Assignment 2

Deep Learning And Its Applications

Group 14

Members:

- Prashant Kumar (B19101)
- Lahari Talla (B19197)
- Ravi Kumar (B19191)

Abstract

This report is based on the assignment 2 of Deep Learning course which includes the optimizers for backpropagation algorithms and autoencoders. This report deals with two major tasks. One of the tasks is to train the FCNN using different optimizers for the backpropagation algorithm and compare the number of epochs that it takes for convergence along with their classification performance. Another major task includes building an autoencoder to obtain the hidden representation and use it for classification.

Data Set

We are given Training, Validation and Test Data. Each contains the subset of the MNIST digit dataset. Digits are 0, 2 , 3 ,7 ,8.

Each Image shape is 28 X 28

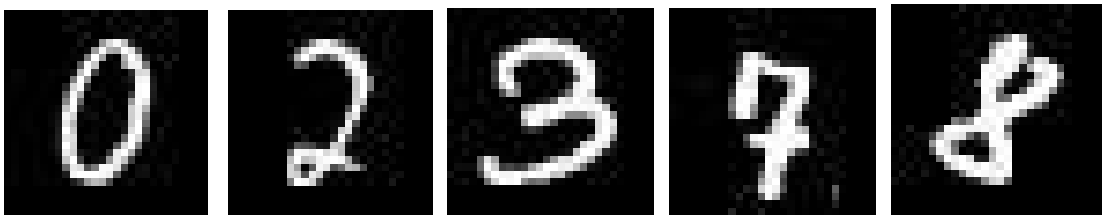


Fig. 1: Images showing Classes of the Data

FCNN with different Optimizers

Implement Feed Forward Neural Network with three hidden layers. Below are the five different architectures we used in this experiment.

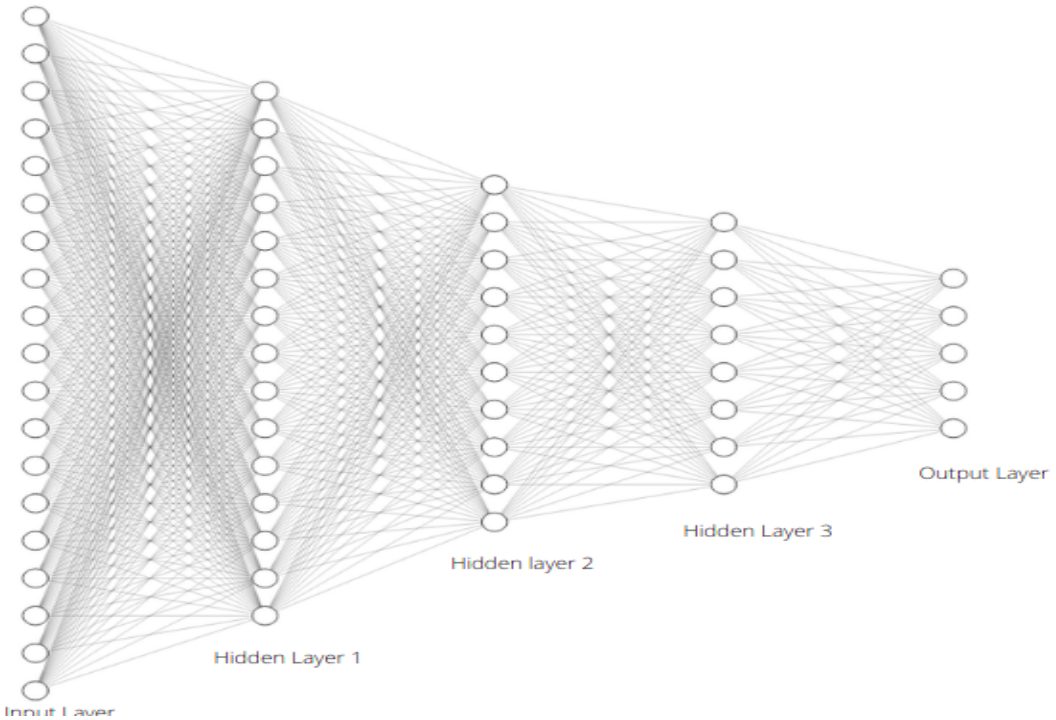


Fig. 2: Architecture of the FCNN

Different FCNN Architecture we used:

Architecture	Input Layer Neurons	Hidden Layer 1 Neurons	Hidden Layer 2 Neurons	Hidden Layer 3 Neurons	Output Layer Neurons
1	784	200	100	50	5
2	784	512	128	32	5
3	784	300	400	100	5
4	784	800	300	50	5
5	784	400	150	60	5

Table 1. Different FCNN architecture used in the assignment

Optimizers Used:

1. stochastic gradient descent (SGD) algorithm - (batch_size=1),
2. batch gradient descent algorithm (vanilla gradient descent) – (batch_size=total number of training examples)
3. SGD with momentum (NAG) – (batch_size=32),
4. RMSProp algorithm – (batch_size=32)
5. Adam optimizer – (batch_size=32).

1. SGD

Training Each Fcnn Architecture using stochastic gradient descent (SGD) algorithm

- Batch size = 1.
- Learning Rate : 0.01
- Convergence Criteria : difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture 1: [784, 200, 100, 50, 5]

A plot for the average error v/s Epoch

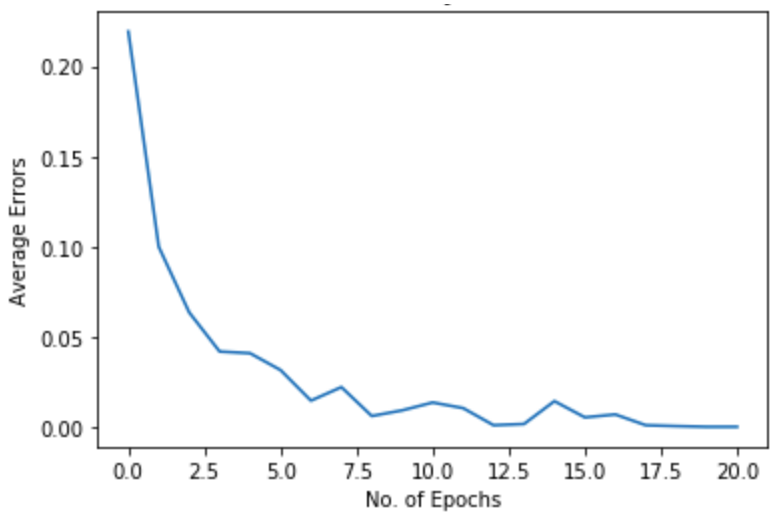


Fig. 3.: Average error vs Epoch for architecture 1 using SGD

Confusion Matrix for Training and Validation Data

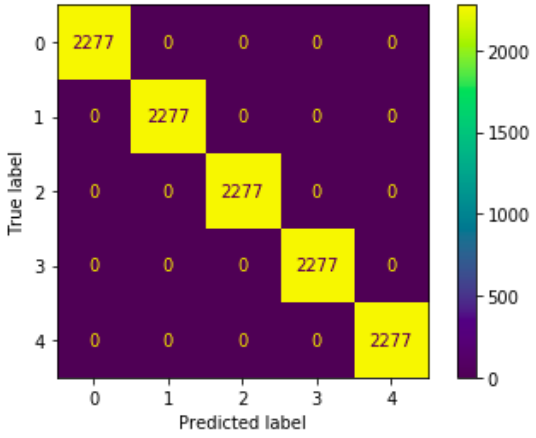


Fig. 4: Confusion matrix for training data for architecture 1 using SGD

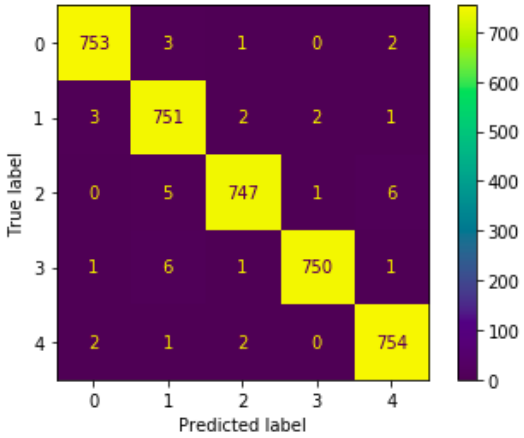


Fig. 5: confusion matrix of validation data for architecture 1 using SGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.94%

Architecture 2: [784, 512, 128, 32, 5]

A plot for the average error v/s Epoch

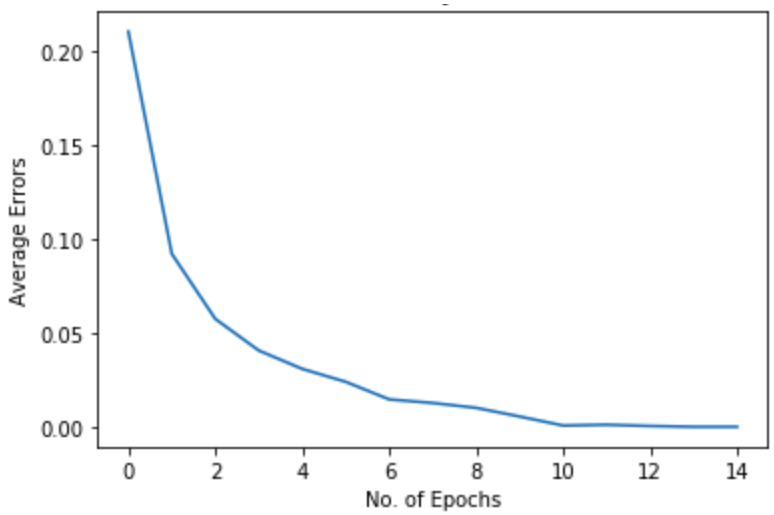


Fig. 6:. Average error vs Epoch for architecture 2 using SGD

Confusion Matrix for Training and Validation Data

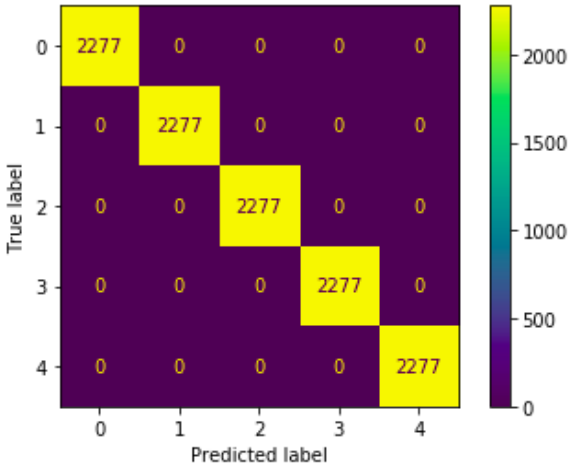


Fig. 7: Confusion matrix for training data architecture 1 using SGD

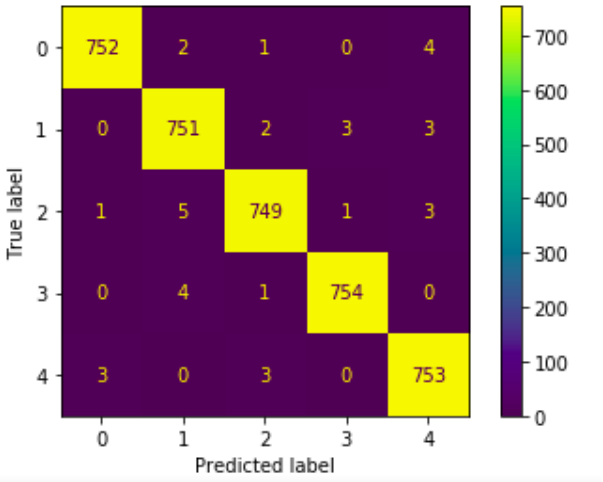


Fig. 8: confusion matrix of validation data for for architecture 1 using SGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 99.05%

Architecture 3: [784, 300, 400, 100, 5]

A plot for the average error v/s Epoch

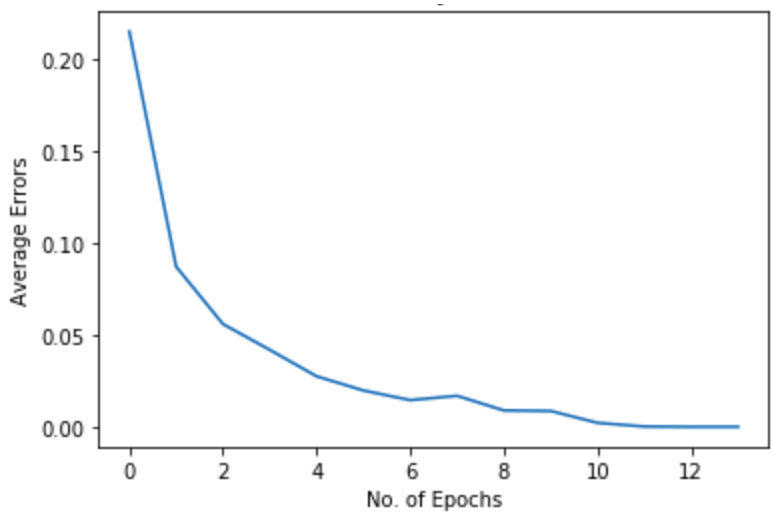


Fig. 9: Average error vs Epoch for architecture 3 using SGD

Confusion Matrix for Training and Validation Data

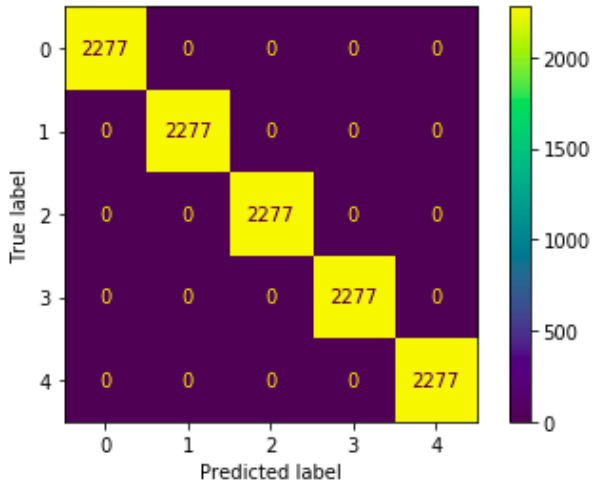


Fig. 10: Confusion matrix for training data architecture 1 using SGD

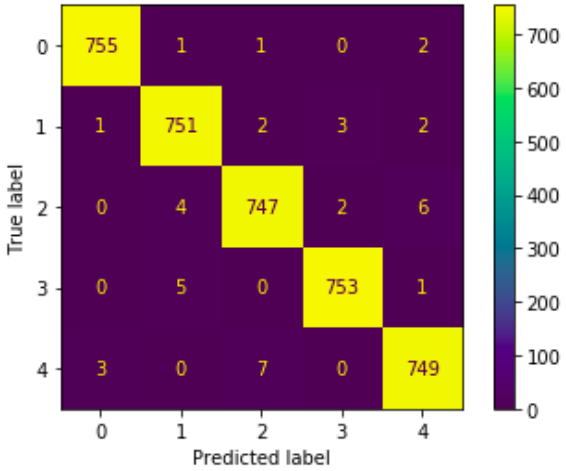


Fig. 11: confusion matrix of validation data for for architecture 1 using SGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.94%

Architecture 4: [784, 800, 300, 50, 5]

A plot for the average error v/s Epoch

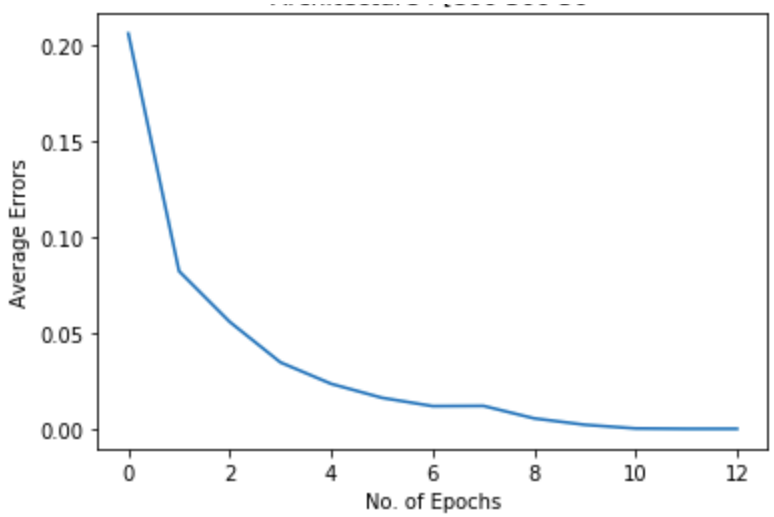


Fig. 12: Average error vs Epoch for architecture 4 using SGD

Confusion Matrix for Training and Validation Data

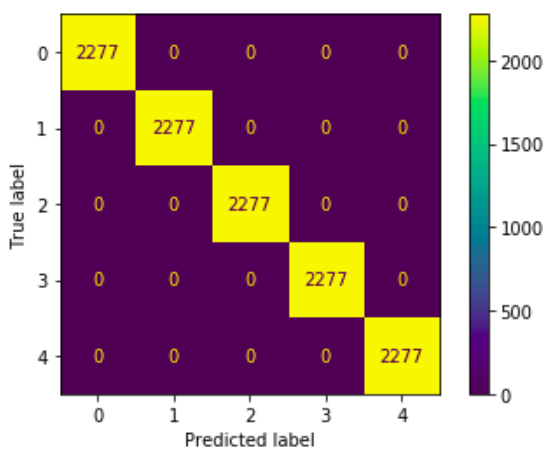


Fig. 13: Confusion matrix for training data architecture 1 using SGD

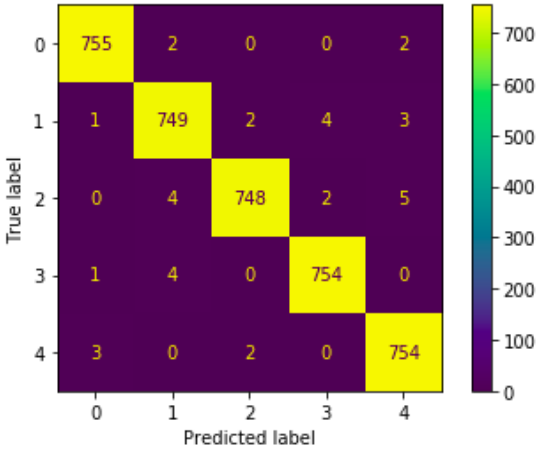


Fig. 14: confusion matrix of validation data for for architecture 1 using SGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 99.08%

Architecture 5: [784, 400, 150, 60, 5]

A plot for the average error v/s Epoch

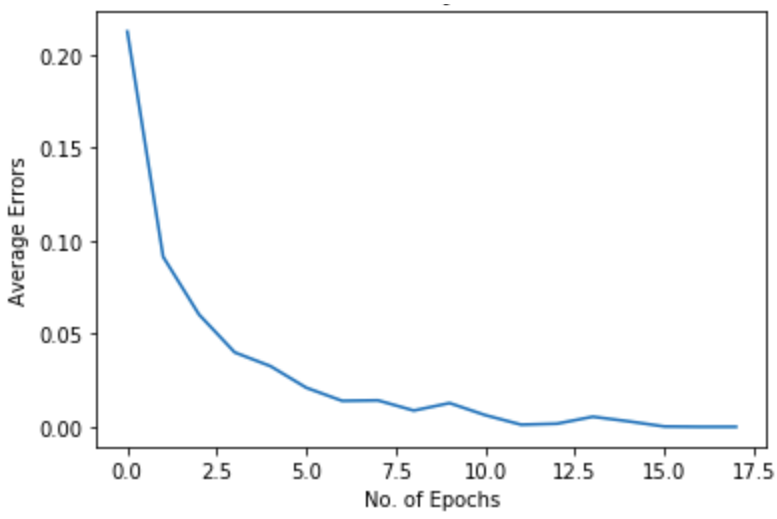


Fig. 15: Average error vs Epoch for architecture 5 using SGD

Confusion Matrix for Training and Validation Data

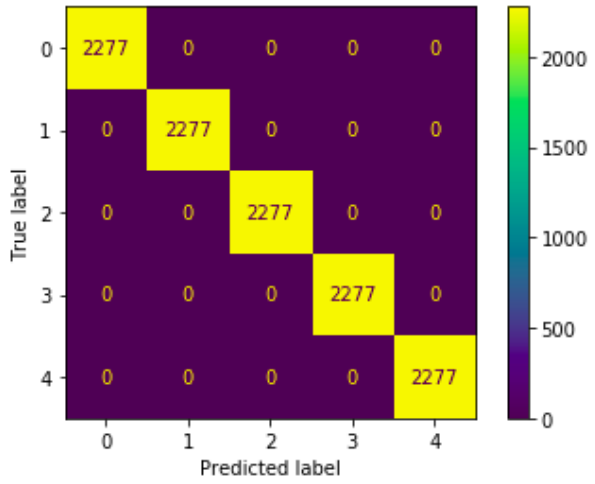


Fig. 16: Confusion matrix for training data architecture 1 using SGD

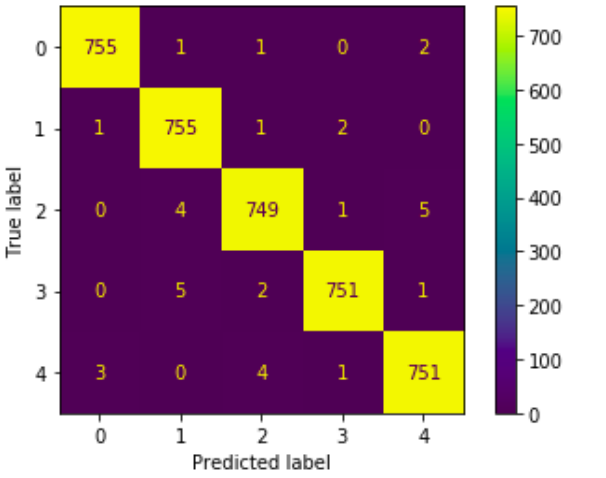


Fig. 17: confusion matrix of validation data for for architecture 1 using SGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 99.104%

Table of Training and Validation Accuracy for each FCNN Architecture

FCNN Architecture	Training Accuracy	Validation Accuracy
Architecture 1: [784, 200, 100, 50, 5]	1	0.989460
Architecture 2: [784, 512, 128, 32, 5]	1	0.990514
Architecture 3: [784, 300, 400, 100, 5]	1	0.989460
Architecture 4: [784, 800, 300, 50, 5]	1	0.990777
Architecture 5: [784, 400, 150, 60, 5]	1	0.991041

Table 2: Comparison of all FCNN architecture using SGD

Observation:

From the above table we observe that Architecture 2 is the best FCNN with SGD optimizer. However we have also trained data on all 5 architectures. Architecture 2 also showed good validation accuracy at that time.

