A Report on

CS671: Assignment 4

Autoencoders

Team Members(Group 04):

Ananya Angra(S22025)
Shilpa Chandra(S22004)
Pushkar Kumar(S22038)

Course Instructor

Dr. Dileep A. D.

IIT Mandi Academic Year 2022-2023

Contents

Chapter 1

Autoencoder

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.

1.1 Need for Autoencoders

Auto encoders have an input layer, hidden layer, and an output layer. The input is forced to be as identical to the output, so its the hidden layer we are interested in.
The hidden layer form a kind of encoding of the input. "The aim of an auto-encoder is to learn a compressed, distributed representation (encoding) for a set of data." If input is a 100 dimensional vector, and you have 60 neurons in the hidden layer, then the auto encoder algorithm will replicate the input as a 100 dimensional vector in the output layer, in the process giving you a 60 dimensional vector that encodes your input.
So the purpose of auto encoders is dimensionality reduction, amongst many others.

1.2 Autoencoder Compenents:

Autoencoders consists of 4 main parts:

1. **Encoder:** In which the model learns how to reduce the input dimensions and compress the input data into an encoded representation.
2. **Bottleneck:** which is the layer that contains the compressed representation of the input data. This is the lowest possible dimensions of the input data.

3. **Decoder:** In which the model learns how to reconstruct the data from the encoded representation to be as close to the original input as possible.

4. **Reconstruction Loss:** This is the method that measures measure how well the decoder is performing and how close the output is to the original input.

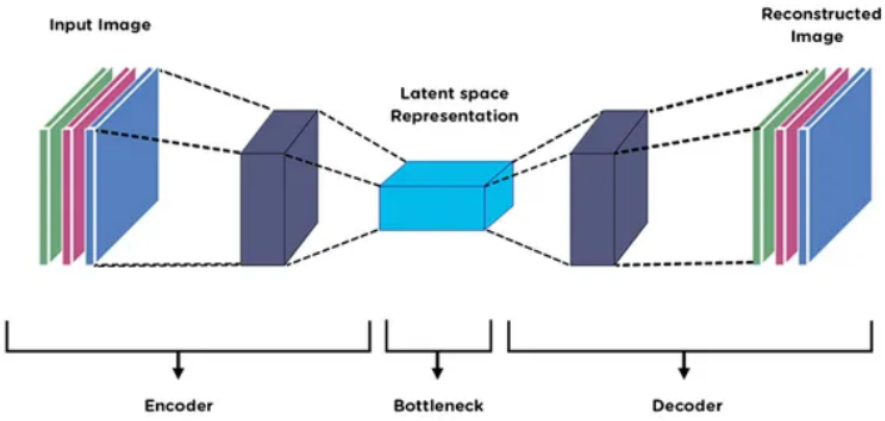


Figure 1.1: Architecture of Autoencoder[2]

The training then involves using back propagation in order to minimize the network's reconstruction loss.

1.3 Principal Component Analysis(PCA):

Principal component analysis is a technique that is widely used for applications such as dimensionality reduction, lossy data compression, feature extraction and data visualization. PCA can be defined as the orthogonal projection of the data onto a lower dimensional space known as the principal subspace such that the variance of the projected data is maximized. PCA employs a linear transformation that is based on preserving the most variance in the data using the least number of dimensions. It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigen values are used to reconstruct a large fraction of variance of the original data.

The data instances are projected onto a lower dimensional space where the new features best represent the entire data in the least squares sense. The eigenvectors of the covariance matrix of the data are referred to as principal axes of the data, and the projection of the data instances on to these principal axes are called the principal components. Dimensionality reduction is then obtained by only retaining those axes (dimensions) that account for most of the variance, and discarding all others.

- To implement PCA in Scikit learn, it is essential to standardize/normalize the data before applying PCA.
- PCA is imported from sklearn.decomposition.

  We need to select the required number of principal components.
- **We have chosen number of components to be 32, 64, 128 and 256**.
- By the fit and transform method, the attributes are passed.

1.4 Denoising Autoencoders

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

$\tilde{x_{ni}} = x_{ni} + \mathcal{N}(0, 1)$

Not all the features in an example is corrupted by noise. They're corrupted by a probability of q and hence retained with a probability of 1-q . Therefore ,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.
It helps because the objective is still to reconstruct the original (uncorrupted) input x_n.

$$\min \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{d}(\hat{x}_{ni} - x_{ni})^2$$

The denoising autoencoder learns 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.

1.5 Weight visualization

Weights visualization here mean to plot the inputs as images that maximally activate each of the neurons of the hidden representations.

The neuron acts as a filter which will fire/activated(or maximally fire) for a certain input configuration x_n.
The neurons will be maximally activated when h_{nj} is high, which in turn depends on a_{nj} which depends on ${w_j}^T x_n$. But weights are fixed after training, so we need to optimize x_n.

$$\max_{\mathbf{x}_n} \quad \mathbf{w}_j^T \mathbf{x}_n$$
$$s.t. \quad \|\mathbf{x}_n\|^2 = \mathbf{x}_n^T \mathbf{x}_n = 1$$

Figure 1.2: Equation to be maximized[1]

$$\text{solution:} \quad \mathbf{x}_n = \frac{\mathbf{w}_j}{\|\mathbf{w}_j\|} = \frac{\mathbf{w}_j}{\sqrt{\mathbf{w}_j^T \mathbf{w}_j}}$$

Figure 1.3: Optimized Solution[1]

We want to plot these x_n as images which maximally activate each of the neurons of the hidden representations learned by an autoencoder. The derivation for Optimized solution can be found in Chapter 7.

Chapter 2

Task 1: Dimension reduction using PCA

from sklearn.decomposition import PCA has been used for Dimension reduction using PCA. The following three architectures have been used throughout the assignment:

- Three hidden layer (64, 32 & 64 neurons respectively in each hidden layer)
- Three hidden layer (512, 128 & 64 neurons respectively in each hidden layer)
- Five hidden layer (512, 512 256, 256 & 256 neurons respectively in each hidden layer)

The results below for 32,64,128 & 256 reduced dimensions are as follows:

2.1 Results for Reduced dimension: 32

Reduced Dimension - 32	
Architecture	Validation Accuracy
64-32-64	0.9715
512-128-64	0.9776
512-512-256-256-256	0.9802

Table 2.1: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-512-256-256-256**

- Test accuracy for best architecture: **0.9794**
- Confusion matrix for best architecture:

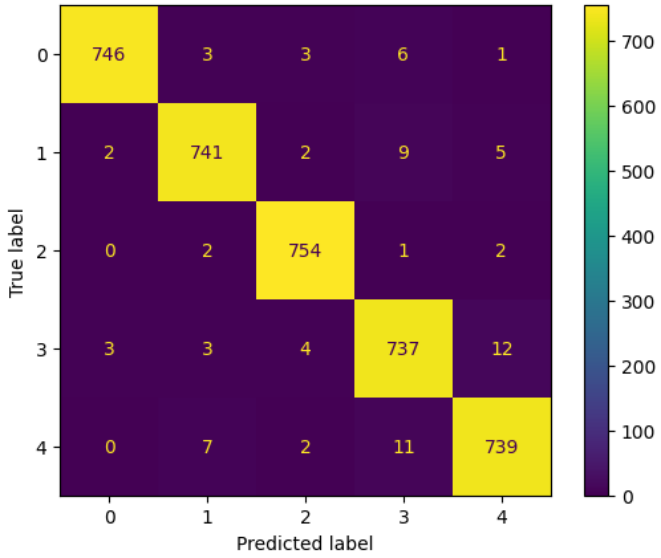


Figure 2.1: Confusion Matrix for best architecture

2.2 Results for Reduced dimension: 64

Reduced Dimension - 64	
Architecture	Validation Accuracy
64-32-64	0.9731
512-128-64	0.9812
512-512-256-256-256	0.9794

Table 2.2: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9810**
- Confusion matrix for best architecture:

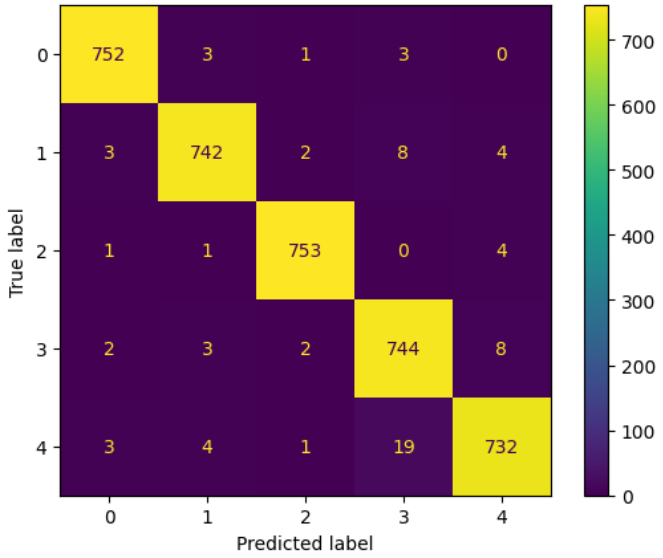


Figure 2.2: Confusion Matrix for best architecture

2.3 Results for Reduced dimension: 128

Reduced Dimension - 128	
Architecture	Validation Accuracy
64-32-64	0.9760
512-128-64	0.9765
512-512-256-256-256	0.9707

Table 2.3: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9752**
- Confusion matrix for best architecture:

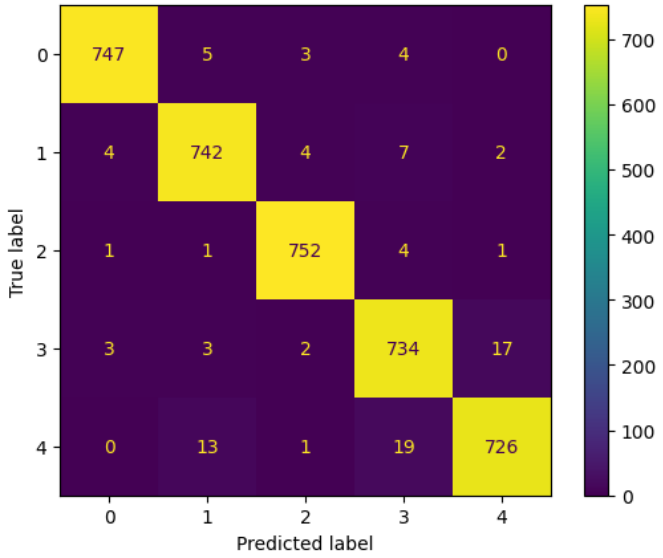


Figure 2.3: Confusion Matrix for best architecture

2.4 Results for Reduced dimension: 256

Reduced Dimension - 256	
Architecture	Validation Accuracy
64-32-64	0.9718
512-128-64	0.9807
512-512-256-256-256	0.9757

Table 2.4: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9784**
- Confusion matrix for best architecture:

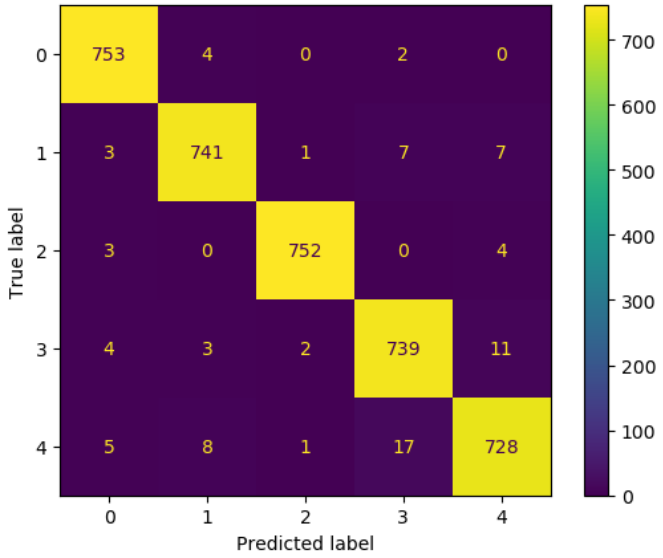


Figure 2.4: Confusion Matrix for best architecture

2.5 Best reduced dimension representation(from the all the test accuracies

Reduced Dimension - 32	Architecture	Test Accuracy
	64-32-64	0.9655
	512-128-64	0.9771
	512-512-256-256-256	0.9794
Reduced Dimension - 64	Architecture	Test Accuracy
	64-32-64	0.9710
	512-128-64	0.9810
	512-512-256-256-256	0.9773
Reduced Dimension - 128	Architecture	Test Accuracy
	64-32-64	0.9739
	512-128-64	0.9752
	512-512-256-256-256	0.9729
Reduced Dimension - 256	Architecture	Test Accuracy
	64-32-64	0.9702
	512-128-64	0.9784
	512-512-256-256-256	0.9755

Table 2.5: test accuracies for all architectures for all reduced dimension representations

From Table 2.5 above the best reduced dimension is from 64 reduced dimension(Architecture: 512-128-64) as it's test accuracy is the highest : 0.9810. The confusion matrix for the same can be seen in Fig 2.2

2.6 Comparison with Assignment 3

- The best architecture for Assignment 3 was with Five hidden layers (512, 512 256, 256 & 256 neurons respectively in each hidden layer) with Adam optimizer. This gave **Test accuracy: 0.9701** and **Validation accuracy: 0.9749**
- The best architecture for Task 1 is with Reduced dimension **64** with Three hidden layer (512, 128 & 64 neurons respectively in each hidden layer) with Adam optimizer. And this gave **Test accuracy: 0.9874** and **Validation accuracy: 0.9807**
- Comparing confusion matrix for the best architectures of Assignment 3 and Task 1

