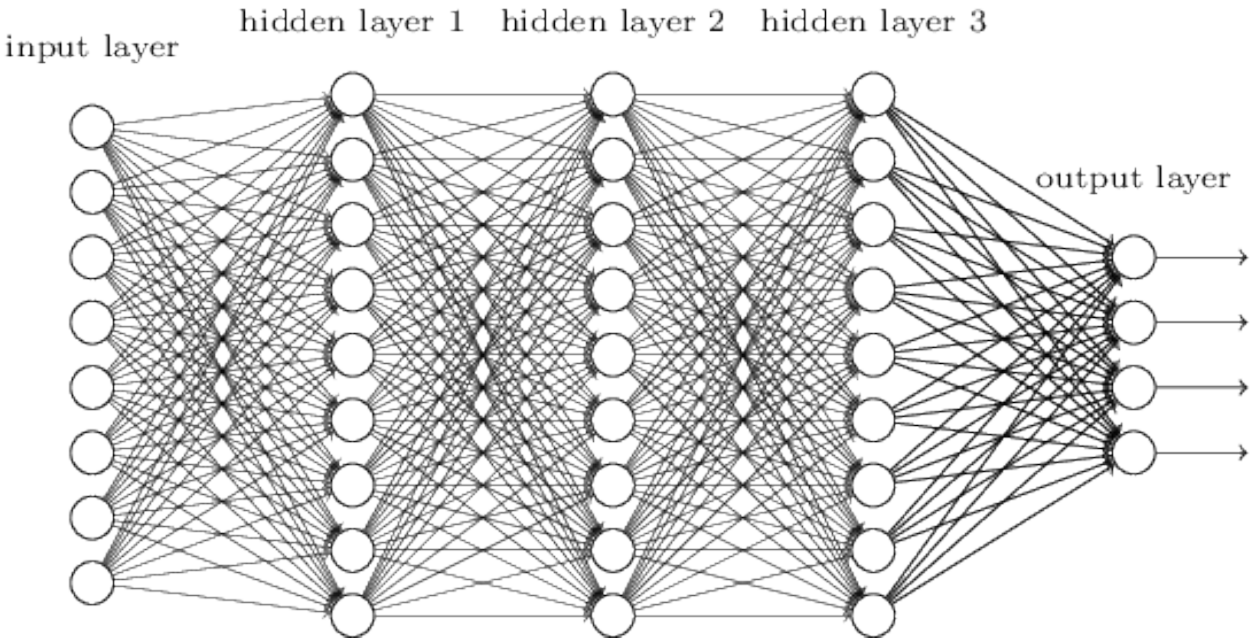
Deep Learning & Applications

Optimizers • Autoencoder

18 April 2022

CONTENTS

Fully Connected Neural Network (FCNN)



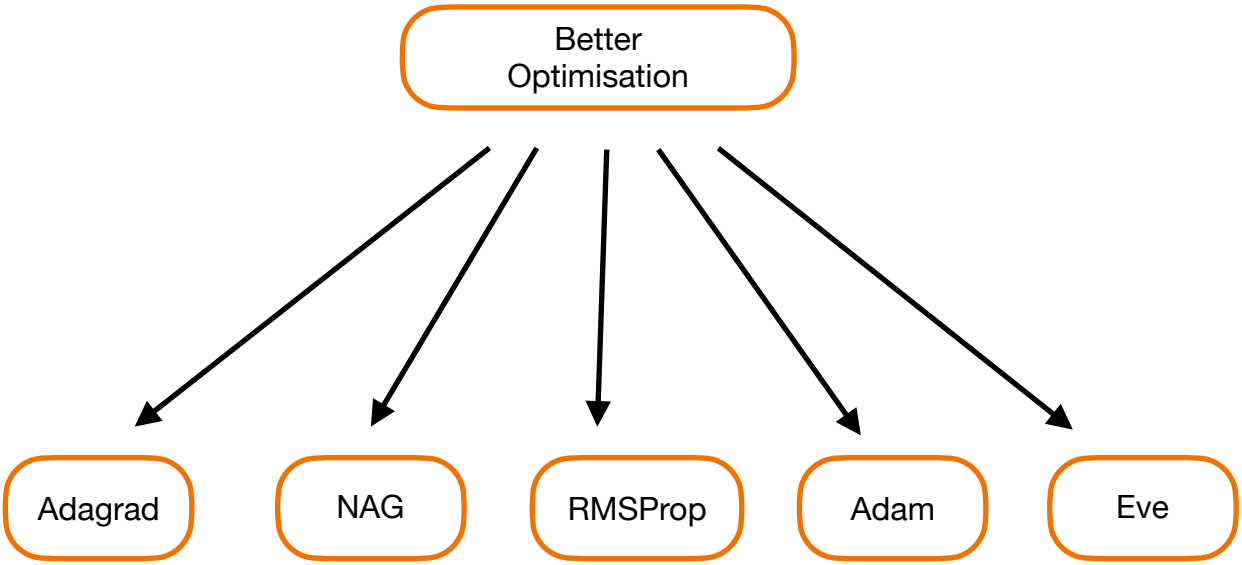
- Fully Connected Neural Network (FCNN) are the type of neural network where all the nodes in one layer are connected to all the nodes in other layer.
- The input layer consists of d (dimension of input vector) neurons which is a linear neuron which simply passes the input as output.
- Hidden layers has the J neurons which is fully connected to the next hidden layer with desired activations.
- Output layer is linear/sigmoidal/relu/softmax neuron depending on the tasks.
- Number of layers and neurons in each of the hidden layers are decided experimentally.
- Weights associated with all the connection between the neurons indicate the parameter of the complex nonlinear discriminant function/nonlinear surface that the network is trying to approximate.
- Network with two or more hidden layers are called deep neural networks.
- Task is to estimate the training parameters weights which is extremely computationally difficult and faces lots of convergence issues. To encounter these challenges, there are methods of optimisers which is used for better weight initialisations for better convergence.

Challenges in Deep Network

- High computing power
- Requires large data for training
- Issues with the gradient descent algorithms such as:

1. Slow convergence
2. Local minima problem
3. Poor weight initialisation

- Issues with activation functions



Better optimisation methods for the back propagation algorithms were developed such as momentum based SGD, Adagrad, RMSProp, Adam, Eve, etc.

The goal of these methods is to accelerate the convergence of the back propagation ie., gradient descent algorithm. These methods provides us better convergence and speed than the regular method and provides considerable results.

STOCHASTIC GRADIENT DESCENT ALGORITHM

This algorithm uses pattern mode and updates the parameters or every single data point.

There are some issues with the pattern mode:
1. It is an approximate gradient because the total gradient is estimated based on the single data point
2. No guarantee that each step will decrease the loss
3. Will encounter many oscillations before convergence

Number of iterations per epoch for N data points is N. It is faster than batch mode.

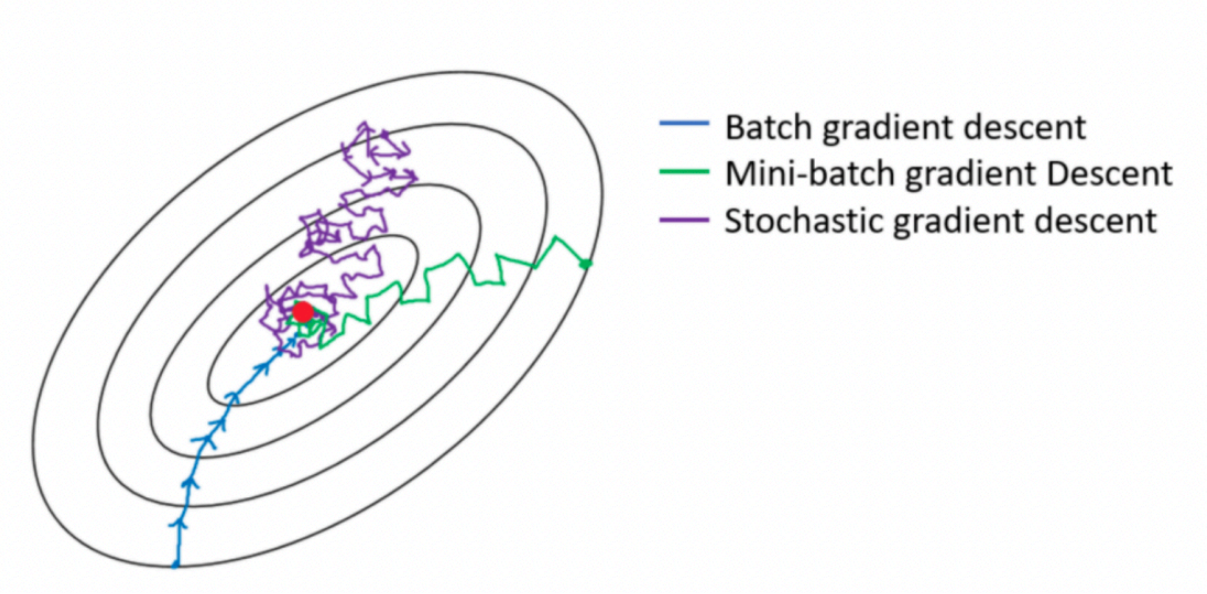
BATCH (VANNILA) GRADIENT DESCENT ALGORITHM

This algorithm iterates over the entire training data once and then updates the parameters. The update rule is based on the true gradient loss which is the sum of all the gradients of the errors corresponding to each data point.

Since there is no approximation, each step guarantees that the loss will decrease.

There are some issues with the pattern mode:
1. It will undergo huge calculations to update one single step
2. It leads to slow convergence

Number of iterations per epoch for N data points is 1.

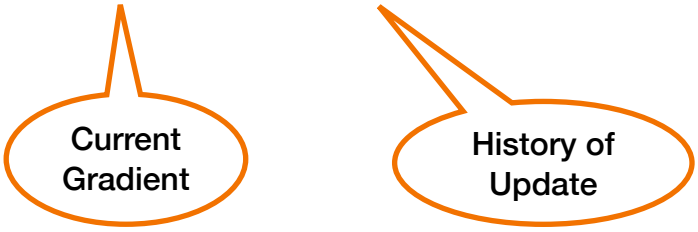


MOMENTUM BASED SGD

Generalised Delta Rule:

The model predicts and updates the weight based on its history of weights so that the updated weight direction will increase large if it is same direction leads to faster convergence.

$\Delta w(m) = -\eta \Delta w(m) + \alpha \Delta w(m-1)$, $0 < \alpha < 1$ is the momentum.



$\Delta w(m) = \dfrac{\partial E_n(m)}{\partial w(m)}$; gradient at m^{th} iterations and the weight is updated by

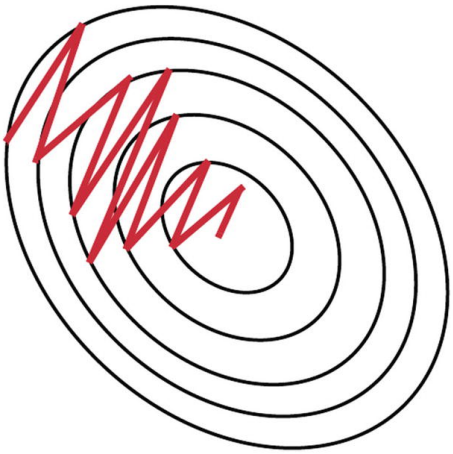
$$w(m+1) = w(m) + \Delta w(m)$$

If the gradient has the same sign on consecutive iterations then the net weight change increases over these iterations.
Momentum causes an acceleration in a downhill direction, because accumulated history of updates is large.

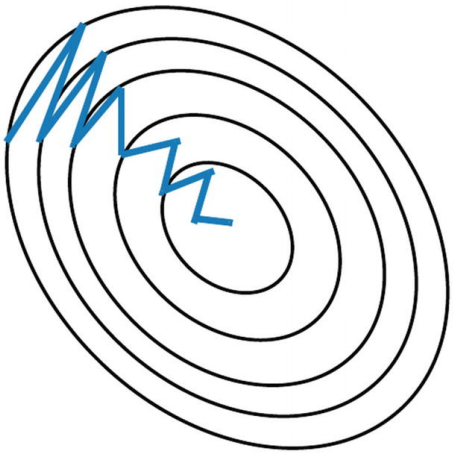
If the gradient has different signs on consecutive iterations then the net weight change decreases over these iterations.
Momentum causes a deceleration in a downhill direction and avoids oscillations.

Advantages:
Irrespective of an error surface having gentle slope or larger slope, (momentum-based gradient descent is able to take large steps forward, because of the momentum carries it along)
In some situation, momentum would cause to run pass the goal.
If the minima is very shallow, this may help to overshoot that minima and thus possibly find deeper local minima
If the minima is sufficiently deep, it causes oscillations in and out of that minima valley as the momentum carries it out of the valley.

It takes a lot of u-turns before finally converging and Despite these oscillations and u-turns, it still converges faster than vanilla gradient descent



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

NESTEROV ACCELERATED GRADIENT DESCENT

To avoid the oscillations in momentum based method, we are making small changes in weight update to predict the weight to avoid oscillations.

$$w(m+1) = w(m) + \alpha\Delta w(m-1) - \eta\Delta w(m)$$

$$w_{lookahead} = w(m) + \alpha\Delta w(m-1)$$

Looking ahead helps NAG in correcting its course quicker than momentum-based gradient descent and the oscillations are smaller.

$\alpha(m) = min\left(1 - 2^{-1-log_2\left(int(\frac{m}{250})+1\right)}, \alpha_{max}\right)$ and α_{max} can be 0.999.

RMSProp

The goal of this method is to achieve speed in parameter update by having different learning rates for different parameters and also to avoid the decaying of learning rate aggressively.

$$v_l(m) = \beta v_l(m-1) + (1-\beta)(\Delta w_l(m))^2 \ \forall l = 1...L$$

$$w_l(m+1) = w_l(m) - \frac{\eta}{\sqrt{v_l(m) + \epsilon}} \Delta w_l(m)$$

Since $v_l(m)$ is exponential moving average value, it smooth the running estimate.

Disadvantage:

The running estimate of squared gradient is initialised to 0 which causes some undesirable bias to second order moment in the early iterations, which disappear over longer iterations.

Adaptive Moments (Adam)

The goal of this method is to solve the decay problem and update the parameter by combining momentum and decaying learning rate.

$$u_l(m) = \beta_1 u_l(m-1) + (1-\beta_1)(\Delta w_l(m)) \ \forall l = 1...L$$

$$v_l(m) = \beta_2 v_l(m-1) + (1-\beta_2)(\Delta w_l(m))^2 \ \forall l = 1...L$$

$\hat{u}_l(m) = \frac{u_l(m)}{1-\beta_1^m}$ and $\hat{v}_l(m) = \frac{v_l(m)}{1-\beta_2^m}$ are the bias corrected terms

$$w_l(m+1) = w_l(m) - \frac{\eta}{\sqrt{\hat{v}_l(m) + \epsilon}} \hat{u}_l(m) \forall l = 1...L$$

Update rule is a combination of momentum and decaying learning rate. Hence the speed up in converging to solution.

Python Implementation

Developed a three hidden layer deep fully connected neural network using tensor flow Keras.

We have MNIST data (hand written images) for 5 classes (1, 4, 7, 8, 9) digits. The deep network is trained using these images each of size (28*28) with different architectures

Architecture	Input layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	784	128	100	128	10
2	784	128	256	128	10
3	784	256	256	256	10
4	784	512	256	512	10
5	784	512	512	512	10
6	784	1024	1024	1024	10

The output layer is fixed with 10 neurons with each representing digits from {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Since the input is given only from the set {1, 4, 7, 8, 9}, it is expected that the model should not give any results from the set of digits {0, 2, 3, 5, 6}.

Convergence criteria is the difference between average error of successive epochs fall below a threshold 10^{-4}

The confusion matrix explains this. The data is trained using the above five models and result (Training accuracy, Validation accuracy and average error vs epoch plot) is attached. Last five iterations of epoch is shown.

Then the best architecture is chosen for this data and the test data is tested. Accuracy and confusion matrix parameters are obtained.

ARCHITECTURE 1

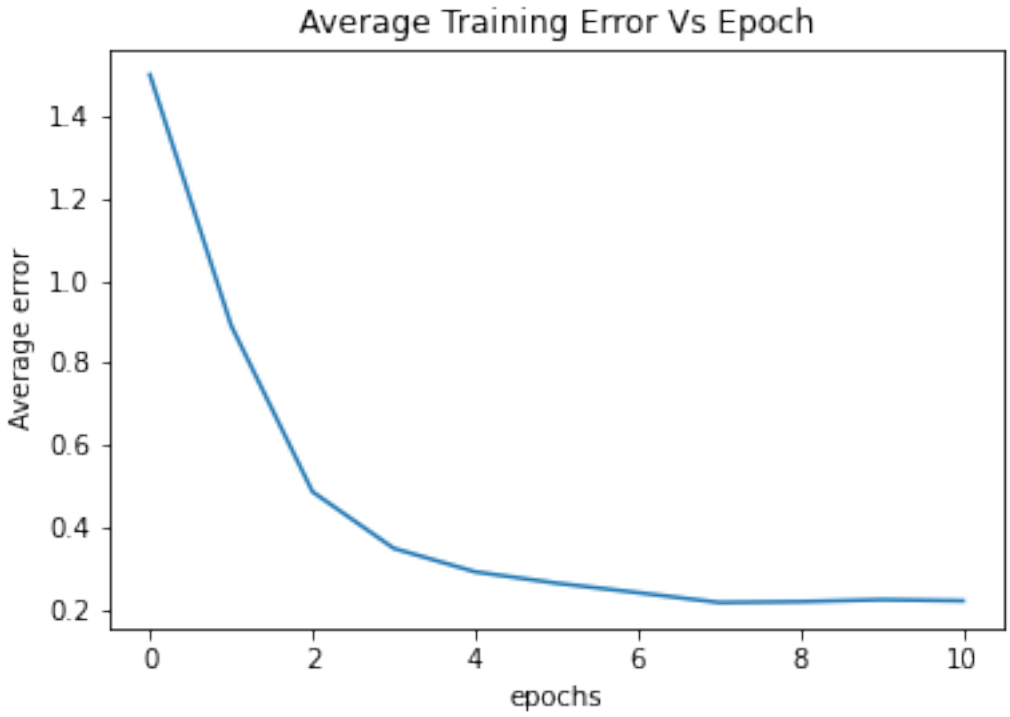
```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 Input_Layer (Flatten)        (None, 784)               0

 Hidden_Layer_1 (Dense)       (None, 128)               100480

 Hidden_Layer_2 (Dense)       (None, 100)               12900

 Hidden_Layer_3 (Dense)       (None, 128)               12928

 Output_Layer (Dense)         (None, 10)                1290

=================================================================
Total params: 127,598
Trainable params: 127,598
Non-trainable params: 0
```

(1) SGD Pattern mode (Batch Size = 1)

Parameters	Values Taken
Learning rate	0.001
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

RESULTS

```
Epoch 7/10000
11385/11385 [==============================] - 4s 310us/step - loss: 0.2446 - accuracy: 0.9322
Epoch 8/10000
11385/11385 [==============================] - 4s 312us/step - loss: 0.2208 - accuracy: 0.9378
Epoch 9/10000
11385/11385 [==============================] - 4s 311us/step - loss: 0.2227 - accuracy: 0.9383
Epoch 10/10000
11385/11385 [==============================] - 4s 310us/step - loss: 0.2275 - accuracy: 0.9348
Epoch 11/10000
11385/11385 [==============================] - 4s 308us/step - loss: 0.2248 - accuracy: 0.9354
```




```
119/119 [==============================] - 0s 414us/step - loss: 0.2004 - accuracy: 0.9418
```

