—---

CS671 : Deep Learning and Applications
Assignment 04 - Report

—---

Autoencoders

Ankit Mehra (S22020)
Urvashi Goswami(S22024)
Ashish Rana (S22019)

1. Introduction

Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation. The code is a compact "summary" or "compression" of the input, also called the *latent-space representation.*

An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.

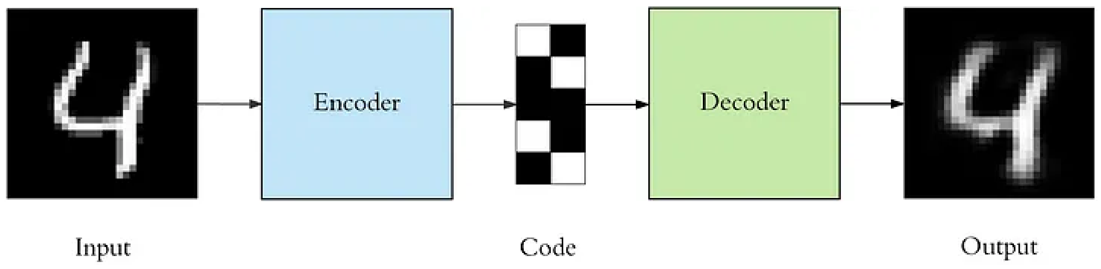


Fig 1. Source: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

To build an autoencoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target. Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different from a standard data compression algorithm like gzip. So we can't expect an autoencoder trained on handwritten digits to compress landscape photos.
- Lossy: The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation.
- Unsupervised: To train an autoencoder we don't need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an *unsupervised* learning technique since they don't need explicit labels to train on. But to be more precise they are *self-supervised* because they generate their own labels from the training data.

2. Architecture

Autoencoders consist of 3 parts:

1. Encoder: A module that compresses the train-validate-test set input data into an encoded representation that is typically several orders of magnitude smaller than the input data.

2. Bottleneck: A module that contains the compressed knowledge representations and is therefore the most important part of the network.

3. Decoder: A module that helps the network“decompress” the knowledge representations and reconstructs the data back from its encoded form. The output is then compared with a ground truth.

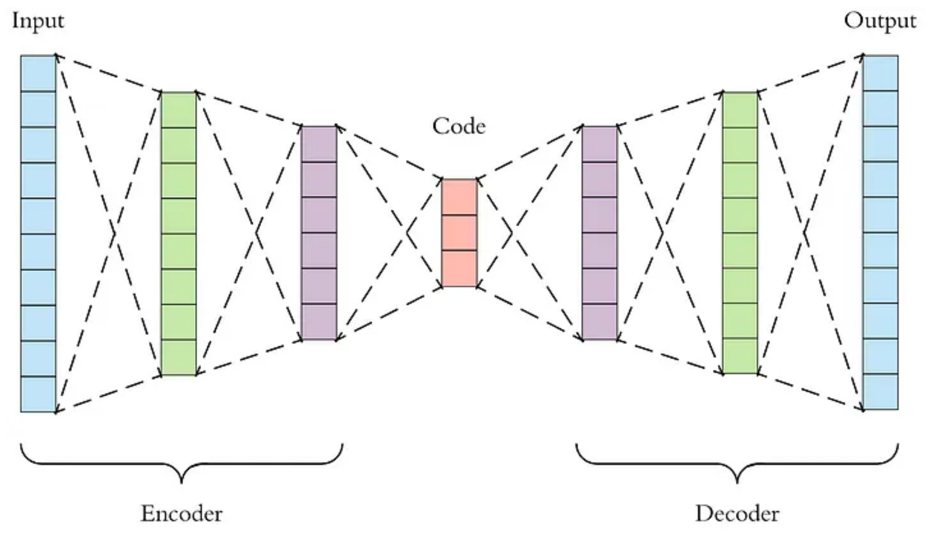


Fig 2. Source : https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

First the input passes through the encoder, which is a fully-connected ANN, to produce the code. The decoder, which has the similar ANN structure, then produces the output only using the code. The goal is to get an output identical with the input. Note that the decoder architecture is the mirror image of the encoder. This is not a requirement but it’s typically the case. The only requirement is the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

How to train autoencoders?