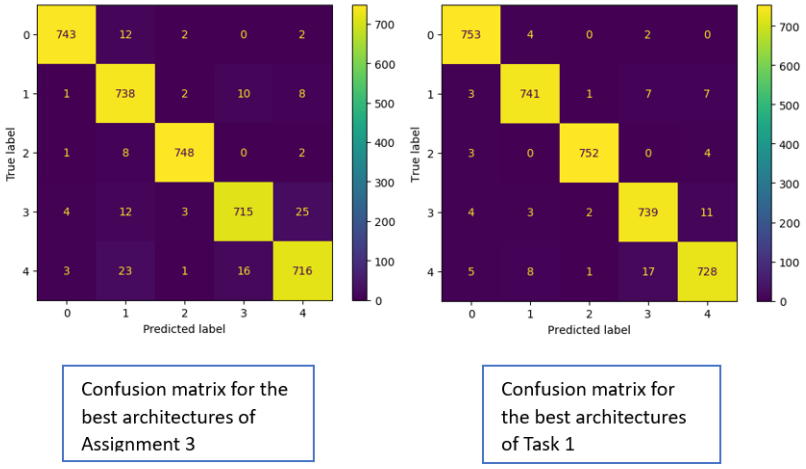


Figure 2.5: Confusion matrix for the best architectures of Assignment 3 and Task 1

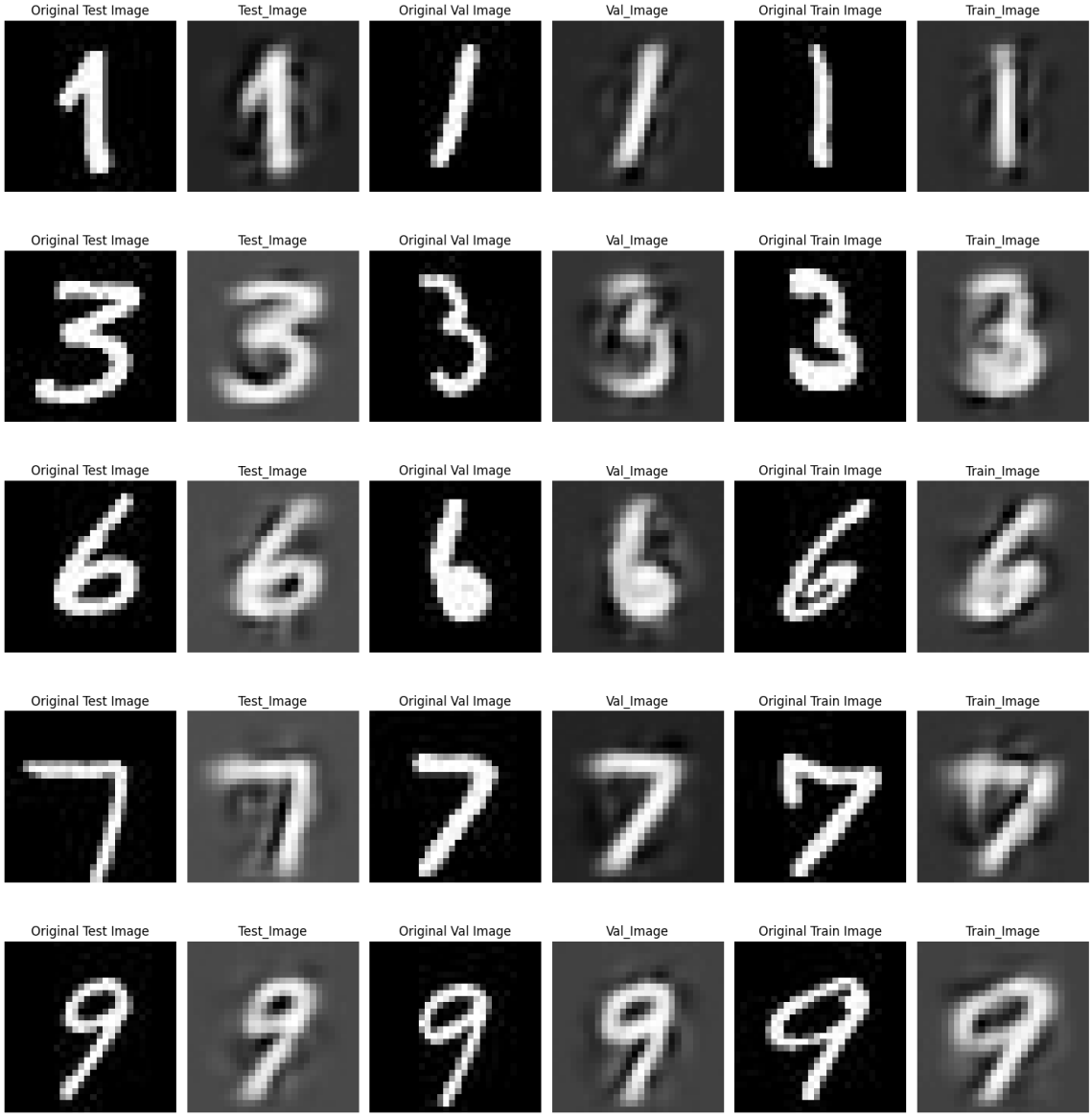
Chapter 3

Task 2: Autoencoders for reconstructing the images

3.1 Autoencoders with one hidden layer

- Considered 32, 64, 128 and 256 neurons for the bottleneck (middle) layer.
- Autoencoder is trained using Adam optimizer
- **tanh** used as activation function for the nodes in all the hidden layers
- Average reconstruction error is computed after the model is trained based on Mean squared error(MSE)
- Same image from each class is reconstructed for test dataset and then same is repeated for training and validation dataset across all architectures

1. **With 32 neurons in the bottleneck layer**

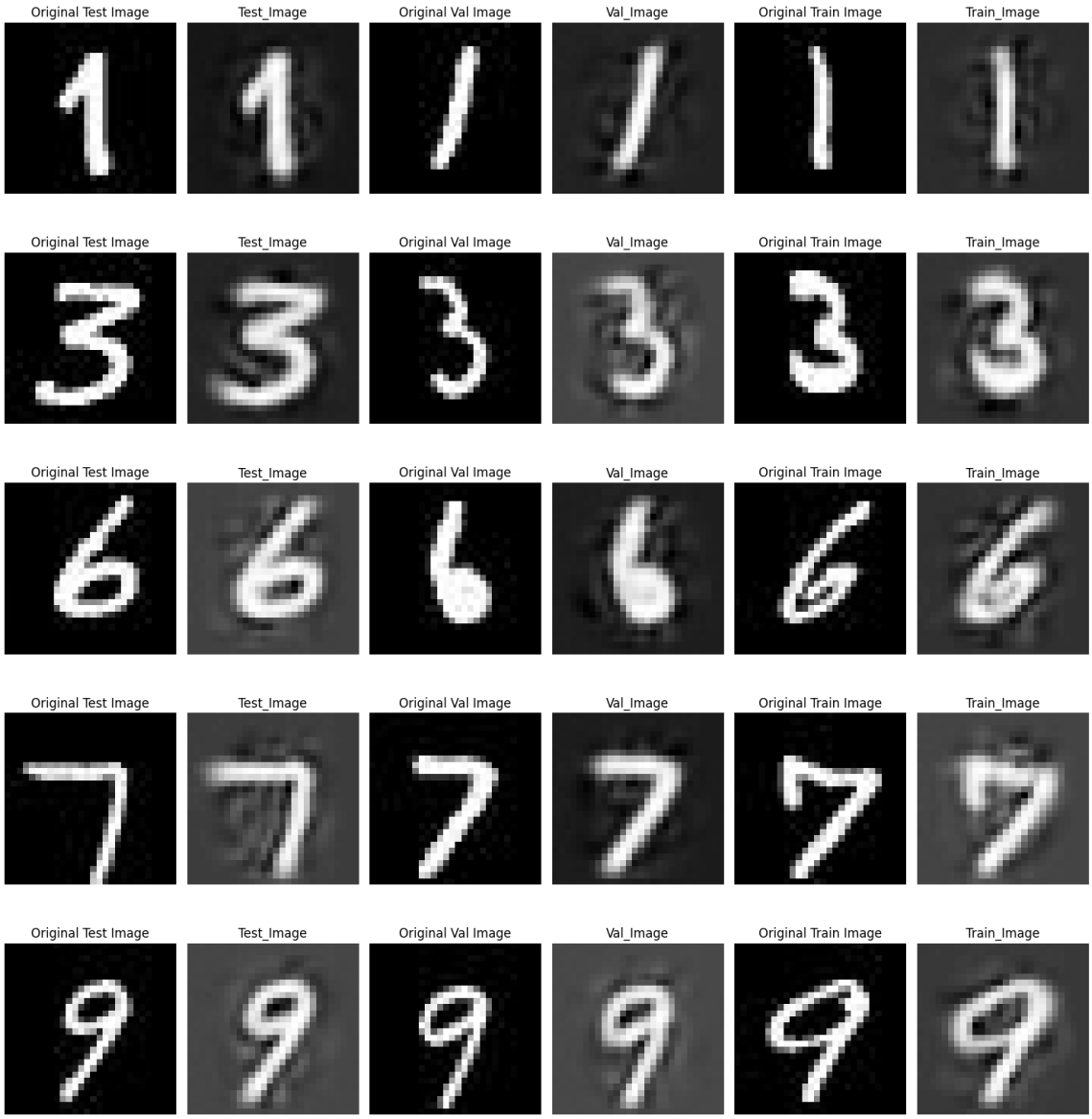


Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 32	
Dataset	Average reconstruction error
Train	0.015676983
Test	0.016127512
Validation	0.016024027

Table 3.1: Average reconstruction errors for training, validation and test data

2. **With 64 neurons in the bottleneck layer**

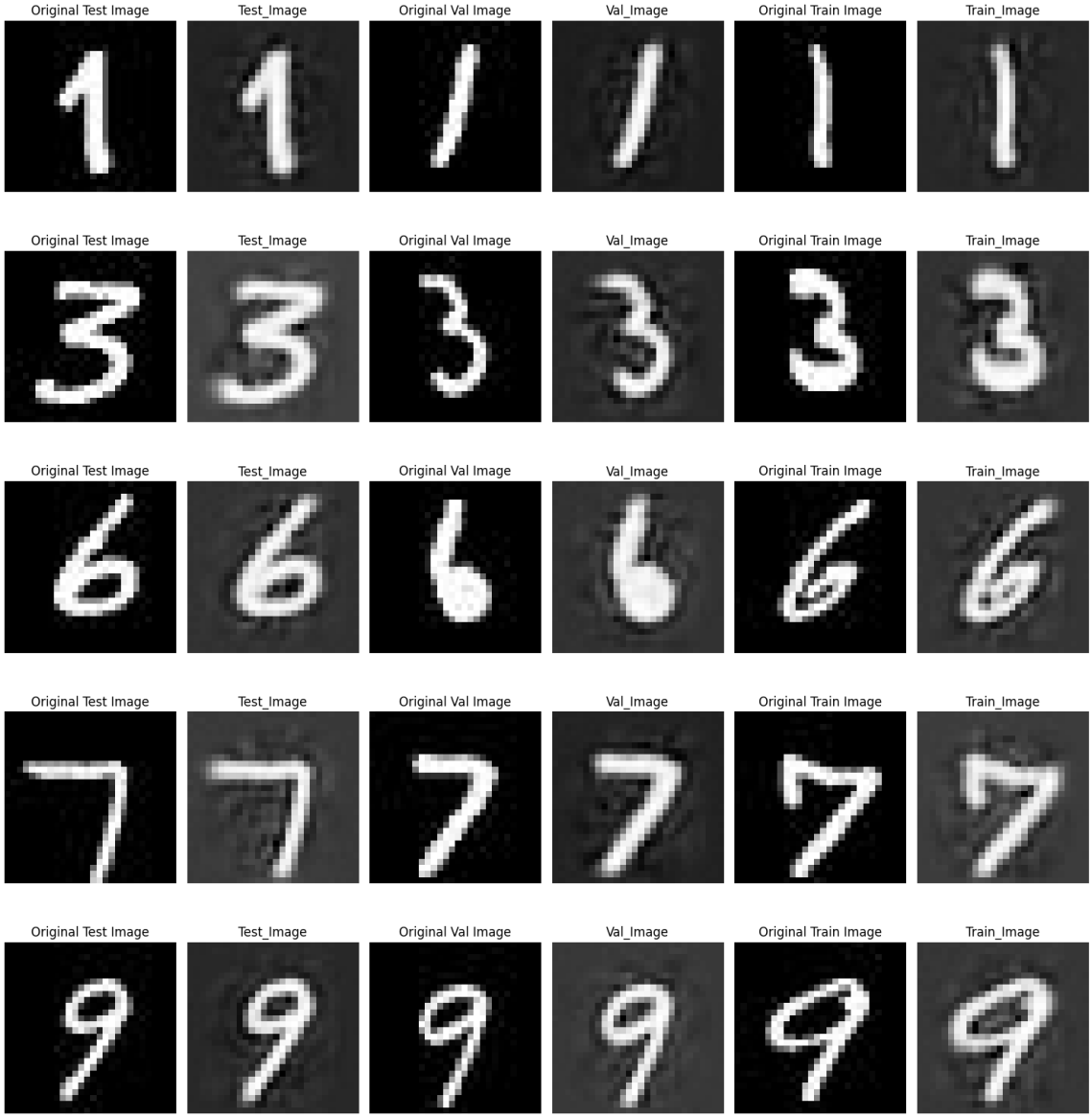


Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 64	
Dataset	Average reconstruction error
Train	0.009816022
Test	0.010185855
Validation	0.010159116