OBSERVATIONS:

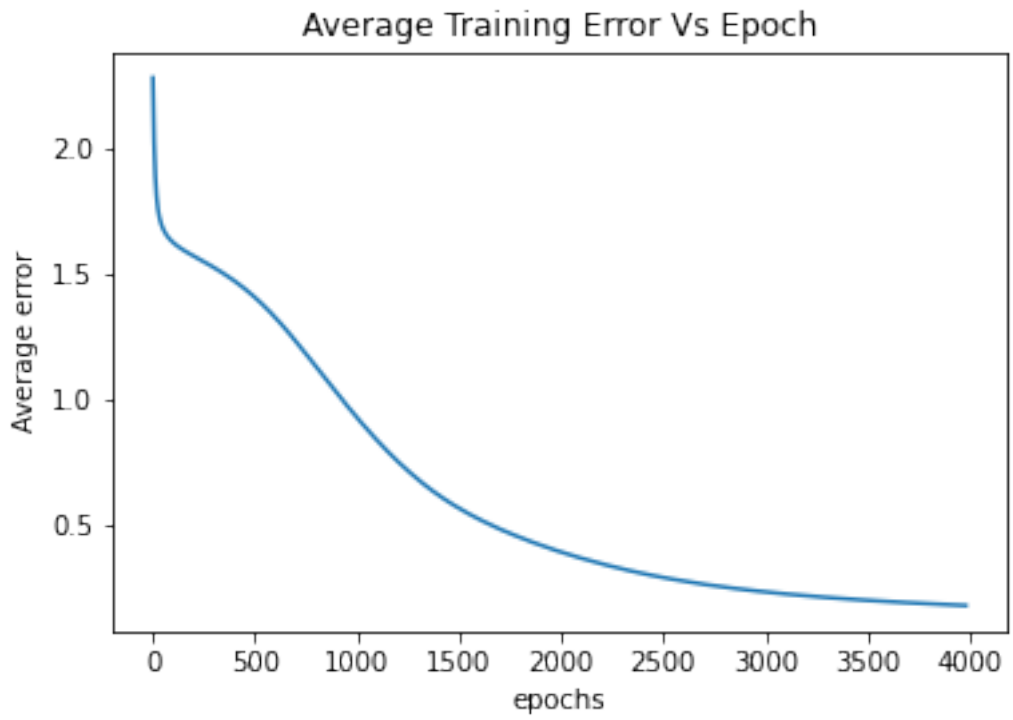
1. No of epochs for convergence: **11**
2. Training Accuracy: **0.9354**
3. Validation Accuracy: **0.9418**

(2) Batch (Vanilla) Gradient Descent (Batch Size = 784)

Parameters	Values Taken
Learning rate	0.001
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

RESULTS

```
Epoch 3978/10000
15/15 [==============================] - 0s 4ms/step - loss: 0.1833 - accuracy: 0.9489
Epoch 3979/10000
15/15 [==============================] - 0s 4ms/step - loss: 0.1833 - accuracy: 0.9490
Epoch 3980/10000
15/15 [==============================] - 0s 4ms/step - loss: 0.1833 - accuracy: 0.9490
Epoch 3981/10000
15/15 [==============================] - 0s 4ms/step - loss: 0.1832 - accuracy: 0.9490
Epoch 3982/10000
15/15 [==============================] - 0s 4ms/step - loss: 0.1832 - accuracy: 0.9489
```




```
119/119 [==============================] - 0s 411us/step - loss: 0.2296 - accuracy: 0.9291
```

OBSERVATIONS:

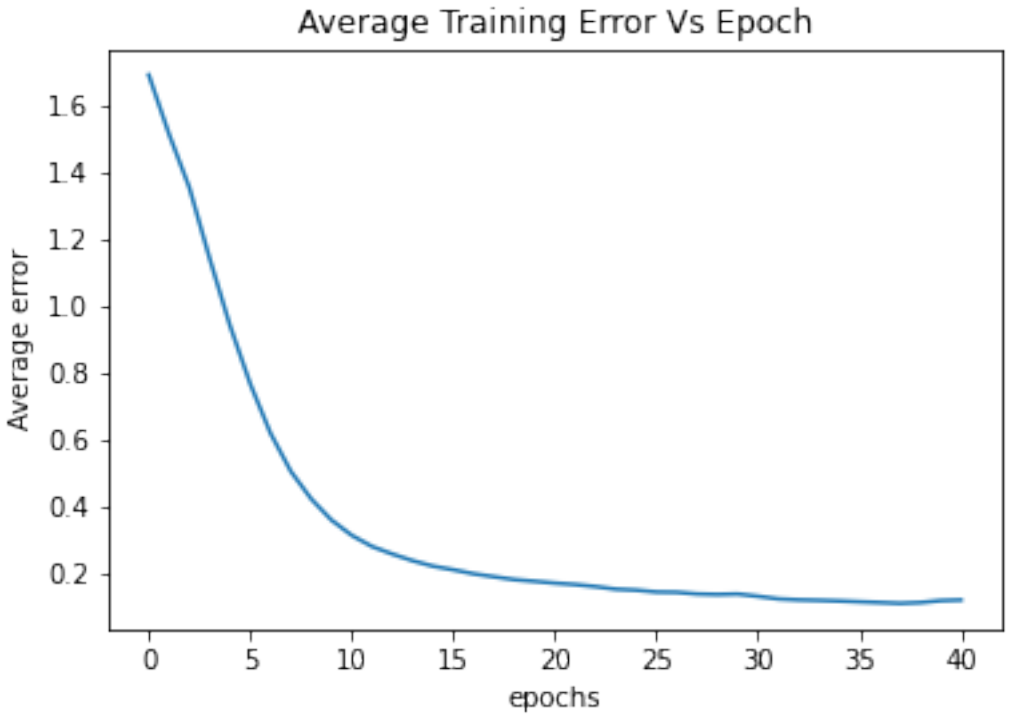
1. No of epochs for convergence: **3982**
2. Training Accuracy: **0.9489**
3. Validation Accuracy: **0.9291**

(3) NAG (Batch Size = 32)

Parameters	Values Taken
Learning rate	0.001
Momentum	0.9
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

RESULTS

```
Epoch 37/10000
356/356 [==============================] - 0s 622us/step - loss: 0.1117 - accuracy: 0.9668
Epoch 38/10000
356/356 [==============================] - 0s 623us/step - loss: 0.1094 - accuracy: 0.9671
Epoch 39/10000
356/356 [==============================] - 0s 625us/step - loss: 0.1117 - accuracy: 0.9676
Epoch 40/10000
356/356 [==============================] - 0s 628us/step - loss: 0.1176 - accuracy: 0.9648
Epoch 41/10000
356/356 [==============================] - 0s 625us/step - loss: 0.1193 - accuracy: 0.9646
```




```
119/119 [==============================] - 0s 405us/step - loss: 0.1353 - accuracy: 0.9584
```

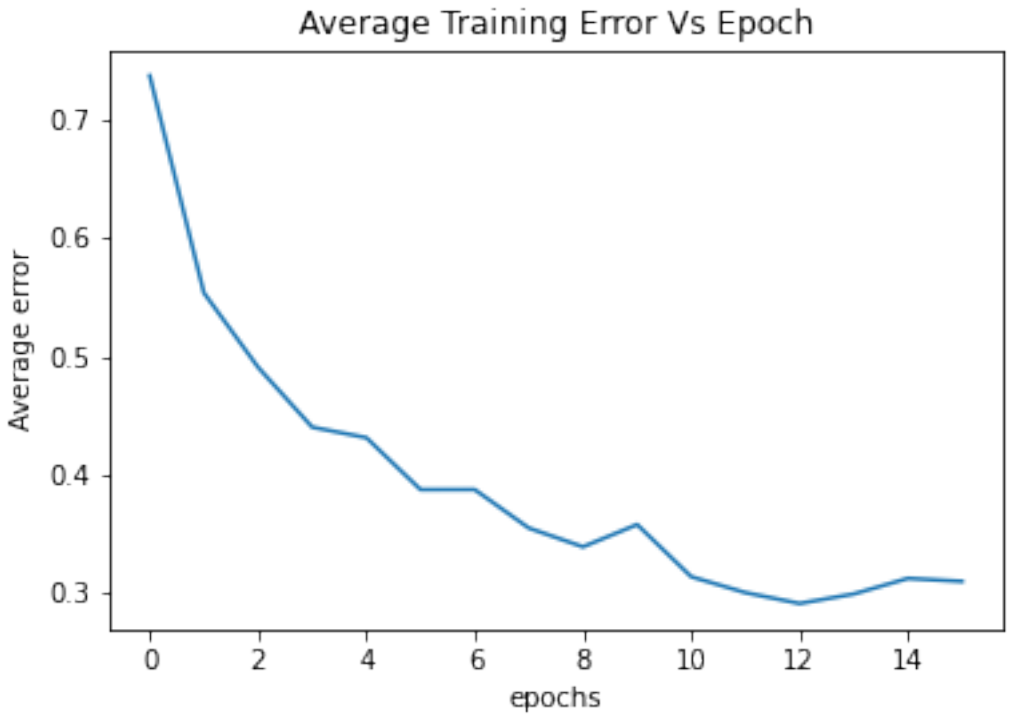
OBSERVATIONS:

1. No of epochs for convergence: **41**
2. Training Accuracy: **0.9646**
3. Validation Accuracy: **0.9584**

(4) RMSProp (Batch Size = 32)

Parameters	Values Taken
Learning rate	0.001
Momentum	0.9
Beta	0.99
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

RESULTS




```
119/119 [==============================] - 0s 422us/step - loss: 0.2754 - accuracy: 0.9094
```

```
Epoch 12/10000
356/356 [==============================] - 0s 674us/step - loss: 0.2992 - accuracy: 0.9004
Epoch 13/10000
356/356 [==============================] - 0s 669us/step - loss: 0.2900 - accuracy: 0.9065
Epoch 14/10000
356/356 [==============================] - 0s 678us/step - loss: 0.2981 - accuracy: 0.9008
Epoch 15/10000
356/356 [==============================] - 0s 676us/step - loss: 0.3114 - accuracy: 0.8950
Epoch 16/10000
356/356 [==============================] - 0s 669us/step - loss: 0.3088 - accuracy: 0.8955
```

OBSERVATIONS:

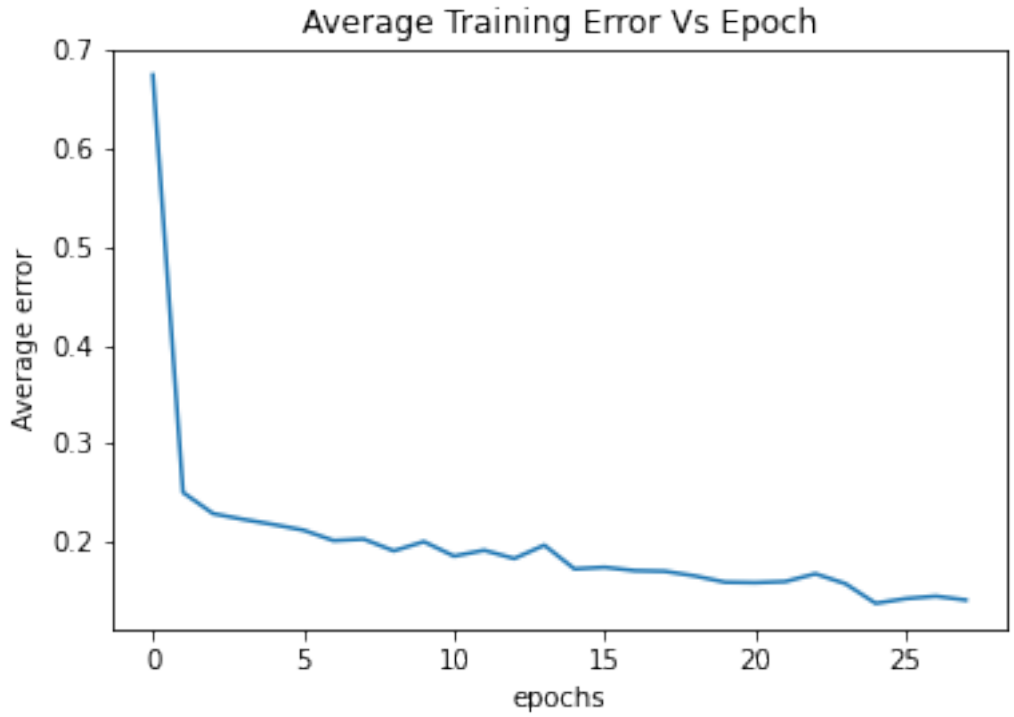
1. No of epochs for convergence: **16**
2. Training Accuracy: **0.8955**
3. Validation Accuracy: **0.9094**

(5) Adam Optimiser (Batch Size = 32)

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

RESULTS

```
Epoch 24/10000
356/356 [==============================] - 0s 731us/step - loss: 0.1580 - accuracy: 0.9480
Epoch 25/10000
356/356 [==============================] - 0s 719us/step - loss: 0.1381 - accuracy: 0.9539
Epoch 26/10000
356/356 [==============================] - 0s 726us/step - loss: 0.1429 - accuracy: 0.9526
Epoch 27/10000
356/356 [==============================] - 0s 728us/step - loss: 0.1453 - accuracy: 0.9515
Epoch 28/10000
356/356 [==============================] - 0s 724us/step - loss: 0.1414 - accuracy: 0.9525
```

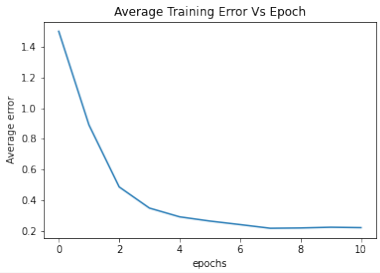
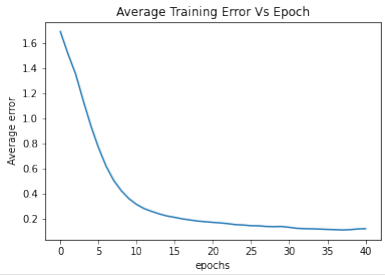
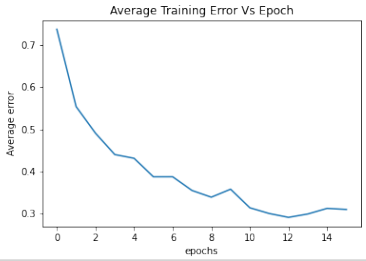
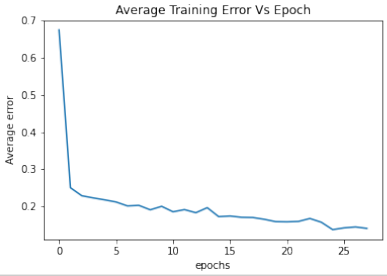



```
119/119 [==============================] - 0s 418us/step - loss: 0.1422 - accuracy: 0.9542
```

OBSERVATIONS:

1. No of epochs for convergence: **28**
2. Training Accuracy: **0.9525**
3. Validation Accuracy: **0.9542**

Summary Comparison of Optimisers for Architecture 1

Optimisers	No of epoch	Error vs Epoch	Training Accuracy	Validation Accuracy
SGD	11		0.9354	0.9418
Batch Mode	3982		0.9489	0.9291
NAG	41		0.9646	0.9584
RMSProp	16		0.8955	0.9094
Adam	28		0.9525	0.9542

ARCHITECTURE 2

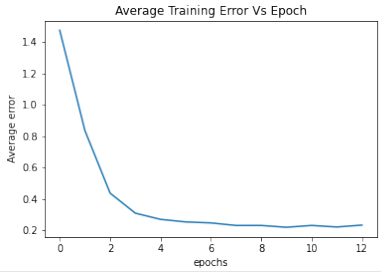
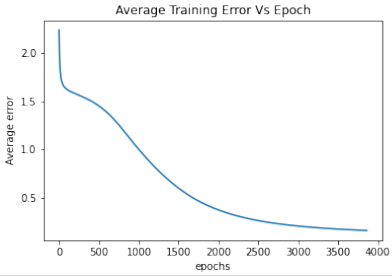
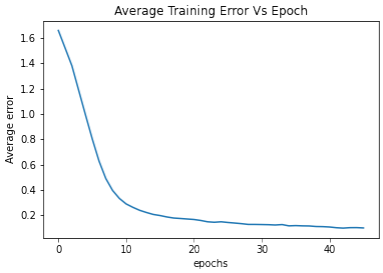
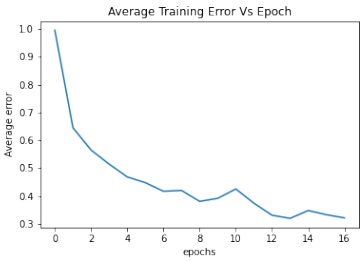
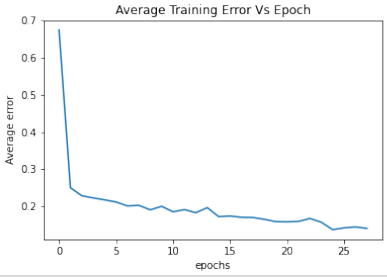
```
Model: "sequential_37"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input_Layer (Flatten)       (None, 784)               0

 Hidden_Layer_1 (Dense)      (None, 128)               100480

 Hidden_Layer_2 (Dense)      (None, 256)               33024

 Hidden_Layer_3 (Dense)      (None, 128)               32896

 Output_Layer (Dense)        (None, 10)                1290

=================================================================
Total params: 167,690
Trainable params: 167,690
Non-trainable params: 0
```

PARAMETERS TAKEN FOR VARIOUS OPTIMIZERS

(It is same for all the architectures)

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

Summary Comparison of Optimisers for Architecture 2

Optimisers	No of epoch	Error vs Epoch	Training Accuracy	Validation Accuracy
SGD	13		0.9281	0.9341
Batch Mode	3862		0.9533	0.9362
NAG	46		0.97	0.9639
RMSProp	17		0.8908	0.8917
Adam	11		0.9377	0.9370