There are 4 hyperparameters that we need to set before training an autoencoder:

- Code size: number of nodes in the middle layer. Smaller size results in more compression.
- Number of layers: the autoencoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.
- Number of nodes per layer: the autoencoder architecture we're working on is called a *stacked autoencoder* since the layers are stacked one after another. Usually stacked autoencoders look like a "sandwich". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.
- Loss function: we either use *mean squared error (mse)* or *binary cross entropy*. If the input values are in the range [0, 1] then we typically use cross entropy, otherwise we use the mean squared error.

Autoencoders are trained the same way as ANNs via backpropagation.

The relationship between the Encoder, Bottleneck, and Decoder

Encoder

The encoder is a set of convolutional blocks followed by pooling modules that compress the input to the model into a compact section called the bottleneck.

The bottleneck is followed by the decoder that consists of a series of upsampling modules to bring the compressed feature back into the form of an image. In case of simple autoencoders, the output is expected to be the same as the input with reduced noise.

Bottleneck

The most important part of the neural network, and ironically the smallest one, is the bottleneck. The bottleneck exists to restrict the flow of information to the decoder from the encoder, thus,allowing only the most vital information to pass through.

Since the bottleneck is designed in such a way that the maximum information possessed by an image is captured in it, we can say that the bottleneck helps us form a *knowledge-representation* of the input.

Thus, the encoder-decoder structure helps us extract the most from an image in the form of data and establish useful correlations between various inputs within the network. A bottleneck as a compressed representation of the input further prevents the neural network from memorizing the input and overfitting on the data.

As a rule of thumb, remember this: The smaller the bottleneck, the lower the risk of overfitting. However,Very small bottlenecks would restrict the amount of information storable, which increases the chances of important information slipping out through the pooling layers of the encoder.

Decoder

Finally, the decoder is a set of upsampling and convolutional blocks that reconstructs the bottleneck's output. Since the input to the decoder is a compressed knowledge representation, the decoder serves as a "decompressor" and builds back the image from its latent attributes.