Table 3.2: Average reconstruction errors for training, validation and test data

3. **With 128 neurons in the bottleneck layer**

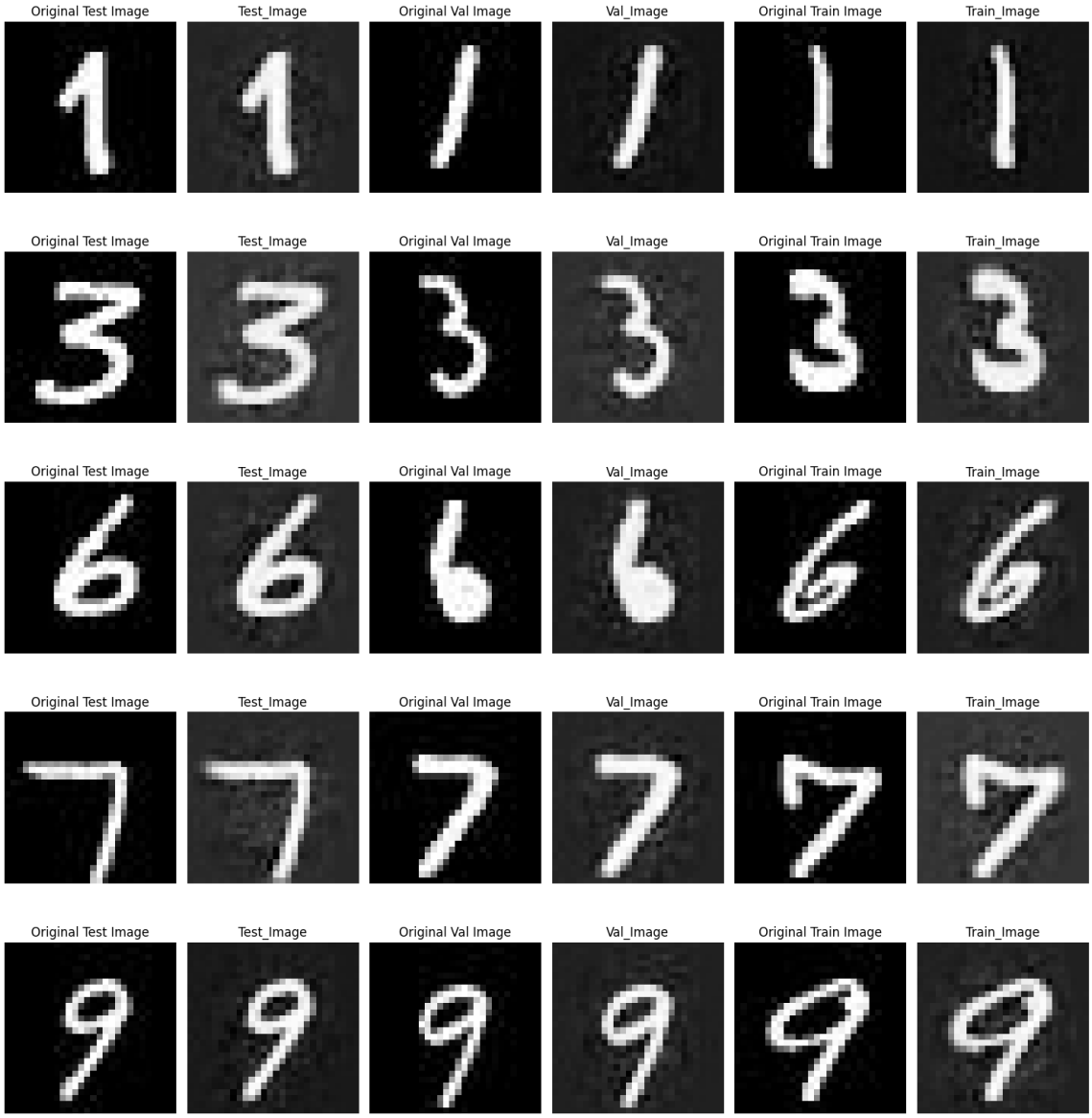


Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 128	
Dataset	Average reconstruction error
Train	0.005362541414797306
Test	0.005715625360608101
Validation	0.0056901593

Table 3.3: Average reconstruction errors for training, validation and test data

4. **With 256 neurons in the bottleneck layer**



Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 256	
Dataset	Average reconstruction error
Train	0.0029235926922410727
Test	0.00318974070250988
Validation	0.0031842138

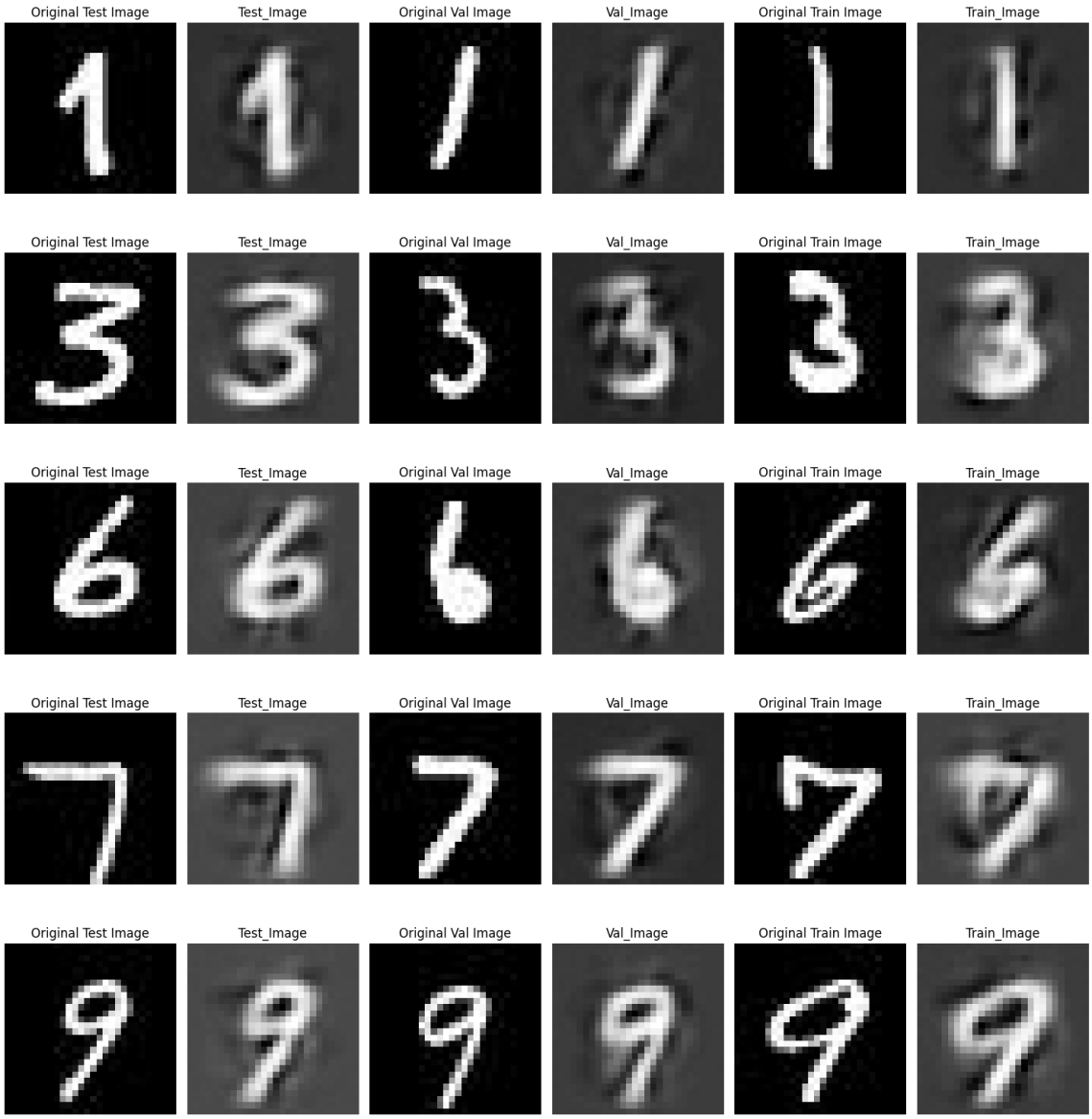
Table 3.4: Average reconstruction errors for training, validation and test data

3.2 Autoencoders with three hidden layer

This autoencoder has 400 neurons in first and third hidden layer.

- Considered 32, 64, 128 and 256 neurons for the bottleneck (middle) layer and considered 400 neurons in the first and third layers.
- Autoencoder is trained using Adam optimizer
- **tanh** used as activation function for the nodes in all the hidden layers for all autoencoders.
- Average reconstruction error is computed after the model is trained based on Mean squared error(MSE)
- Same image from each class is reconstructed for test dataset and then same is repeated for training and validation dataset across all architectures

1. **With 32 neurons in the bottleneck layer**

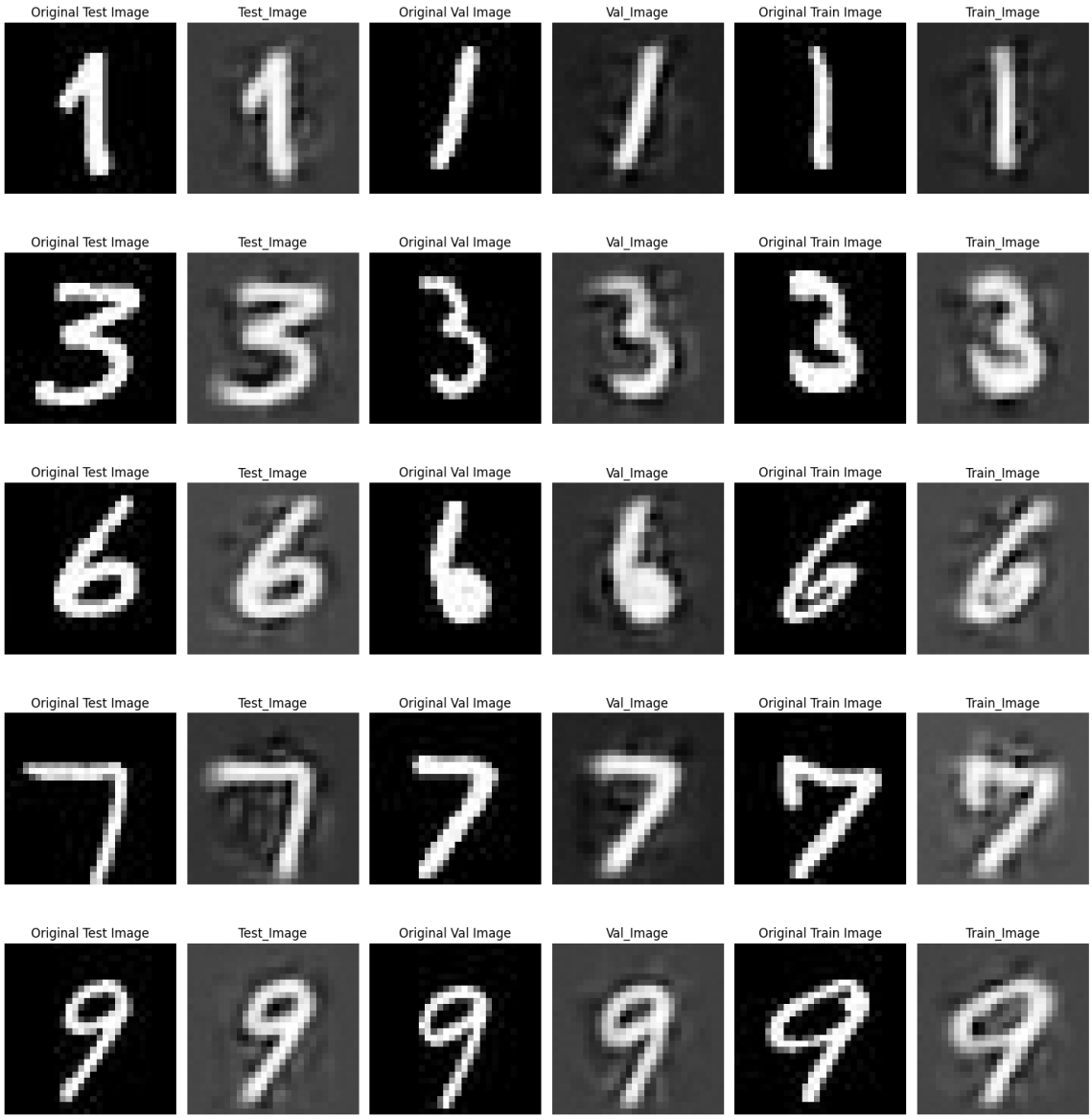


Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 32	
Dataset	Average reconstruction error
Train	0.015832422
Test	0.016306141
Validation	0.016185721