ARCHITECTURE 3

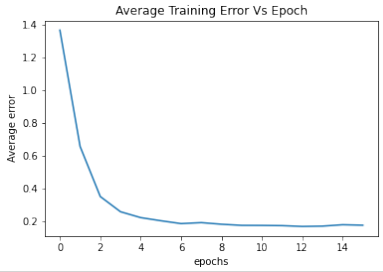
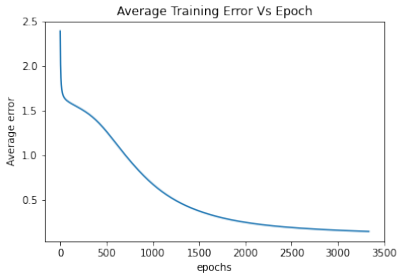
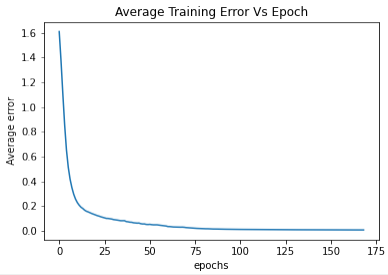
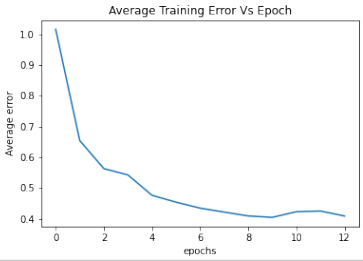
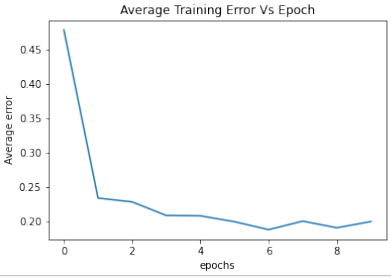
```
Model: "sequential_42"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 Input_Layer (Flatten)        (None, 784)               0

 Hidden_Layer_1 (Dense)       (None, 256)               200960

 Hidden_Layer_2 (Dense)       (None, 256)               65792

 Hidden_Layer_3 (Dense)       (None, 256)               65792

 Output_Layer (Dense)         (None, 10)                2570

=================================================================
Total params: 335,114
Trainable params: 335,114
Non-trainable params: 0
```

PARAMETERS TAKEN FOR VARIOUS OPTIMIZERS

(It is same for all the architectures)

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

Summary Comparison of Optimisers for Architecture 3

Optimisers	No of epoch	Error vs Epoch	Training Accuracy	Validation Accuracy
SGD	16		0.9458	0.9460
Batch Mode	3335		0.9554	0.9370
NAG	169		0.9989	0.9742
RMSProp	13		0.8635	0.8870
Adam	10		0.9332	0.9439

ARCHITECTURE 4

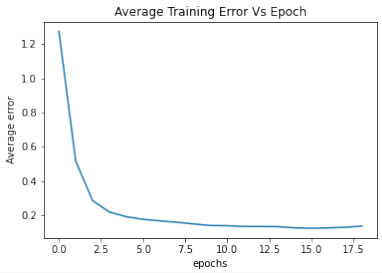
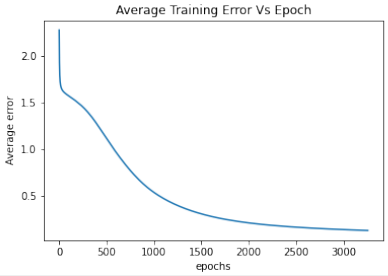
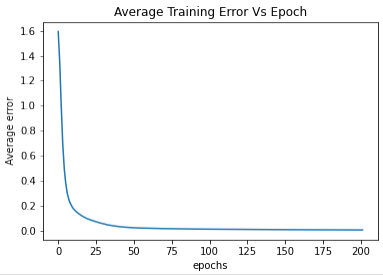
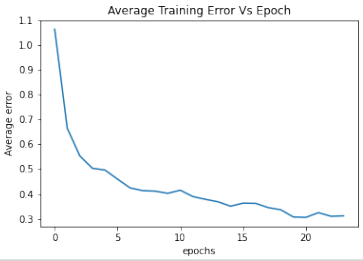
```
Model: "sequential_47"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input_Layer (Flatten)       (None, 784)               0

 Hidden_Layer_1 (Dense)      (None, 512)               401920

 Hidden_Layer_2 (Dense)      (None, 256)               131328

 Hidden_Layer_3 (Dense)      (None, 512)               131584

 Output_Layer (Dense)        (None, 10)                5130

=================================================================
Total params: 669,962
Trainable params: 669,962
Non-trainable params: 0
```

PARAMETERS TAKEN FOR VARIOUS OPTIMIZERS

(It is same for all the architectures)

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

Summary Comparison of Optimisers for Architecture 4

Optimisers	No of epoch	Error vs Epoch	Training Accuracy	Validation Accuracy
SGD	19		0.9585	0.9576
Batch Mode	3256		0.9622	0.9455
NAG	202		0.9993	0.9713
RMSProp	24		0.9055	0.9120
Adam	17		0.9439	0.9441

ARCHITECTURE 5

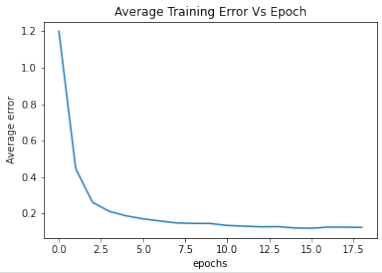
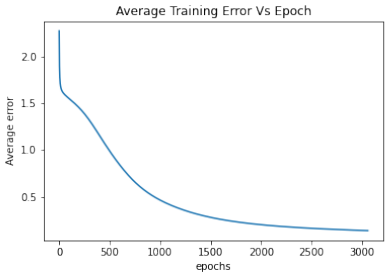
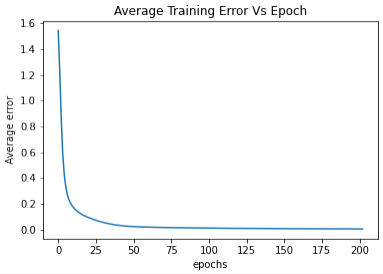
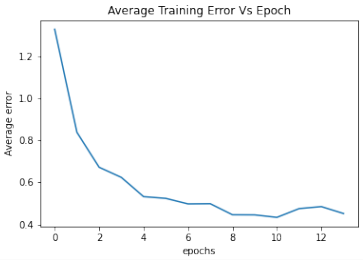
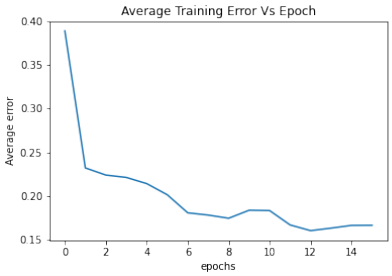
```
Model: "sequential_53"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input_Layer (Flatten)       (None, 784)               0

 Hidden_Layer_1 (Dense)      (None, 512)               401920

 Hidden_Layer_2 (Dense)      (None, 512)               262656

 Hidden_Layer_3 (Dense)      (None, 512)               262656

 Output_Layer (Dense)        (None, 10)                5130

=================================================================
Total params: 932,362
Trainable params: 932,362
Non-trainable params: 0
```

PARAMETERS TAKEN FOR VARIOUS OPTIMIZERS

(It is same for all the architectures)

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

Summary Comparison of Optimisers for Architecture 5

Optimisers	No of epoch	Error vs Epoch	Training Accuracy	Validation Accuracy
SGD	19		0.9643	0.9584
Batch Mode	3057		0.9601	0.9412
NAG	203		0.9994	0.9692
RMSProp	14		0.8620	0.8693
Adam	16		0.9439	0.9528

ARCHITECTURE 6

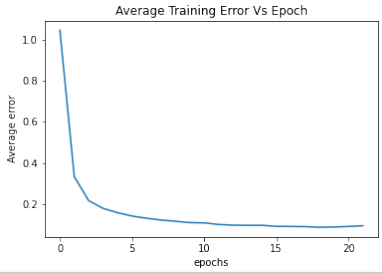
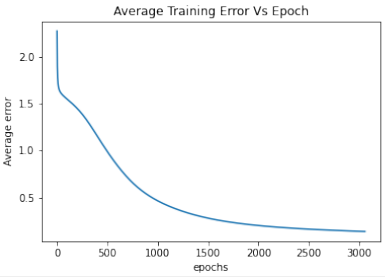
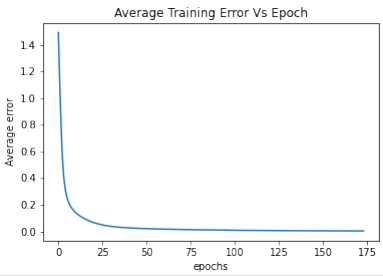
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input_Layer (Flatten)       (None, 784)               0

 Hidden_Layer_1 (Dense)      (None, 1024)              803840

 Hidden_Layer_2 (Dense)      (None, 1024)              1049600

 Hidden_Layer_3 (Dense)      (None, 1024)              1049600

 Output_Layer (Dense)        (None, 10)                10250

=================================================================
Total params: 2,913,290
Trainable params: 2,913,290
Non-trainable params: 0
```

PARAMETERS TAKEN FOR VARIOUS OPTIMIZERS

(It is same for all the architectures)

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

Summary Comparison of Optimisers for Architecture 6

Optimisers	No of epoch	Error vs Epoch	Training Accuracy	Validation Accuracy
SGD	22		0.9715	0.9628
Batch Mode	2702		0.9665	0.9431
NAG	174		0.9998	0.9671
RMSProp	21		0.5120	0.5731
Adam	12		0.9419	0.9520

BEST ARCHITECTURE

The best architecture is chosen based on the validation accuracy.

1) SGD

Architecture 6 with 784 input neurons, **1024** neurons in hidden layer 1, **1024** neurons in hidden layer 2, **1024** neurons in hidden layer 3 and 10 neurons in output layer gives the best **validation accuracy** of **0.9628**.

2) Batch (Vanilla) Gradient Descent Algorithm

Architecture 4 with 784 input neurons, **512** neurons in hidden layer 1, **256** neurons in hidden layer 2, **512** neurons in hidden layer 3 and 10 neurons in output layer gives the best **validation accuracy** of **0.9455**.

3) NAG

Architecture 4 with 784 input neurons, **512** neurons in hidden layer 1, **256** neurons in hidden layer 2, **512** neurons in hidden layer 3 and 10 neurons in output layer gives the best **validation accuracy** of **0.9713**.

4) RMSProp

Architecture 4 with 784 input neurons, **512** neurons in hidden layer 1, **256** neurons in hidden layer 2, **512** neurons in hidden layer 3 and 10 neurons in output layer gives the best **validation accuracy** of **0.9120**.

5) Adam

Architecture 1 with 784 input neurons, **128** neurons in hidden layer 1, **100** neurons in hidden layer 2, **128** neurons in hidden layer 3 and 10 neurons in output layer gives the best **validation accuracy** of **0.9542**.

TEST DATA RESULTS USING BEST ARCHITECTURE

(1) SGD

Training Accuracy	Test Accuracy
0.9715	0.9626

CONFUSION MATRIX-1

PREDICTED CLASS \ ACTUAL CLASS										
	0	0	0	0	0	0	0	0	0	0
	0	747	0	0	0	0	0	4	8	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	744	0	0	2	2	10
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	8	0	0	8	0	0	731	1	11
	0	6	0	0	10	0	0	2	736	5
	0	6	0	0	26	0	0	22	10	695

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 747,   0,   0,   0,   0,   0,   4,   8,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   1,   0,   0, 744,   0,   0,   2,   2,  10],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   8,   0,   0,   8,   0,   0, 731,   1,  11],
       [  0,   6,   0,   0,  10,   0,   0,   2, 736,   5],
       [  0,   6,   0,   0,  26,   0,   0,  22,  10, 695]], dtype=int32)>
```

```
119/119 [==============================] - 0s 3ms/step - loss: 0.1264 - accuracy: 0.9626
```

(2) Batch (vanilla) Gradient Descent

Training Accuracy	Test Accuracy
0.9617	0.9381

CONFUSION MATRIX-2										
	ACTUAL CLASS									
PREDICTED CLASS	0	0	0	0	0	0	0	0	0	0
	0	742	0	0	1	0	0	5	11	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	2	0	0	727	0	0	2	3	25
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	6	0	0	8	0	0	707	2	36
	0	12	0	0	10	0	0	4	722	11
	0	2	0	0	28	0	0	47	20	662

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 746,   0,   0,   0,   0,   0,   2,  10,   1],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   2,   0,   0, 730,   0,   0,   2,   2,  23],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   5,   0,   0,  12,   0,   0, 721,   1,  20],
       [  0,   4,   0,   0,  10,   0,   0,   3, 734,   8],
       [  0,   3,   0,   0,  27,   0,   0,  19,  12, 698]], dtype=int32)>
```

```
119/119 [==============================] - 0s 1ms/step - loss: 0.1957 - accuracy: 0.9381
```

(3) NAG

Training Accuracy	Test Accuracy
0.9990	0.9657

CONFUSION MATRIX-3

PREDICTED CLASS \ ACTUAL CLASS										
	0	0	0	0	0	0	0	0	0	0
	0	749	0	0	0	0	0	1	8	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	743	0	0	1	2	12
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	3	0	0	9	0	0	730	2	15
	0	6	0	0	8	0	0	4	737	4
	0	3	0	0	18	0	0	18	14	706

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 749,   0,   0,   0,   0,   0,   1,   8,   1],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   1,   0,   0, 743,   0,   0,   1,   2,  12],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   3,   0,   0,   9,   0,   0, 730,   2,  15],
       [  0,   6,   0,   0,   8,   0,   0,   4, 737,   4],
       [  0,   3,   0,   0,  18,   0,   0,  18,  14, 706]], dtype=int32)>
```

```
119/119 [==============================] - 0s 1ms/step - loss: 0.1229 - accuracy: 0.9657
```

(4) RMSProp

Training Accuracy	Test Accuracy
0.8549	0.8859

CONFUSION MATRIX-4

PREDICTED CLASS \ ACTUAL CLASS										
	0	0	0	0	0	0	0	0	0	0
	0	753	0	0	0	0	0	1	5	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	13	0	0	640	0	0	2	26	78
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	23	0	0	2	0	0	663	9	62
	0	43	0	0	7	0	0	4	698	7
	0	18	0	0	40	0	0	44	49	608

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 753,   0,   0,   0,   0,   0,   1,   5,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,  13,   0,   0, 640,   0,   0,   2,  26,  78],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,  23,   0,   0,   2,   0,   0, 663,   9,  62],
       [  0,  43,   0,   0,   7,   0,   0,   4, 698,   7],
       [  0,  18,   0,   0,  40,   0,   0,  44,  49, 608]], dtype=int32)>
```

```
119/119 [==============================] - 0s 991us/step - loss: 0.4676 - accuracy: 0.8859
```


Figure 1

Figure 2

(5) Adam Optimizer

Training Accuracy	Test Accuracy
0.9350	0.9312

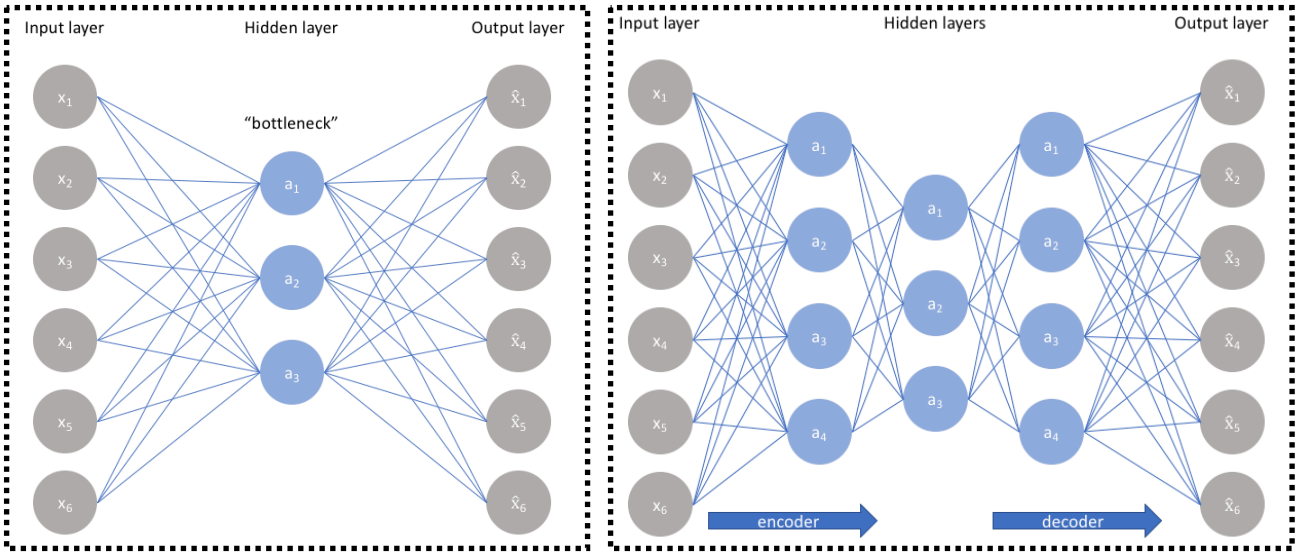
CONFUSION MATRIX-5

PREDICTED CLASS \ ACTUAL CLASS										
	0	0	0	0	0	0	0	0	0	0
	0	746	0	0	0	0	0	2	11	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	2	0	0	677	0	0	10	4	68
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	2	0	0	2	0	0	730	7	18
	0	9	0	0	6	0	0	6	690	48
	0	1	0	0	19	0	0	38	10	691

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 746,   0,   0,   0,   0,   0,   2,  11,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0, 677,   0,   0,  10,   4,  68],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   2,   0,   0,   2,   0,   0, 730,   7,  18],
       [  0,   9,   0,   0,   6,   0,   0,   6, 690,  48],
       [  0,   1,   0,   0,  19,   0,   0,  38,  10, 691]], dtype=int32)>
```

```
119/119 [==============================] - 0s 480us/step - loss: 0.2148 - accuracy: 0.9312
```

AUTOENCODERS



- Autoencoders are similar to FCNN with the exception of (1) the number of neurons in the output layer should be equal to the number of neuron in the input i.e., dimension of input vector. (2) The hidden layer should be odd in numbers and should contain less nodes compared to input node like bottle neck as shown in figure 1.
- Three hidden layer auto encoder is also shown in figure 2.
- Auto encoder can be used as non linear dimension reduction technique because hidden representation is obtained by non linear encoding operation.
- Autoencoder captures all the important characteristics of input.

Python Implementation

The same classes of input is used for auto encoders. Here we developed a single hidden layer auto encoder and three hidden layer auto encoder with different architectures.

Used Adam Optimiser and Sigmoidal Activation Function for the nodes in all hidden layers. The stopping criteria used here is the difference between average error of successive epochs fall below a threshold 10^{-4}.

The Autoencoder is split into compression tasks and classification task.

DATA COMPRESSION

(1) Single Hidden Layer - Autoencoder

Presented five different architectures with their results and best architecture is chosen based on the validation MSE loss and presented the results and accuracy parameters.

Architecture	Input layer	Hidden Layer 1	Output Layer
1	784	64	784
2	784	100	784
3	784	144	784
4	784	256	784
5	784	196	784

The hyper parameters are chosen and the Adam optimiser is used with sigmoidal activation.

