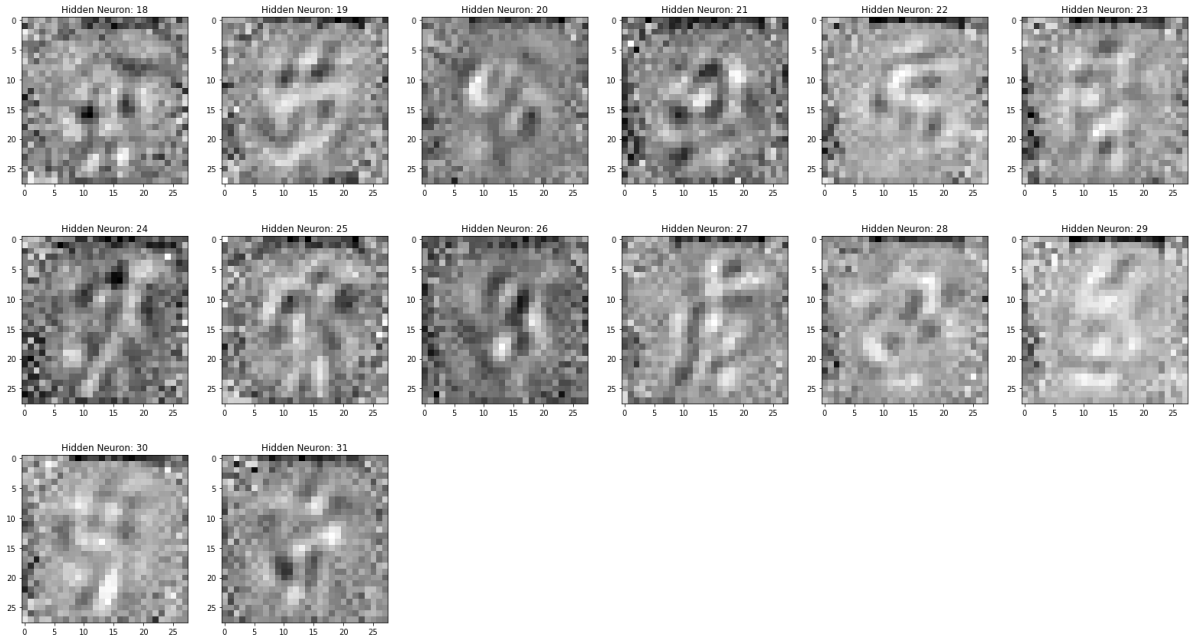
Input vs Reconstructed



Classification Accuracy = 0.9866

Weight Visualisation





From the experiments with our denoising encoders, we can clearly see that the classification accuracy results improved as we added noise to our inputs. Referring to Table 12, Vanilla autoencoder gave us a classification accuracy of 0.9797 while 20% denoising autoencoder and 40% denoising autoencoder gave classification accuracies of 0.9804 and 0.9866 respectively.

This confirms our understanding that the vanilla autoencoder was a slightly overfit model which was also learning the noise in the data while the denoising autoencoder no longer has the ability to perfectly match input to output since they are different to each other with a probability of 20(40)%.

Hence, the denoising autoencoder is forced to learn the characteristics of the data rather than the data itself which in turn leads to a more in-depth understanding of the data and hence a better classification model.