Table 3.5: Average reconstruction errors for training, validation and test data

2. **With 64 neurons in the bottleneck layer**

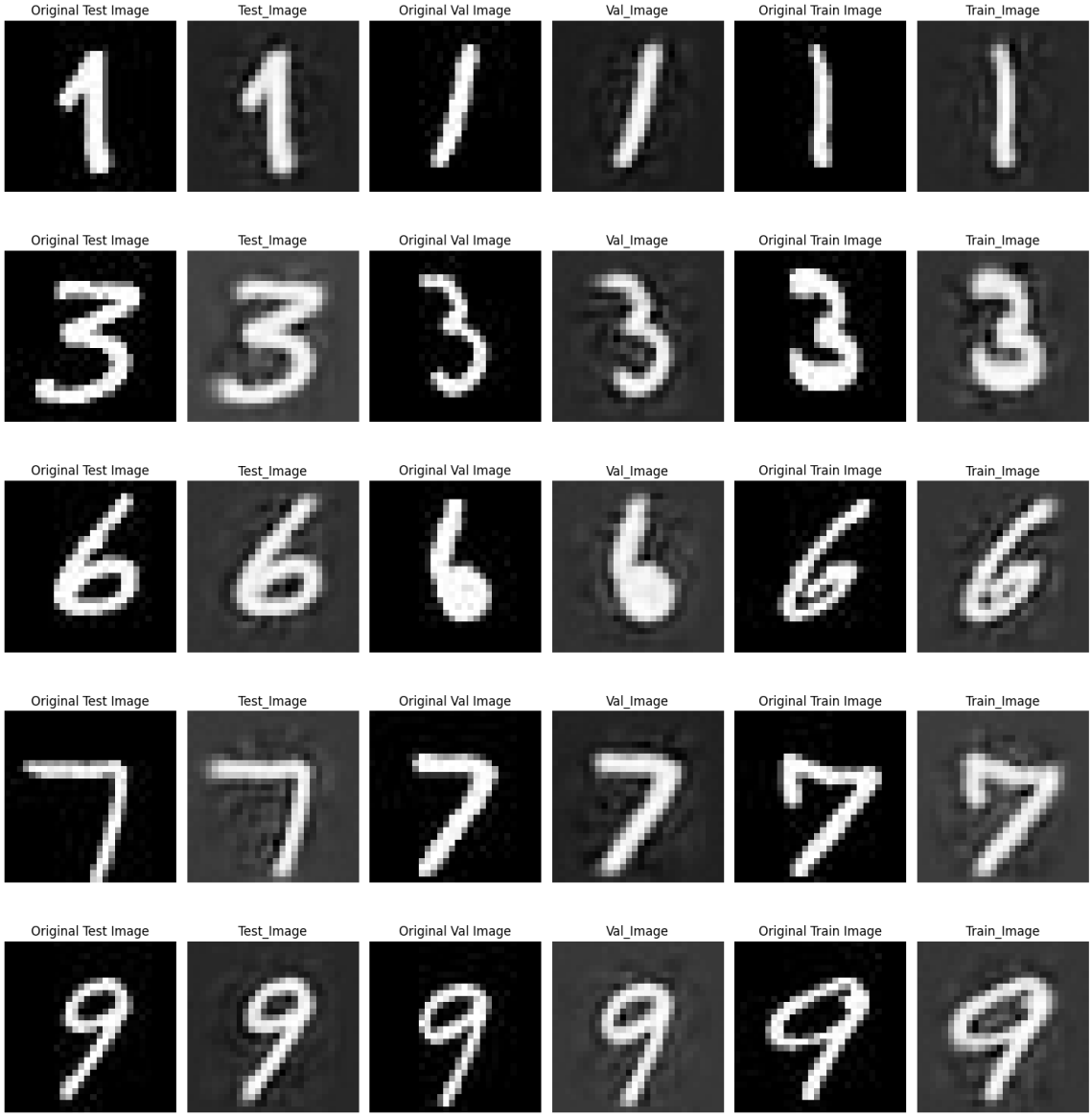


Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 64	
Dataset	Average reconstruction error
Train	0.010095606
Test	0.010464314
Validation	0.0104413815

Table 3.6: Average reconstruction errors for training, validation and test data

3. **With 128 neurons in the bottleneck layer**

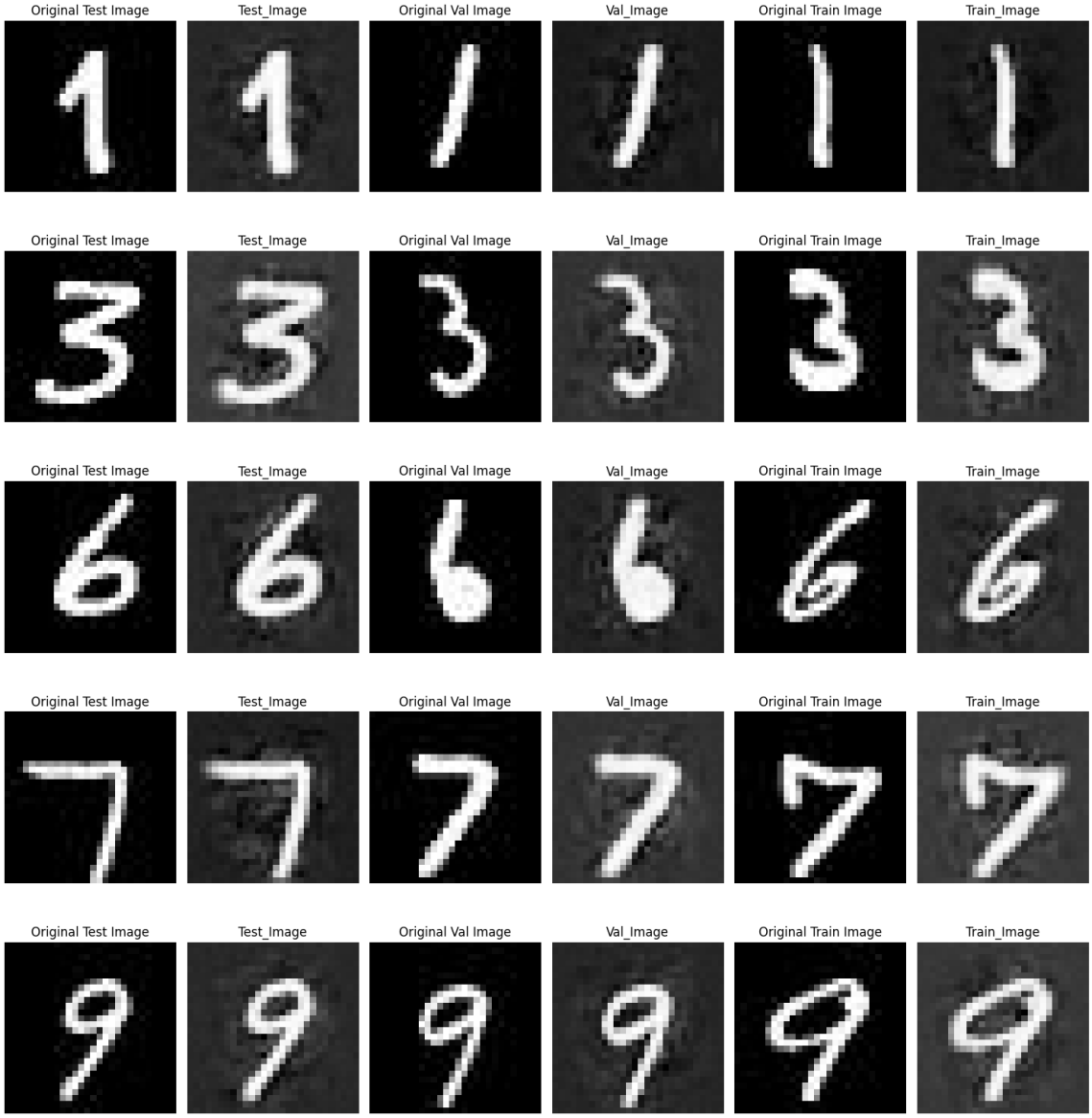


Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 128	
Dataset	Average reconstruction error
Train	0.0061078802682459354
Test	0.0064576915465295315
Validation	0.0064360267

Table 3.7: Average reconstruction errors for training, validation and test data

4. **With 256 neurons in the bottleneck layer**



Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 256	
Dataset	Average reconstruction error
Train	0.0037964447401463985
Test	0.004061944782733917
Validation	0.0040484727

Table 3.8: Average reconstruction errors for training, validation and test data

Chapter 4

Task 3: Classification using the compressed representation from the 1-hidden layer autoencoder

- Considered 32, 64, 128 and 256 neurons for the bottleneck (middle) layer.
- Autoencoder is trained using Adam optimizer
- **tanh** used as activation function for the nodes in all the hidden layers
- Saved the output of the middle layer (compressed layer). Use the reduced dimension representation as input to the FCNN.
- The following three architectures have been used throughout the assignment:
  - Three hidden layer (64, 32 & 64 neurons respectively in each hidden layer)
  - Three hidden layer (512, 128 & 64 neurons respectively in each hidden layer)
  - Five hidden layer (512, 512 256, 256 & 256 neurons respectively in each hidden layer)

4.1 Results for reduced dimension representation as input to FCNN: 32

Compressed representation from the 1-hidden layer autoencoder - 32	
Architecture	Validation Accuracy
64-32-64	0.9810
512-128-64	0.9852
512-512-256-256-256	0.9733

Table 4.1: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9781**
- Confusion matrix for best architecture:

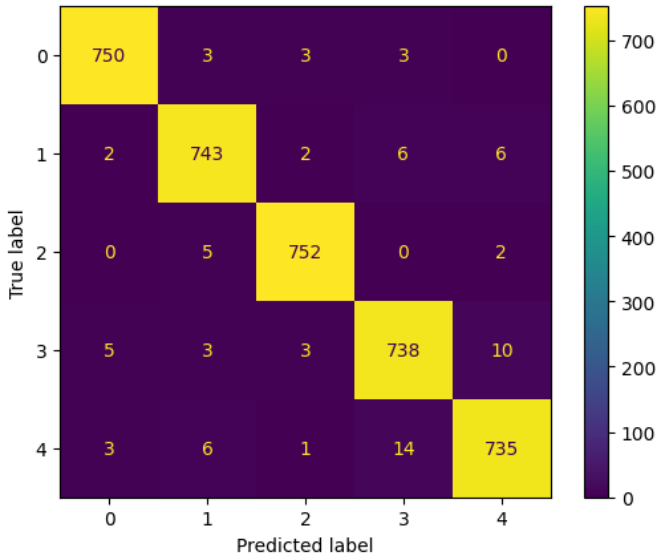


Figure 4.1: Confusion Matrix for best architecture

4.2 Results for Reduced dimension: 64

Compressed representation from the 1-hidden layer autoencoder - 64	
Architecture	Validation Accuracy
64-32-64	0.9839
512-128-64	0.9823
512-512-256-256-256	0.9791

Table 4.2: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **64-32-64**
- Test accuracy for best architecture: **0.9807**
- Confusion matrix for best architecture:

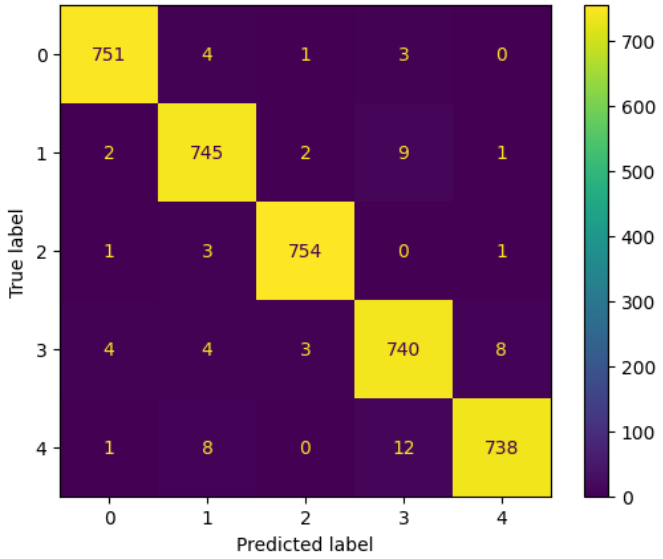


Figure 4.2: Confusion Matrix for best architecture

4.3 Results for Reduced dimension: 128

Compressed representation from the 1-hidden layer autoencoder - 128	
Architecture	Validation Accuracy
64-32-64	0.9820
512-128-64	0.9847
512-512-256-256-256	0.9786

Table 4.3: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9833**
- Confusion matrix for best architecture:

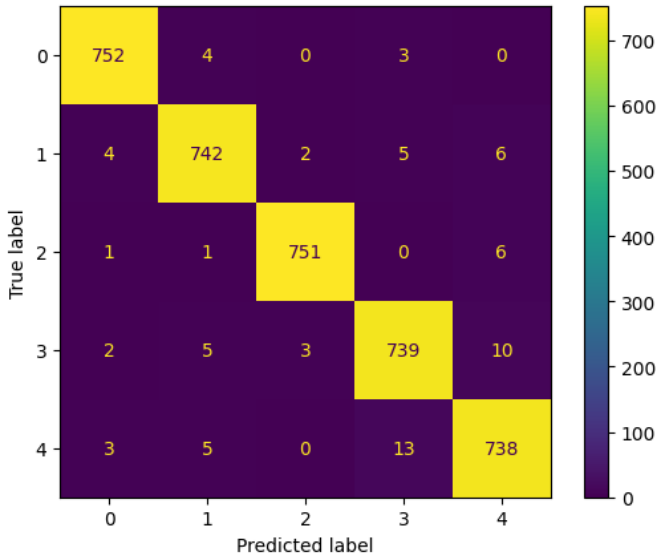


Figure 4.3: Confusion Matrix for best architecture

4.4 Results for Reduced dimension: 256

Compressed representation from the 1-hidden layer autoencoder - 256	
Architecture	Validation Accuracy
64-32-64	0.9815
512-128-64	0.9802
512-512-256-256-256	0.9757