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

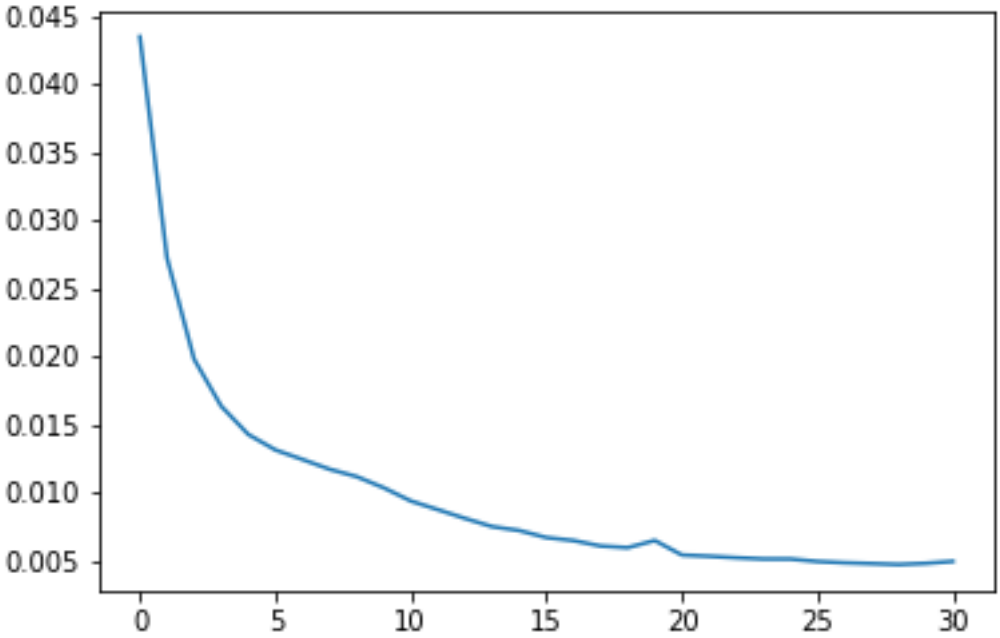
ARCHITECTURE 1

```
Model: "Autoencoder1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 784)               50960

 reshape (Reshape)           (None, 28, 28, 1)         0

=================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Epoch 27/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0049 - accuracy: 0.6725 - val_loss: 0.0059 - val_accuracy: 0.6715
Epoch 28/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0048 - accuracy: 0.6725 - val_loss: 0.0056 - val_accuracy: 0.6716
Epoch 29/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0048 - accuracy: 0.6725 - val_loss: 0.0064 - val_accuracy: 0.6715
Epoch 30/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0048 - accuracy: 0.6725 - val_loss: 0.0073 - val_accuracy: 0.6712
Epoch 31/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0050 - accuracy: 0.6724 - val_loss: 0.0056 - val_accuracy: 0.6716
```

Training Error Vs Epoch



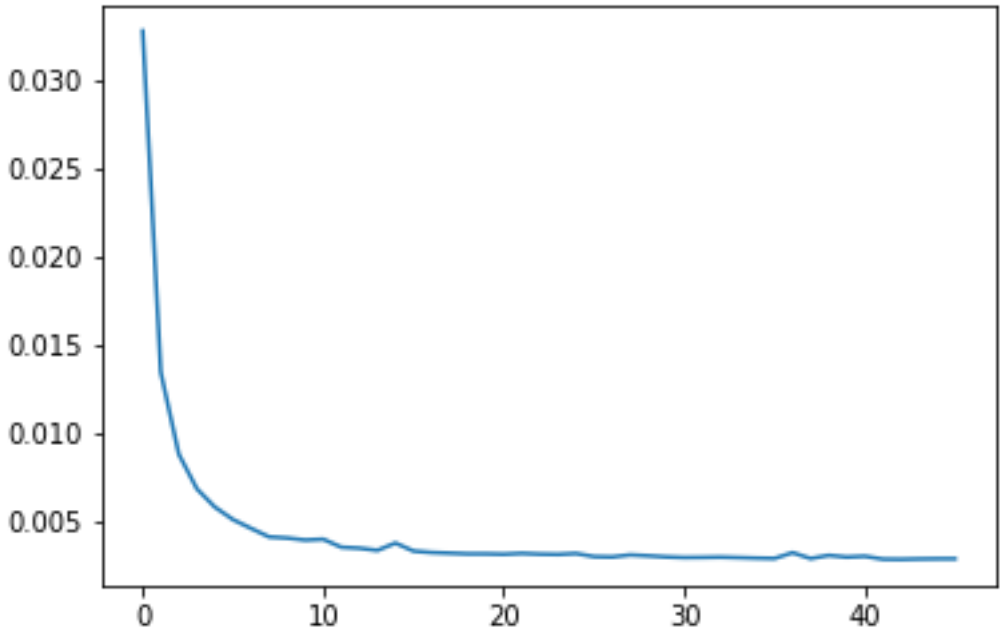
ARCHITECTURE 2

```
Model: "Autoencoder2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 100)               78500

 dense_3 (Dense)             (None, 784)               79184

 reshape_1 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 157,684
Trainable params: 157,684
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Epoch 42/100
358/358 [==============================] - 1s 4ms/step - loss: 0.0028 - accuracy: 0.6728 - val_loss: 0.0039 - val_accuracy: 0.6719
Epoch 43/100
358/358 [==============================] - 1s 3ms/step - loss: 0.0028 - accuracy: 0.6728 - val_loss: 0.0038 - val_accuracy: 0.6719
Epoch 44/100
358/358 [==============================] - 1s 3ms/step - loss: 0.0028 - accuracy: 0.6728 - val_loss: 0.0038 - val_accuracy: 0.6719
Epoch 45/100
358/358 [==============================] - 1s 4ms/step - loss: 0.0028 - accuracy: 0.6728 - val_loss: 0.0038 - val_accuracy: 0.6720
Epoch 46/100
358/358 [==============================] - 1s 3ms/step - loss: 0.0028 - accuracy: 0.6728 - val_loss: 0.0042 - val_accuracy: 0.6719
```

Training Error Vs Epoch



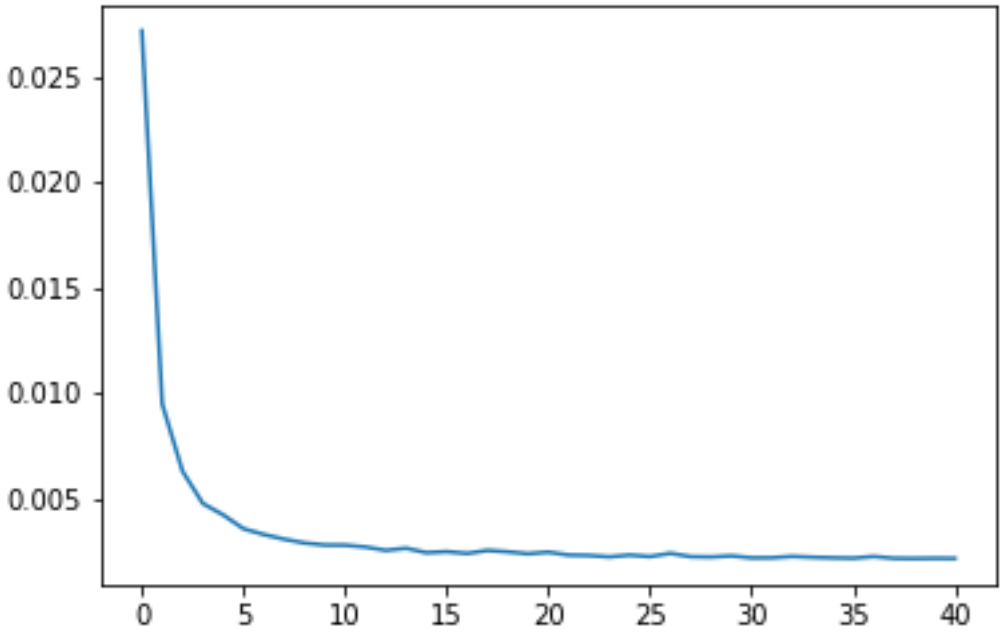
ARCHITECTURE 3

```
Model: "Autoencoder3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_2 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 144)               113040

 dense_5 (Dense)             (None, 784)               113680

 reshape_2 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 226,720
Trainable params: 226,720
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Epoch 37/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0023 - accuracy: 0.6728 - val_loss: 0.0032 - val_accuracy: 0.6721
Epoch 38/100
358/358 [==============================] - 2s 4ms/step - loss: 0.0022 - accuracy: 0.6728 - val_loss: 0.0031 - val_accuracy: 0.6720
Epoch 39/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0022 - accuracy: 0.6728 - val_loss: 0.0032 - val_accuracy: 0.6721
Epoch 40/100
358/358 [==============================] - 2s 4ms/step - loss: 0.0022 - accuracy: 0.6728 - val_loss: 0.0033 - val_accuracy: 0.6721
Epoch 41/100
358/358 [==============================] - 2s 4ms/step - loss: 0.0022 - accuracy: 0.6728 - val_loss: 0.0036 - val_accuracy: 0.6720
```

Training Error Vs Epoch



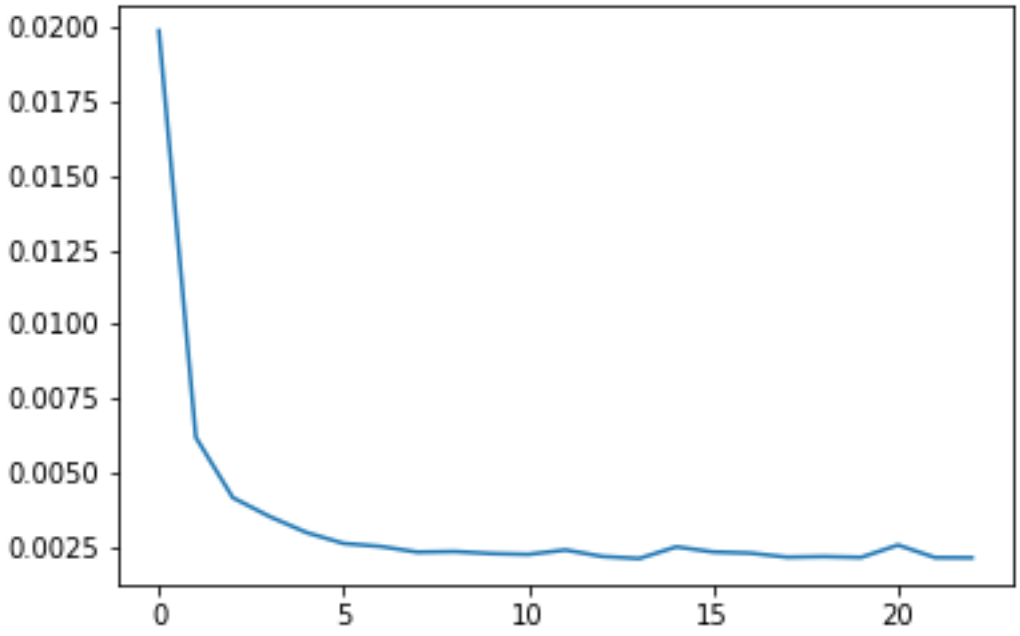
ARCHITECTURE 4

```
Model: "Autoencoder5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_4 (Flatten)         (None, 784)               0

 dense_8 (Dense)             (None, 256)               200960

 dense_9 (Dense)             (None, 784)               201488

 reshape_4 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 402,448
Trainable params: 402,448
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Epoch 30/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0018 - accuracy: 0.6728 - val_loss: 0.0031 - val_accuracy: 0.6721
Epoch 31/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0019 - accuracy: 0.6728 - val_loss: 0.0028 - val_accuracy: 0.6721
Epoch 32/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0018 - accuracy: 0.6728 - val_loss: 0.0031 - val_accuracy: 0.6721
Epoch 33/100
358/358 [==============================] - 2s 4ms/step - loss: 0.0019 - accuracy: 0.6728 - val_loss: 0.0035 - val_accuracy: 0.6720
Epoch 34/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0020 - accuracy: 0.6728 - val_loss: 0.0030 - val_accuracy: 0.6721
```

Training Error Vs Epoch



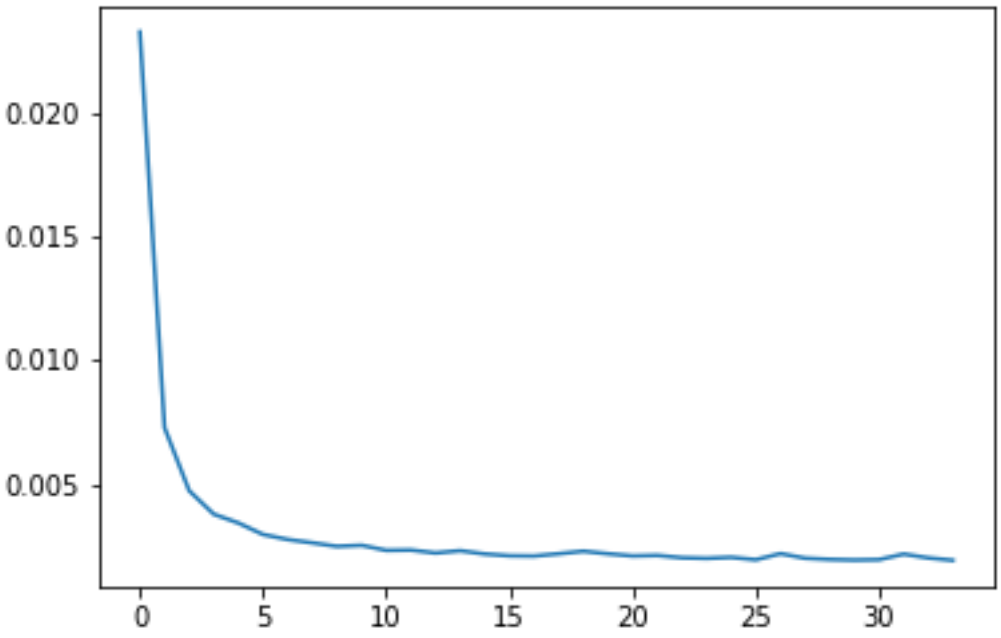
ARCHITECTURE 5

```
Model: "Autoencoder4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_3 (Flatten)         (None, 784)               0

 dense_6 (Dense)             (None, 196)               153860

 dense_7 (Dense)             (None, 784)               154448

 reshape_3 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 308,308
Trainable params: 308,308
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Epoch 30/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0019 - accuracy: 0.6728 - val_loss: 0.0032 - val_accuracy: 0.6721
Epoch 31/100
358/358 [==============================] - 2s 4ms/step - loss: 0.0019 - accuracy: 0.6728 - val_loss: 0.0037 - val_accuracy: 0.6720
Epoch 32/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0022 - accuracy: 0.6728 - val_loss: 0.0033 - val_accuracy: 0.6721
Epoch 33/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0020 - accuracy: 0.6728 - val_loss: 0.0031 - val_accuracy: 0.6721
Epoch 34/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0019 - accuracy: 0.6728 - val_loss: 0.0041 - val_accuracy: 0.6719
```

Training Error Vs Epoch



OBSERVATION

Architecture	No of hidden neuron	No of epoch	Training MSE error	Validation MSE error
1	64	31	0.0050	0.0056
2	100	46	0.0028	0.0042
3	144	41	0.0022	0.0036
4	256	34	0.0020	0.0030
5	196	34	0.0019	0.0040

Considered five under complete and five over complete architecture. It is observed that the over complete architecture does not perform any compression but just simple copies the input. Have not included the over complete architecture since it is compression task. Out of the under complete architecture, best is chosen based on validation MSE error.

BEST ARCHITECTURE

Architecture 4 is the best that works on the given data in this case with the validation error of **0.0030**.

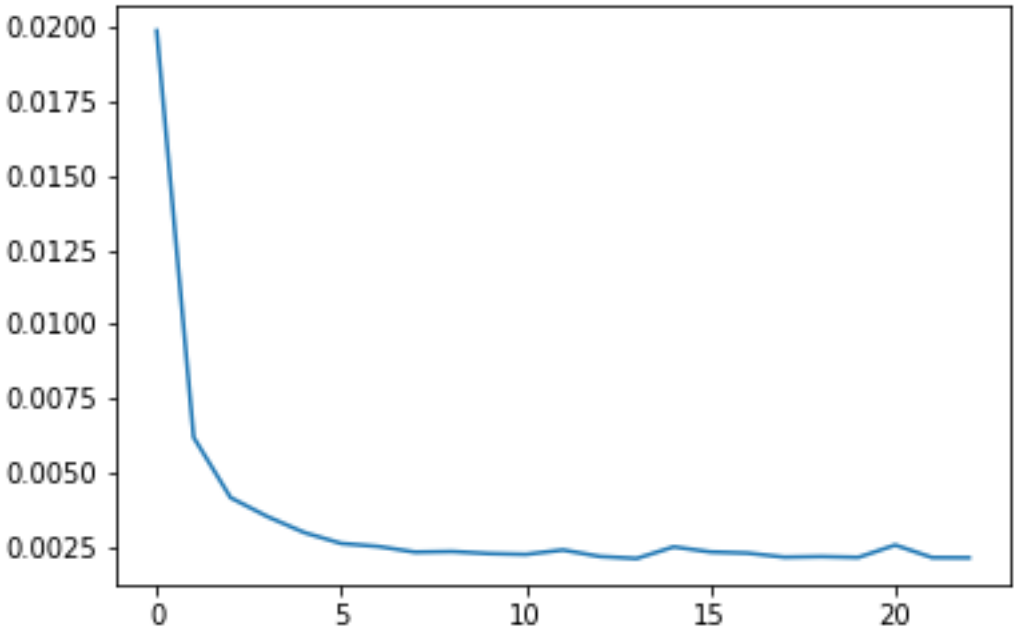
```
Model: "Autoencoder5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_4 (Flatten)         (None, 784)               0

 dense_8 (Dense)             (None, 256)               200960

 dense_9 (Dense)             (None, 784)               201488

 reshape_4 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 402,448
Trainable params: 402,448
Non-trainable params: 0
_________________________________________________________________
```

Reconstruction Errors:

Training Loss	0.0020
Validation Loss	0.0030
Test Reconstruction Error	0.0031

Error Vs Epoch Graph:

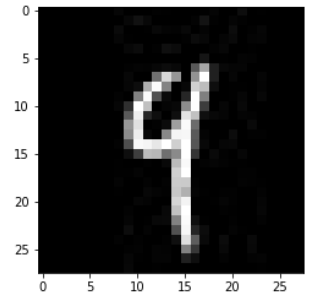
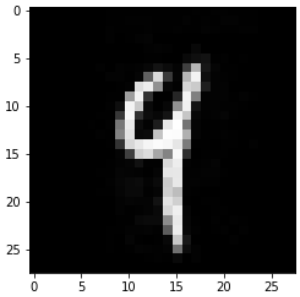
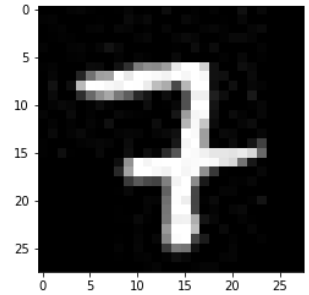
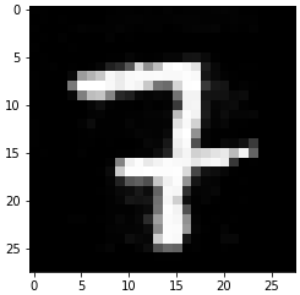
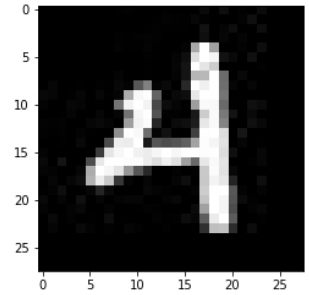
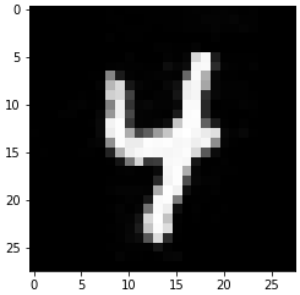
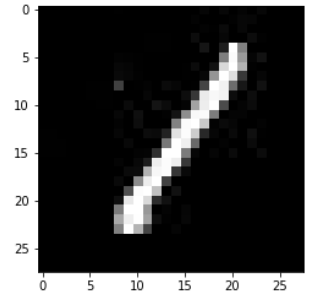
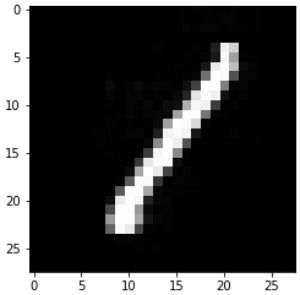
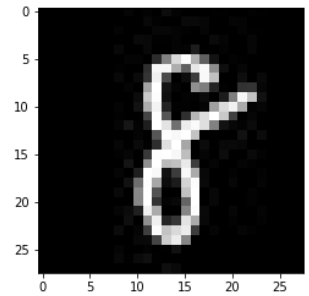
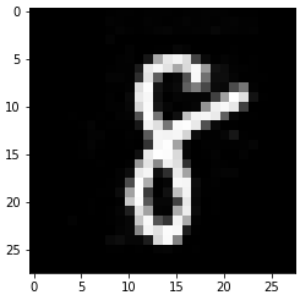
The architecture took 34 number of epoch to converge with a patience level of 3