Now Test this architecture using Test Data.

Confusion Matrix of Test Data:

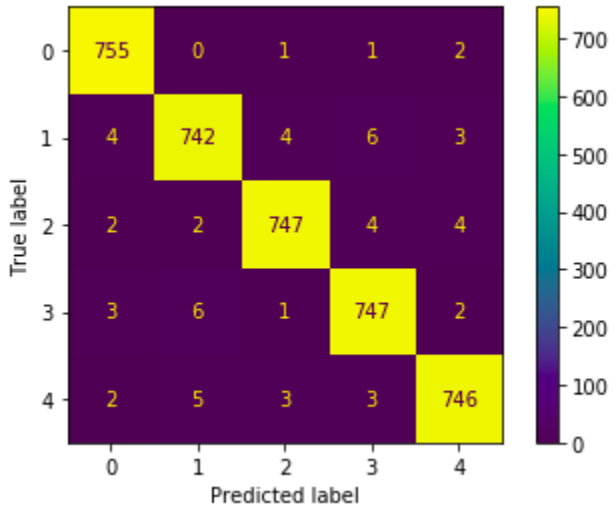


Fig. 18: Confusion matrix of best architecture

Accuracy Score of Test Data: 98.472%

2. VGD

Training Each Fcnn Architecture using Vanilla gradient descent (VGD) algorithm

- Batch size = Training Data
- Learning Rate : 0.01
- Convergence Criteria : difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture 1: [784, 200, 100, 50, 5]

A plot for the average error v/s Epoch

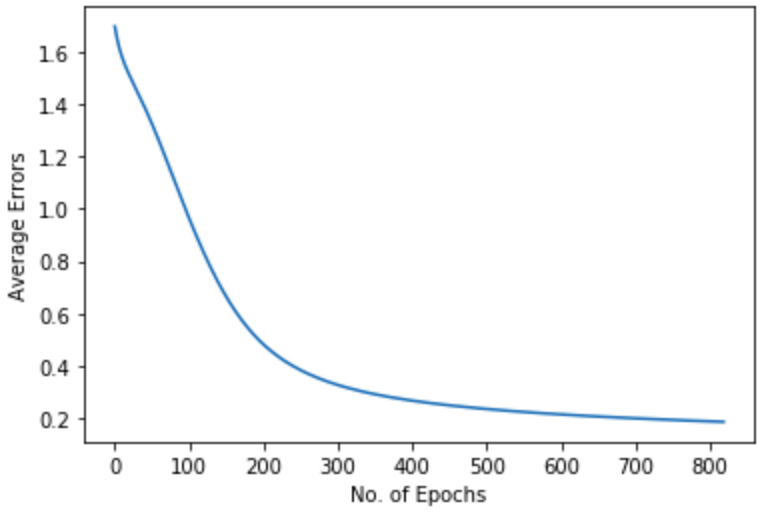


Fig. 19: Average error vs Epoch for architecture 1 using VGD

Confusion Matrix for Training and Validation Data

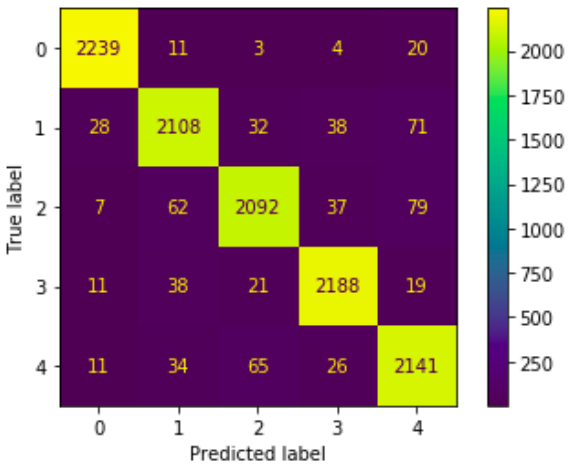


Fig. 20: Confusion matrix for training data architecture 1 using VGD

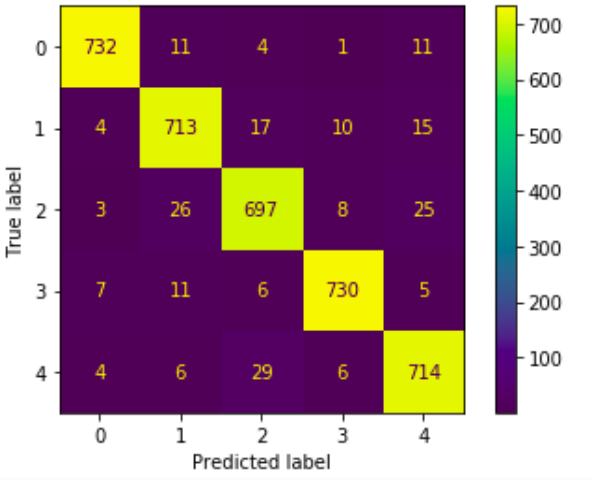


Fig. 21: confusion matrix of validation data for for architecture 1 using VGD

Accuracy:

- Training Accuracy: 94.58%
- Validation Accuracy: 94.49%

Architecture 2: [784, 512, 128, 32, 5]

A plot for the average error v/s Epoch

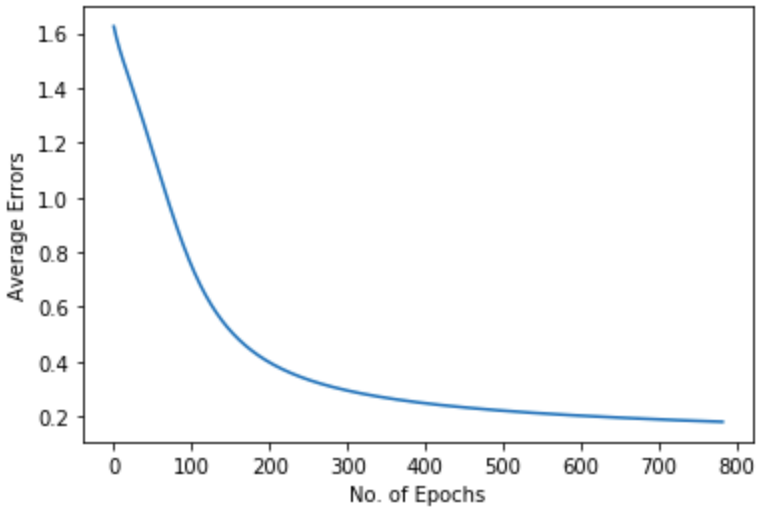


Fig. 22: Average error vs Epoch for architecture 2 using VGD

Confusion Matrix for Training and Validation Data

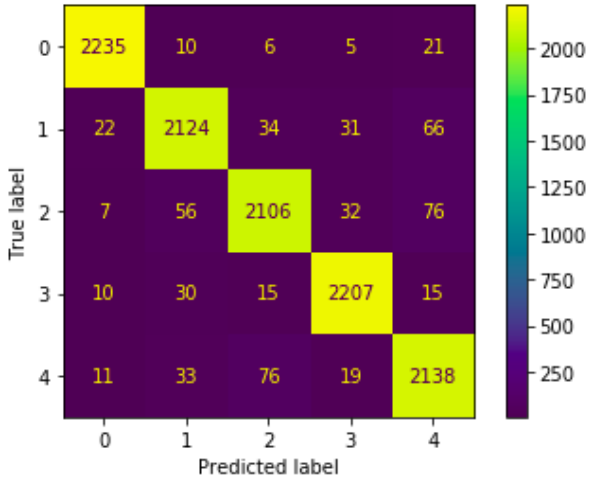


Fig. 23: Confusion matrix for training data architecture 2 using VGD

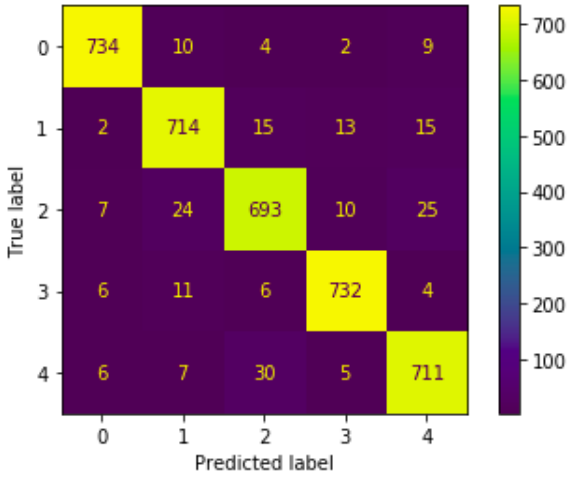


Fig. 24: confusion matrix of validation data for for architecture 2 using VGD

Accuracy:

- Training Accuracy: 94.95%
- Validation Accuracy: 94.44%

Architecture 3: [784, 300, 400, 100, 5]

A plot for the average error v/s Epoch

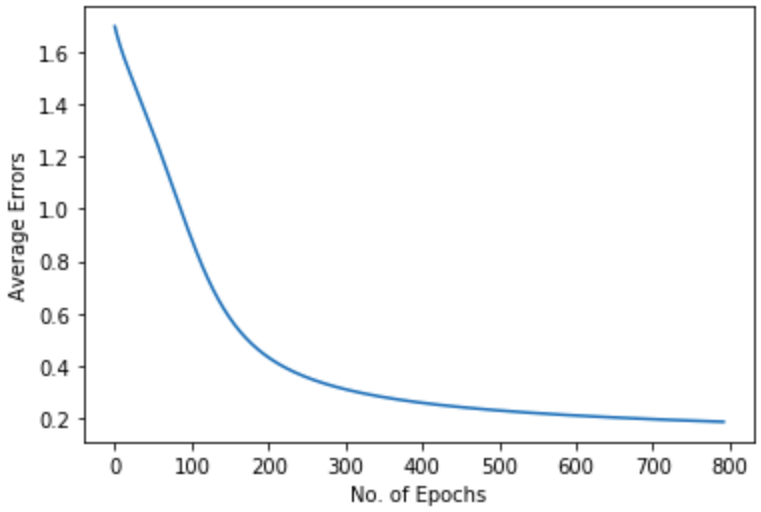


Fig. 25: Average error vs Epoch for architecture 3 using VGD

Confusion Matrix for Training and Validation Data

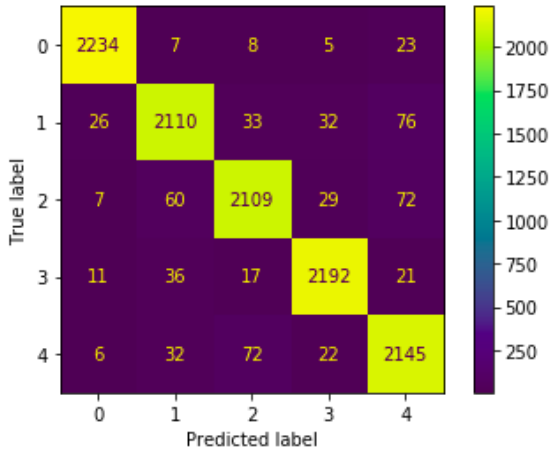


Fig. 26: Confusion matrix for training data architecture 3 using VGD

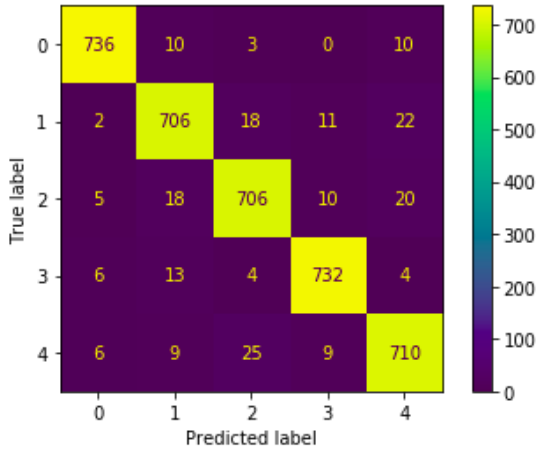


Fig. 27: confusion matrix of validation data for for architecture 3 using VGD

Accuracy:

- Training Accuracy: 94.77%
- Validation Accuracy: 94.6%

Architecture 4: [784, 800, 300, 50, 5]

A plot for the average error v/s Epoch

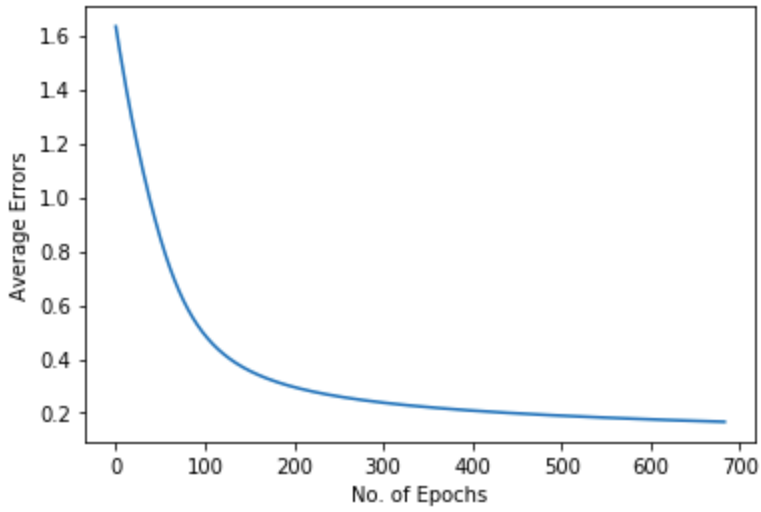


Fig. 28: Average error vs Epoch for architecture 4 using VGD

Confusion Matrix for Training and Validation Data

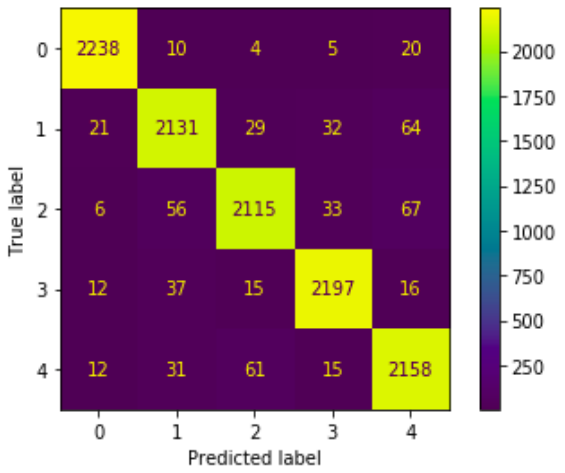


Fig. 29: Confusion matrix for training data architecture 4 using VGD

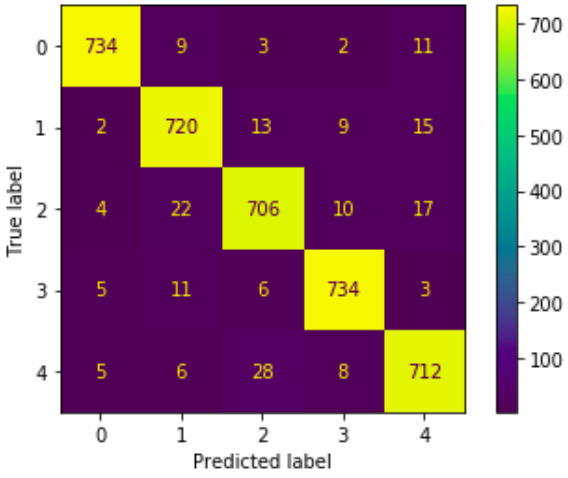


Fig. 30: confusion matrix of validation data for for architecture 4 using VGD

Accuracy:

- Training Accuracy: 95.20%
- Validation Accuracy: 95.02%

Architecture 5: [784, 400, 150, 60, 5]

A plot for the average error v/s Epoch

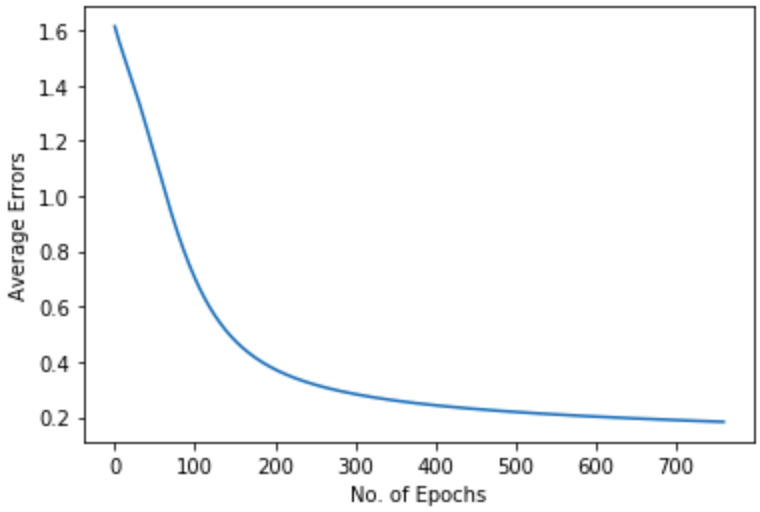


Fig. 31: Average error vs Epoch for architecture 5 using VGD

Confusion Matrix for Training and Validation Data

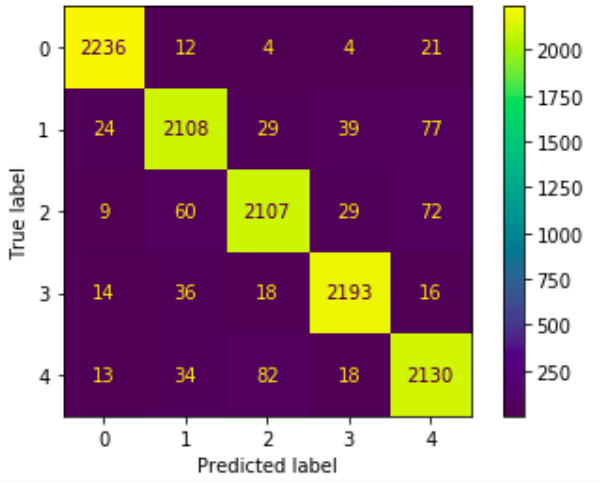


Fig. 32: Confusion matrix for training data architecture 5 using VGD

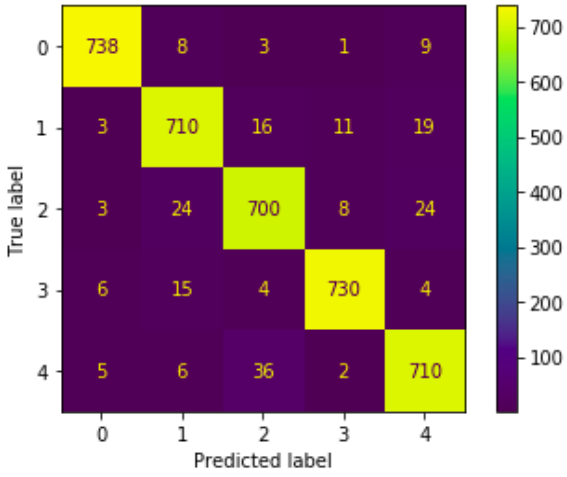


Fig. 33: confusion matrix of validation data for for architecture 5 using VGD

Accuracy:

- Training Accuracy: 94.63%
- Validation Accuracy: 94.54%

Table of Training and Validation Accuracy for each FCNN Architecture

FCNN Architecture	Training Accuracy	Validation Accuracy
Architecture 1: [784, 200, 100, 50, 5]	0.945806	0.944928
Architecture 2: [784, 512, 128, 32, 5]	0.949495	0.944401
Architecture 3: [784, 300, 400, 100, 5]	0.947738	0.945982
Architecture 4: [784, 800, 300, 50, 5]	0.952042	0.950198
Architecture 5: [784, 400, 150, 60, 5]	0.946333	0.945455

Table 3: Comparison of Train, Test, Validation accuracy for FCNN Architecture

Observation:

From the above table we observe that Architecture 4 is the best FCNN with VGD optimizer. We also observe that it takes more than approx 700 epochs to train the model and still give the accuracy upto 95% . Reason for too many epochs is due to the batch size. As the batch size is equal to the training data. So backpropagation happens only once for one epoch.

Now Test the best architecture using Test Data.