Table 4.4: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **64-32-64**
- Test accuracy for best architecture: **0.9781**
- Confusion matrix for best architecture:

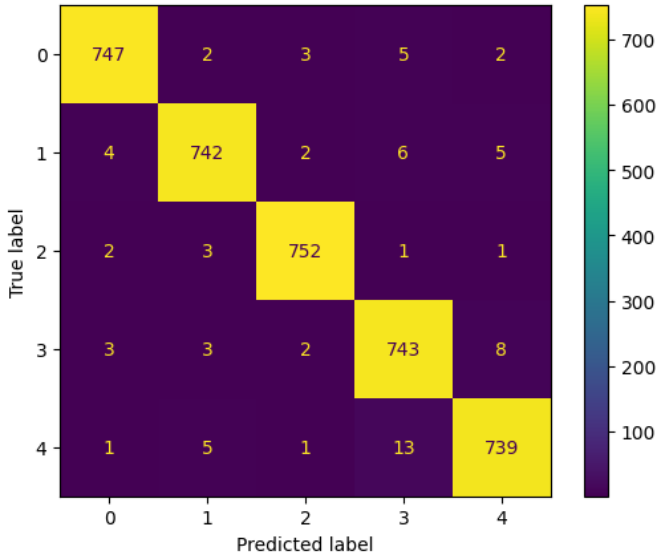


Figure 4.4: Confusion Matrix for best architecture

4.5 Best reduced dimension representation(from the all the test accuracies

Compressed representation - 32	Architecture	Test Accuracy
	64-32-64	0.9781
	512-128-64	0.9833
	512-512-256-256-256	0.9728
Compressed representation - 64	Architecture	Test Accuracy
	64-32-64	0.9807
	512-128-64	0.9776
	512-512-256-256-256	0.9791
Compressed representation- 128	Architecture	Test Accuracy
	64-32-64	0.9815
	512-128-64	0.9807
	512-512-256-256-256	0.9765
Compressed representation - 256	Architecture	Test Accuracy
	64-32-64	0.9781
	512-128-64	0.9786
	512-512-256-256-256	0.9741

Table 4.5: Test accuracies for all architectures for all reduced dimension representations

From Table 4.5 above the best reduced dimension is from 32 reduced dimension(Architecture: 512-128-64) as it's test accuracy is the highest : 0.9833.

4.6 Comparison with Assignment 3 and Task 1

- The best architecture for Assignment 3 was with Five hidden layers (512, 512 256, 256 & 256 neurons respectively in each hidden layer) with Adam optimizer. This gave **Test accuracy: 0.9701** and **Validation accuracy: 0.9749**
- The best architecture for Task 1 is with Reduced dimension **64** (with Three hidden layer 512, 128 & 64 neurons respectively in each hidden layer with Adam optimizer). And this gave **Test accuracy: 0.9874** and **Validation accuracy: 0.9807**

- The best architecture for Task 3 is with Reduced dimension **32** (with Three hidden layer 512, 128 & 64 neurons respectively in each hidden layer with Adam optimizer). And this gave **Test accuracy: 0.9833** and **Validation accuracy: 0.9852**
- Comparing confusion matrix for the best results of Assignment 3, Task 1 and Task 3

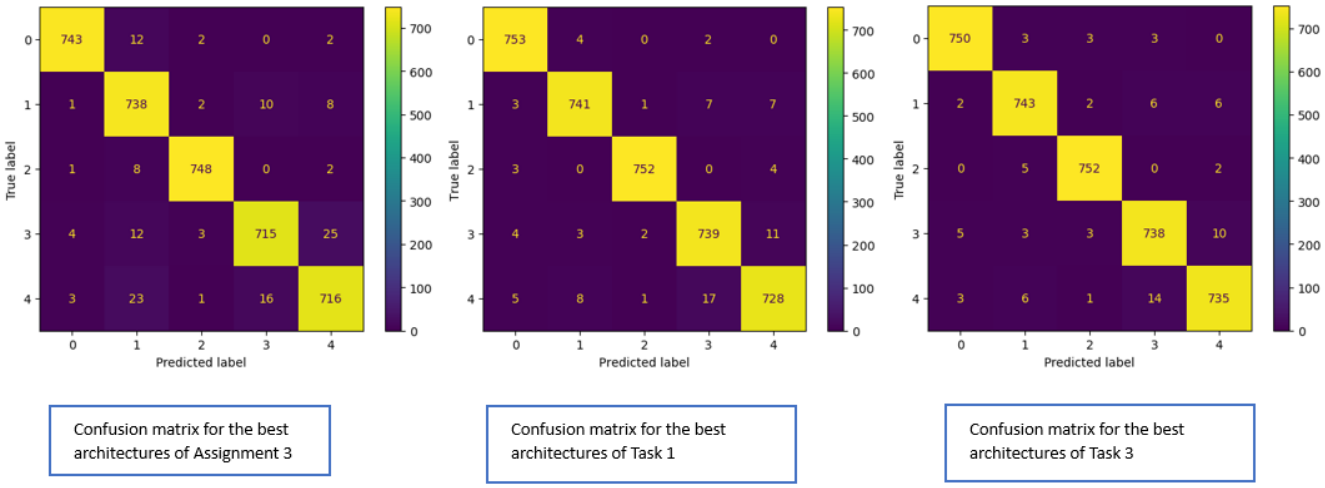


Figure 4.5: Confusion matrix for the best results of Assignment 3, Task 1 and Task 3

Chapter 5

Task 4: Classification using the compressed representation from the 3-hidden layer autoencoder

- Considered 32, 64, 128 and 256 neurons for the bottleneck (middle) layer.
- Autoencoder is trained using Adam optimizer
- **tanh** used as activation function for the nodes in all the hidden layers
- Saved the output of the middle layer (compressed layer). Use the reduced dimension representation as input to the FCNN.
- The following three architectures have been used throughout the assignment:
  - Three hidden layer (64 , 32 & 64 neurons respectively in each hidden layer)
  - Three hidden layer (512, 128 & 64 neurons respectively in each hidden layer)
  - Five hidden layer (512, 512 256, 256 & 256 neurons respectively in each hidden layer)
- There are 400 neurons in first and third hidden layers with middle layer being the bottleneck layer.

5.1 Results for reduced dimension representation as input to FCNN: 32

Compressed representation from the 3-hidden layer autoencoder - 32	
Architecture	Validation Accuracy
64-32-64	0.9805
512-128-64	0.9828
512-512-256-256-256	0.9786

Table 5.1: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9802**
- Confusion matrix for best architecture:

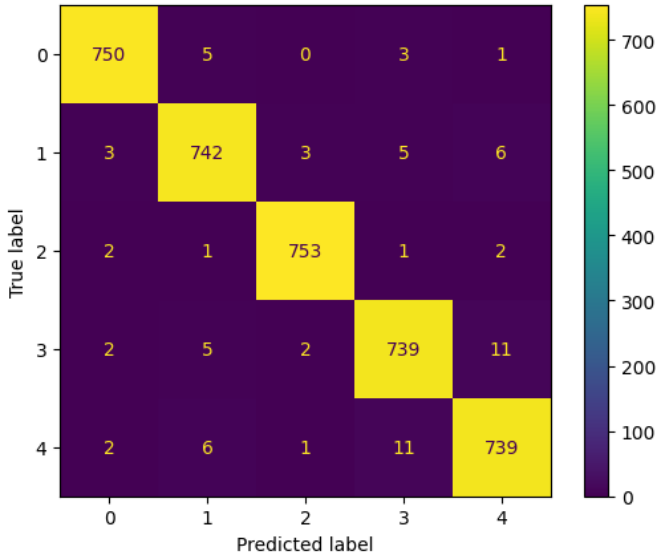


Figure 5.1: Confusion Matrix for best architecture

5.2 Results for Reduced dimension: 64

Compressed representation from the 3-hidden layer autoencoder - 64	
Architecture	Validation Accuracy
64-32-64	0.9833
512-128-64	0.9841
512-512-256-256-256	0.9781

Table 5.2: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9828**
- Confusion matrix for best architecture:

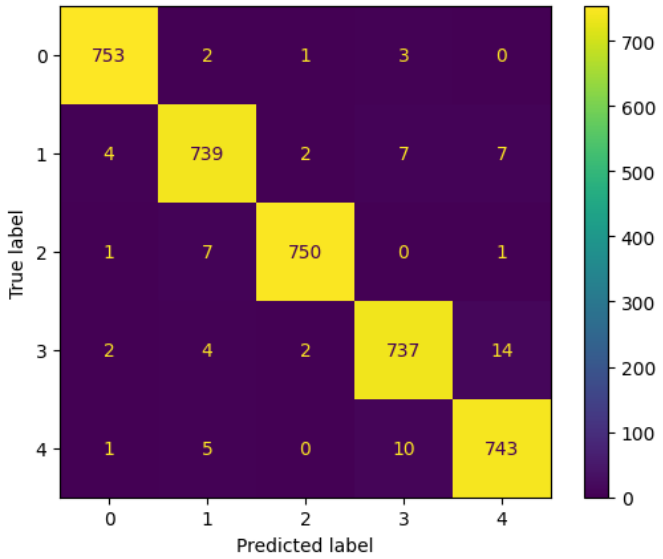


Figure 5.2: Confusion Matrix for best architecture

5.3 Results for Reduced dimension: 128

Compressed representation from the 3-hidden layer autoencoder - 128	
Architecture	Validation Accuracy
64-32-64	0.9776
512-128-64	0.9783
512-512-256-256-256	0.9723

Table 5.3: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9807**
- Confusion matrix for best architecture:

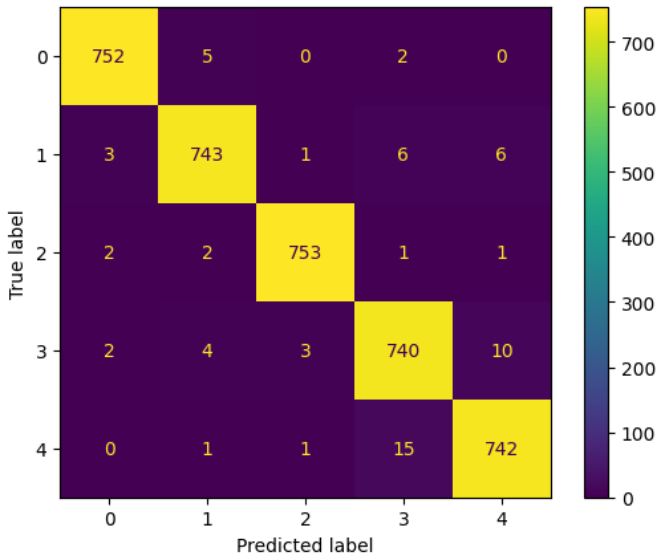


Figure 5.3: Confusion Matrix for best architecture

5.4 Results for Reduced dimension: 256

Compressed representation from the 3-hidden layer autoencoder - 256	
Architecture	Validation Accuracy
64-32-64	0.9765
512-128-64	0.9781
512-512-256-256-256	0.9768

Table 5.4: Validation Accuracy for all architectures

- Best architecture based on validation accuracy: **512-128-64**
- Test accuracy for best architecture: **0.9786**
- Confusion matrix for best architecture:

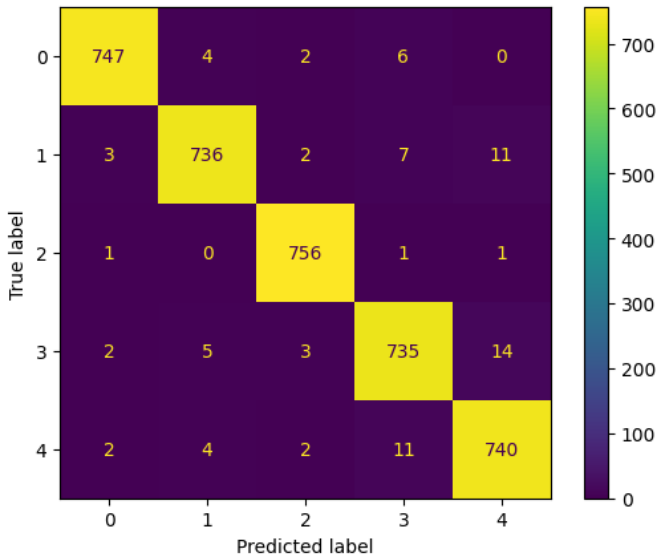


Figure 5.4: Confusion Matrix for best architecture

5.5 Best reduced dimension representation(from all the test accuracies)

Compressed representation - 32	Architecture	Test Accuracy
	64-32-64	0.9791
	512-128-64	0.9802
	512-512-256-256-256	0.9765
Compressed representation - 64	Architecture	Test Accuracy
	64-32-64	0.9799
	512-128-64	0.9828
	512-512-256-256-256	0.9725
Compressed representation- 128	Architecture	Test Accuracy
	64-32-64	0.9797
	512-128-64	0.9807
	512-512-256-256-256	0.9699
Compressed representation - 256	Architecture	Test Accuracy
	64-32-64	0.9781
	512-128-64	0.9786
	512-512-256-256-256	0.9741