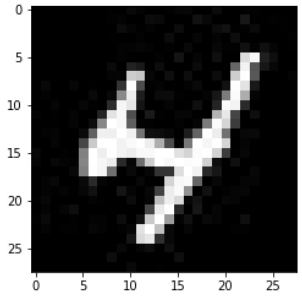
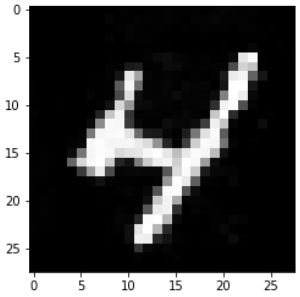
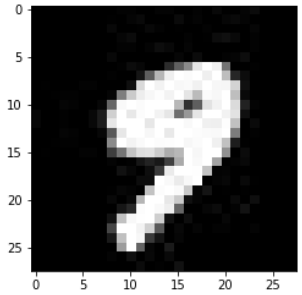
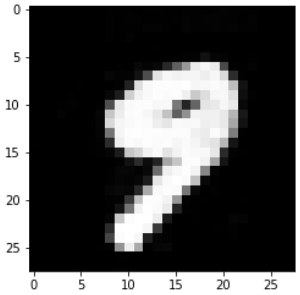
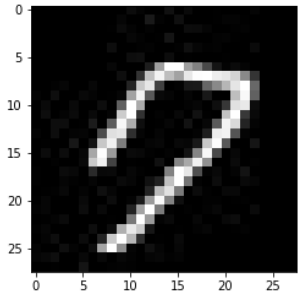
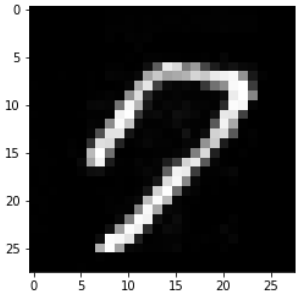
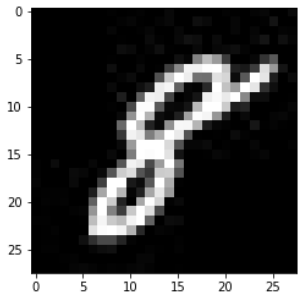
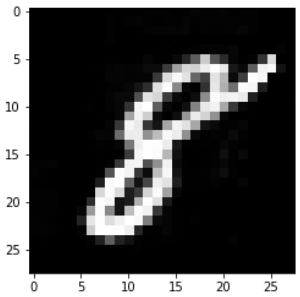
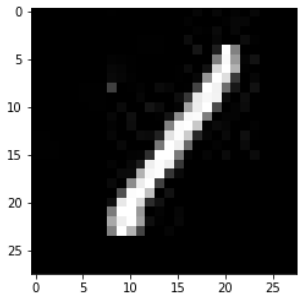
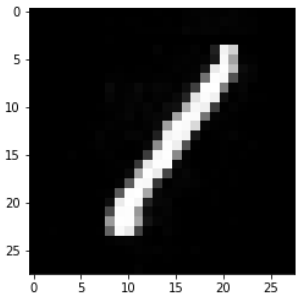
Confusion Matrix:

		ACTUAL CLASS				
		1	**4**	**7**	**8**	**9**
PREDICTED CLASS	**1**	760	0	3	4	2
	4	1	765	2	2	9
	7	1	4	736	3	15
	8	0	8	3	742	6
	9	0	10	89	7	753

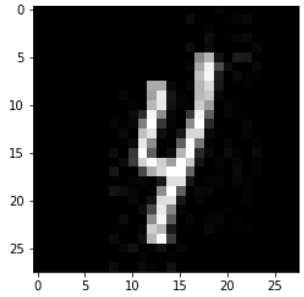
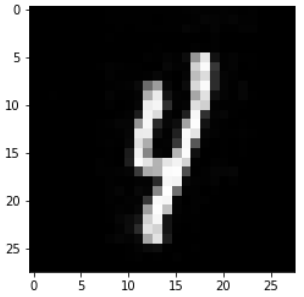
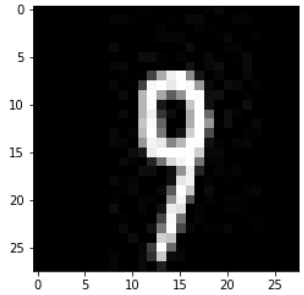
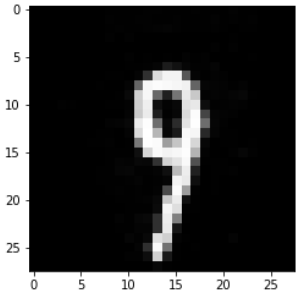
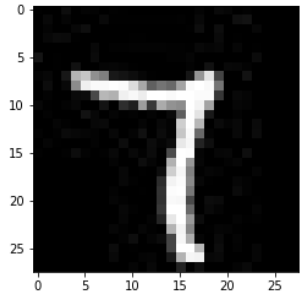
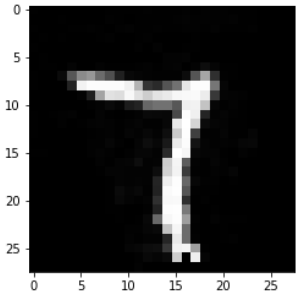
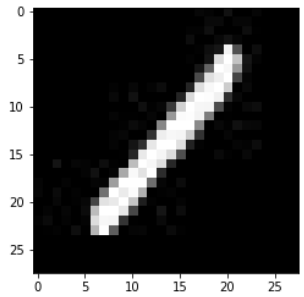
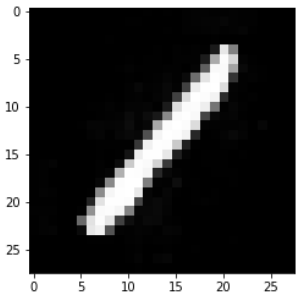
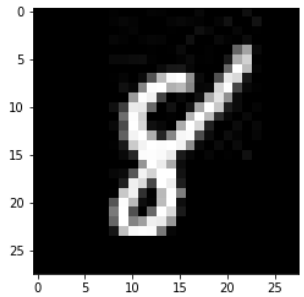
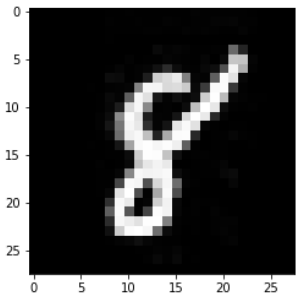
DATA FROM TRAINING SET

Original	Reconstructed
	
	
	
	
	

DATA FROM VALIDATION SET

Original	Reconstructed
	
	
	
	
	

DATA FROM TEST SET

Original	Reconstructed
	
	
	
	
	

(2) Three Hidden Layer - Autoencoder

Presented five different architectures with their results and best architecture is chosen based on the validation MSE loss and presented the results and accuracy parameters.

Architecture	Input layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	784	144	100	144	784
2	784	100	64	100	784
3	784	225	64	225	784
4	784	256	100	256	784
5	784	324	144	324	784

The hyper parameters are chosen and the Adam optimiser is used with sigmoidal activation.

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

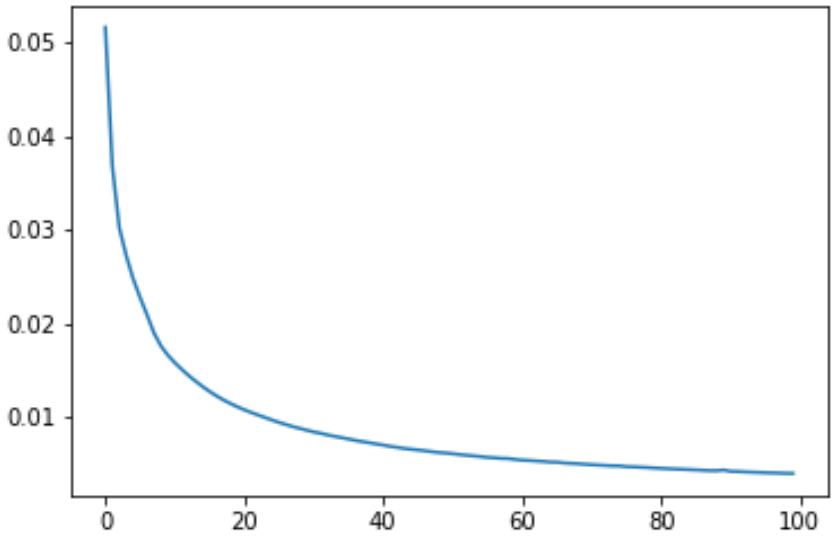
ARCHITECTURE 1

```
Model: "Autoencoder31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_5 (Flatten)         (None, 784)               0

 dense_10 (Dense)            (None, 144)               113040

 dense_11 (Dense)            (None, 100)               14500

 dense_12 (Dense)            (None, 144)               14544

 dense_13 (Dense)            (None, 784)               113680

 reshape_5 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 255,764
Trainable params: 255,764
Non-trainable params: 0
_________________________________________________________________
```

RESULTS

```
Epoch 54/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0053 - accuracy: 0.6724 - val_loss: 0.0086 - val_accuracy: 0.6709
Epoch 55/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0077 - val_accuracy: 0.6711
Epoch 56/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0053 - accuracy: 0.6724 - val_loss: 0.0087 - val_accuracy: 0.6707
Epoch 57/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0058 - accuracy: 0.6723 - val_loss: 0.0102 - val_accuracy: 0.6704
Epoch 58/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0056 - accuracy: 0.6724 - val_loss: 0.0079 - val_accuracy: 0.6710
```

Error Vs Epoch



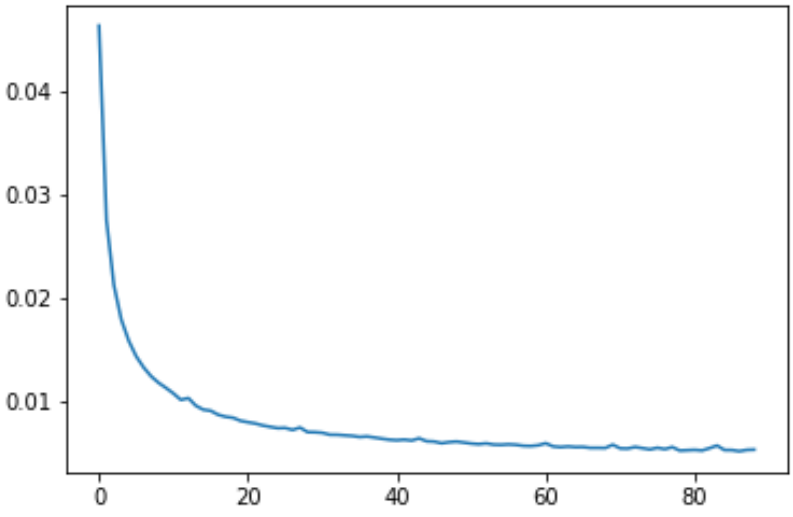
ARCHITECTURE 2

```
Model: "Autoencoder32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_6 (Flatten)         (None, 784)               0

 dense_14 (Dense)            (None, 100)               78500

 dense_15 (Dense)            (None, 64)                6464

 dense_16 (Dense)            (None, 100)               6500

 dense_17 (Dense)            (None, 784)               79184

 reshape_6 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 170,648
Trainable params: 170,648
Non-trainable params: 0
_________________________________________________________________
```

RESULTS

```
Epoch 85/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0077 - val_accuracy: 0.6710
Epoch 86/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0077 - val_accuracy: 0.6711
Epoch 87/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0053 - accuracy: 0.6724 - val_loss: 0.0078 - val_accuracy: 0.6711
Epoch 88/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0093 - val_accuracy: 0.6706
Epoch 89/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0055 - accuracy: 0.6724 - val_loss: 0.0088 - val_accuracy: 0.6708
```

Error Vs Epoch



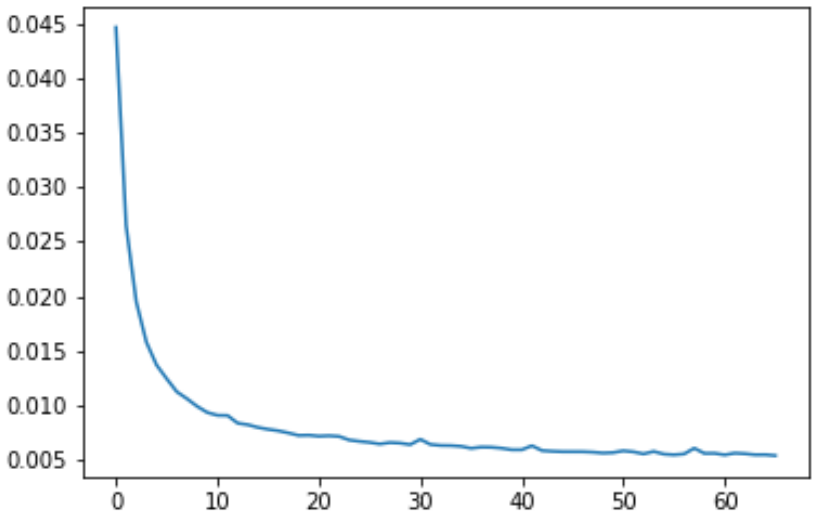
ARCHITECTURE 3

```
Model: "Autoencoder33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_9 (Flatten)         (None, 784)               0

 dense_22 (Dense)            (None, 225)               176625

 dense_23 (Dense)            (None, 64)                14464

 dense_24 (Dense)            (None, 225)               14625

 dense_25 (Dense)            (None, 784)               177184

 reshape_9 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 382,898
Trainable params: 382,898
Non-trainable params: 0
_________________________________________________________________
```

RESULTS

```
Epoch 62/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0055 - accuracy: 0.6724 - val_loss: 0.0085 - val_accuracy: 0.6708
Epoch 63/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0055 - accuracy: 0.6724 - val_loss: 0.0083 - val_accuracy: 0.6709
Epoch 64/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0082 - val_accuracy: 0.6709
Epoch 65/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0083 - val_accuracy: 0.6709
Epoch 66/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0053 - accuracy: 0.6724 - val_loss: 0.0086 - val_accuracy: 0.6708
```

Error Vs Epoch



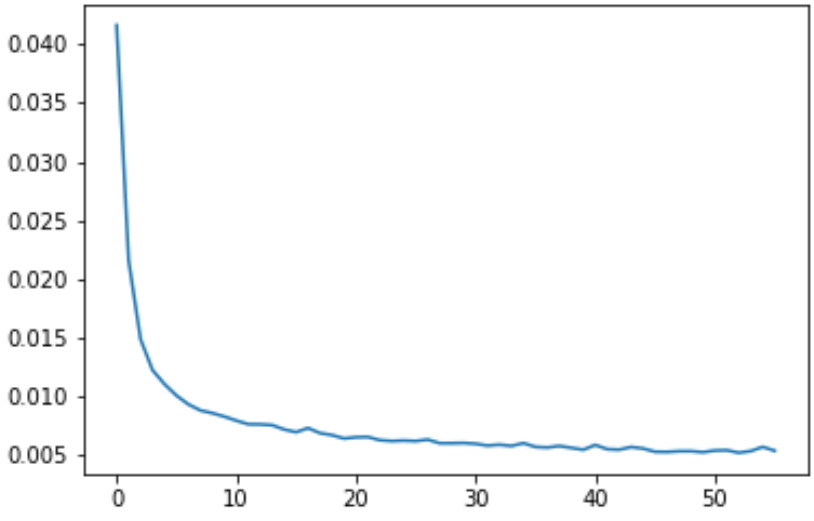
ARCHITECTURE 4

```
Model: "Autoencoder34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_7 (Flatten)         (None, 784)               0

 dense_18 (Dense)            (None, 256)               200960

 dense_19 (Dense)            (None, 100)               25700

 dense_20 (Dense)            (None, 256)               25856

 dense_21 (Dense)            (None, 784)               201488

 reshape_7 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 454,004
Trainable params: 454,004
Non-trainable params: 0
_________________________________________________________________
```

RESULTS

```
Epoch 57/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0054 - accuracy: 0.6724 - val_loss: 0.0087 - val_accuracy: 0.6708
Epoch 58/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0052 - accuracy: 0.6725 - val_loss: 0.0083 - val_accuracy: 0.6709
Epoch 59/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0052 - accuracy: 0.6725 - val_loss: 0.0089 - val_accuracy: 0.6708
Epoch 60/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0053 - accuracy: 0.6724 - val_loss: 0.0082 - val_accuracy: 0.6709
Epoch 61/100
358/358 [==============================] - 2s 5ms/step - loss: 0.0052 - accuracy: 0.6725 - val_loss: 0.0092 - val_accuracy: 0.6706
```

Error Vs Epoch



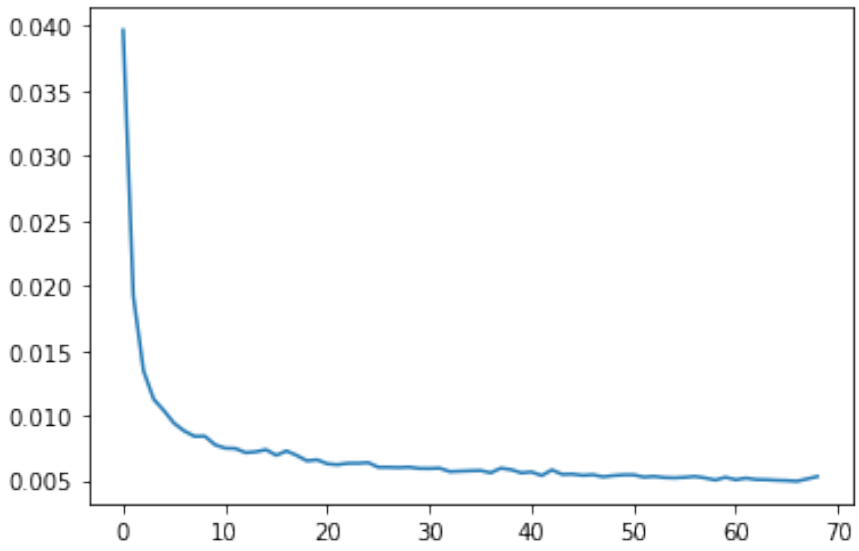
ARCHITECTURE 5

```
Model: "Autoencoder35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_8 (Flatten)         (None, 784)               0

 dense_22 (Dense)            (None, 324)               254340

 dense_23 (Dense)            (None, 144)               46800

 dense_24 (Dense)            (None, 324)               46980

 dense_25 (Dense)            (None, 784)               254800

 reshape_8 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 602,920
Trainable params: 602,920
Non-trainable params: 0
_________________________________________________________________
```

RESULTS

```
Epoch 65/100
358/358 [==============================] - 3s 7ms/step - loss: 0.0050 - accuracy: 0.6725 - val_loss: 0.0090 - val_accuracy: 0.6707
Epoch 66/100
358/358 [==============================] - 3s 8ms/step - loss: 0.0050 - accuracy: 0.6725 - val_loss: 0.0091 - val_accuracy: 0.6707
Epoch 67/100
358/358 [==============================] - 2s 7ms/step - loss: 0.0050 - accuracy: 0.6725 - val_loss: 0.0091 - val_accuracy: 0.6707
Epoch 68/100
358/358 [==============================] - 3s 7ms/step - loss: 0.0051 - accuracy: 0.6725 - val_loss: 0.0094 - val_accuracy: 0.6706
Epoch 69/100
358/358 [==============================] - 3s 7ms/step - loss: 0.0053 - accuracy: 0.6724 - val_loss: 0.0093 - val_accuracy: 0.6706
```

Error Vs Epoch



OBSERVATION

Architecture	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	No of epoch	Training MSE error	Validation MSE error
1	144	100	144	58	0.0056	0.0079
2	100	64	100	89	0.0055	0.0088
3	225	64	225	66	0.0053	0.0086
4	256	100	256	61	0.0052	0.0092
5	324	144	324	69	0.0053	0.0093

Considered five under complete and five over complete architecture. It is observed that the over complete architecture does not perform any compression but just simple copies the input. Have not included the over complete architecture since it is compression task. Out of the under complete architecture, best is chosen based on validation MSE error.