Confusion Matrix of Test Data:

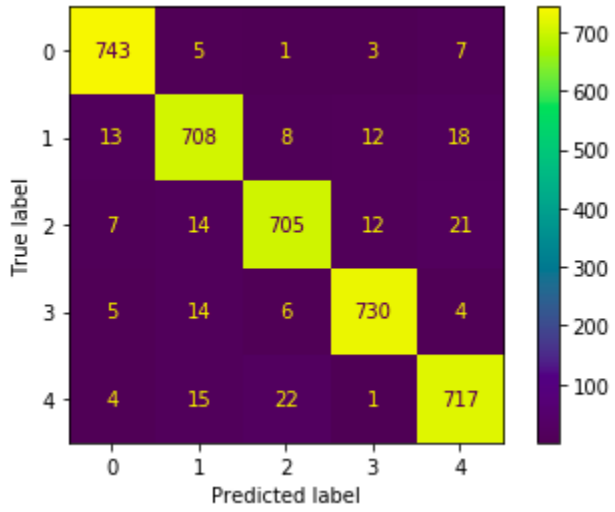


Fig. 34: Confusion matrix of the best architecture using VGD

Accuracy of Test Data: 94.94%

3. NAG

Training Each Fcnn Architecture using Nesterov Accelerated Gradient (NAG) algorithm

- Batch size = 32
- Learning Rate : 0.01
- Momentum: 0.9
- Convergence Criteria : difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture 1: [784, 200, 100, 50, 5]

A plot for the average error v/s Epoch

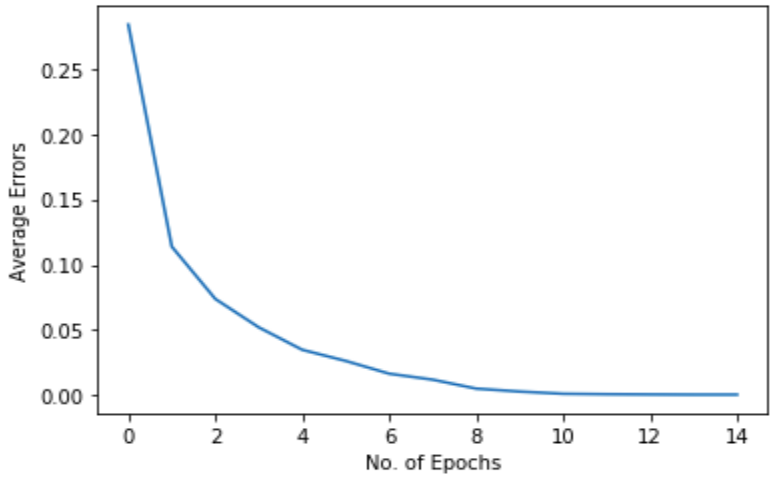


Fig. 35: Average error vs Epoch for architecture 1 using NAG

Confusion Matrix for Training and Validation Data

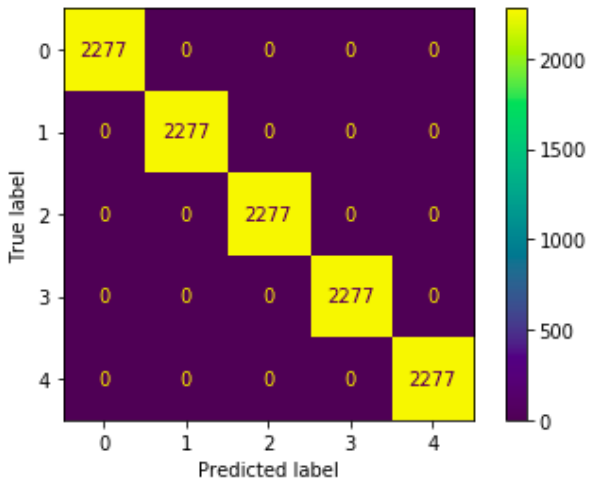


Fig. 36: Confusion matrix for training data architecture 1 using NAG

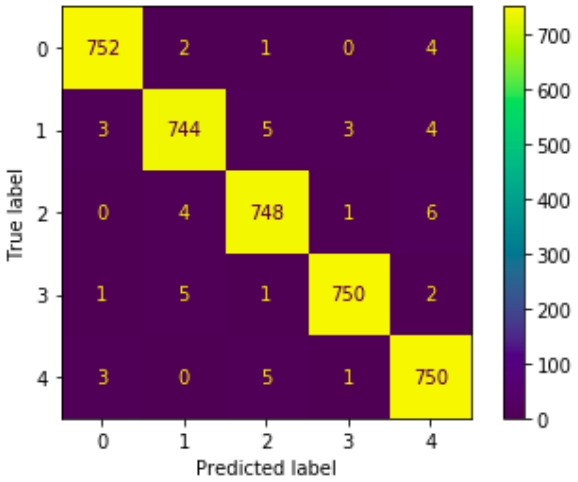


Fig. 37: confusion matrix of validation data for for architecture 1 using NAG

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.66%

Architecture 2: [784, 512, 128, 32, 5]

A plot for the average error v/s Epoch

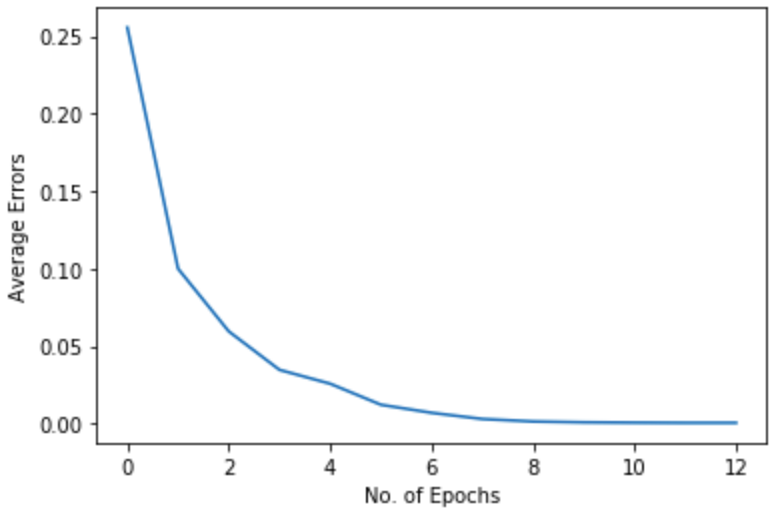


Fig. 38: Average error vs Epoch for architecture 2 using NAG

Confusion Matrix for Training and Validation Data

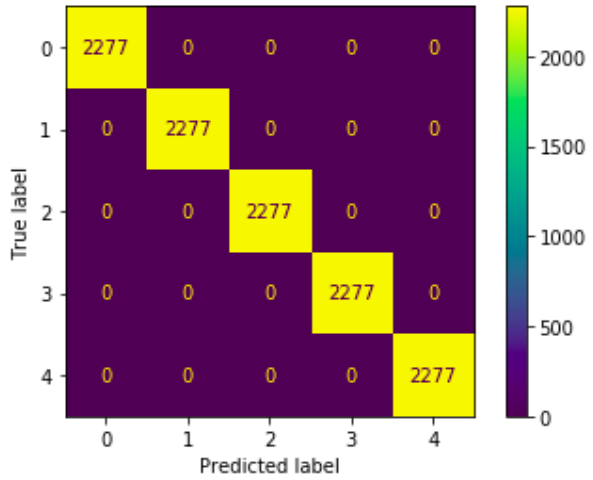


Fig. 39: Confusion matrix for training data architecture 2 using VGD

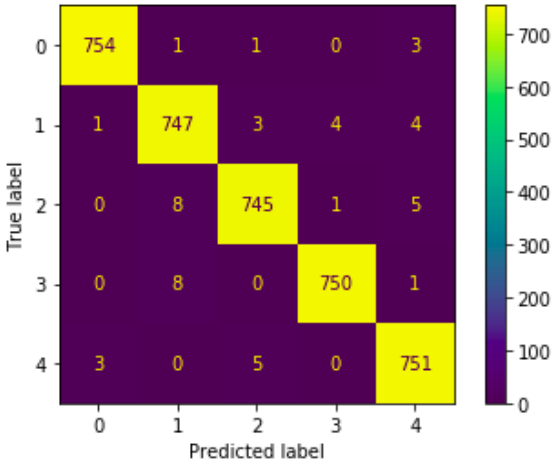


Fig. 40: confusion matrix of validation data for for architecture 2 using VGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.74%

Architecture 3: [784, 300, 400, 100, 5]

A plot for the average error v/s Epoch

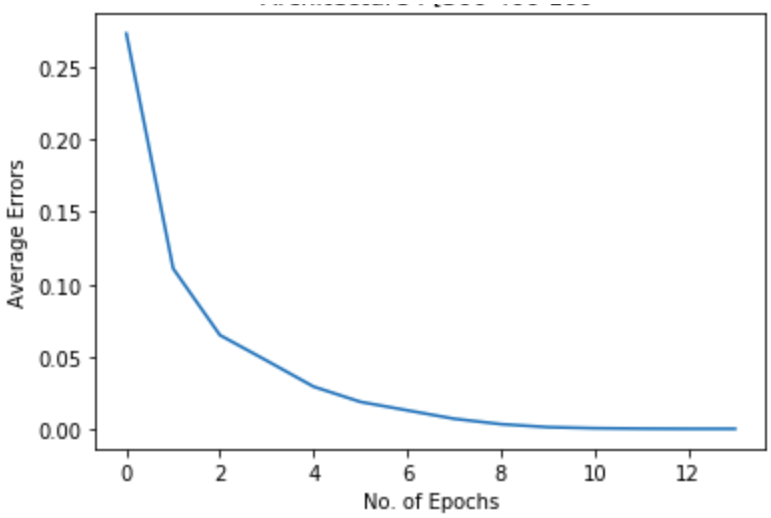


Fig. 41: Average error vs Epoch for architecture 3 using NAG

Confusion Matrix for Training and Validation Data

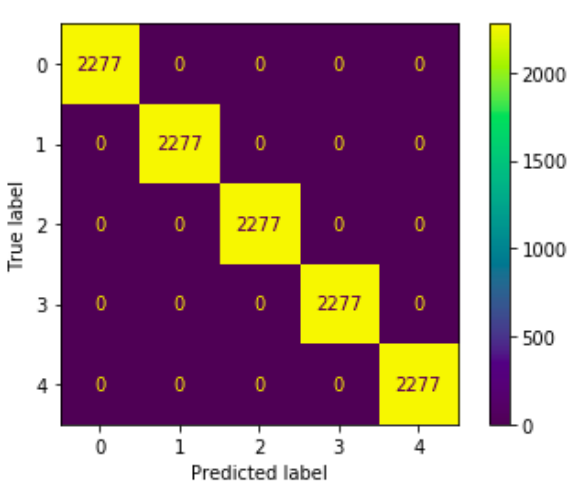


Fig. 42: Confusion matrix for training data architecture 3 using NAG

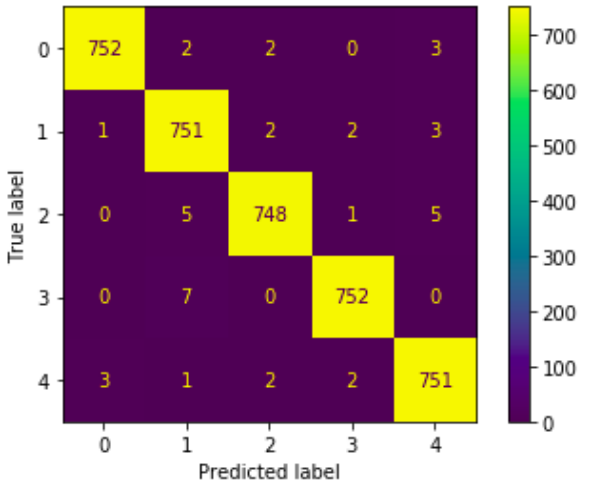


Fig. 43: confusion matrix of validation data for for architecture 3 using NAG

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.92%

Architecture 4: [784, 800, 300, 50, 5]

A plot for the average error v/s Epoch

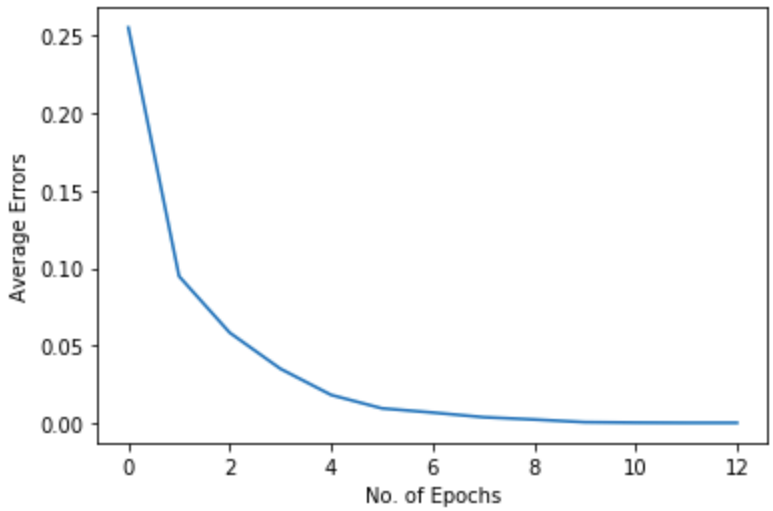


Fig. 44: Average error vs Epoch for architecture 4 using NAG

Confusion Matrix for Training and Validation Data

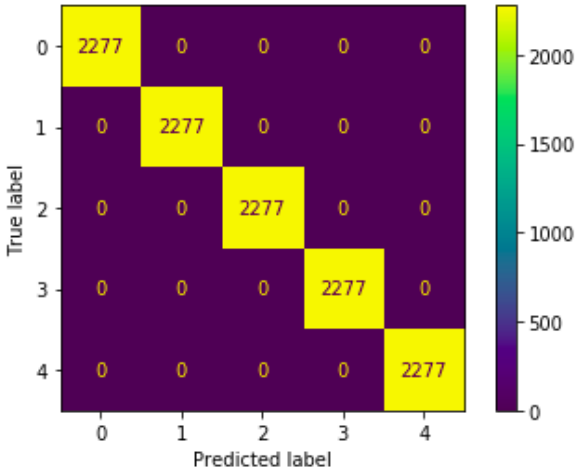


Fig. 45: Confusion matrix for training data architecture 4 using VGD

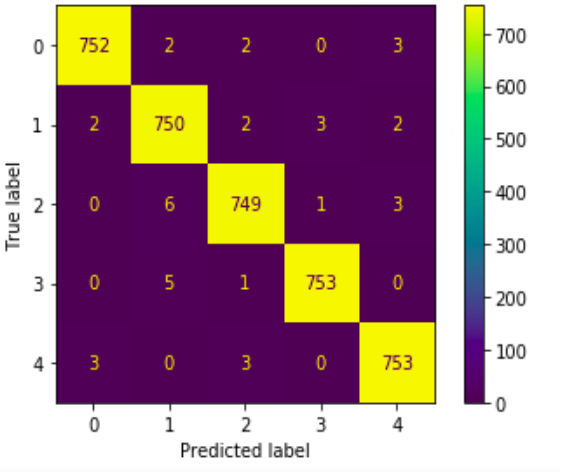


Fig. 46: confusion matrix of validation data for for architecture 4 using VGD

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.99%

Architecture 5: [784, 400, 150, 60, 5]

A plot for the average error v/s Epoch

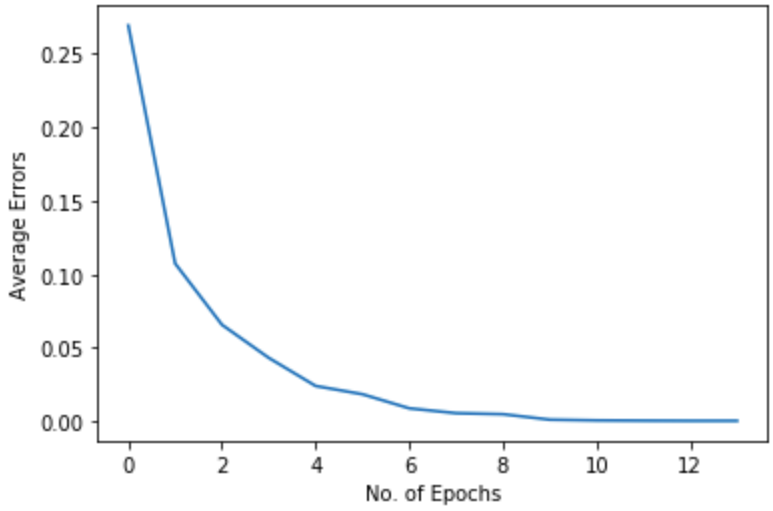


Fig. 47: Plot for the average error v/s Epoch

Confusion Matrix for Training and Validation Data

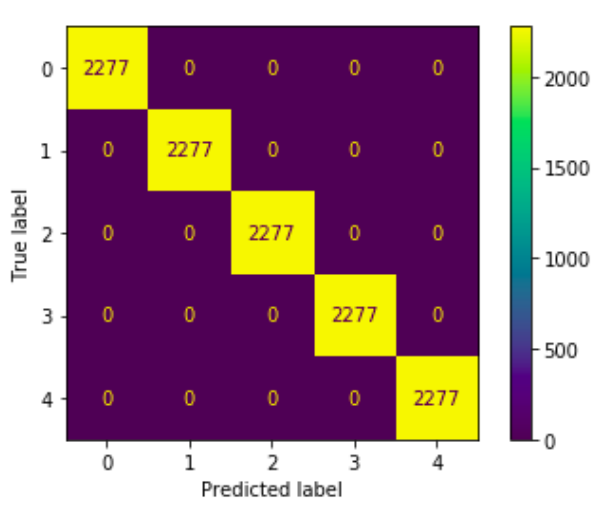


Fig. 48: Confusion matrix for training data architecture 5 using NAG

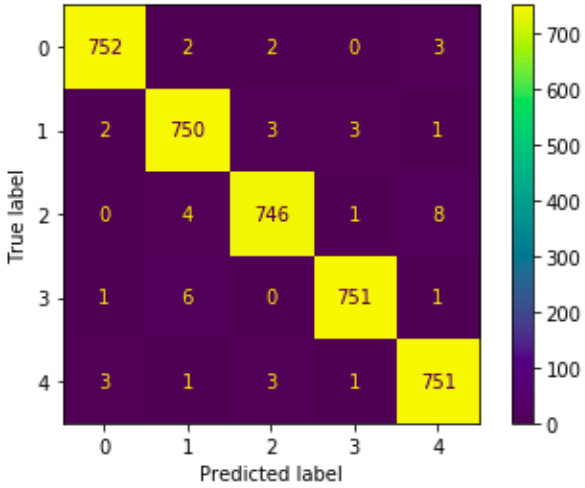


Fig. 49: confusion matrix of validation data for for architecture 5 using NAG

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.814%

Table of Training and Validation Accuracy for each FCNN Architecture

FCNN Architecture	Training Accuracy	Validation Accuracy
Architecture 1: [784, 200, 100, 50, 5]	1	0.986561
Architecture 2: [784, 512, 128, 32, 5]	1	0.987352
Architecture 3: [784, 300, 400, 100, 5]	1	0.989196
Architecture 4: [784, 800, 300, 50, 5]	1	0.989987
Architecture 5: [784, 400, 150, 60, 5]	1	0.988142

Table 4: Comparison of Train, Test, Validation accuracy for FCNN Architecture using NAG

Observation:

From the above table and also experiment 4-5 times we observe that Architecture 3 and 4 both show 98.9% accuracy. Hence architecture 3 and 4 are the best FCNN using NAG optimizer. Now Test this architecture using Test Data.

Confusion Matrix of Test Data:

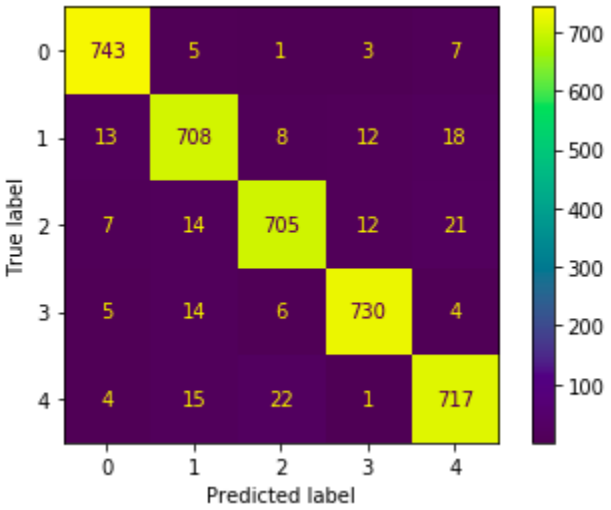


Fig. 50: Confusion matrix for test data using NAG

Accuracy of Test Data: 94.94%

4. RMSProp

Training each FCNN Architecture using RMSProp algorithm For this optimizer we have tried with different learning rate like 0.01 and 0.001 with beta (β) 0.99 and 0.9. We observe that model training takes too much oscillation and even give very less accuracy. Below are the outputs we observe.