Table 5.5: test accuracies for all architectures for all reduced dimension representations

From Table 5.5 above the best reduced dimension is from 64 reduced dimension(Architecture: 512-128-64) as it's test accuracy is the highest : 0.9828. The confusion matrix for the same can be seen in Fig 5.3

5.6 Comparison with Assignment 3, Task 1 and Task 3

- The best architecture for Assignment 3 was with Five hidden layers (512, 512 256, 256 & 256 neurons respectively in each hidden layer) with Adam optimizer. This gave **Test accuracy: 0.9701** and **Validation accuracy: 0.9749**
- The best architecture for Task 1 is with Reduced dimension **64** (with Three hidden layer 512, 128 & 64 neurons respectively in each hidden

layer with Adam optimizer). And this gave **Test accuracy: 0.9874** and **Validation accuracy: 0.9807**

- The best architecture for Task 3 is with Reduced dimension **32** (with Three hidden layer 512, 128 & 64 neurons respectively in each hidden layer with Adam optimizer). And this gave **Test accuracy: 0.9833** and **Validation accuracy: 0.9852**
- The best architecture for Task 4 is with Reduced dimension **64** (with Three hidden layer 512, 128 & 64 neurons respectively in each hidden layer with Adam optimizer). And this gave **Test accuracy: 0.9828** and **Validation accuracy: 0.9841**
- Comparing confusion matrix for the best results of Assignment 3, Task 1, Task 3 and Task 4

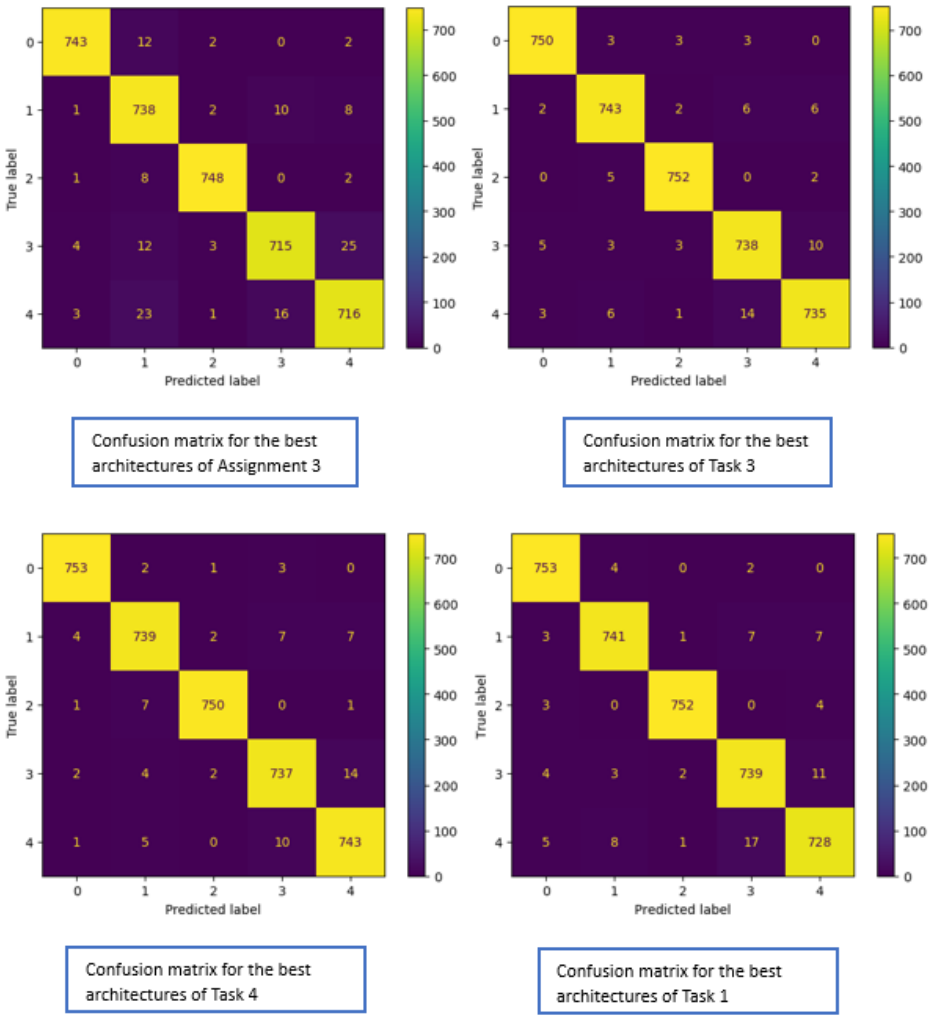


Figure 5.5: Confusion matrix for the best results of Assignment 3, Task 1, Task 3 and Task 4

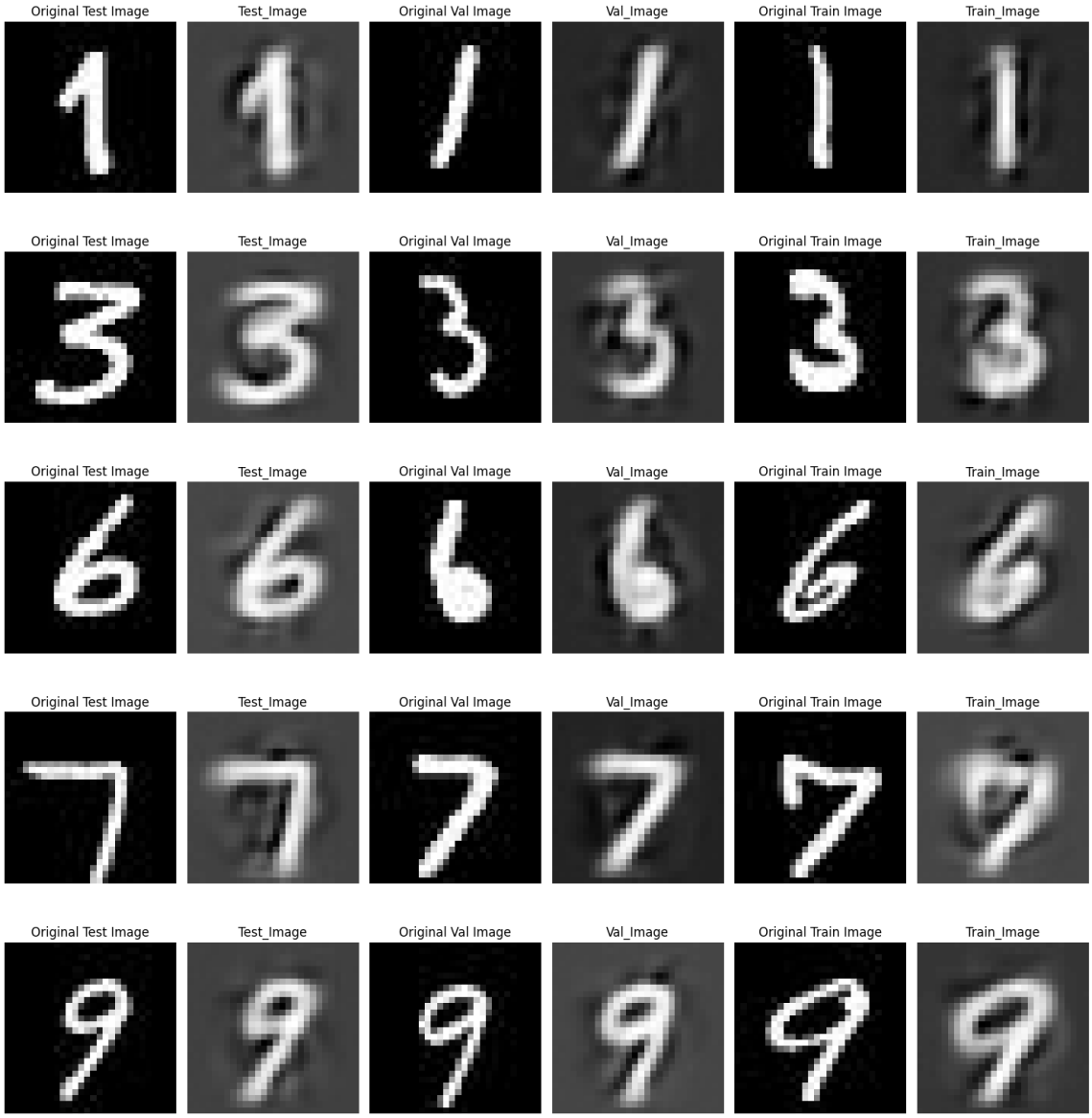
Chapter 6

Task 5: Denoising autoencoders for reconstructing the images

- Considered the number of neurons for the bottleneck (middle) layer = 32 based on the best test accuracy of the best reduced dimensional representation from 1 hidden layer autoencoder.
- Denoising Autoencoder with 20% and 40% noise is trained using Adam optimizer
- **tanh** used as activation function for the nodes in all the hidden layers
- Average reconstruction error is computed after the model is trained based on Mean squared error(MSE)
- Same image from each class is reconstructed for test dataset and then same is repeated for training and validation dataset across all architectures
- Considered the best architecture obtained in Task-3 i.e., three hidden layer (512, 128 & 64 neurons respectively in each hidden layer)

6.1 Denoising autoencoders with one hidden layer with 20% noise

With 32 neurons in the bottleneck layer



Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 32 | noise = 0.20	
Dataset	Average reconstruction error
Train	0.01613149
Test	0.016476374
Validation	0.016398426

Table 6.1: Average reconstruction errors for training, validation and test data

Reduced dimension representation as input to a fully connected neural network (FCNN)

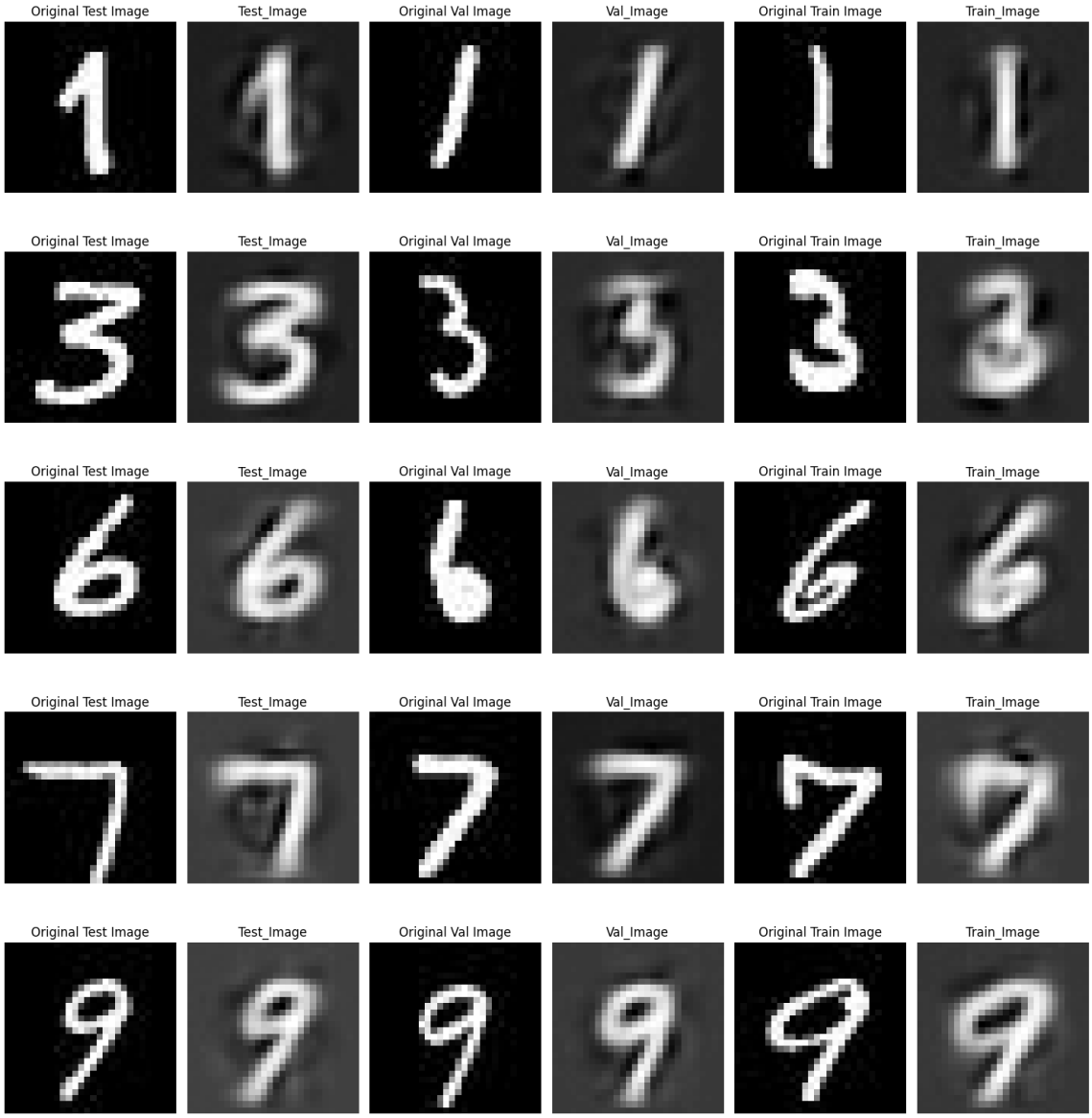
Best Architecture(Task 3): 512-128-64 | Denoising Autoencoder = 0.20	
Dataset	Accuracy
Test	0.9781
Validation	0.9802

Table 6.2: Accuracy for best architecture FCNN from Task 3

- Best architecture based on Task 3: **512-128-64**

6.2 Denoising autoencoders with one hidden layer with 40% noise

With 32 neurons in the bottleneck layer



Reconstruction errors for training, validation and test data:

bottleneck (middle) layer - 32 | noise = 0.40	
Dataset	Average reconstruction error
Train	0.017458517
Test	0.01779007
Validation	0.017729674

Table 6.3: Average reconstruction errors for training, validation and test data

Reduced dimension representation as input to a fully connected neural network (FCNN)

Best Architecture(Task 3): 512-128-64 | Denoising Autoencoder = 0.40	
Dataset	Accuracy
Test	0.9823
Validation	0.9828

Table 6.4: Accuracy for best architecture FCNN from Task 3

- Best architecture based on Task 3: **512-128-64**

Chapter 7

Task 6: Weight visualization

We want to plot those x_n as images which maximally activate each of the neurons of the hidden representations learned by an autoencoder.