BEST ARCHITECTURE

Architecture 1 is the best that works on the given data in this case with the validation error of **0.0079**.

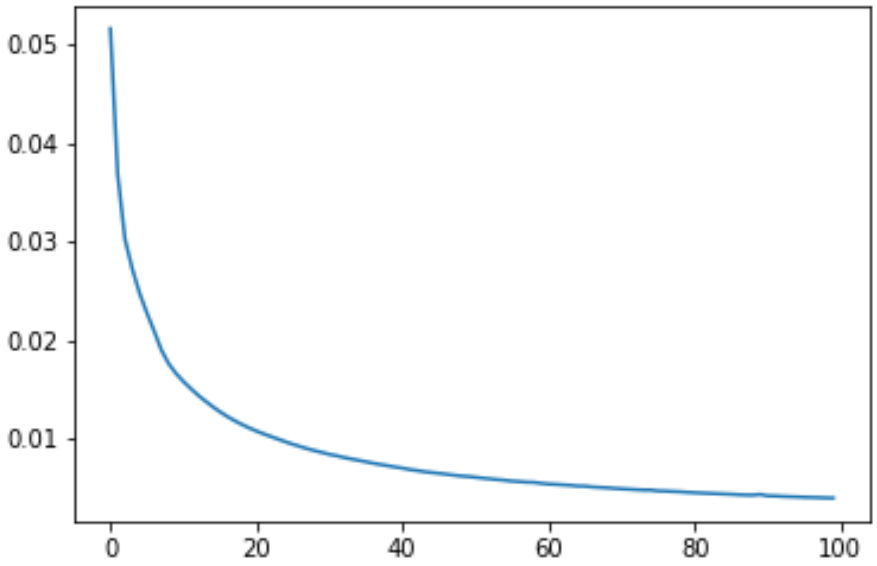
```
Model: "Autoencoder31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_5 (Flatten)         (None, 784)               0

 dense_10 (Dense)            (None, 144)               113040

 dense_11 (Dense)            (None, 100)               14500

 dense_12 (Dense)            (None, 144)               14544

 dense_13 (Dense)            (None, 784)               113680

 reshape_5 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 255,764
Trainable params: 255,764
Non-trainable params: 0
_________________________________________________________________
```

Reconstruction Errors:

Training Loss	0.0056
Validation Loss	0.0079
Test Reconstruction Error	0.0082

Error Vs Epoch Graph:

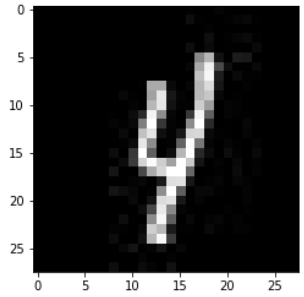
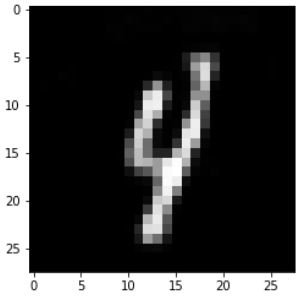
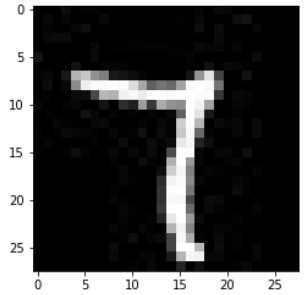
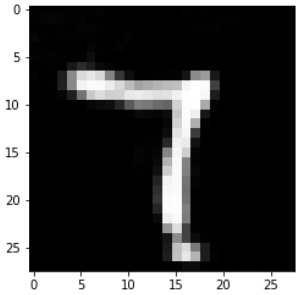
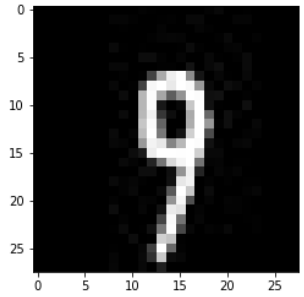
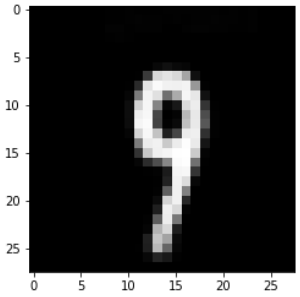
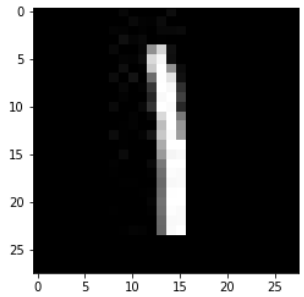
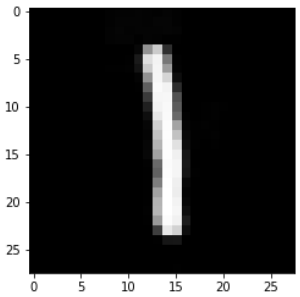
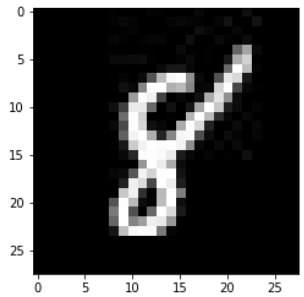
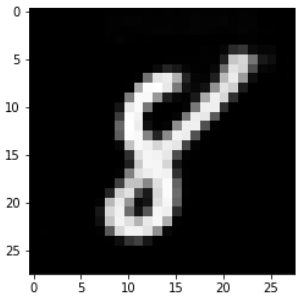
The architecture took 58 number of epoch to converge with a patience level of 10



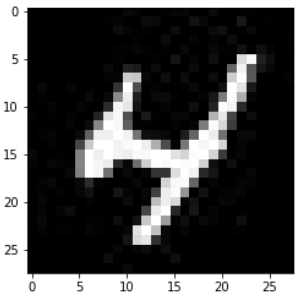
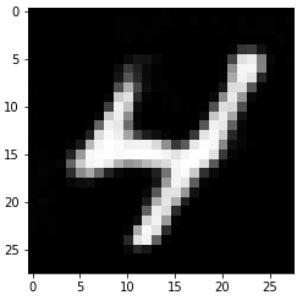
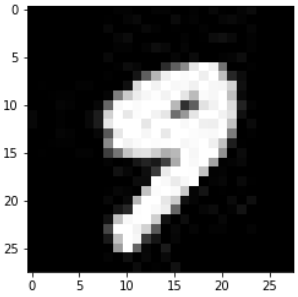
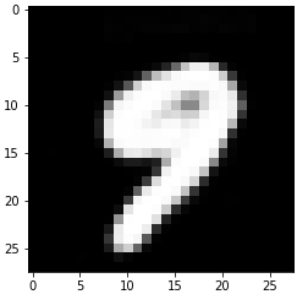
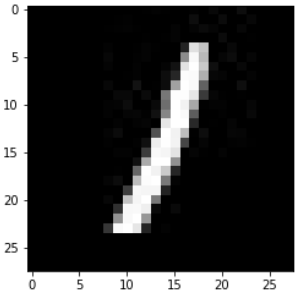
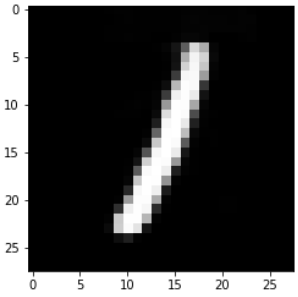
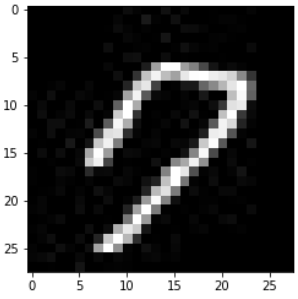
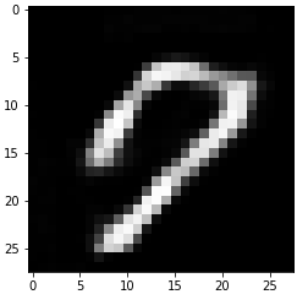
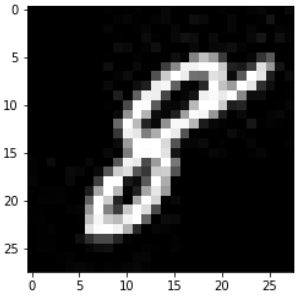
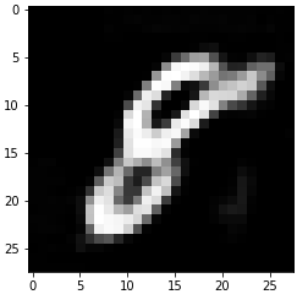
Confusion Matrix:

		ACTUAL CLASS				
		1	**4**	**7**	**8**	**9**
PREDICTED CLASS	**1**	754	0	3	8	1
	4	4	744	5	6	20
	7	9	6	728	3	13
	8	10	5	2	733	9
	9	6	24	13	13	723

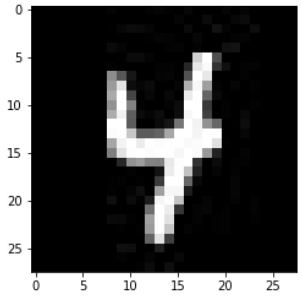
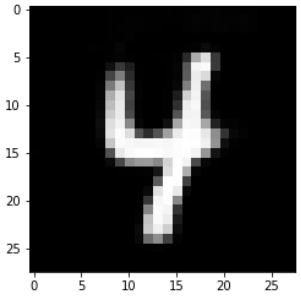
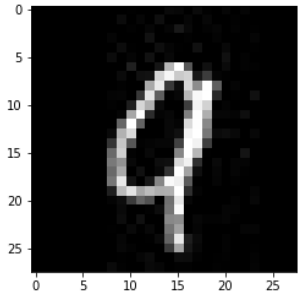
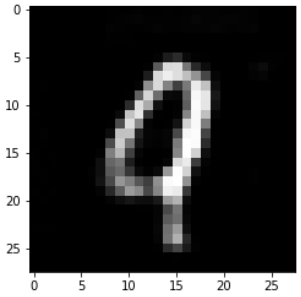
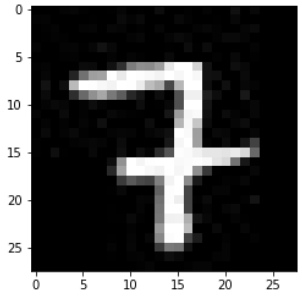
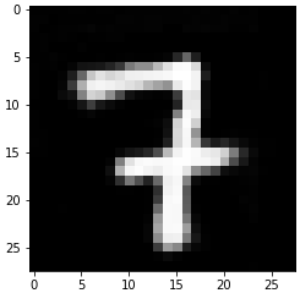
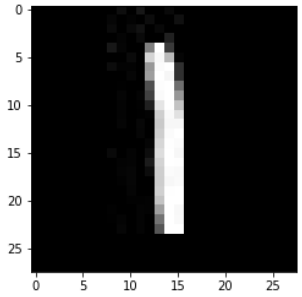
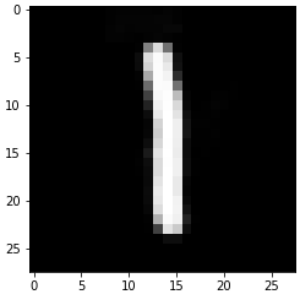
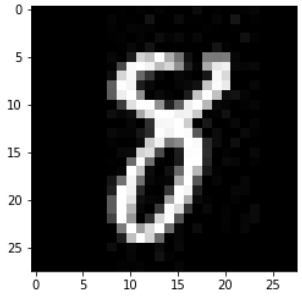
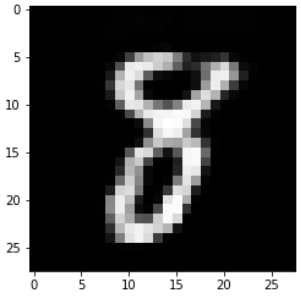
DATA FROM TRAINING SET

Original	Reconstructed
	
	
	
	
	

DATA FROM VALIDATION SET

Original	Reconstructed
	
	
	
	
	

DATA FROM TEST SET

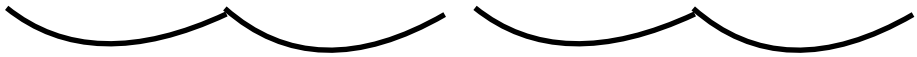
Original	Reconstructed
	
	
	
	
	

CLASSIFICATION

(1) Single Hidden Layer - Autoencoder

Autoencoder can be used as classifier by chopping of the output layer and by introducing FCNN classification model. The data is tested with sufficient number of architectures and three is presented as example.

Architecture	Input layer	AE Hidden layer	FCNN Hidden Layer 1	FCNN Hidden Layer 2	FCNN Hidden Layer 3	Output Layer
1	784	256	361	289	361	5
2	784	256	289	256	256	5
3	784	256	361	289	400	5



AUTOENCODER **FCNN**

Three layer FCNN with different architecture is considered. Using the compressed data, the classification is performed and results are obtained. The parameters considered for the FCNN as well as the auto encoder is as follows.

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

ARCHITECTURE 1

No of neurons in auto encoder hidden layer:	256

FCNN	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	256	361	289	361	5

RESULTS

```
Epoch 46/10000
358/358 [==============================] - 1s 4ms/step - loss: 8.4313e-04 - accuracy: 0.9999
Epoch 47/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0355 - accuracy: 0.9891
Epoch 48/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0046 - accuracy: 0.9987
Epoch 49/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0060 - accuracy: 0.9981
Epoch 50/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0102 - accuracy: 0.9968
```

No of epoch for convergence: **50**
Training loss: **0.0102**
Training Accuracy: **0.9968**

```
121/121 [==============================] - 1s 3ms/step - loss: 0.1121 - accuracy: 0.9787
```

Validation Accuracy: **0.9787**

ARCHITECTURE 2

No of neurons in auto encoder hidden layer:	256

FCNN	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	256	289	256	256	5

RESULTS

```
Epoch 73/10000
358/358 [==============================] - 3s 7ms/step - loss: 1.7794e-05 - accuracy: 1.0000
Epoch 74/10000
358/358 [==============================] - 1s 4ms/step - loss: 1.4675e-05 - accuracy: 1.0000
Epoch 75/10000
358/358 [==============================] - 1s 4ms/step - loss: 1.2283e-05 - accuracy: 1.0000
Epoch 76/10000
358/358 [==============================] - 1s 4ms/step - loss: 1.0400e-05 - accuracy: 1.0000
Epoch 77/10000
358/358 [==============================] - 1s 4ms/step - loss: 8.4387e-06 - accuracy: 1.0000
```

No of epoch for convergence: **77**
Training loss: **0.000008**
Training Accuracy: **1.0000**

```
121/121 [==============================] - 0s 3ms/step - loss: 0.1529 - accuracy: 0.9785
```

Validation Accuracy: **0.9785**

ARCHITECTURE 3

No of neurons in auto encoder hidden layer:	**256**

FCNN	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	256	361	289	400	5

RESULTS

```
Epoch 67/10000
358/358 [==============================] - 2s 4ms/step - loss: 1.8285e-05 - accuracy: 1.0000
Epoch 68/10000
358/358 [==============================] - 2s 5ms/step - loss: 1.4541e-05 - accuracy: 1.0000
Epoch 69/10000
358/358 [==============================] - 2s 6ms/step - loss: 1.1892e-05 - accuracy: 1.0000
Epoch 70/10000
358/358 [==============================] - 2s 6ms/step - loss: 9.8415e-06 - accuracy: 1.0000
Epoch 71/10000
358/358 [==============================] - 3s 7ms/step - loss: 8.0425e-06 - accuracy: 1.0000
```

No of epoch for convergence: **71**
Training loss: **0.000008**
Training Accuracy: **1.0000**

```
121/121 [==============================] - 1s 3ms/step - loss: 0.1588 - accuracy: 0.9774
```

Validation Accuracy: **0.9774**

SUMMARY

Architecture	AE Hidden	FCNN H1	FCNN H2	FCNN H3	No of epoch	Training Accuracy	Validation Accuracy
1	256	361	289	361	50	0.9968	0.9787
2	256	289	256	256	77	1.00	0.9785
3	256	361	289	400	71	1.00	0.9774

BEST ARCHITECTURE

The Architecture 1 give the best validation accuracy of **0.9787**.
No of epoch for convergence: **50**
Training loss: **0.0102**
Training Accuracy: **0.9968**

Test Accuracy: **0.9769**

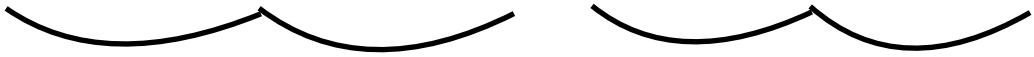
COMPARISON WITH FCNN USING OPTIMIZERS

The result obtained using auto encoder gives better performance than the traditional FCNN even using optimisers. In the first part, it is observed that the best accuracy obtained was 0.96. Here the the accuracy is improved and speed, test accuracy and even epoch is improved considerably.

(2) Three Hidden Layer - Autoencoder

Autoencoder can be used as classifier by chopping of the output layer and by introducing FCNN classification model. The data is tested with sufficient number of architectures and three is presented as example.

Architecture	Input layer	AE Hidden layer 1	AE Hidden layer 2	AE Hidden layer 3	FCNN Hidden Layer 1	FCNN Hidden Layer 2	FCNN Hidden Layer 3	Output Layer
1	784	144	100	144	289	361	289	5
2	784	144	100	144	256	289	256	5
3	784	144	100	144	289	361	400	5



AUTOENCODER **FCNN**

Three layer FCNN with different architecture is considered. Using the compressed data, the classification is performed and results are obtained. The parameters considered for the FCNN as well as the auto encoder is as follows.