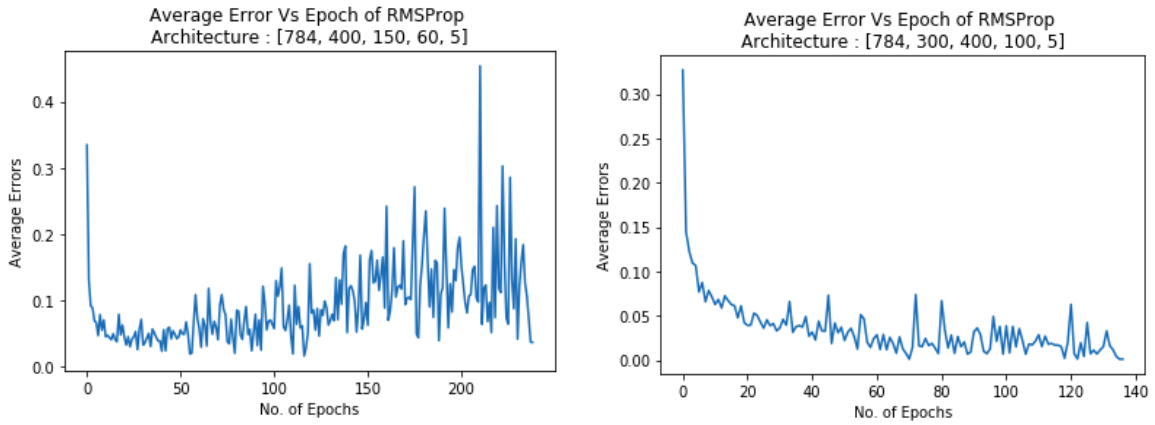


Fig. 51 : Avg error v/s epoch (learning rate = 0.001, β = 0.99)

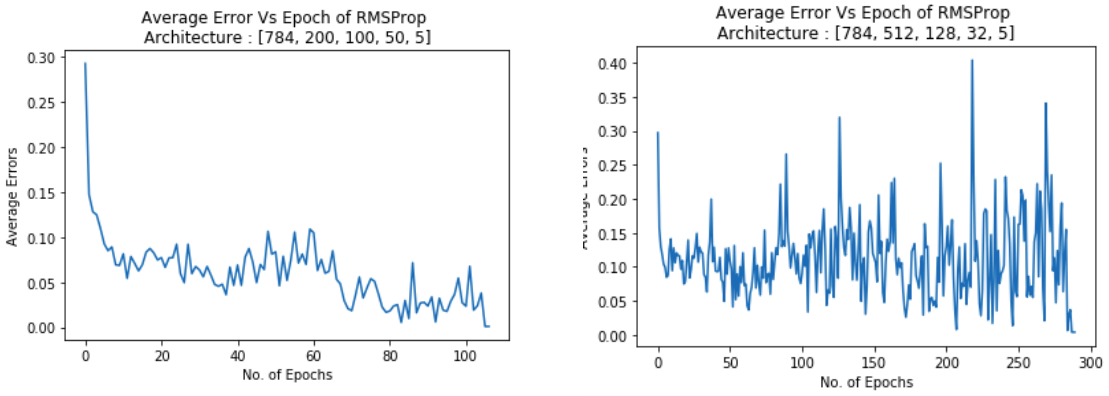


Fig. 52 : Avg error v/s epoch (learning rate = 0.001, β = 0.90)

Hence after observation we used below feature

- Batch size = 32
- Learning Rate : 0.0001
- Momentum: 0.9
- Beta (β): 0.90
- Convergence Criteria : difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture 1: [784, 200, 100, 50, 5]

A plot for the average error v/s Epoch

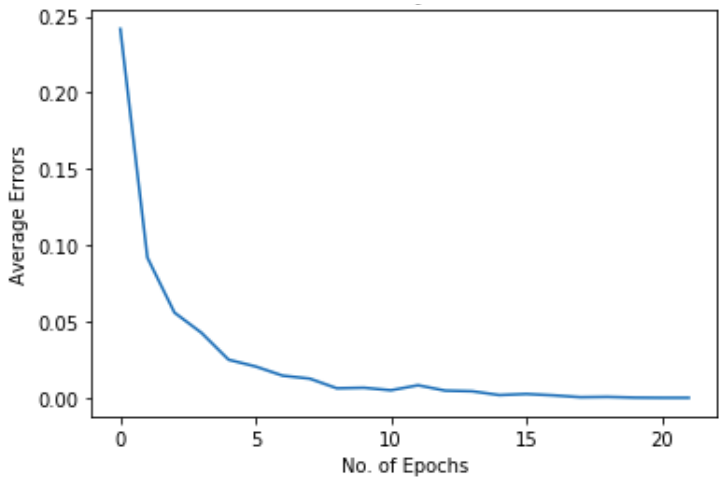


Fig. 53: Average error vs Epoch for architecture 1 using RMSProp

Confusion Matrix for Training and Validation Data

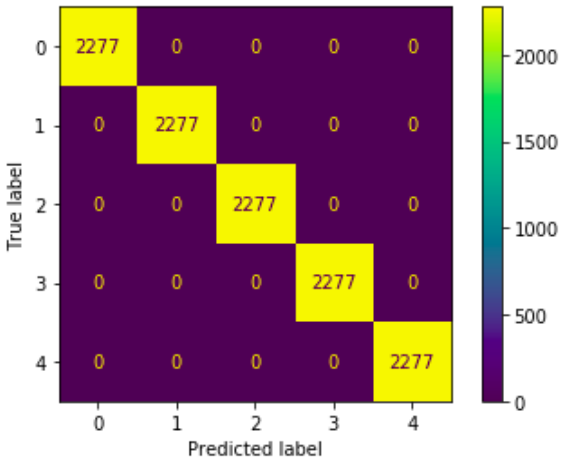


Fig. 54: Confusion matrix for training data architecture 1 using RMSProp

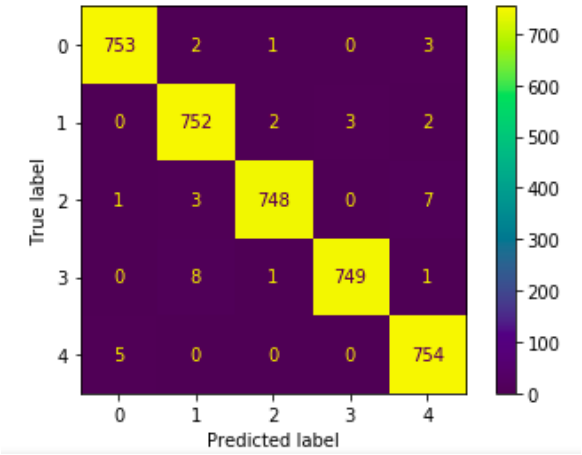


Fig. 55: confusion matrix of validation data for for architecture 1 using RMSProp

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.97%

Architecture 2: [784, 512, 128, 32, 5]

A plot for the average error v/s Epoch

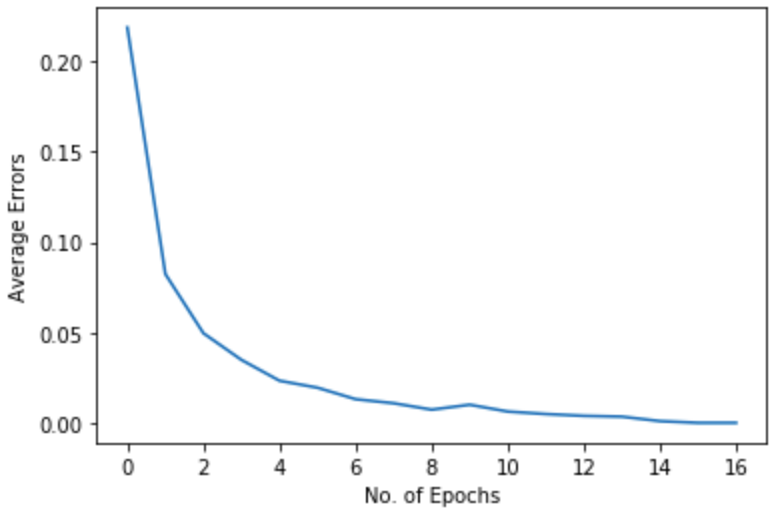


Fig. 56: Average error vs Epoch for architecture 2 using RMSProp

Confusion Matrix for Training and Validation Data

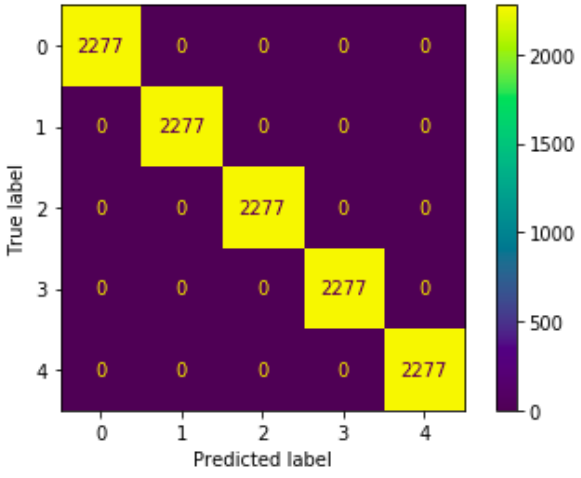


Fig. 57: Confusion matrix for training data architecture 2 using RMSProp

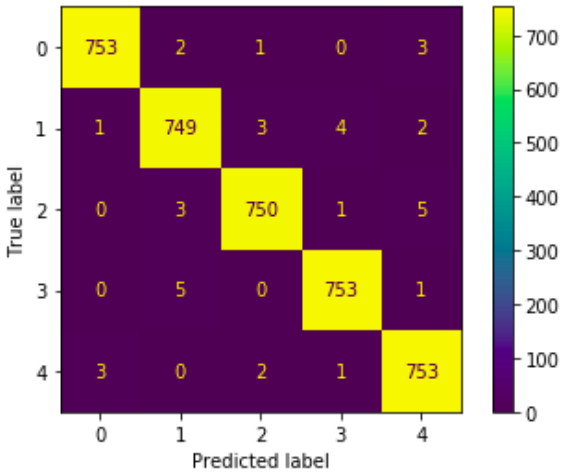


Fig. 58: confusion matrix of validation data for for architecture 2 using RMSProp

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 99.02%

Architecture 3: [784, 300, 400, 100, 5]

A plot for the average error v/s Epoch

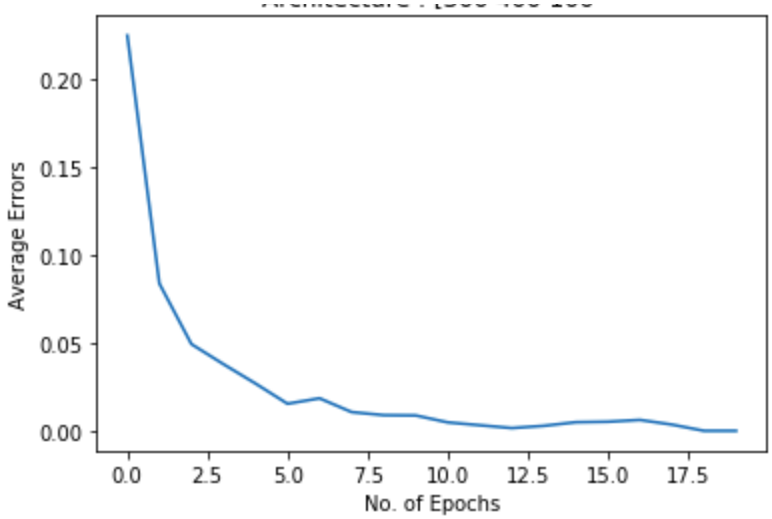


Fig. 59: Average error vs Epoch for architecture 3 using RMSProp

Confusion Matrix for Training and Validation Data

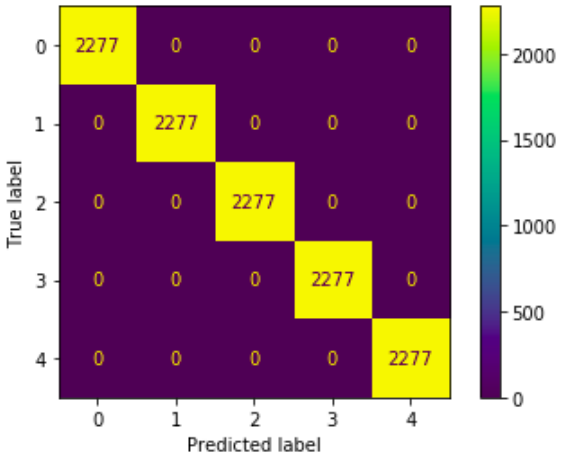


Fig. 60: Confusion matrix for training data architecture 2 using RMSProp

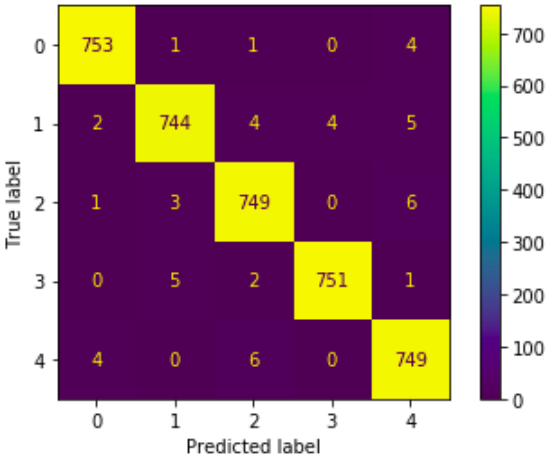


Fig. 61: confusion matrix of validation data for for architecture 2 using RMSProp

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.7%

Architecture 4: [784, 800, 300, 50, 5]

A plot for the average error v/s Epoch

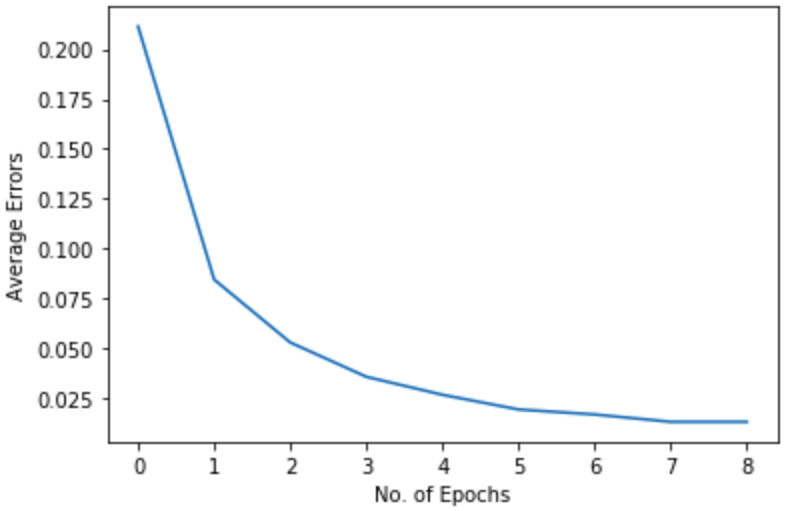


Fig. 62: Average error vs Epoch for architecture 2 using RMSProp

Confusion Matrix for Training and Validation Data

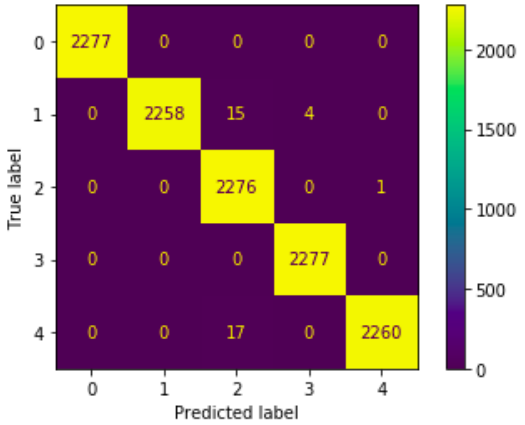


Fig. 63: Confusion matrix for training data architecture 2 using RMSProp

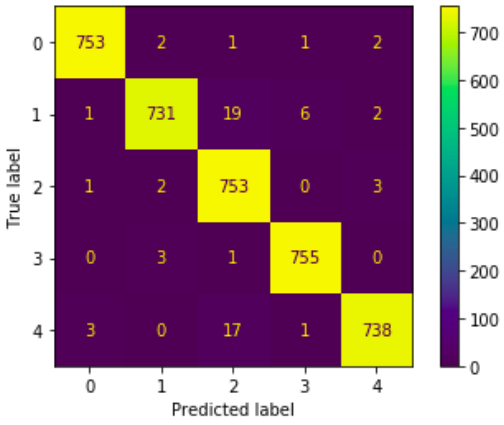


Fig. 64: confusion matrix of validation data for for architecture 2 using RMSProp

Accuracy:

- Training Accuracy: 99.67%
- Validation Accuracy: 98.29%

Architecture 5: [784, 400, 150, 60, 5]

A plot for the average error v/s Epoch

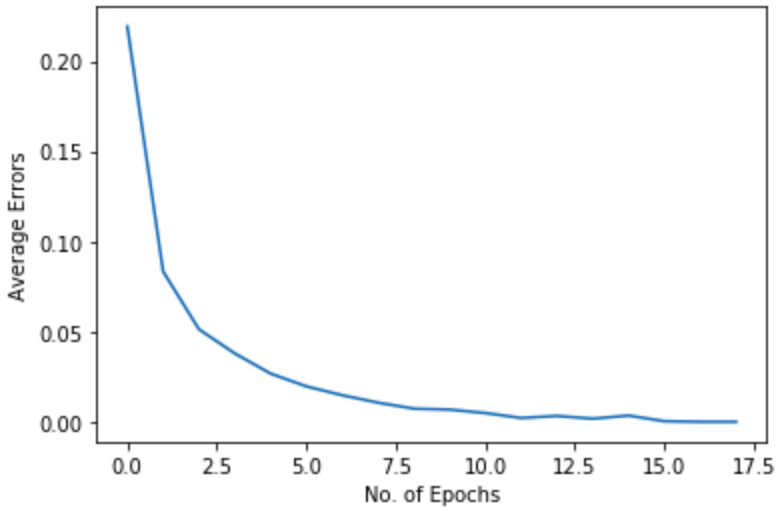


Fig. 65: Average error vs Epoch for architecture 2 using RMSProp

Confusion Matrix for Training and Validation Data

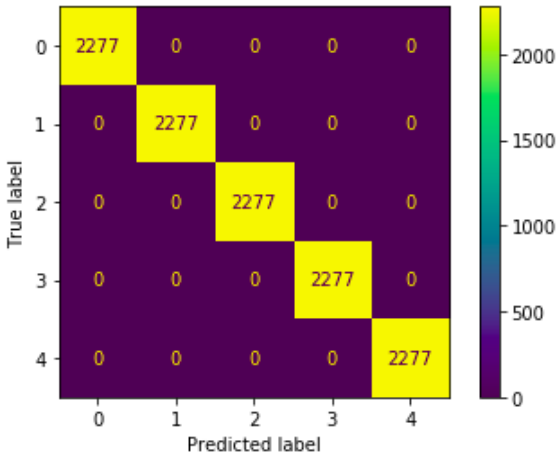


Fig. 66: Confusion matrix for training data architecture 2 using RMSProp

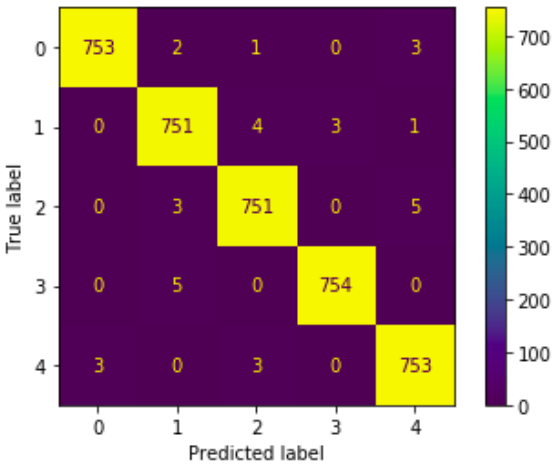


Fig. 67: confusion matrix of validation data for for architecture 2 using RMSProp

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 99.13%

Table of Training and Validation Accuracy for each FCNN Architecture

FCNN Architecture	Training Accuracy	Validation Accuracy
Architecture 1: [784, 200, 100, 50, 5]	1	0.989723
Architecture 2: [784, 512, 128, 32, 5]	1	0.990250
Architecture 3: [784, 300, 400, 100, 5]	1	0.987088
Architecture 4: [784, 800, 300, 50, 5]	0.99675	0.982872
Architecture 5: [784, 400, 150, 60, 5]	1	0.991304

Table 5: Comparison of Train, Test, Validation accuracy for FCNN Architecture using RMSProp

Observation:

From the above table we observe that Architecture 2 and 5 are the best FCNN with RMSProp optimizer. Now Test this architecture using Test Data.