Deviation for the objective function:

$$maximize : {w_j}^T x_n \tag{7.1}$$

with constraint

$$||x_n||^2 = {x_n}^T x_n = 1 \tag{7.2}$$

Solution:

Applying Lagrangian Multiliplier

$$L(\alpha, x_n) = {\mathbf{w_j}}^T \mathbf{x_n} + \alpha(1 - ||\mathbf{x_n}||^2) \tag{7.3}$$

$$\frac{\partial L}{\partial \mathbf{x_n}} = \mathbf{w_j} - 2\alpha \mathbf{x_n} = 0 \tag{7.4}$$

$$\therefore \mathbf{x_n} = \frac{\mathbf{w_j}}{2\alpha} \tag{7.5}$$

On substituting 7.5 in 7.1:

$$||\frac{\mathbf{w_j}}{2\alpha}|| = 1 \tag{7.6}$$

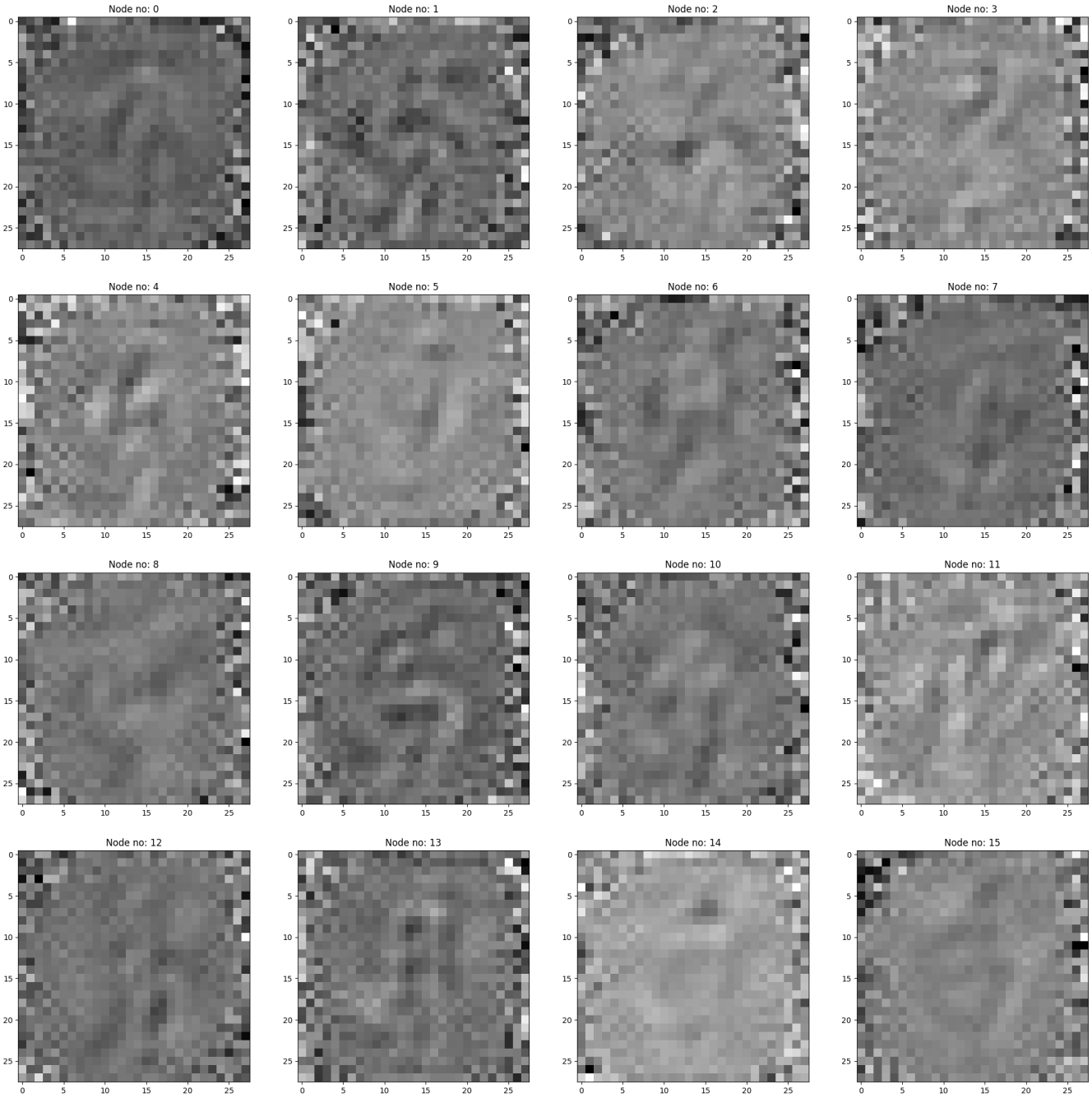
$$\therefore \alpha = \frac{||\mathbf{w_j}||}{2} \tag{7.7}$$

On substituting 7.7 in 7.5:

$$\mathbf{x_n} = \frac{\mathbf{w_j} * 2}{2 * ||\mathbf{w_j}||} \tag{7.8}$$

$$\therefore \mathbf{x_n} = \frac{\mathbf{w_j}}{||\mathbf{w_j}||} \tag{7.9}$$

7.1 Best compressed representation in one hidden layer autoencoder



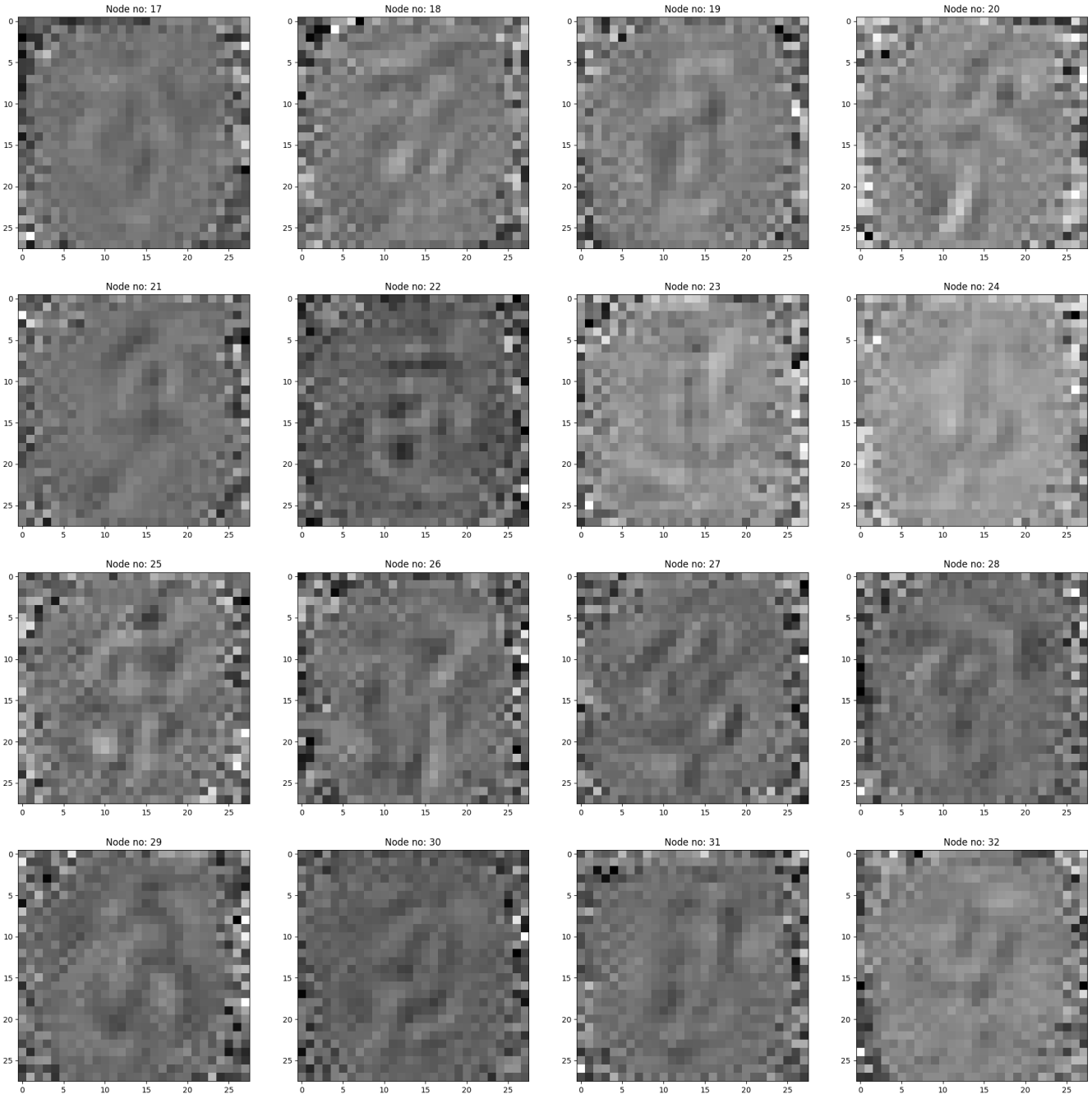
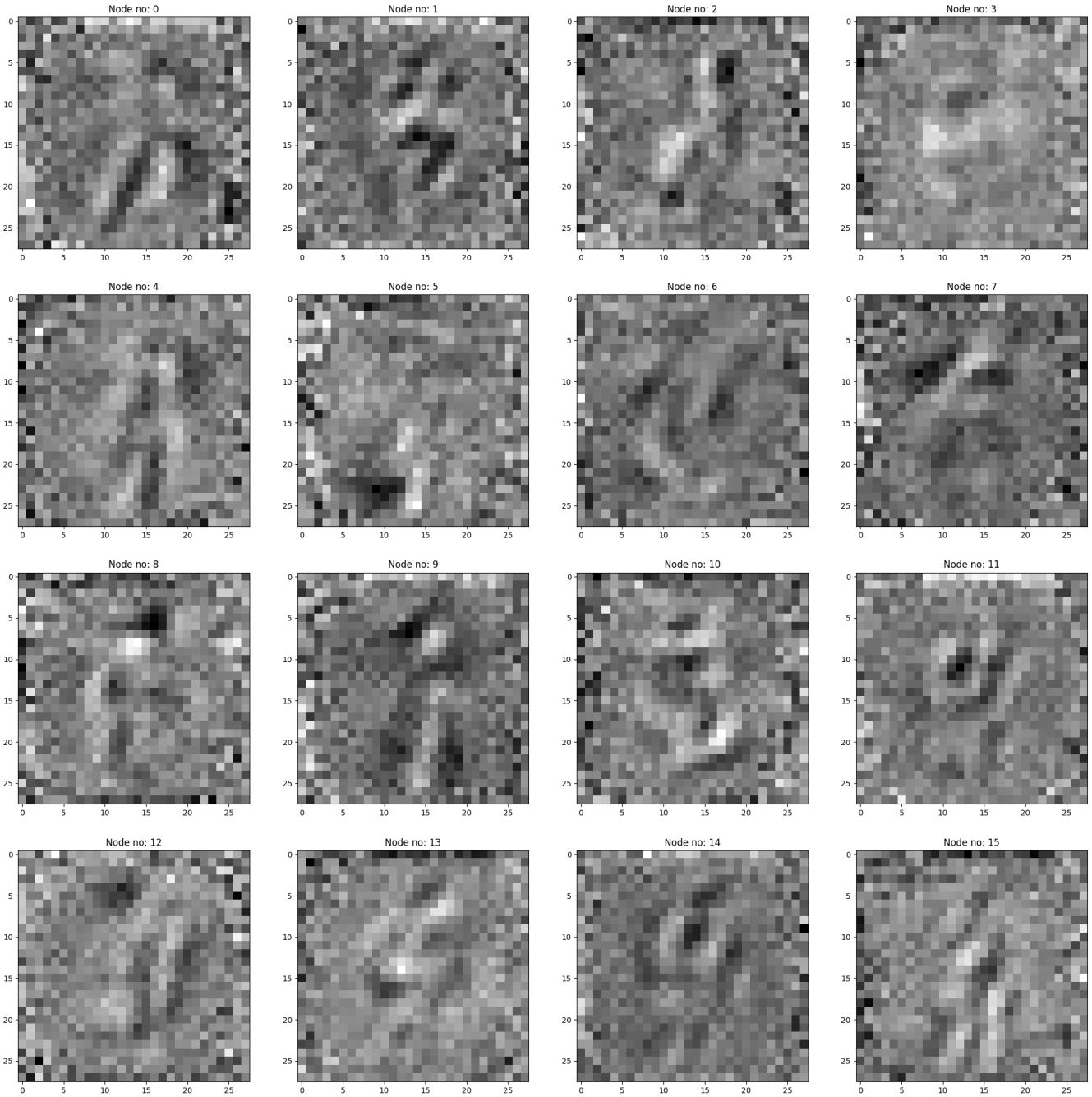