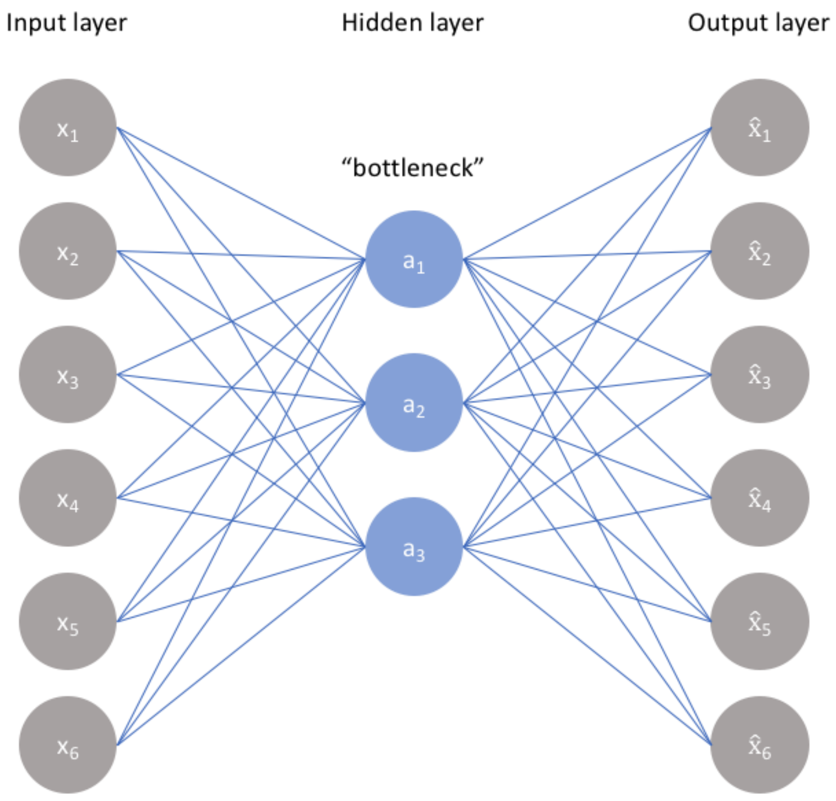
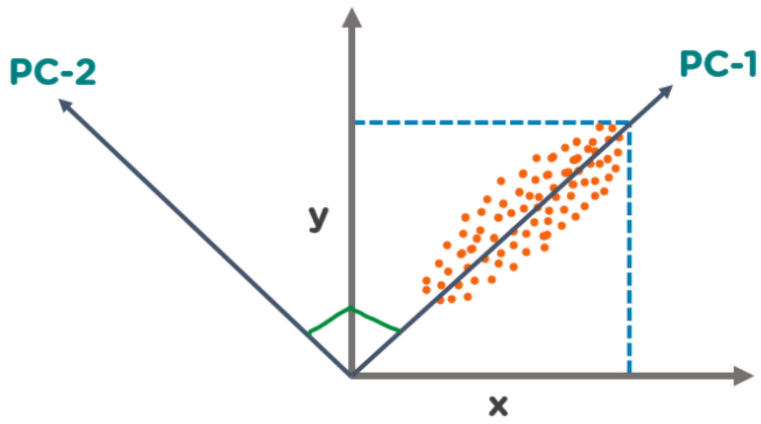


Fig 3 : Source : https://www.v7labs.com/blog/autoencoders-guide

Principal Component Analysis

The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables.



How Does Principal Component Analysis Work?

1. Normalize the Data

Standardize the data before performing PCA. This will ensure that each feature has a mean = 0 and variance = 1.

$$Z = \frac{x - \mu}{\sigma}$$

2. Build the Covariance Matrix

Construct a square matrix to express the correlation between two or more features in a multidimensional dataset.

3. Find the Eigenvectors and Eigenvalues

Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.

4. Sort the Eigenvectors in Highest to Lowest Order and Select the Number of Principal Components.

Denoising Autoencoders

Keeping the code layer small forced our autoencoder to learn an intelligent representation of the data. There is another way to force the autoencoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data. This way the autoencoder can't simply copy the input to its output because the input also contains random noise. We are asking it to subtract the noise and produce the underlying meaningful data. This is called a denoising autoencoder.

We add random Gaussian noise to them and the noisy data becomes the input to the autoencoder. The autoencoder doesn't see the original image at all. But then we expect the autoencoder to regenerate the noise-free original image.