Confusion Matrix of Test Data:

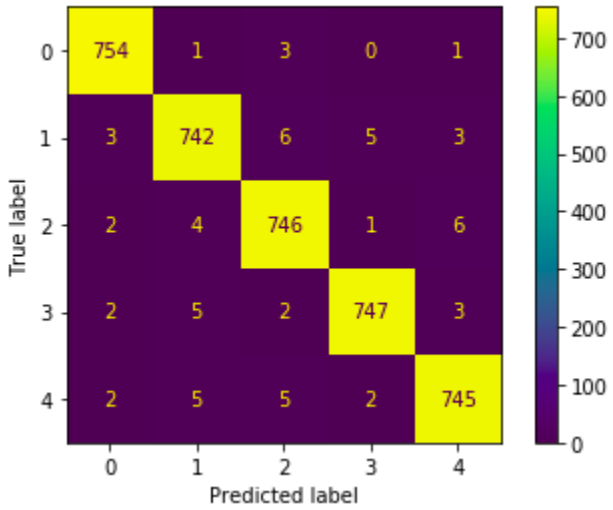


Fig. 68: Confusion matrix for test data using RMSProp

Accuracy of Test Data: 98.39%

5. ADAM

Training Each Fcnn Architecture using stochastic gradient descent (SGD) algorithm

- Batch size = 32
- Learning Rate : 0.001
- Beta_1 (β_1): 0.9
- Beta_2 (β_2): 0.999
- Epsilon (ε) = 10^{-8}
- Convergence Criteria : difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture 1: [784, 200, 100, 50, 5]

A plot for the average error v/s Epoch

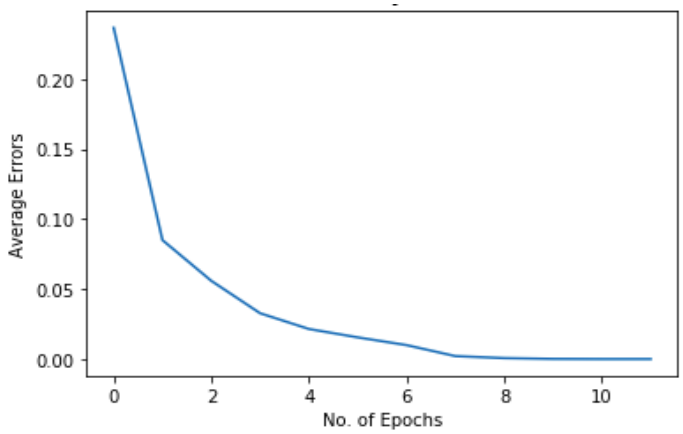


Fig. 69: Average error vs Epoch for architecture 1 using ADAM

Confusion Matrix for Training and Validation Data

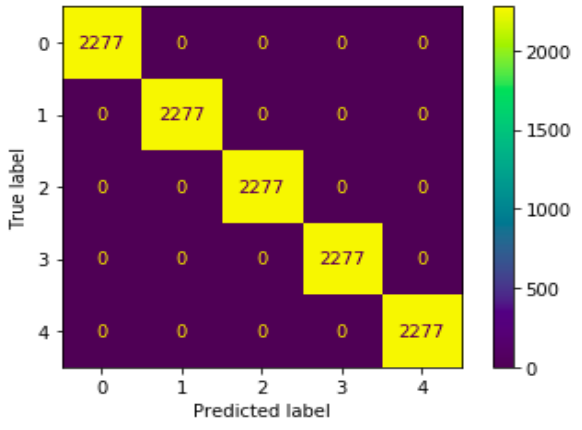


Fig. 70: Confusion matrix for training data architecture 1 using ADAM

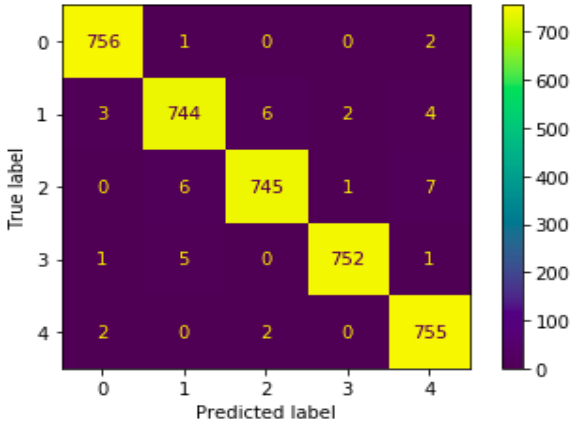


Fig. 71: confusion matrix of validation data for for architecture 1 using ADAM

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.87%

Architecture 2: [784, 512, 128, 32, 5]

A plot for the average error v/s Epoch

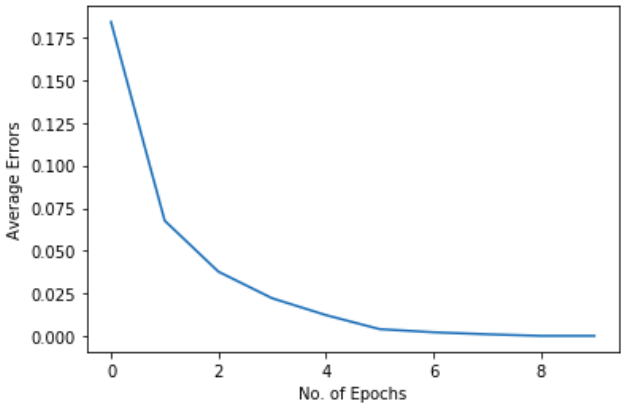


Fig. 72: Average error vs Epoch for architecture 2 using ADAM

Confusion Matrix for Training and Validation Data

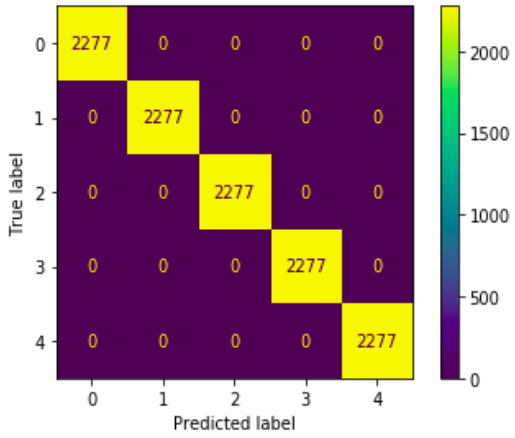


Fig. 73: Confusion matrix for training data architecture 2 using ADAM

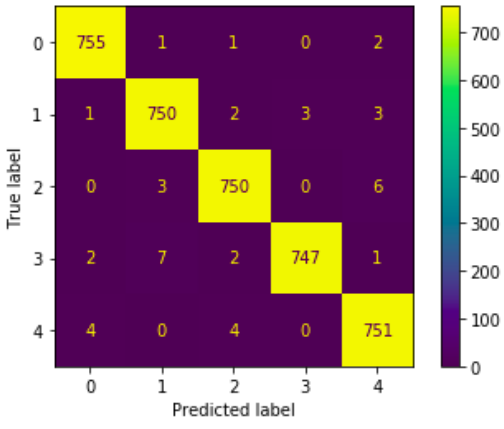


Fig. 74: confusion matrix of validation data for for architecture 2 using ADAM

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.89%

Architecture 3: [784, 300, 400, 100, 5]

A plot for the average error v/s Epoch

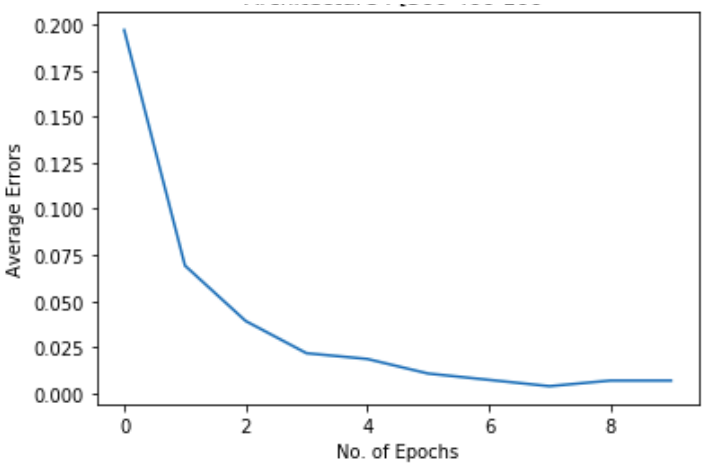


Fig. 75: Average error vs Epoch for architecture 3 using ADAM

Confusion Matrix for Training and Validation Data

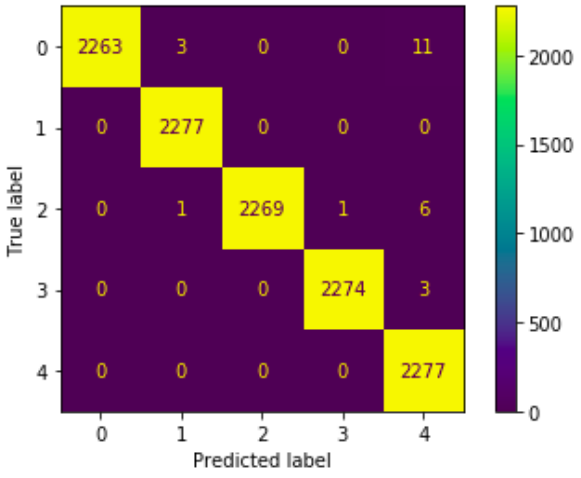


Fig. 76: Confusion matrix for training data architecture 3 using ADAM

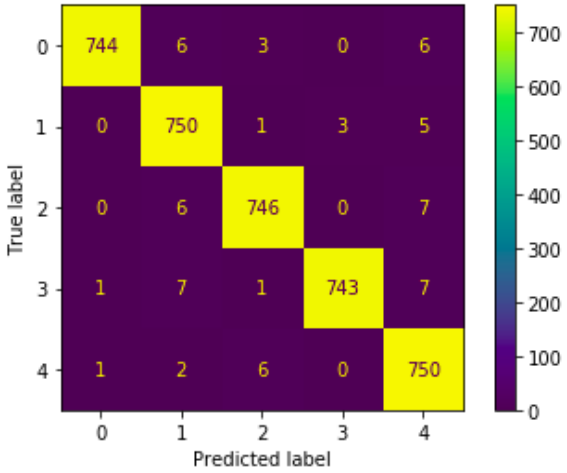


Fig. 77: confusion matrix of validation data for for architecture 3 using ADAM

Accuracy:

- Training Accuracy: 99.78%
- Validation Accuracy: 98.36%

Architecture 4: [784, 800, 300, 50, 5]

A plot for the average error v/s Epoch

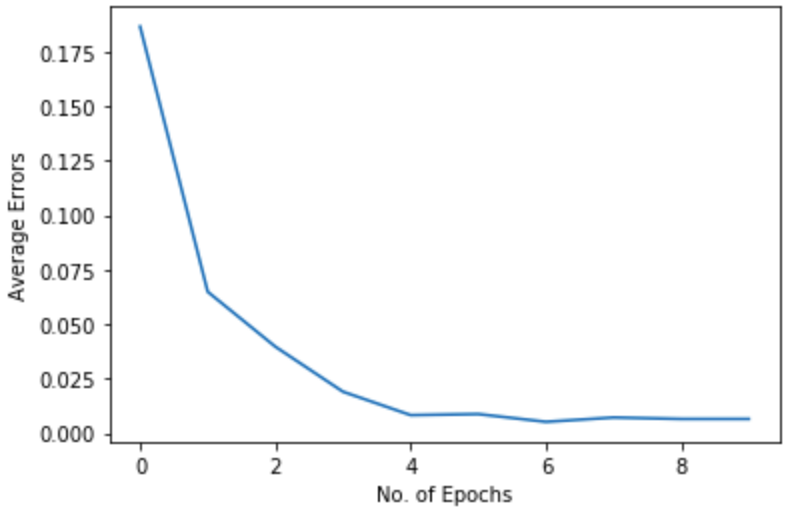


Fig. 78: Average error vs Epoch for architecture 4 using ADAM

Confusion Matrix for Training and Validation Data

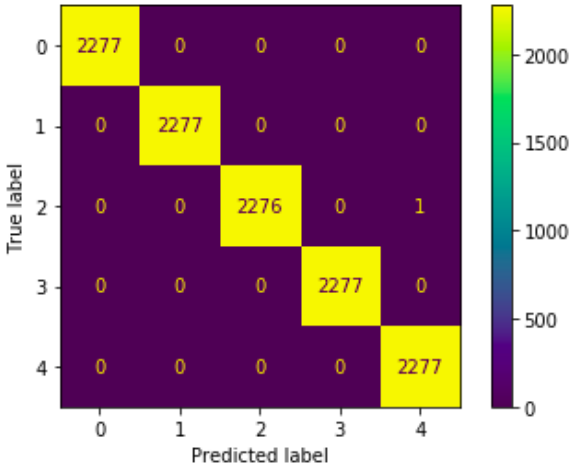


Fig. 79: Confusion matrix for training data architecture 4 using ADAM

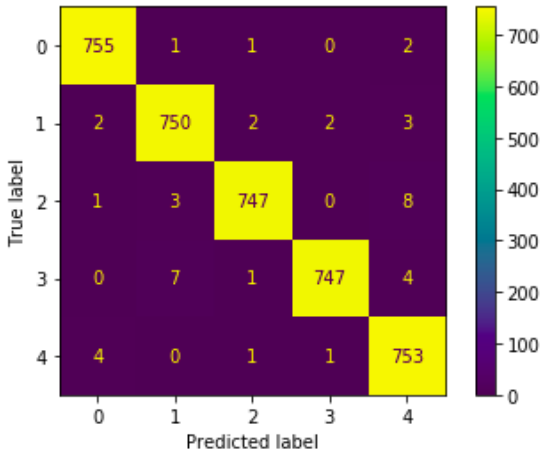


Fig. 80: confusion matrix of validation data for for architecture 5 using ADAM

Accuracy:

- Training Accuracy: 99.99%
- Validation Accuracy: 98.87%

Architecture 5: [784, 400, 150, 60, 5]

A plot for the average error v/s Epoch

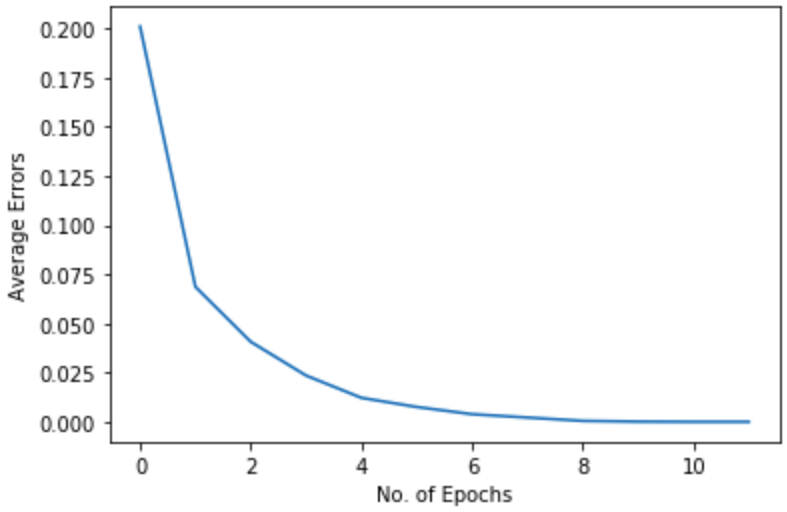


Fig. 81: Average error vs Epoch for architecture 5 using ADAM

Confusion Matrix for Training and Validation Data

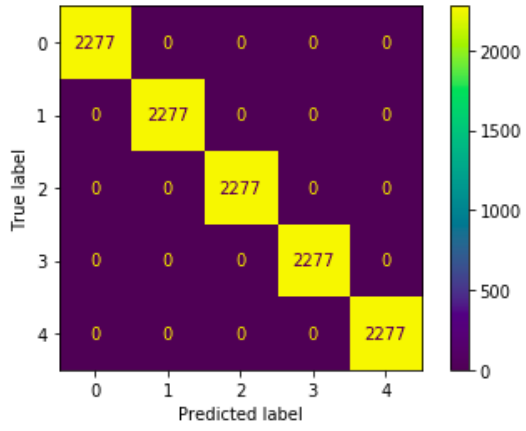


Fig. 82: Confusion matrix for training data architecture 1 using ADAM

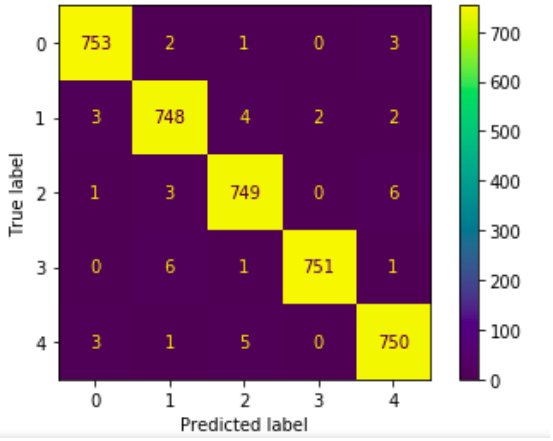


Fig. 83: confusion matrix of validation data for for architecture 1 using ADAM

Accuracy:

- Training Accuracy: 100%
- Validation Accuracy: 98.84%

Table of Training and Validation Accuracy for each FCNN Architecture

FCNN Architecture	Training Accuracy	Validation Accuracy
Architecture 1: [784, 200, 100, 50, 5]	1	0.988669
Architecture 2: [784, 512, 128, 32, 5]	1	0.988933
Architecture 3: [784, 300, 400, 100, 5]	0.997804	0.983663
Architecture 4: [784, 800, 300, 50, 5]	0.999912	0.988669
Architecture 5: [784, 400, 150, 60, 5]	1	0.988406

Table 6: Comparison of Train, Test, Validation accuracy for FCNN Architecture using RMSProp

Observation:

From the above table we observe that Architecture 2 is the best FCNN for Adam optimizer. Now Test this architecture using Test Data.

Confusion Matrix of Test Data:

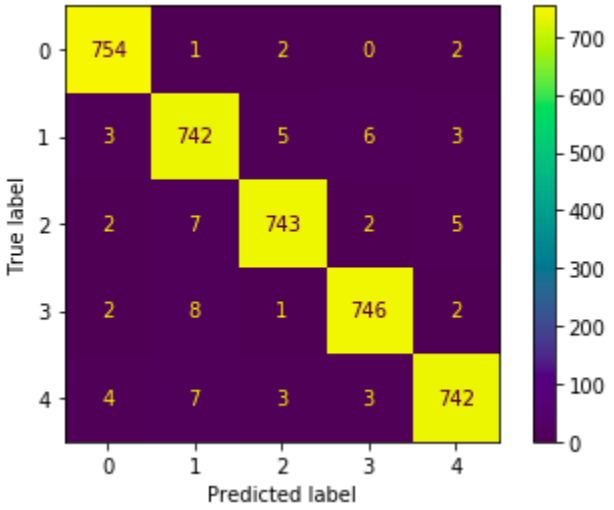


Fig. 84: Confusion matrix for test data using ADAM

Accuracy Test Data: 98.208%

Epochs Taken by each Optimizer for each architecture:

Tabulation of the number of epochs considered by each of the optimizers for each architecture

FCNN Architecture	SGD	VGD	NAG	RMSProp	Adam
Architecture 1: [784, 200, 100, 50, 5]	20	818	14	21	11
Architecture 2: [784, 512, 128, 32, 5]	14	782	12	16	9
Architecture 3: [784, 300, 400, 100, 5]	13	792	13	19	9
Architecture 4: [784, 800, 300, 50, 5]	12	683	12	8	9
Architecture 5: [784, 400, 150, 60, 5]	17	760	13	17	11

Table 7: Comparison of each optimizer using number of epochs

Observation:

From the above table we observe that Adam is the best optimizer for most of the FCNN architecture. As it takes very less number of epochs for training models and converges faster as compared to other optimizers.

AutoEncoder

Task is to implement an autoencoder with one hidden layer and three hidden layers and test with a varying number of neurons in the hidden layer. After getting the compressed representation we will further use this for classification.

Autoencoder with one Hidden Layer

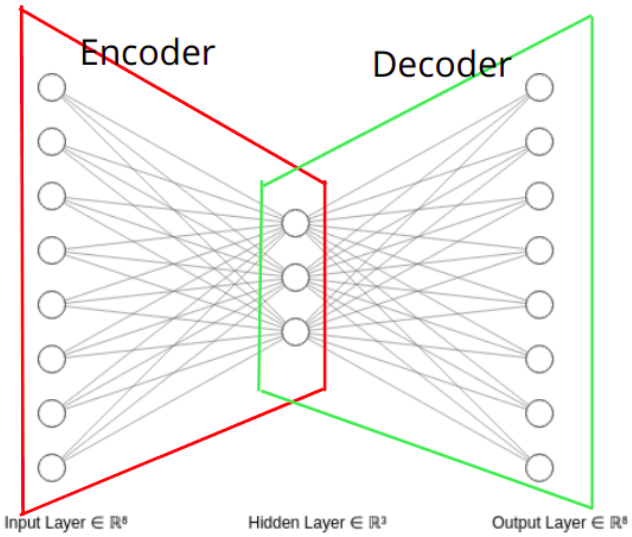


Fig. 85: Autoencoders with one hidden layer

Different Architecture we used:

Autoencoder Architecture	Input Layer	Hidden Layer	Output Layer
1	784	128	784
2	784	64	784
3	784	100	784

Table 8: Different architecture used in autoencoder

Training above all three architecture we have observe the average reconstruction error for training and validation data:

Architecture	Training Reconstruction Error	Validation Reconstruction Error
Architecture 1: [784,128,784]	2.553928	3.021354
Architecture 2: [784,64,784]	4.712975	5.138082
ARchitecture 3: [784,100,784]	2.913603	3.366698

Table 9: Reconstruction error on training and validation set with different architecture

Observation:

From the above validation reconstruction error we can say that Architecture 1 is the best Autoencoder with one hidden layer.

Test Reconstruction Error: 3.0504

Below is the plot of Average Error v/s epoch of the best architecture we obtained.

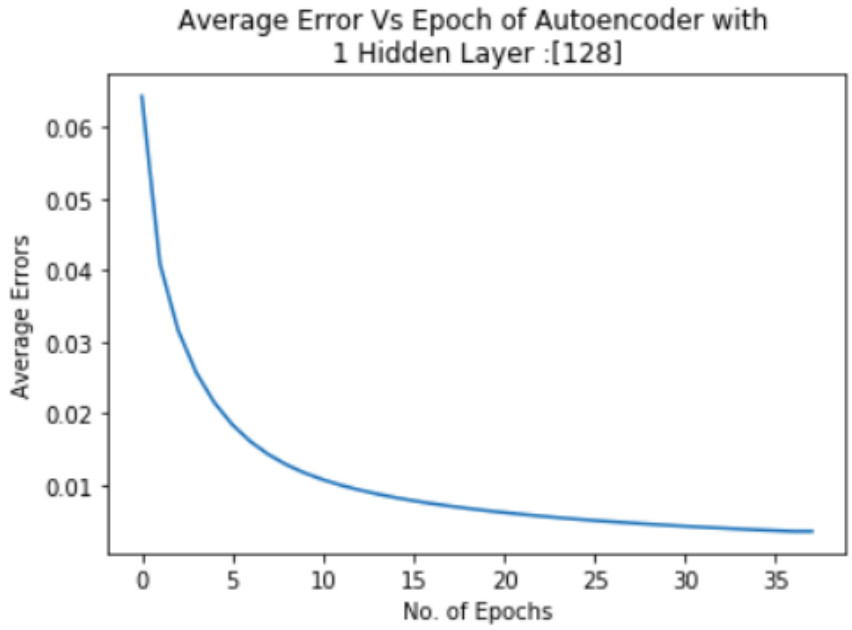


Fig. 86: Average Error v/s epoch of the best architecture

Now pass the 8 random images from the training set into the autoencoder and get the reconstructed image as an output. Below is the plot for the same.