Parameters	Values Taken
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}
Activation (Hidden)	Sigmoidal
Activation (Output)	Softmax
Loss	Cross Entropy

ARCHITECTURE 1

No of neurons in auto encoder hidden layer 1:	**144**
No of neurons in auto encoder hidden layer 2:	**100**
No of neurons in auto encoder hidden layer 3:	**144**

FCNN	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	100	289	361	289	5

RESULTS

```
Epoch 201/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0692 - accuracy: 0.9745
Epoch 202/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0697 - accuracy: 0.9743
Epoch 203/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0754 - accuracy: 0.9709
Epoch 204/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0666 - accuracy: 0.9746
Epoch 205/10000
358/358 [==============================] - 2s 5ms/step - loss: 0.0671 - accuracy: 0.9744
```

No of epoch for convergence: **205**
Training loss: **0.0671**
Training Accuracy: **0.9744**

```
121/121 [==============================] - 1s 3ms/step - loss: 0.1169 - accuracy: 0.9655
```

Validation Accuracy: **0.9655**

ARCHITECTURE 2

No of neurons in auto encoder hidden layer 1:	**144**
No of neurons in auto encoder hidden layer 2:	**100**
No of neurons in auto encoder hidden layer 3:	**144**

FCNN	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	100	256	289	256	5

RESULTS

```
Epoch 104/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1792 - accuracy: 0.9390
Epoch 105/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1675 - accuracy: 0.9433
Epoch 106/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1532 - accuracy: 0.9454
Epoch 107/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1526 - accuracy: 0.9471
Epoch 108/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1508 - accuracy: 0.9470
```

No of epoch for convergence: **108**
Training loss: **0.1508**
Training Accuracy: **0.9470**

```
121/121 [==============================] - 1s 3ms/step - loss: 0.1334 - accuracy: 0.9523
```

Validation Accuracy: **0.9523**

ARCHITECTURE 3

No of neurons in auto encoder hidden layer 1:	**144**
No of neurons in auto encoder hidden layer 2:	**100**
No of neurons in auto encoder hidden layer 3:	**144**

FCNN	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	100	289	361	400	5

RESULTS

```
Epoch 209/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1091 - accuracy: 0.9641
Epoch 210/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1036 - accuracy: 0.9639
Epoch 211/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.1005 - accuracy: 0.9655
Epoch 212/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0913 - accuracy: 0.9690
Epoch 213/10000
358/358 [==============================] - 1s 4ms/step - loss: 0.0856 - accuracy: 0.9707
```

No of epoch for convergence: **213**
Training loss: **0.0856**
Training Accuracy: **0.9707**

```
121/121 [==============================] - 1s 3ms/step - loss: 0.1227 - accuracy: 0.9595
```

Validation Accuracy: **0.9595**

SUMMARY

Architecture	AE H1	AE H2	AE H3	FCNN H1	FCNN H2	FCNN H3	No of epoch	Tr Acc.	Val Acc.
1	144	100	144	289	361	289	205	0.974	0.965
2	144	100	144	256	289	256	108	0.947	0.952
3	144	100	144	289	361	400	213	0.970	0.959

BEST ARCHITECTURE

The Architecture 1 give the best validation accuracy of **0.9759**.
No of epoch for convergence: **213**
Training loss: **0.0856**
Training Accuracy: **0.9707**

Test Accuracy: **0.9584**

COMPARISON WITH FCNN USING OPTIMIZERS & SINGLE HIDDEN LAYER AUTOENCODER

The result obtained using auto encoder gives better performance than the traditional FCNN even using optimisers. In the first part, it is observed that the best accuracy obtained was 0.96. Here the the accuracy is improved and speed, test accuracy and even epoch is improved considerably but still for this data single hidden layer auto encoder compression works better and gives suitable results.

WEIGHT VISUALIZATION

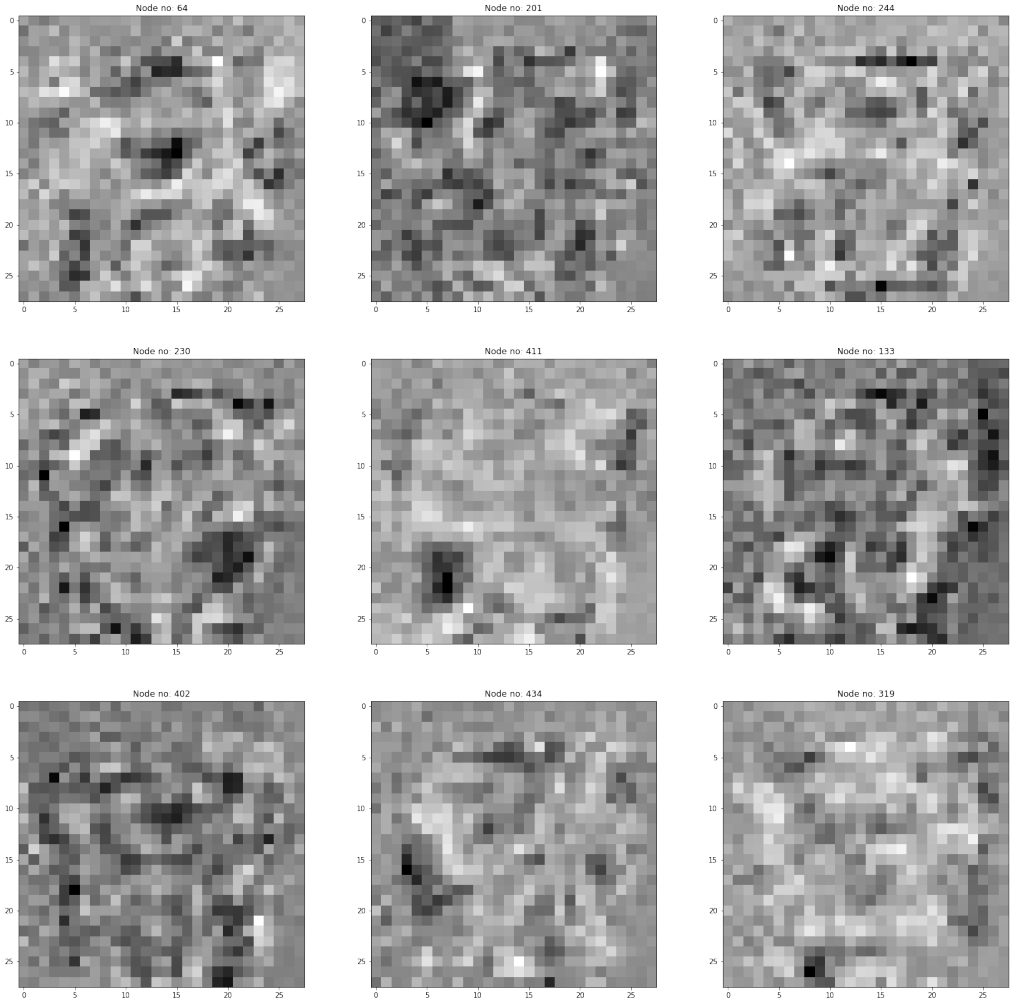
For the best compressed representation in one hidden layer autoencoder, the inputs as images that maximally activate each of the neurons of the hidden representations is plotted. (weights from the input layer to the compressed layer)

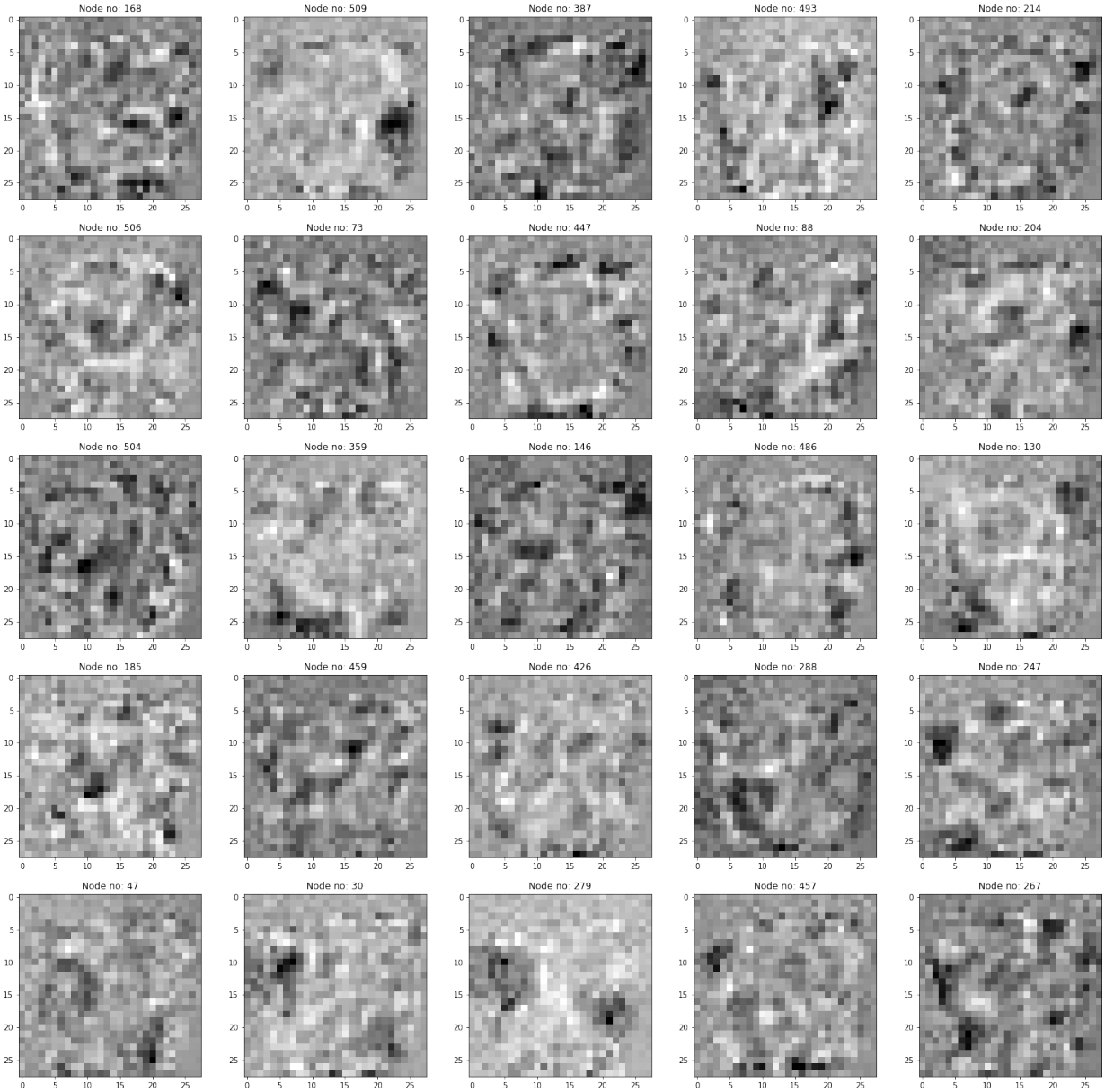
The neurons will be maximally activated when,

$$max \qquad w_j^T x_n \qquad \text{where.,}$$

$$x_n = \frac{w_j}{\sqrt{w_j^T w_j}}$$

These cause hidden neuron to maximally fire. Plotting these x_n as images which maximally activate each of the neurons of the hidden representations learned by anautoencoder, we get




Node no: 168
Node no: 509
Node no: 387
Node no: 493
Node no: 214
Node no: 506
Node no: 73
Node no: 447
Node no: 88
Node no: 204
Node no: 504
Node no: 359
Node no: 146
Node no: 486
Node no: 130
Node no: 185
Node no: 459
Node no: 426
Node no: 288
Node no: 247
Node no: 47
Node no: 30
Node no: 279
Node no: 457
Node no: 267

DENOISING AUTOENCODER

A denoising encoder simply corrupts the input data using a probabilistic process before feeding it to the auto encoder network. With a probability of q, a Gaussian noise is added to ith input value (x_{ni}),

$$\hat{x}_{ni} = x_{ni} + N(0,1)$$

Not all the input variables (features) in an example is corrupted by noise. By this, at each epoch, there is a chance that different set of values in the same example will be corrupted by noise. This is done to address the issue of overfitting. The denoising autoencoder learn many meaningful patterns, As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a pattern.

Python Implementation

Here the value of q, probabilities is taken as 0.2 and 0.4 with the best one hidden layer architecture chosen above.

The best Architecture of auto encoder compression obtained was with:

No of Hidden neuron: 256
Validation error of **0.0030**.
Training loss: **0.0020**
Test loss: **0.0031**

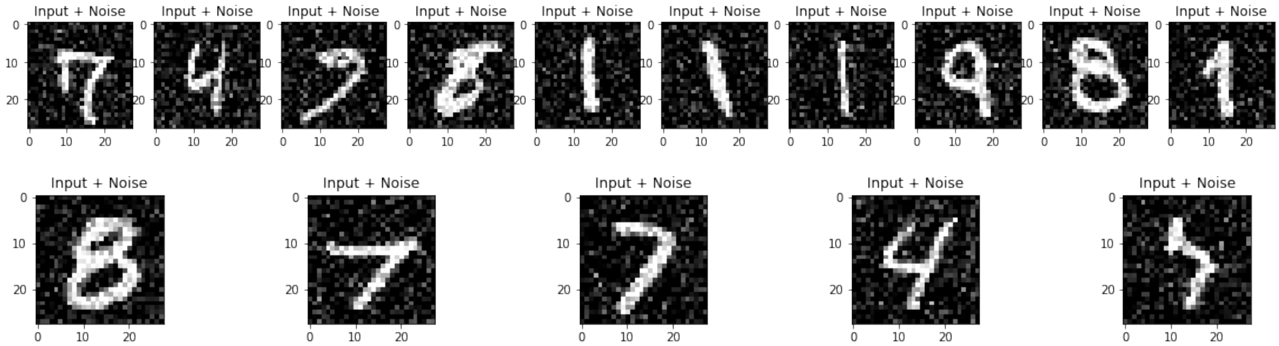
The best Architecture of auto encoder classification obtained was with:

Validation accuracy of **0.9787**
Training loss: **0.0102**
Training Accuracy: **0.9968**

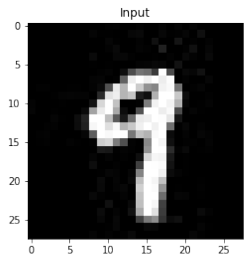
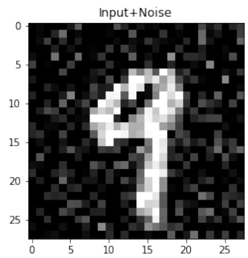
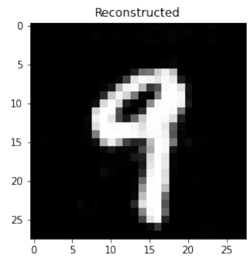
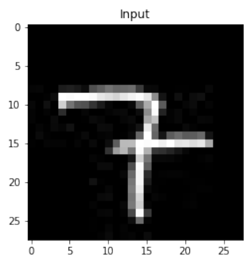
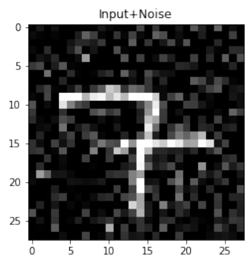
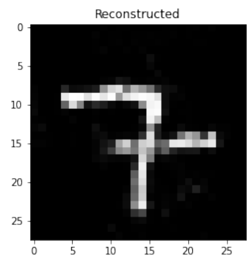
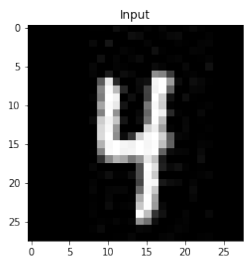
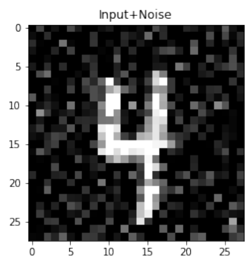
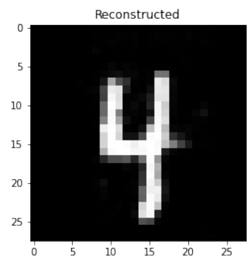
Test Accuracy: **0.9769**

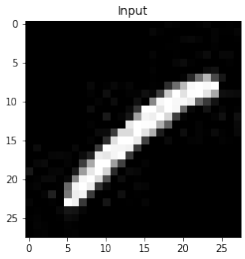
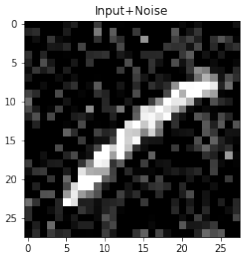
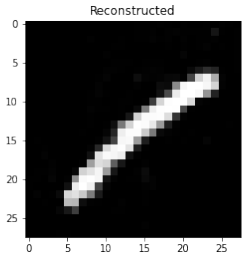
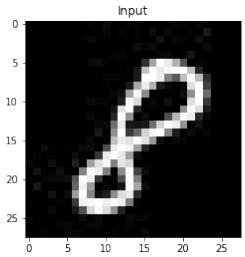
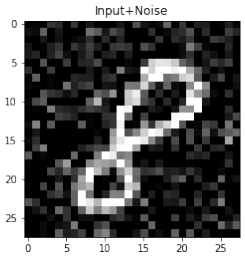
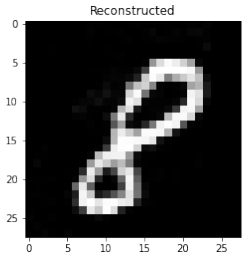
(1) Noise factor of 20%

Some sample inputs corrupted by noise is shown as example with gaussian noise 20%.