Figure 7.1: Plot of weights from the input layer to the compressed layer in one hidden layer autoencoder

7.2 Denoising Autoencoders with 20% noise

We want to plot the inputs as images that maximally activate each of the neurons of the hidden representations obtained using the denoising autoencoders with 20 % noise Denoising autoencoder - 20 % Noise



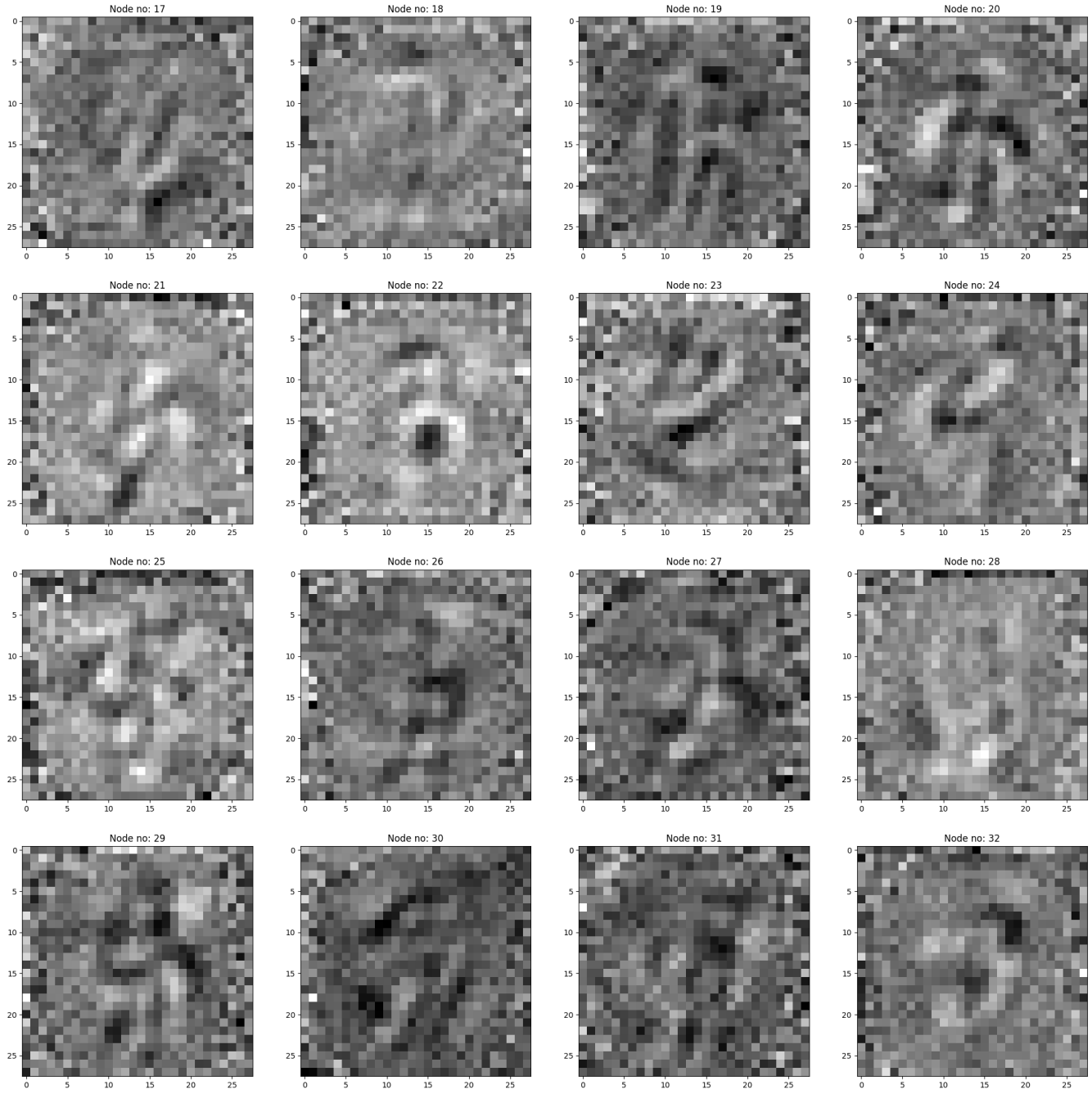
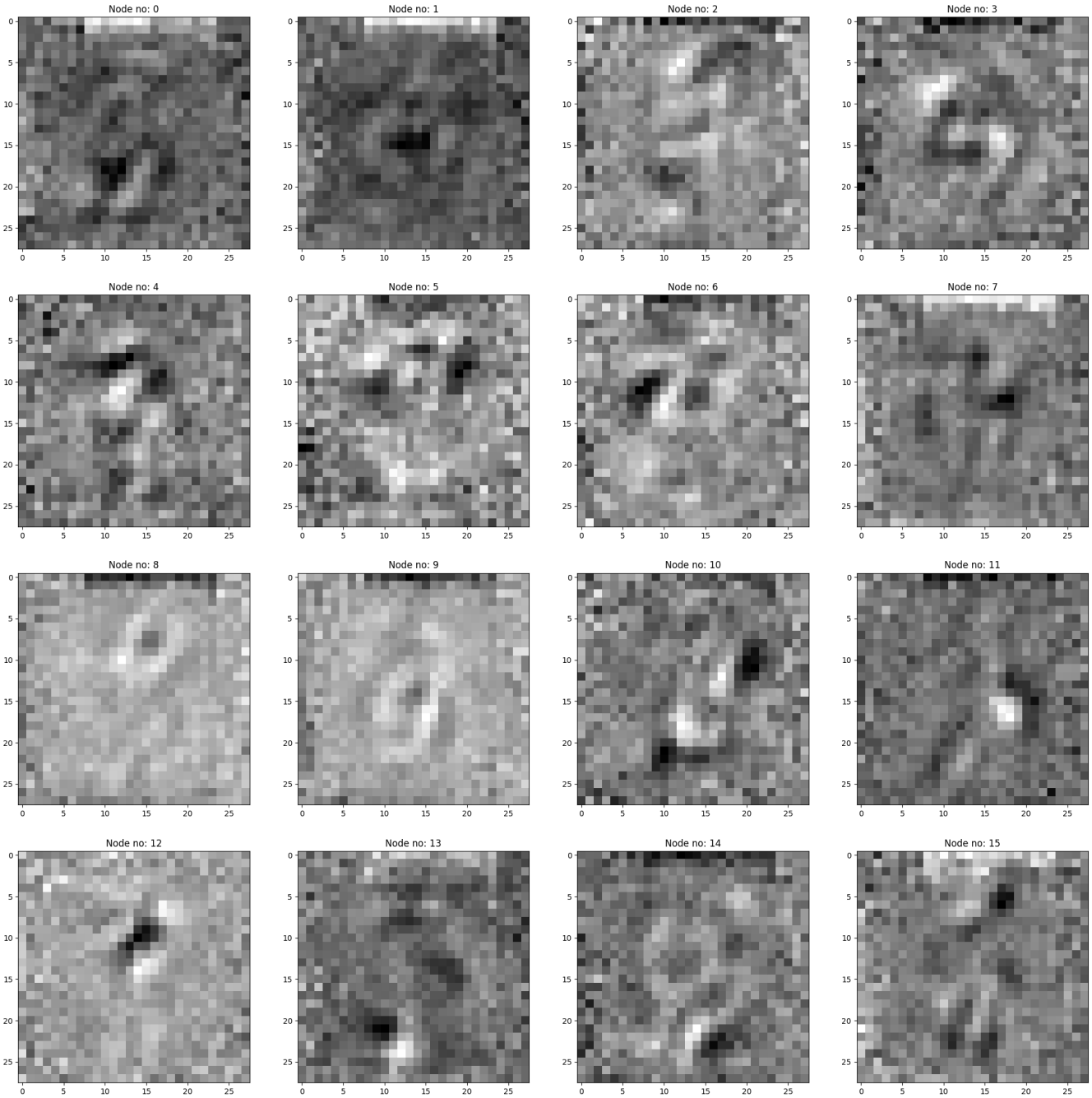


Figure 7.2: Plot of weights from the input layer to the compressed layer in denoising autoencoders with one hidden layer with 20 % noise

7.3 Denoising Autoencoders with 40% noise

We want to plot the inputs as images that maximally activate each of the neurons of the hidden representations obtained using the denoising autoencoders with 40 % noise Denoising autoencoder - 40 % Noise



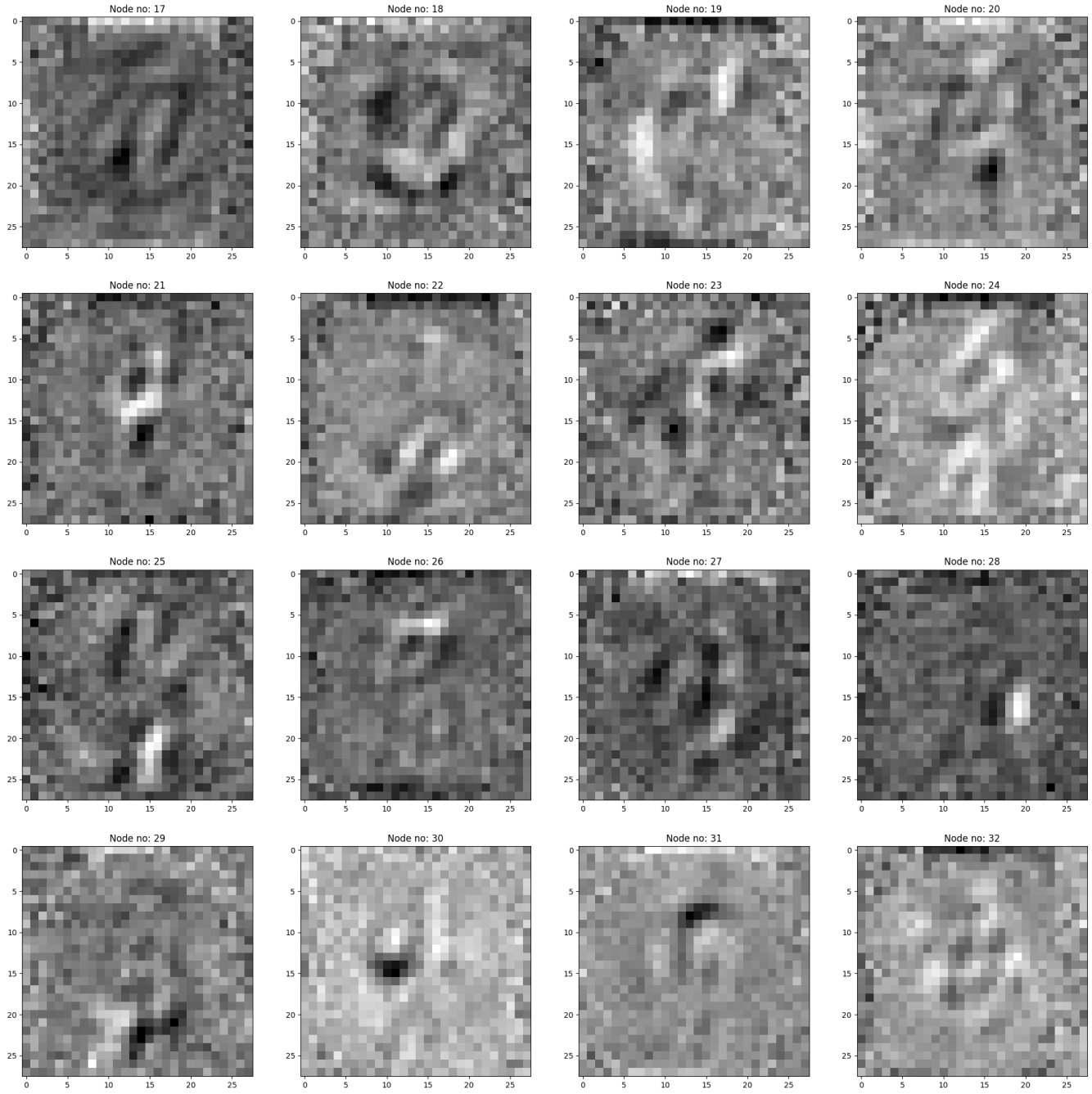


Figure 7.3: Plot of weights from the input layer to the compressed layer in denoising autoencoders with one hidden layer with 40 % noise

We can observe from Fig 7.1,7.2 and 7.3 that the hidden neurons seem to act like pen-stroke detectors.

When the model is trained without noise it is possible that it starts overfitting itself to the pattern only available in the training dataset. Whereas, while modeling using denoising autoencoders, in each epoch with some 'q' probability noise is added to x_n, this helps the model to avoid overfitting by not letting the noise to be memorised as with each epoch noise is added to

different samples.

References

[1] Lecture notes of CS671 Feb 2023-24 by Dr.Dileep A D

[2] https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f