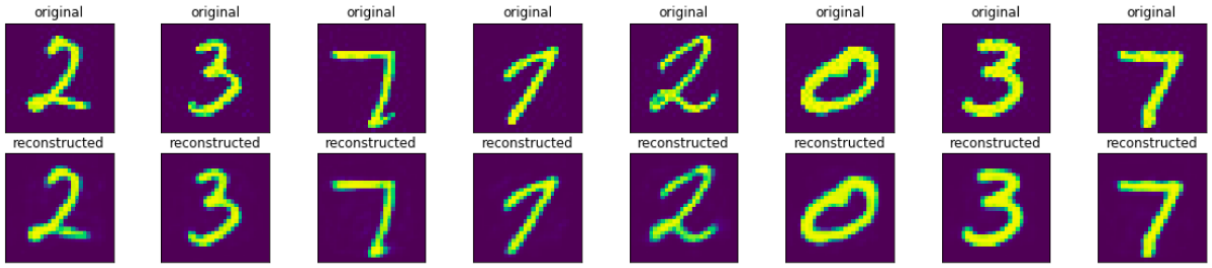


Fig. 87: Original image and reconstructed image obtained from the autoencoders on train data

Similar for the validation data set.

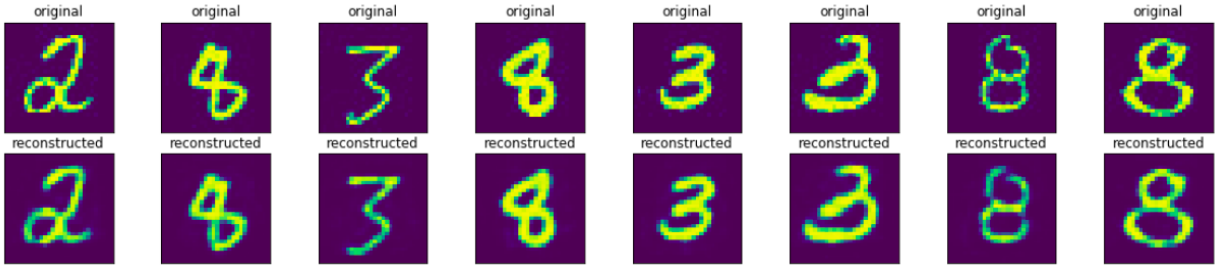


Fig. 87: Original image and reconstructed image obtained from the autoencoders on validation data

Classification Using the compressed representation on different FCNN architecture:

We have used three different architectures for FCNN.

- Learning Rate: 0.001
- Optimizer: Adam
- Batch Size: 32
- Convergence Criteria: Difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture	Input Neuron(Compressed data get from Encoder)	Hidden Layers	Output Layer Neuron
1	128	[64]	5
2	128	[64,32,16]	5
3	128	[128, 32]	5

Table 10: Different architecture used for classification

Architecture 1:

Confusion Matrix and Accuracy of train, validation and test dataset.

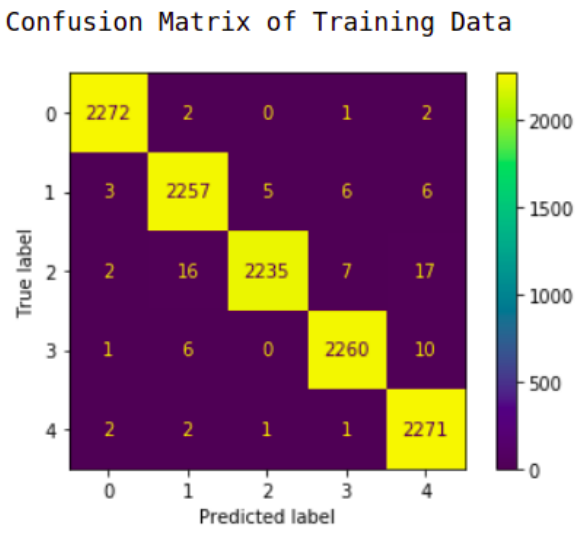


Fig. 88: Confusion matrix for training data for architecture 1

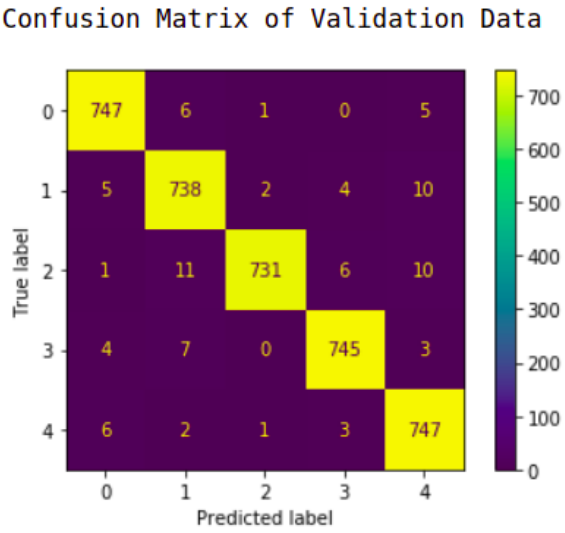


Fig. 89: confusion matrix of validation data for architecture 1

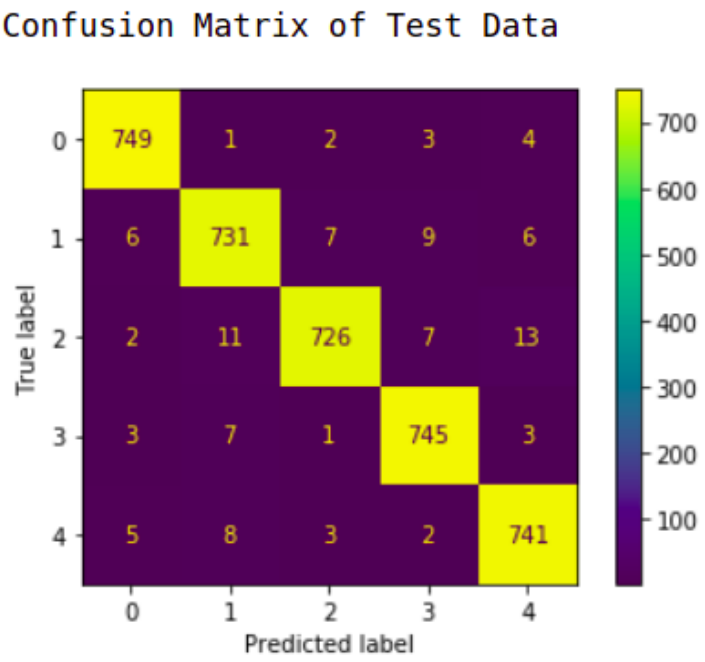


Fig. 90 Confusion matrix for test data

Accuracy:

- Train Data: 99.21%
- Validation Data: 97.71%
- Test Data: 97.28%

Architecture 2:

Confusion Matrix and Accuracy of train, validation and test dataset.

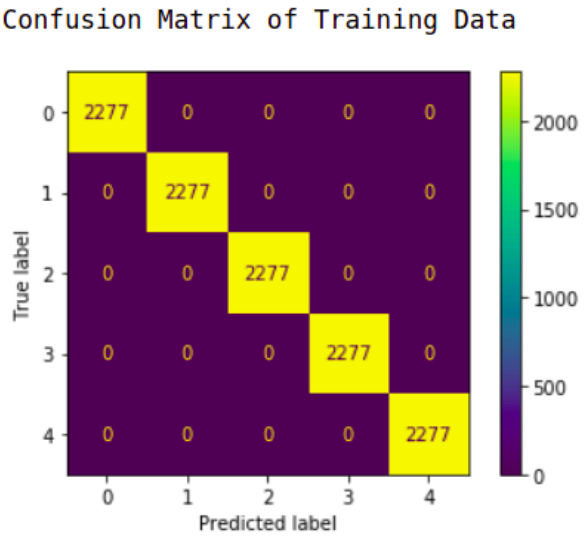


Fig. 91 Confusion matrix for training data for architecture 2

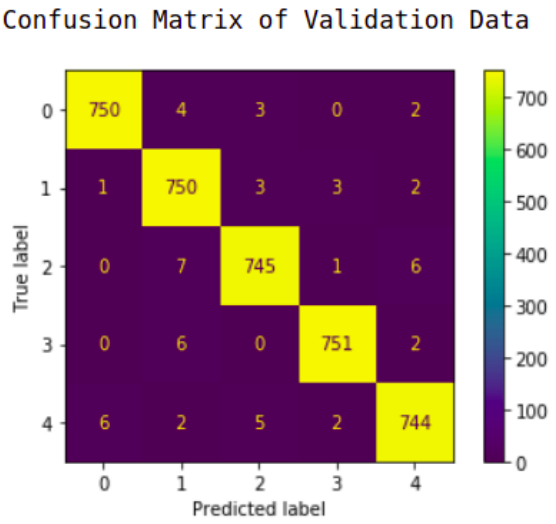


Fig. 92 confusion matrix of validation data for architecture 2

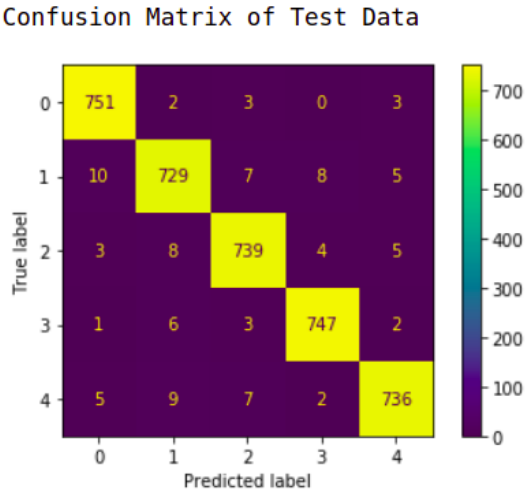


Fig. 93 Confusion matrix for test data for architecture 2

Accuracy:

- Train Data: 100%
- Validation Data: 98.55%
- Test Data: 97.55%

Architecture 3:

Confusion Matrix and Accuracy of train, validation and test dataset.

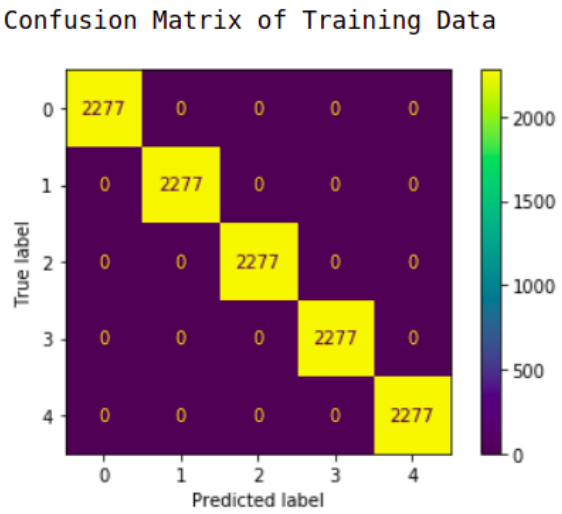


Fig. 94 Confusion matrix of training data architecture 3

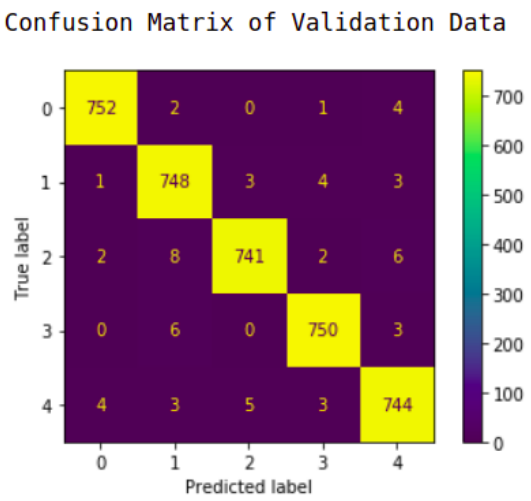


Fig. 95 confusion matrix of validation data for architecture 3

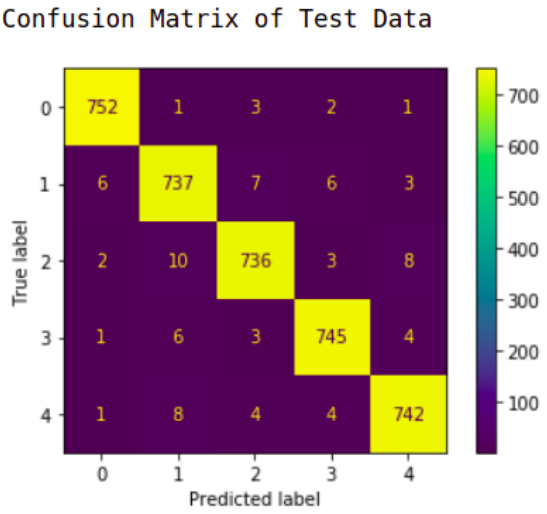


Fig. 96 Confusion matrix for test data for architecture 3

Accuracy:

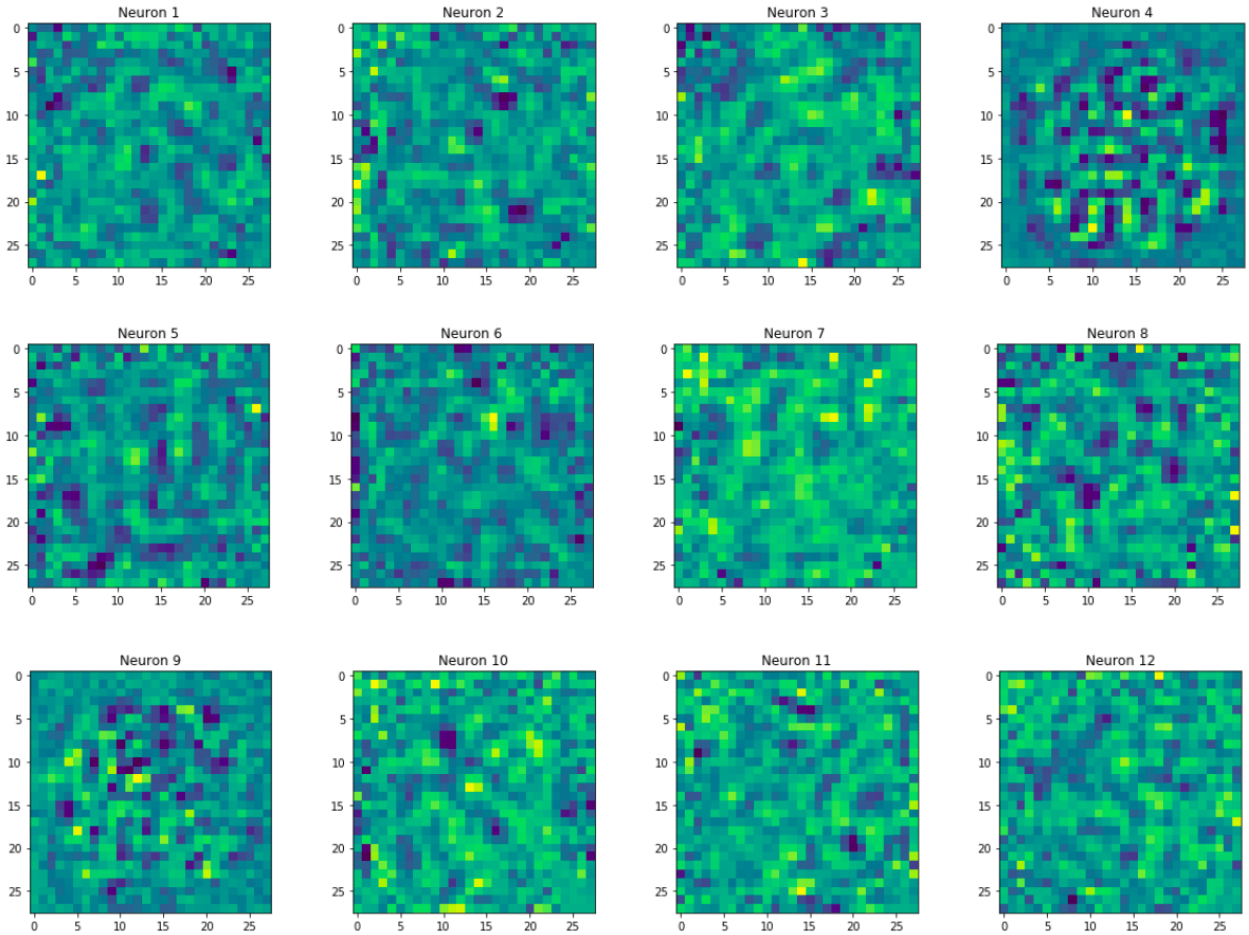
- Train Data: 100%
- Validation Data: 98.42%
- Test Data: 97.81%

Observation:

In comparison for the classification using the original data with the compressed data there is a unit decrease in the accuracy in case of compressed data that we obtained from encoder. This is the comparison with the best architecture of FCNN and one hidden layer encoder that gets from autoencoder. In the case of FCNN using an adam optimizer we almost get 98% accuracy for test data.

Weight Visualization:

Out of 128 Neurons I have shown some of the weight visualization of some of the neurons.



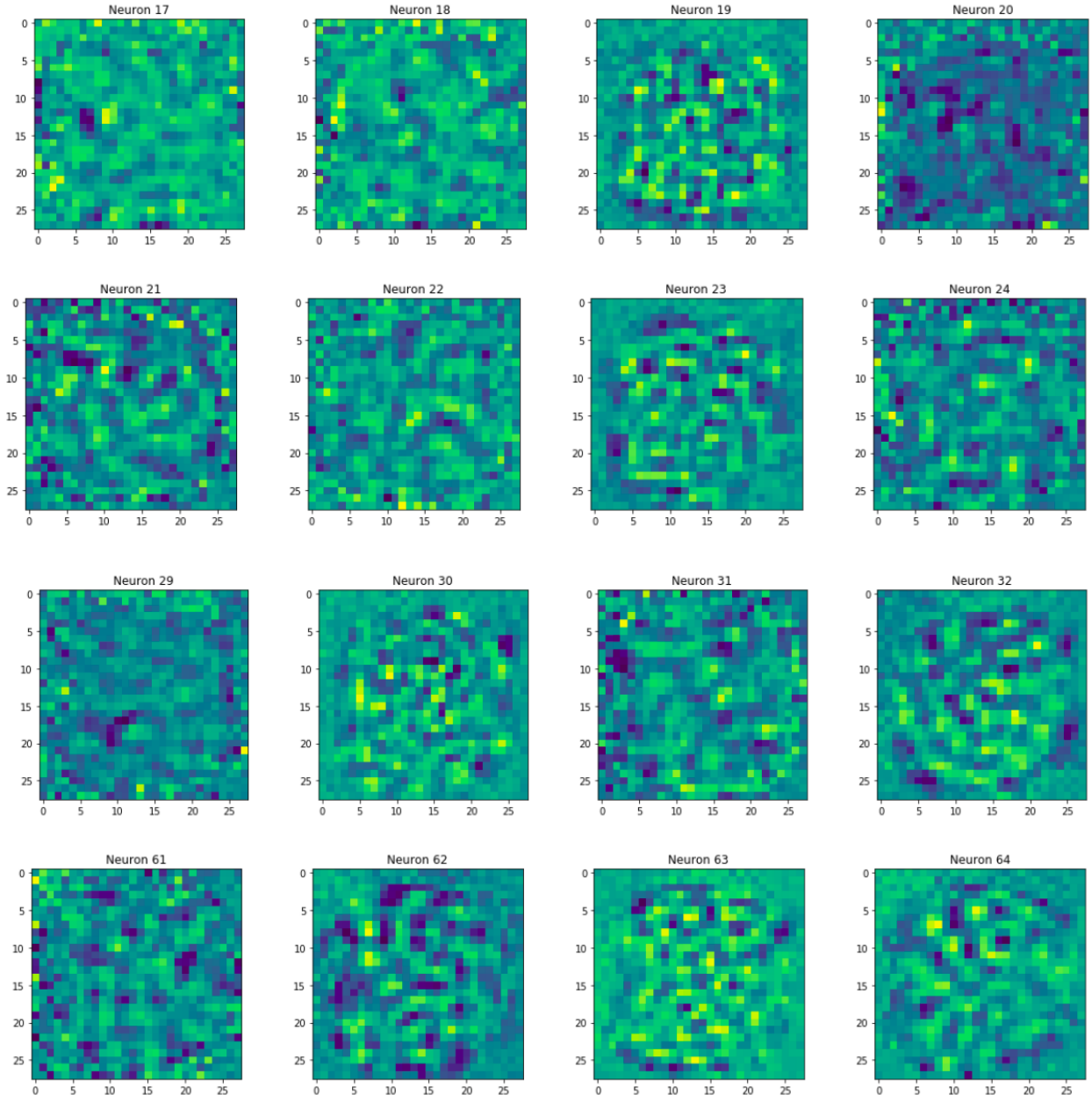


Fig. 97 Weight visualization of some of the neurons

Autoencoder with Three Hidden Layer

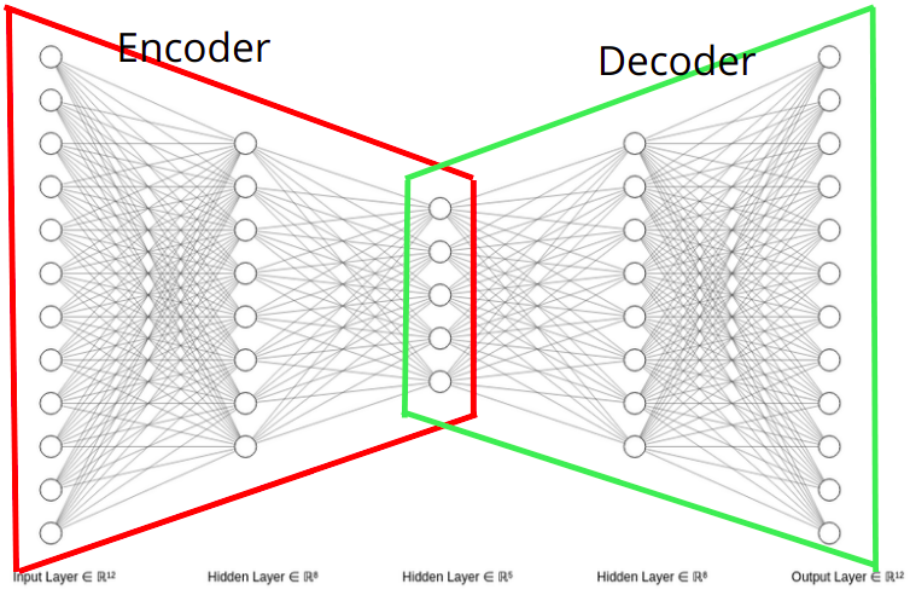


Fig. 98 Autoencoder with 3 hidden layers

Different Architecture we used:

Architecture	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	784	128	64	128	784
2	784	512	128	512	784
3	784	200	100	200	784

Table 11. Different architecture used in Autoencoders with 3 hidden layers

Training above all three architecture we have observe the average reconstruction error for training and validation data:

Architecture	Training Reconstruction Error	Validation Reconstruction Error
Architecture 1: [784,128,64,128,,784]	7.281954	8.095446
Architecture 2: [784,512,128,512,,784]	4.152769	5.265795
ARchitecture 3: [784,200,100,200,784]	5.6715014	6.575053

Table 12. Reconstruction error for different architecture

Observation:

From the above validation reconstruction error we can say that Architecture 2 is the best Autoencoder with three hidden layers.