RESULTS

Original	Noisy	Reconstructed
		
		
		

Original	Noisy	Reconstructed
		
		

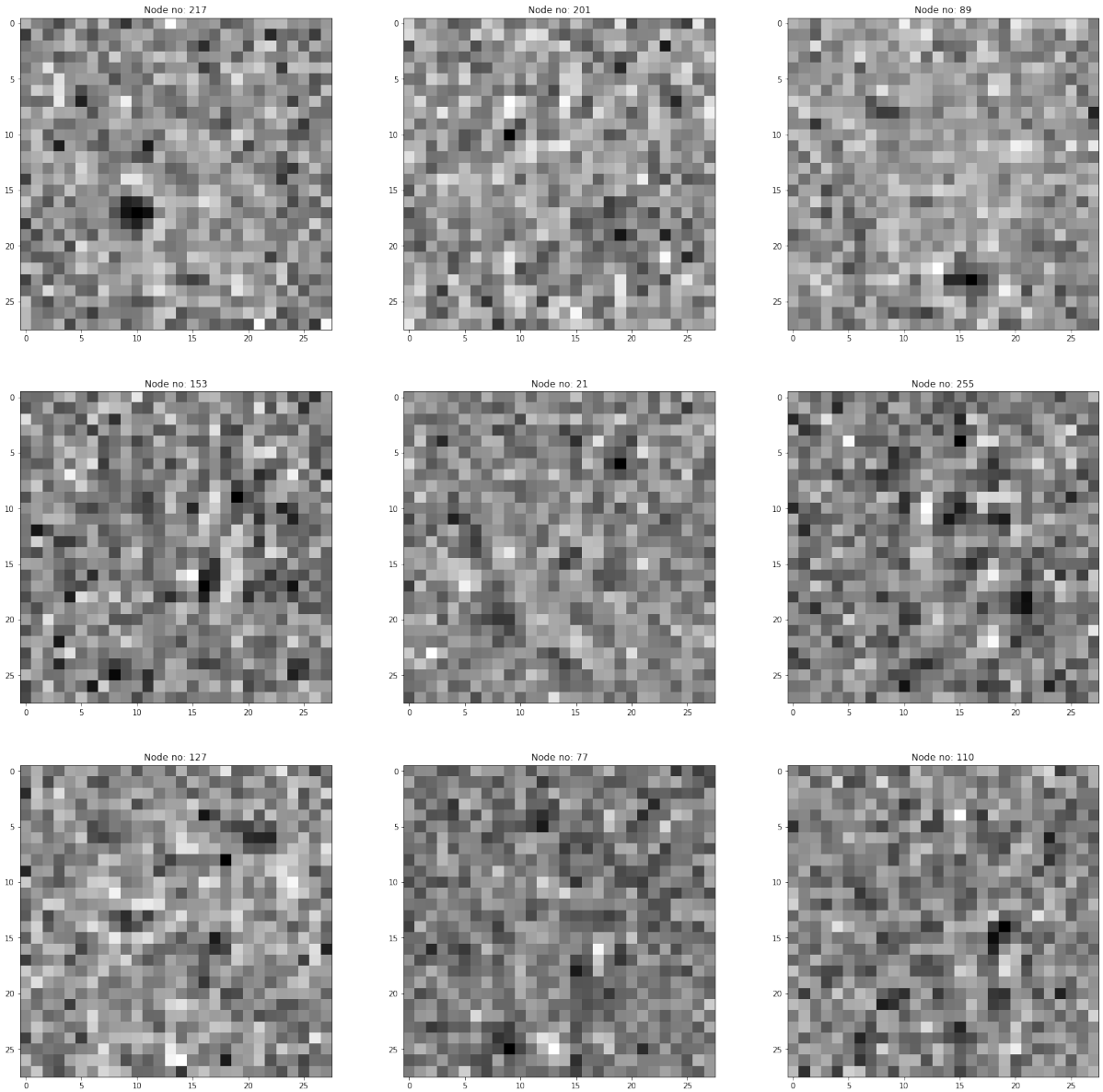
ACCURACY PARAMETERS & COMPARISON

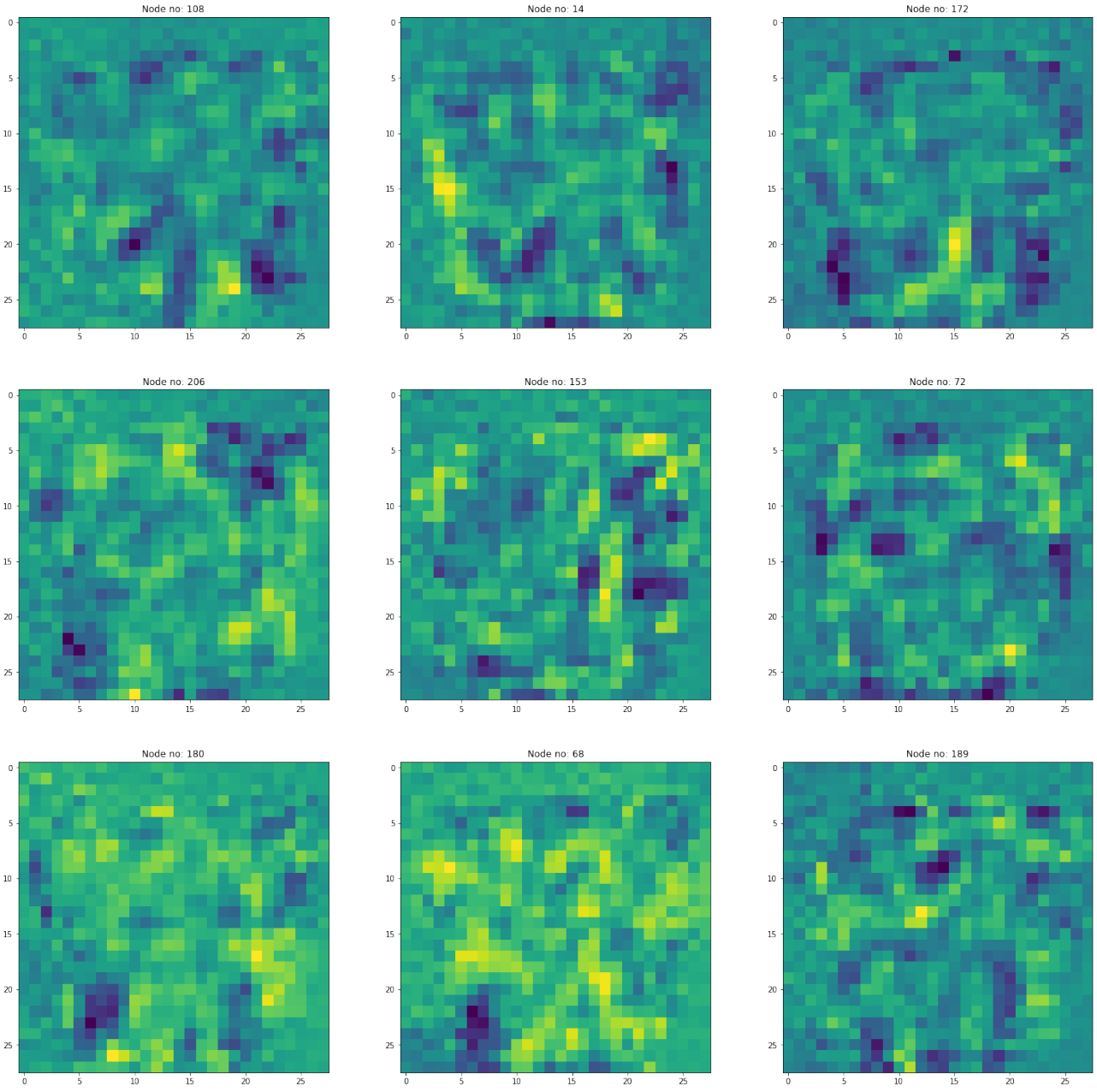
Training Accuracy	95.28%
Classification Accuracy (With Noise)	
Validation	95.67%
Test	97.18%
Classification Accuracy (Without Noise)	
Validation	96.44%
Test	97.16%

Confusion Matrix:

ACTUAL CLASS						
PREDICTED CLASS		**1**	**4**	**7**	**8**	**9**
	1	752	0	3	4	0
	4	0	752	1	1	5
	7	4	5	722	3	25
	8	1	5	2	742	9
	9	3	20	11	5	720

WEIGHT VISUALIZATION



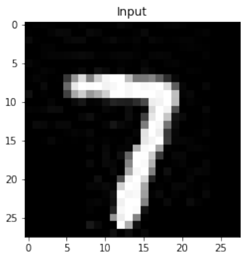
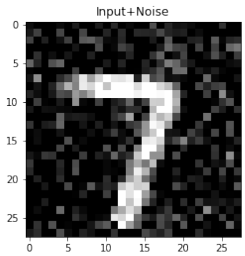
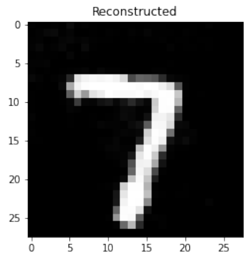
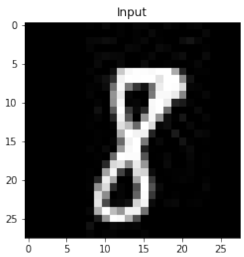
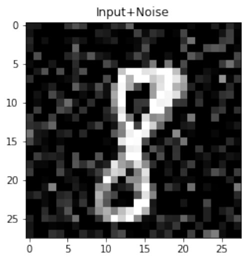
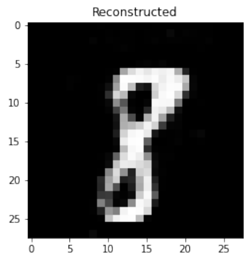
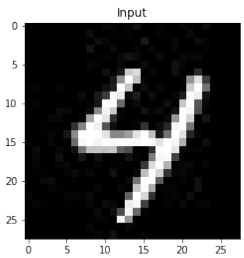
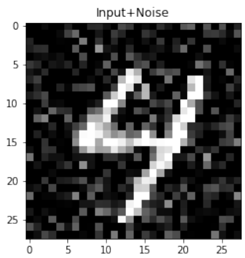
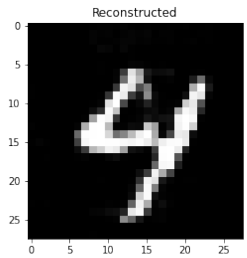
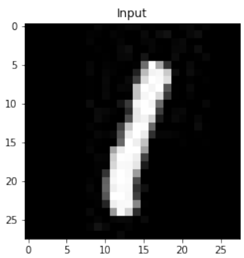
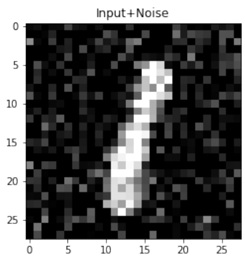
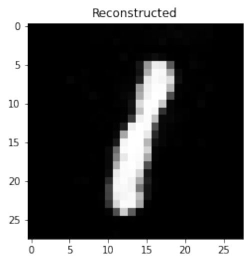


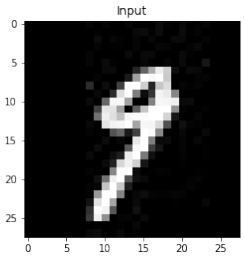
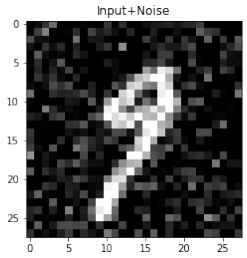
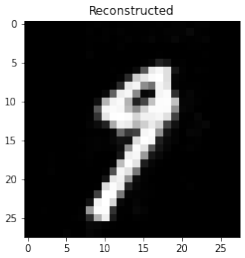
Comparing these weights with the obtained weights earlier, that is with and without noise: In without noise it is possible to see my structure or patterns clearly and there is high chance of model to memorise those pattern. In with noise, there are lots of irregularities and here too pattern is visible, but avoids overfitting by not letting the noise to memorise.

(1) Noise factor of 40%

Some sample inputs corrupted by noise is shown as example with gaussian noise 40%.

RESULTS

Original	Noisy	Reconstructed
		
		
		
		

Original	Noisy	Reconstructed
		

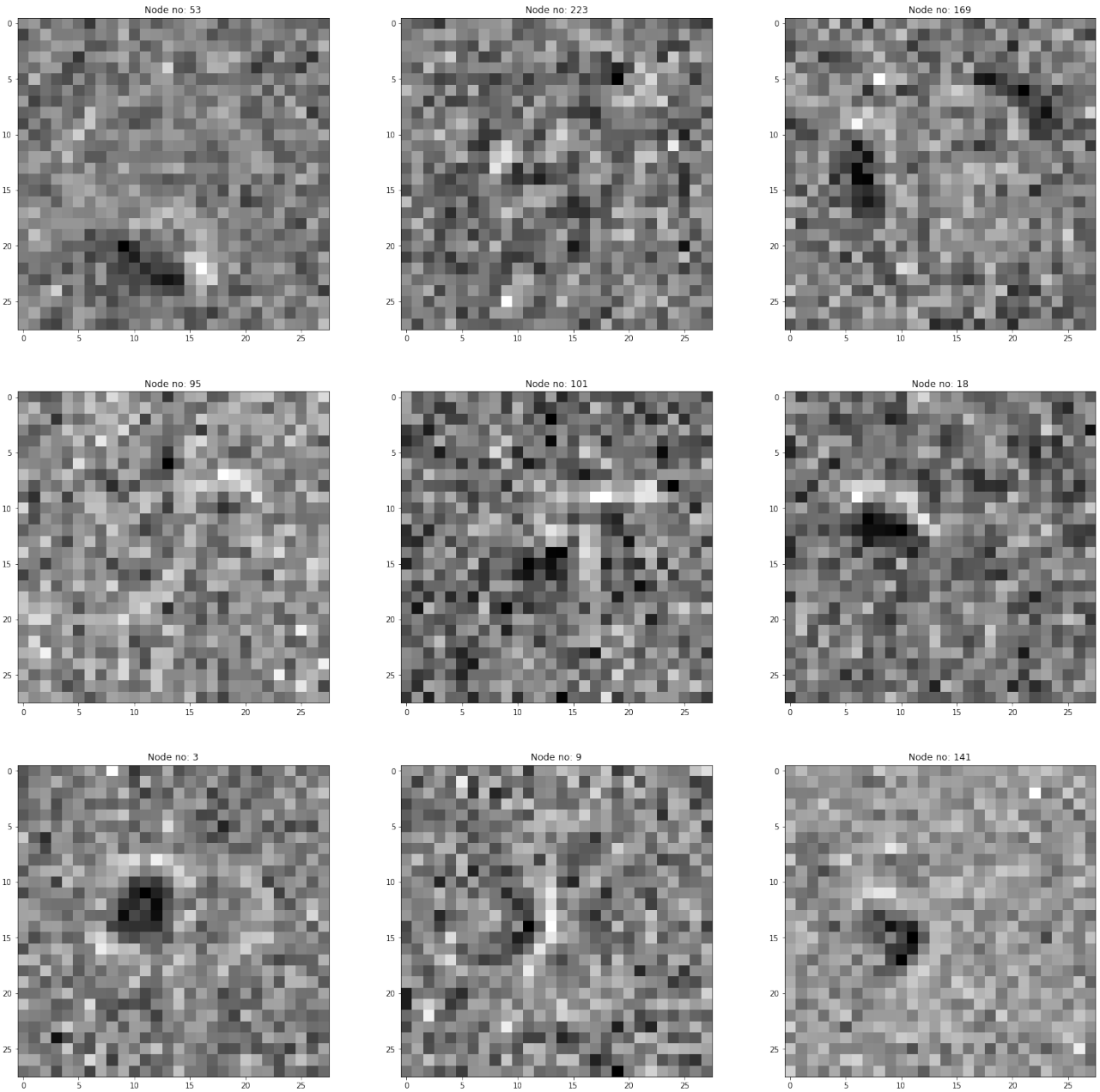
ACCURACY PARAMETERS & COMPARISON

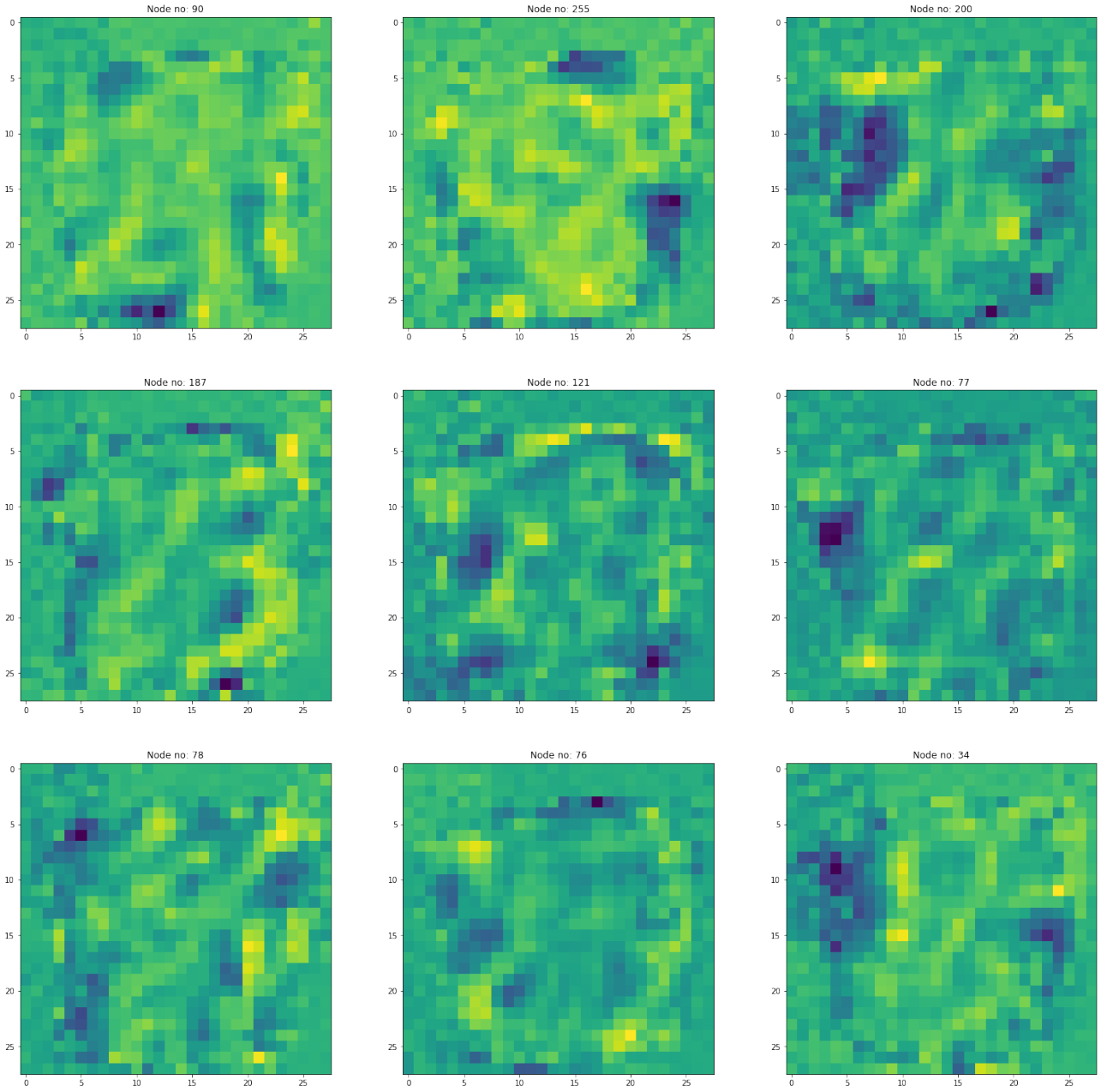
Training Accuracy	96.89%
Classification Accuracy (With Noise)	
Validation	96.89%
Test	97.07%
Classification Accuracy (Without Noise)	
Validation	96.44%
Test	97.16%

Confusion Matrix:

		ACTUAL CLASS				
		1	**4**	**7**	**8**	**9**
PREDICTED CLASS	**1**	749	0	3	7	0
	4	4	748	2	1	4
	7	5	6	740	1	7
	8	6	7	6	728	12
	9	2	17	16	5	719

WEIGHT VISUALIZATION





(Same observation). Comparing these weights with the obtained weights earlier, that is with and without noise: In without noise it is possible to see my structure or patterns clearly and there is high chance of model to memorise those pattern. In with noise, there are lots of irregularities and here too pattern is visible, but avoids overfitting by not letting the noise to memorise.

GITHUB LINK

https://github.com/Rajesh-Smartino/Deep-Learning

TEAM DETAILS
(Group 22)

Name	Rollno	Mailid
Rajesh R	S21005	s21005@students.iitmandi.ac.in
Naisarg Pandya	S21012	s21012@students.iitmandi.ac.in
Ashok Kumar	D19042	d19042@students.iitmandi.ac.in