Test Reconstruction Error: 5.3642063

Below is the plot of Average Error v/s epoch of the best architecture we obtained.

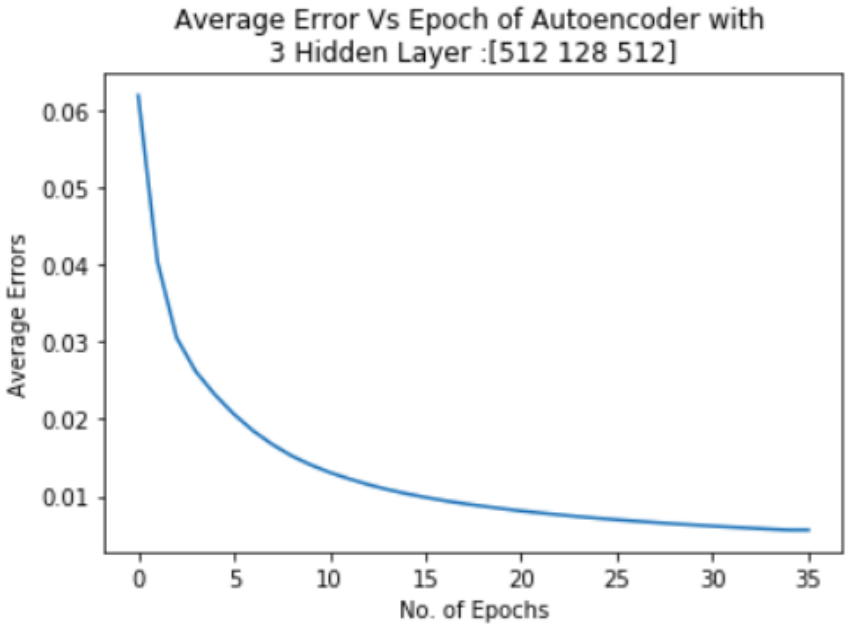


Fig. 99 Avergae error vs epoch of autoencoder

Now pass the 8 random images from the training set into the autoencoder and get the reconstructed image as an output. Below is the plot for the same.

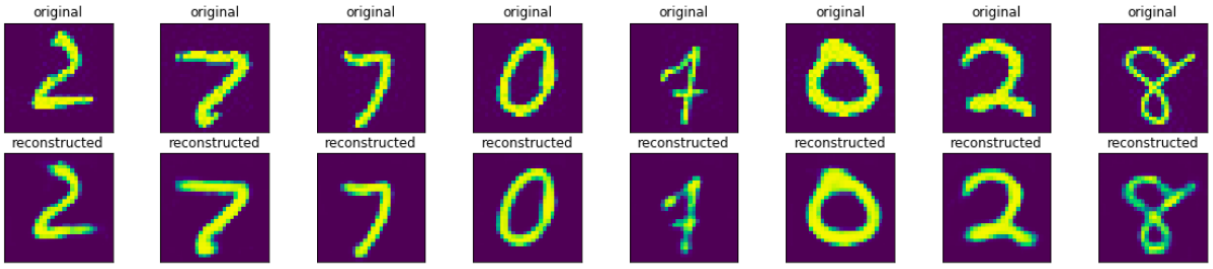


Fig. 100 Original image vs reconstructed image of training set

Similar for the validation data set.

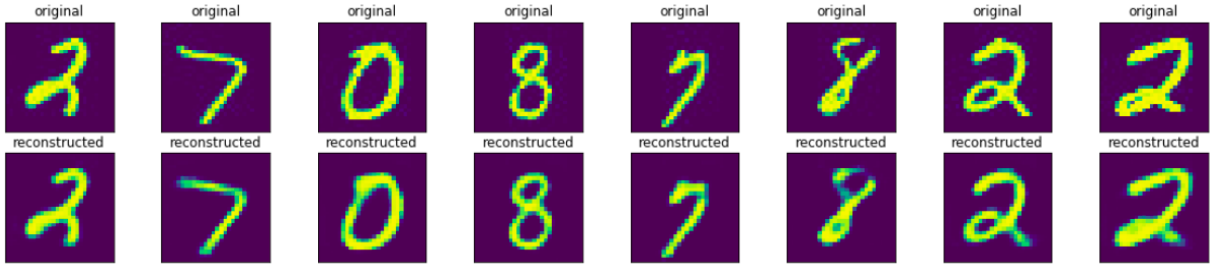


Fig. 101 Original image vs reconstructed image of validataion set

Classification Using the compressed representation on different FCNN architecture:

We have used three different architectures similar to the one layer autoencoder for FCNN.

- Learning Rate: 0.001
- Optimizer: Adam
- Batch Size: 32
- Convergence Criteria: Difference between average error of successive epochs fall below a threshold 10^{-4}

Architecture	Input Neuron(Compressed data get from Encoder)	Hidden Layers	Output Layer Neuron
1	128	[64]	5
2	128	[64,32,16]	5
3	128	[128, 32]	5

Table 13. Different FCNN architecture used for the classification

Architecture 1:

Confusion Matrix and Accuracy of train, validation and test dataset.

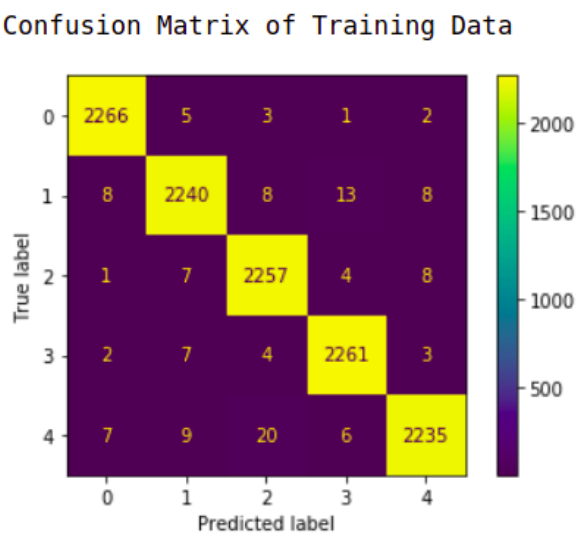


Fig. 102 Confusion matrix for training data for architecture 1

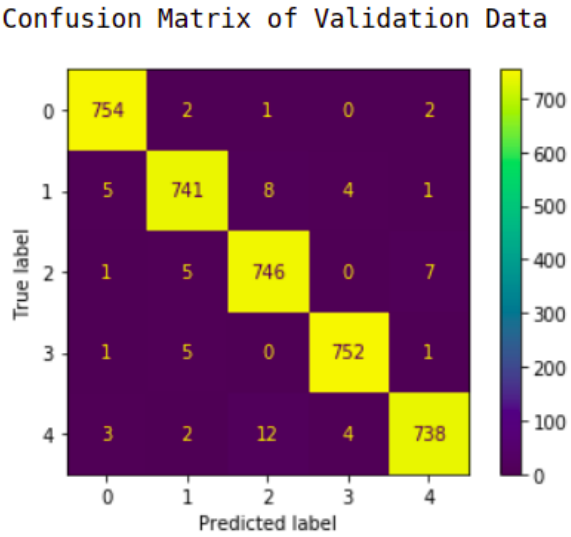


Fig. 103 confusion matrix of validation data for architecture 1

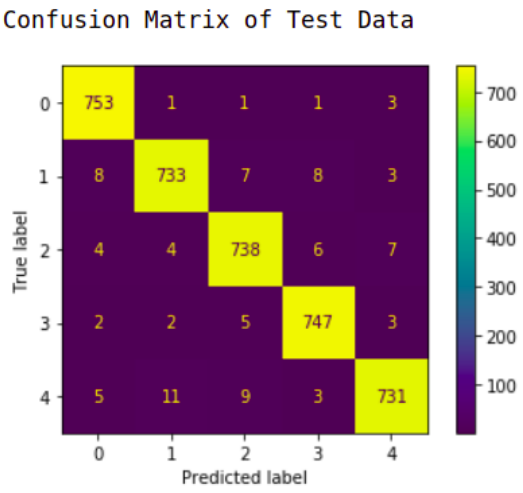


Fig. 104 Confusion matrix for test data for architecture 1

Accuracy:

- Train Data: 98.89%
- Validation Data: 98.31%
- Test Data: 97.55%

Architecture 2:

Confusion Matrix and Accuracy of train, validation and test dataset.

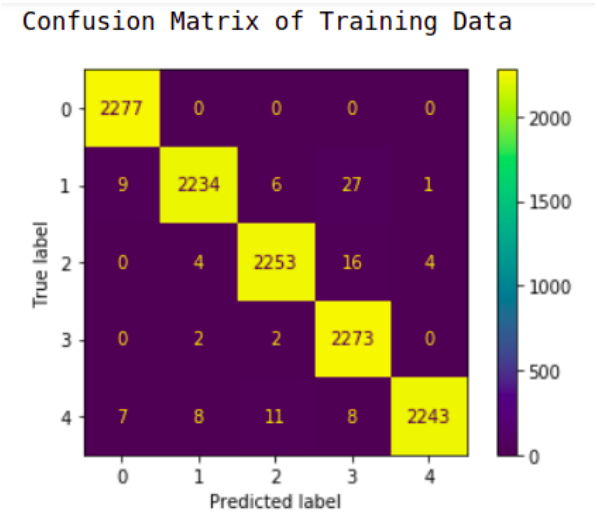


Fig. 105 Confusion matrix for training data for architecture 2

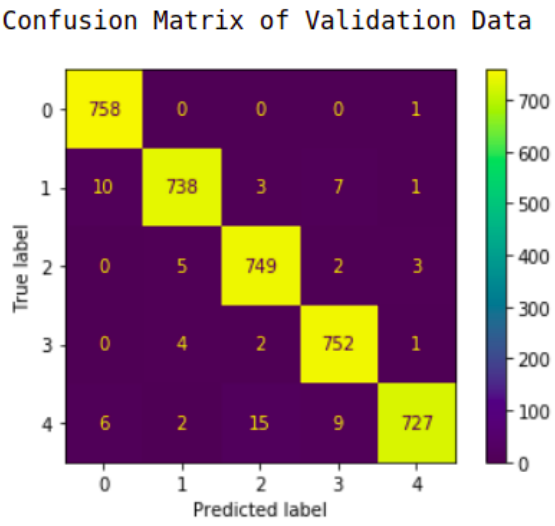


Fig. 106 confusion matrix of validation data for architecture 2

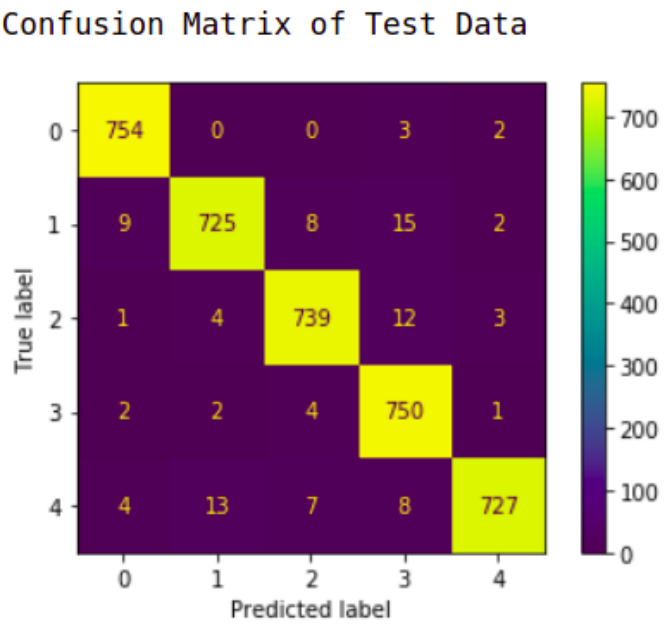


Fig. 107 Confusion matrix for test data for architecture 3

Accuracy:

- Train Data: 99.07%
- Validation Data: 98.13%
- Test Data: 97.36%

Architecture 3:

Confusion Matrix and Accuracy of train, validation and test dataset.

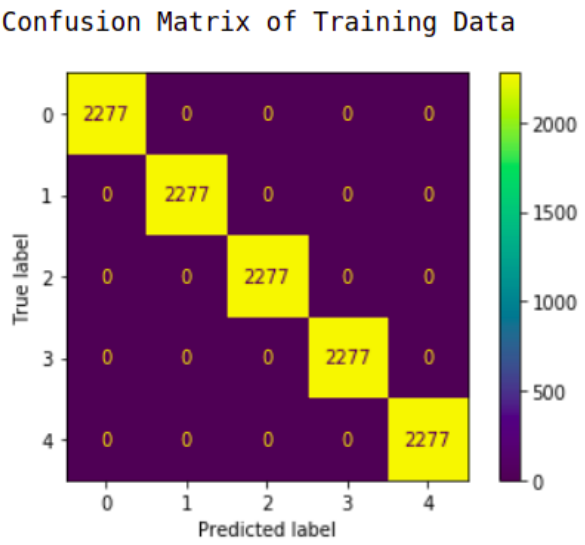


Fig. 108 Confusion matrix for training data for architecture 3

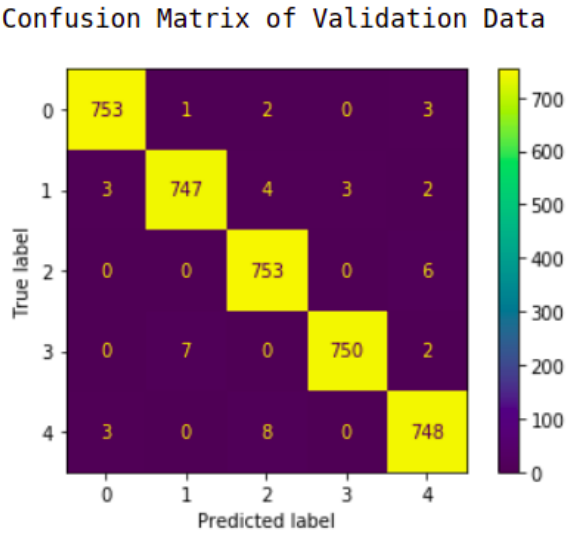


Fig. 109 confusion matrix of validation data for architecture 3

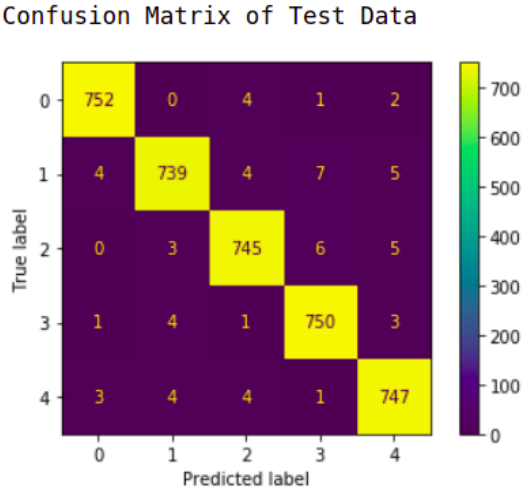


Fig. 110 Confusion matrix for test data for architecture 3

Accuracy:

- Train Data: 100%
- Validation Data: 98.36%
- Test Data: 98.84%

Observation:

In the case of an autoencoder with three hidden layers. The compressed data we obtained is used further for classification. We observed that there is not much difference in accuracy with the FCNN we used for classification. However, compressed data obtained from the encoder of three hidden layers autoencoders give better accuracy on test data as compared to the compressed data we used for classification that we get from one hidden layer autoencoder.

Denoising Autoencoder

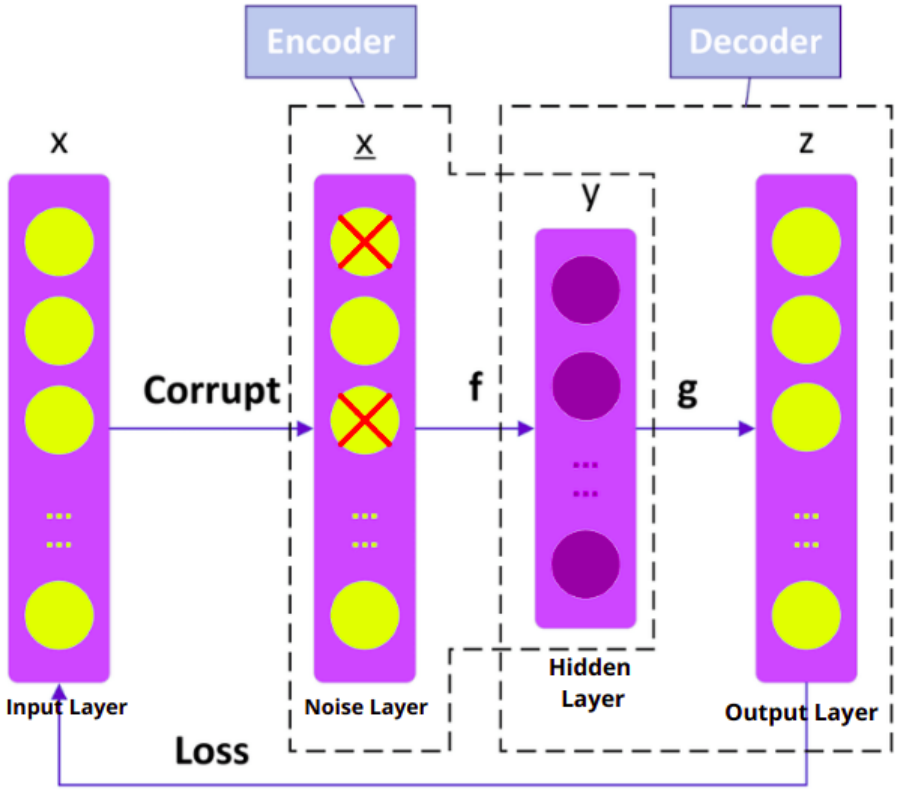


Fig. 111 Denoising Autoencoder

In Denoising Autoencoder we corrupt the input with a probability of “q” a Gaussian noise is added to ith input value (x_{ni}).

$$\tilde{X}_{ni} = X_{ni} + N(0,1)$$

Below is the Denoising Autoencoder Architecture we used :

Encoder

Decoder

Layer (type)	Output Shape	Param #
flatten_18 (Flatten)	(None, 784)	0
noise_5 (Noise)	(None, 784)	0
dense_66 (Dense)	(None, 128)	100480
dense_67 (Dense)	(None, 784)	101136
reshape_12 (Reshape)	(None, 28, 28)	0

Total params: 201,616
Trainable params: 201,616
Non-trainable params: 0

Fig. 112 Denoising Autoencoder Architecture

1. Adding Noise with probability of 20% i.e 0.2

After training Model below is the plot between original and the reconstructed image after adding noise

On training data

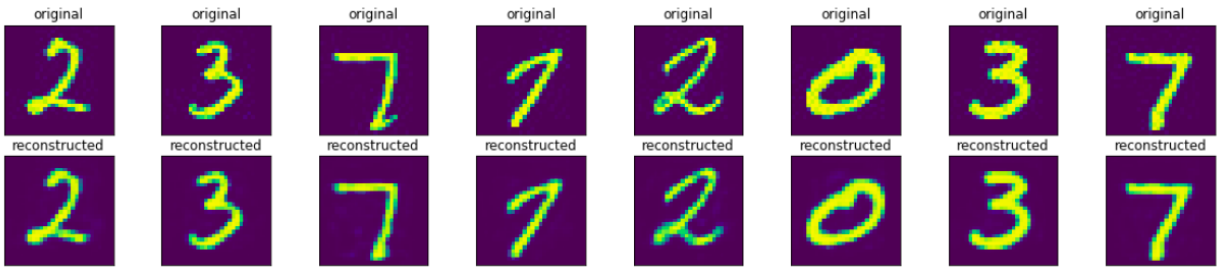


Fig. 113 Original and the reconstructed image after adding noise on training data

On validation Data

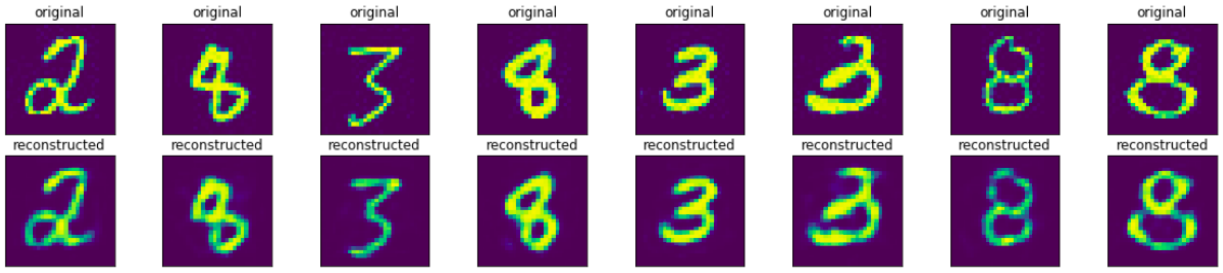


Fig. 113 Original and the reconstructed image after adding noise on validation data

Now used this compressed data for classification with different FCNN architecture.

Architecture	**Input Neuron(Compressed data get from Encoder)**	**Hidden Layers**	**Output Layer Neuron**	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**
1	128	[64]	5	0.999385	0.977339	0.970487
2	128	[64,32,16]	5	0.995169	0.970224	0.966535
3	128	[128, 32]	5	0.989548	0.968906	0.965217
4	128	[64, 32]	5	1	0.977866	0.973123
5	128	[32]	5	0.975933	0.966008	0.958103

Table 14. Different architecture used for the classification

Observation:

From the test accuracy we observe that classification using the compressed data obtained from the denoising autoencoder does not give enough good accuracy as compared to normal FCNN or autoencoder. On the basis of validation accuracy Architecture 4 is the best architecture. Below is the confusion matrix for the Architecture 4 of training, validation and test data.

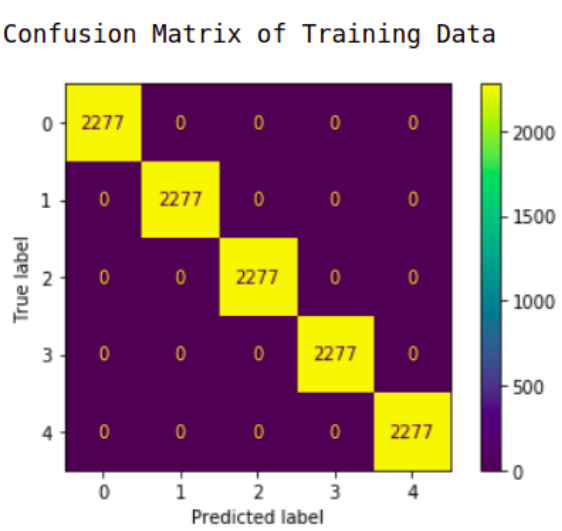


Fig. 114 Confusion matrix for training data for architecture 4

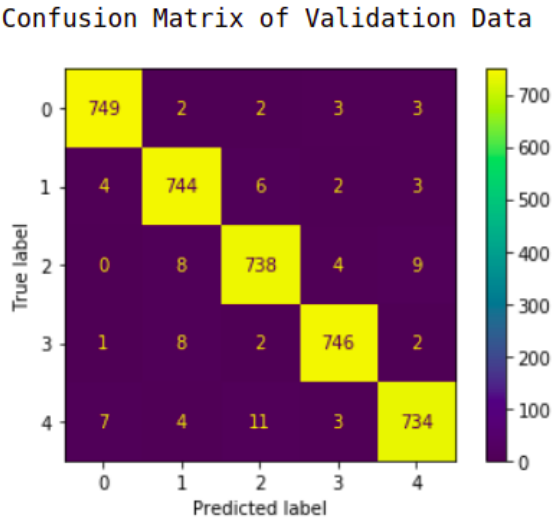


Fig. 115 confusion matrix of validation data for architecture 4

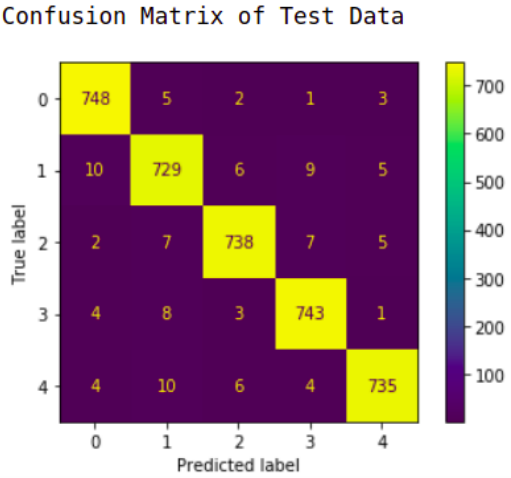
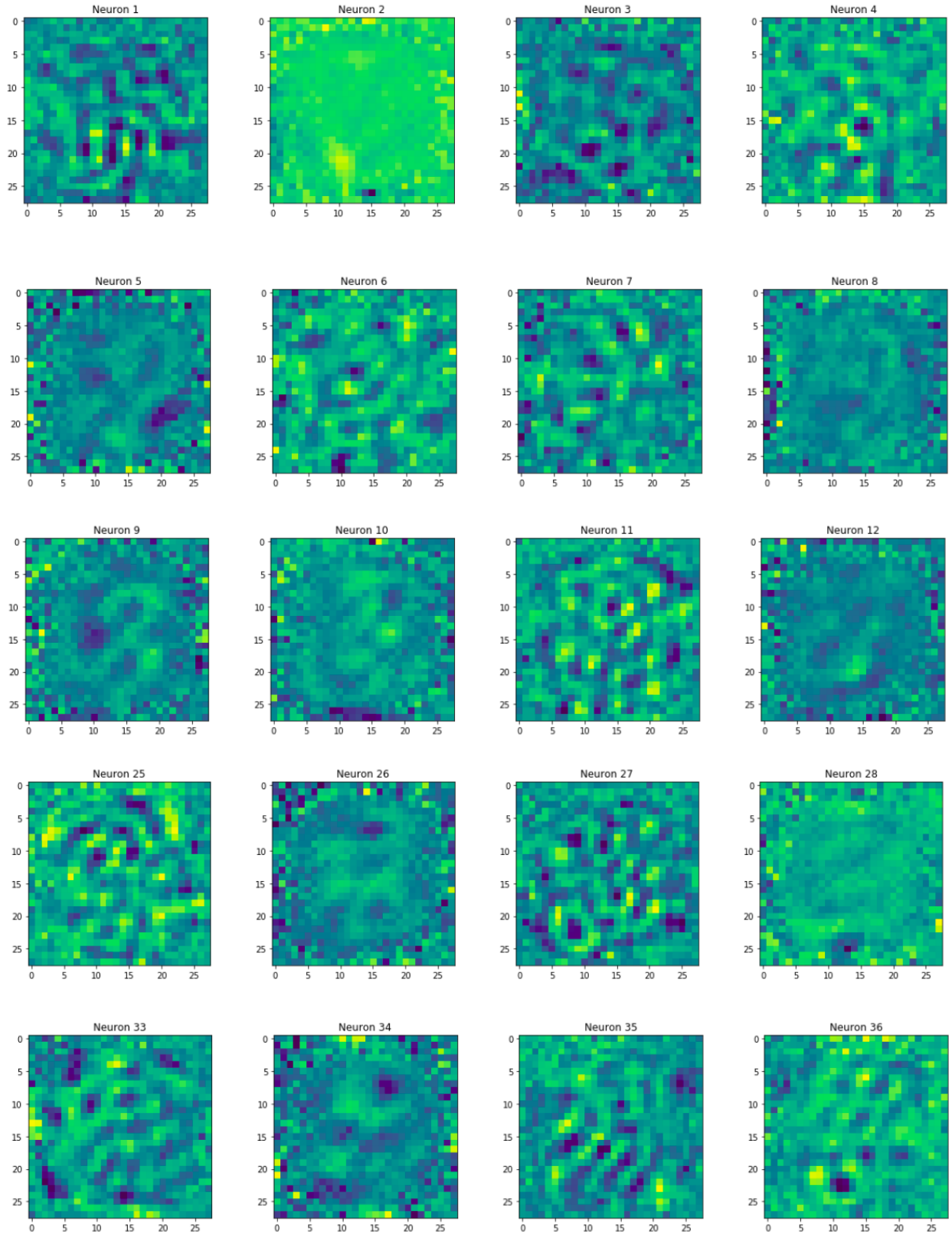


Fig. 116 Confusion matrix for test data for architecture 4

Weight Visualization:

We have showed weights visualization of some of the neurons


Neuron 1
Neuron 2
Neuron 3
Neuron 4
Neuron 5
Neuron 6
Neuron 7
Neuron 8
Neuron 9
Neuron 10
Neuron 11
Neuron 12
Neuron 25
Neuron 26
Neuron 27
Neuron 28
Neuron 33
Neuron 34
Neuron 35
Neuron 36

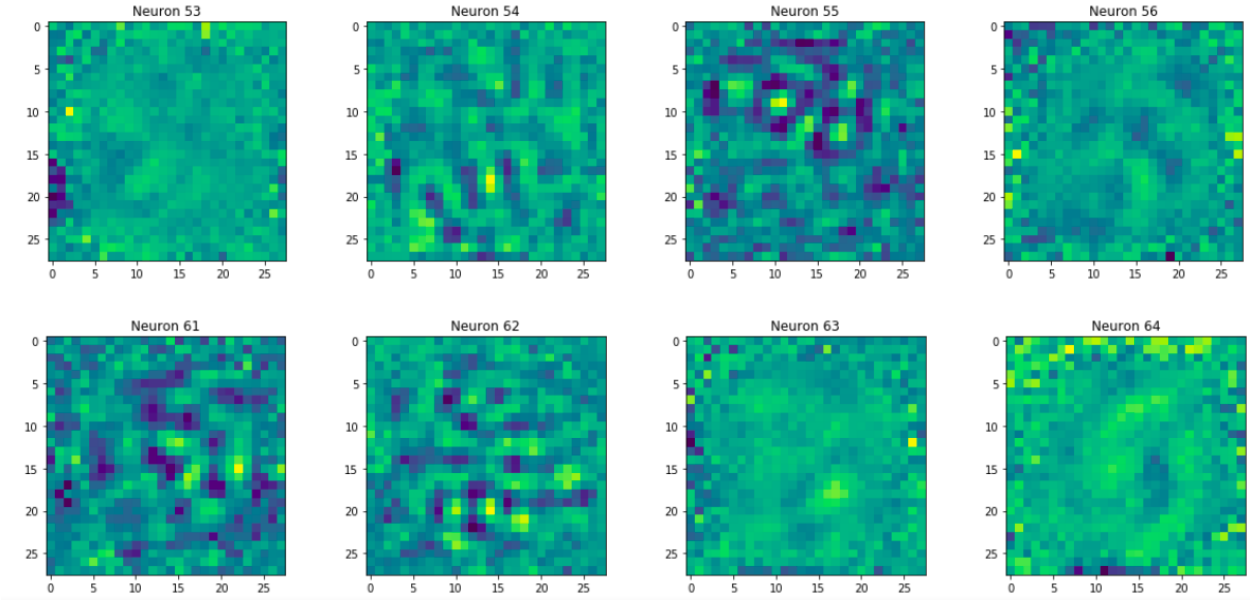


Fig. 117 weights visualization of some of the neurons

From above plots we can see that due to noise some of the neurons are not showing some particular number.

2. Adding Noise with probability of 40% i.e 0.4

After training Model below is the plot between original and the reconstructed image after adding noise

On training data

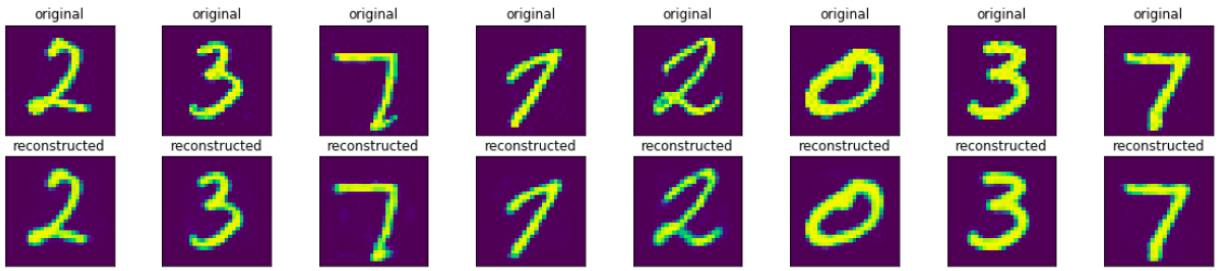


Fig. 118 Original and the reconstructed image after adding noise for training data

On validation Data

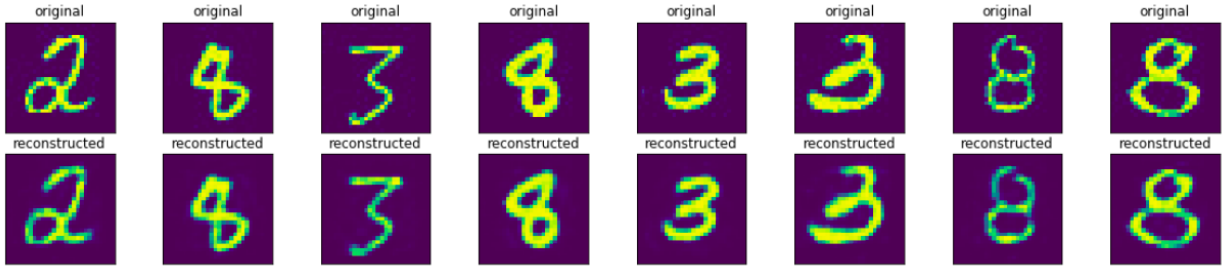


Fig. 119 Original and the reconstructed image after adding noise for validation data

Now used this compressed data for classification with different FCNN architecture.

Architecture	Input Neuron(Compressed data get from Encoder)	Hidden Layers	Output Layer Neuron	Training Accuracy	Validation Accuracy	Test Accuracy
1	128	[64]	5	1	0.979974	0.974177
2	128	[64,32,16]	5	0.999912	0.982345	0.972332
3	128	[128, 32]	5	1	0.982609	0.975758
4	128	[64, 32]	5	0.998419	0.978393	0.970224
5	128	[32]	5	0.998331	0.974704	0.967062

Table 15. Different architecture used for the classification

Observation:

After adding 40% noise, we used the compressed data for classification. We observe a very good result in this case as compared to Denoising autoencoder with 20% noise. However still this denoising autoencoder can't beat the FCNN model. In this case on the basis of validation accuracy architecture 3 is the best architecture for this classification. And also show a good accuracy on test data as compared to other architectures. Below is the confusion matrix of Architecture 3 of the training, validation and test data.

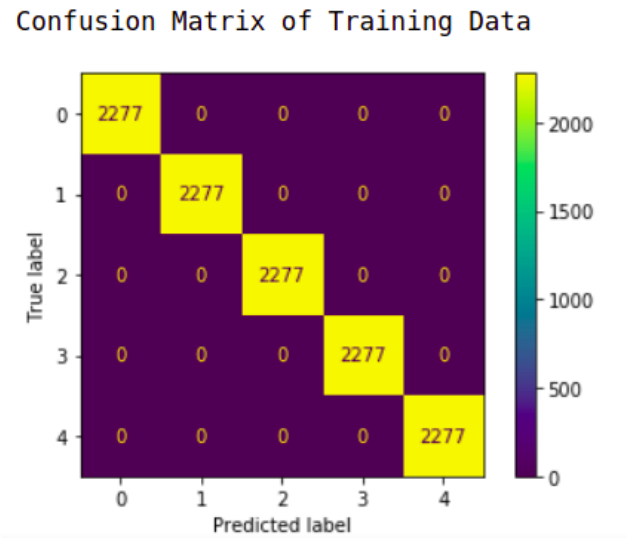


Fig. 120 Confusion matrix for training data for architecture 3

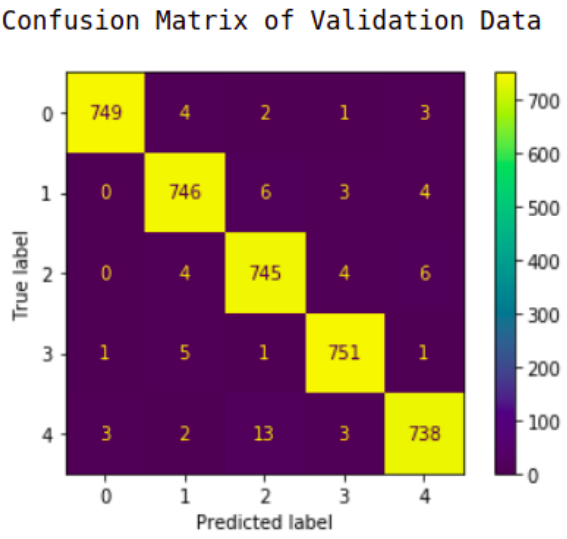


Fig. 121 confusion matrix of validation data for architecture 3

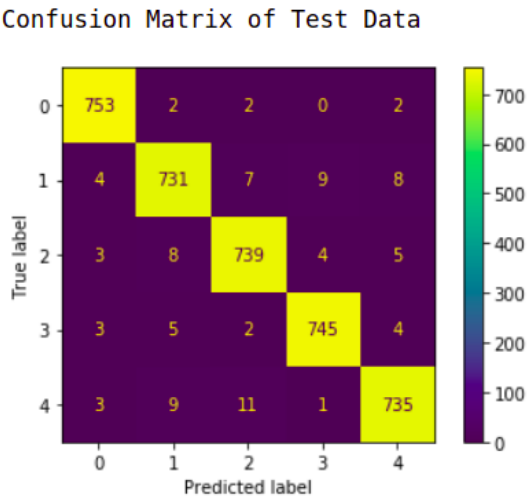


Fig. 122 Confusion matrix for test data for architecture 3

Weight Visualization:

We have showed weights visualization of some of the neurons

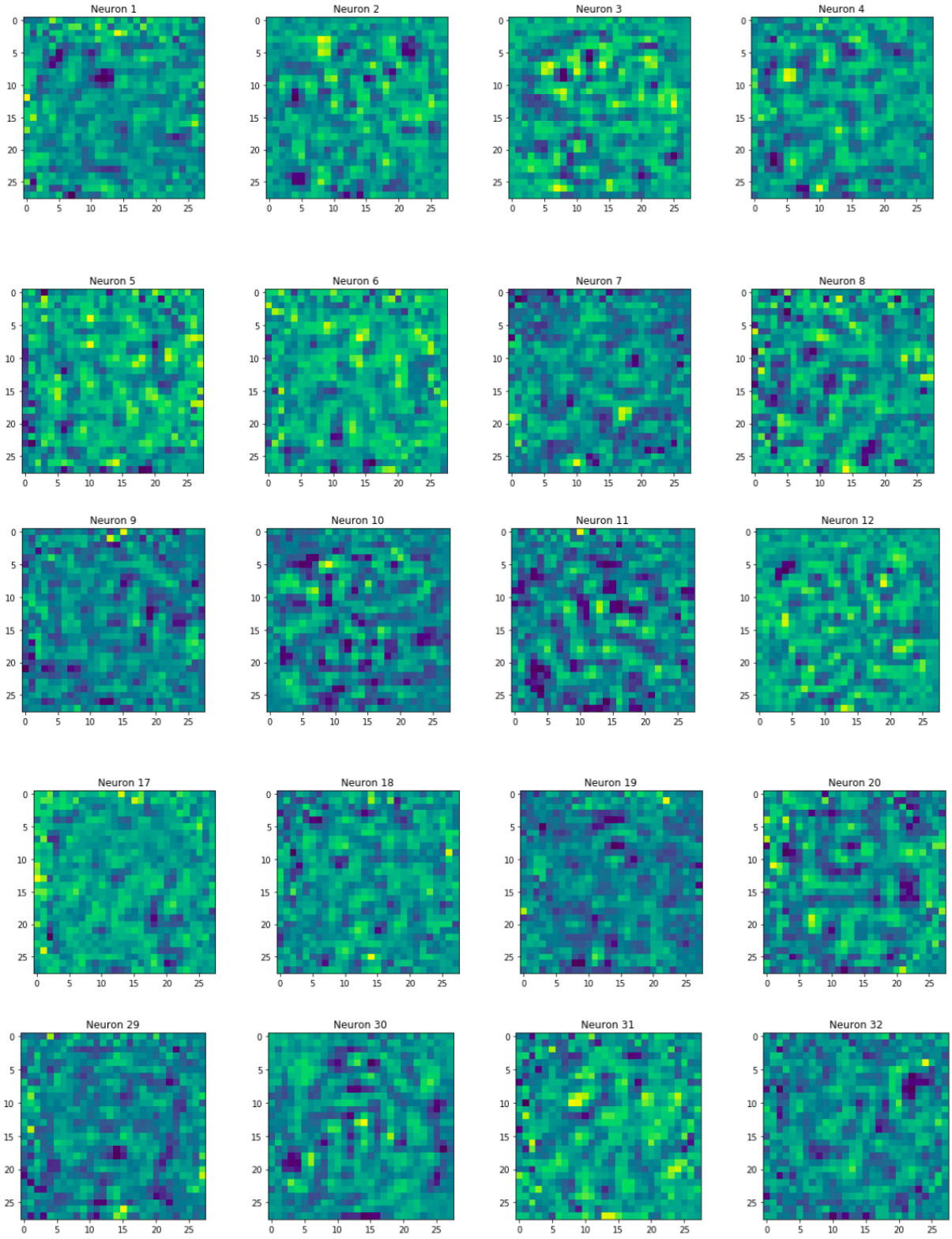


Fig. 123 weights visualization of some of the neurons

From above we can see that due to noise it was hard to figure out which class is trying to maximize that particular neuron.