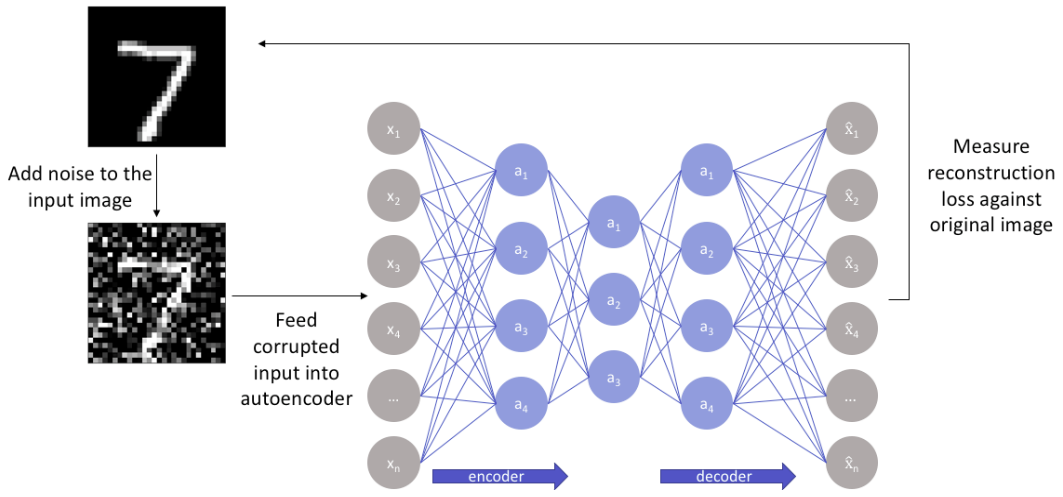

Add noise to the input image
Feed corrupted input into autoencoder
Measure reconstruction loss against original image
encoder
decoder

Fig 4 . Source : https://www.v7labs.com/blog/autoencoders-guide

While removing noise directly from the image seems difficult, the autoencoder performs this by mapping the input data into a lower-dimensional manifold , where filtering of noise becomes much easier.

Essentially, denoising autoencoders work with the help of non-linear dimensionality reduction. The loss function generally used in these types of networks is L2 or L1 loss.

Number of datapoints used to perform the task

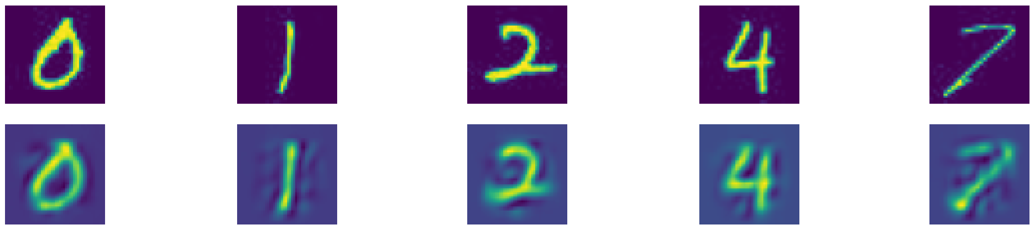
Digits	Training Data	Validation Data	Testing Data
0	2277	759	759
1	2277	759	759
2	2277	759	759
4	2277	759	759
7	2277	759	759
Total	**11385**	**3795**	**3795**

We have a MNIST dataset with 5 sets of digits [0,1,2,4,7] . The Deep Neural network is trained using these images of size (28 $\times$ 28) using different architectures.

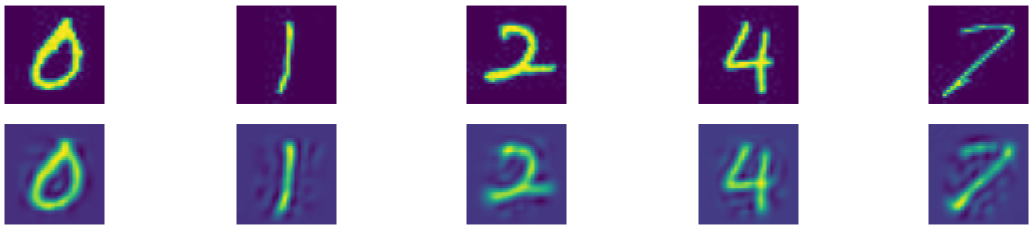
TASK - 1 : Dimension reduction using PCA:

Original Images V/S Reconstructed Images

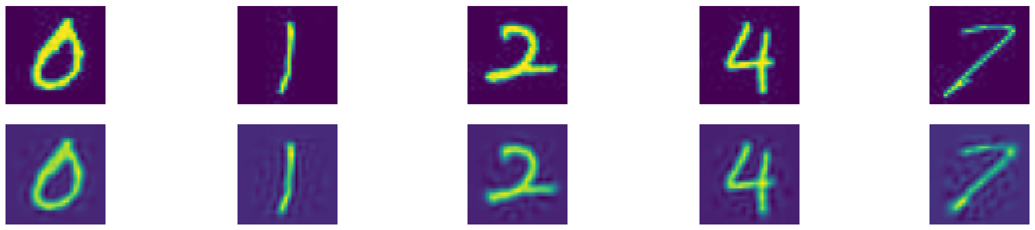
32 Dimension Reduction



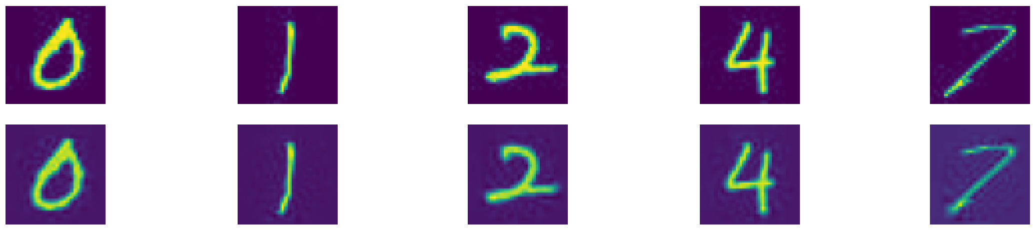
64 Dimension Reduction



128 Dimension Reduction



256 Dimension Reduction



Values of parameters taken over all architectures.

Parameters	Values
Learning Rate	0.001
Activation (Hidden)	Logistic
Activation (Output)	Softmax
Batch Size	32
Loss Function	Cross Entropy
Stopping Criterion	Early Stopping

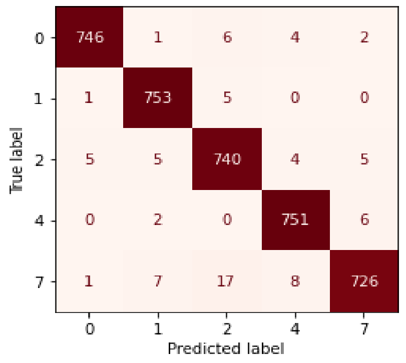
Architectures

Architectures	Inputs				H L 1	H L 2	H L 3	H L 4
Architecture 1	32	64	128	256	512	256	256	-
Architecture 2	32	64	128	256	512	512	512	-
Architecture 3	32	64	128	256	512	256	128	64

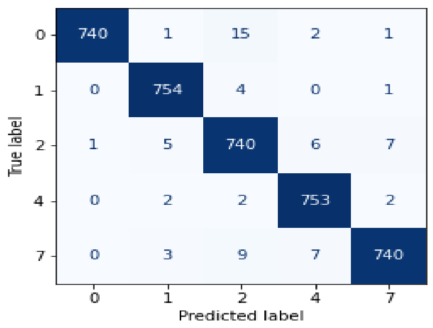
Validation and Test accuracy of various architectures

Architecture		Validation Accuracy	Test Accuracy
1	32	98.76%	97.97%
	64	98.13%	98.13%
	128	98.31%	98.23%
	256	97.79%	97.92%
2	32	98.60%	98.42%
	64	98.71%	98.08%
	128	98.52%	97.84%
	256	97.81%	97.68%
3	32	98.21%	97.76%
	64	98.31%	98.31%
	128	98.42%	97.92%
	256	98.45%	97.89%

Best Architecture based on validation accuracy

Input Dimension	**Architecture**	**Confusion Matrix**	**Val Accuracy**	**Test Accuracy**
32	512-256-256		**98.76 %**	**97.97%**

Best reduced dimension representation based on the test accuracy.

Input Dimension	Architecture	Confusion Matrix	Val Accuracy	Test Accuracy
32	512-512-512		**98.60%**	**98.42 %**

True label \ Predicted label	0	1	2	4	7
0	740	1	15	2	1
1	0	754	4	0	1
2	1	5	740	6	7
4	0	2	2	753	2
7	0	3	9	7	740

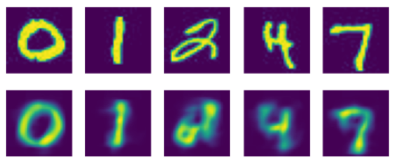
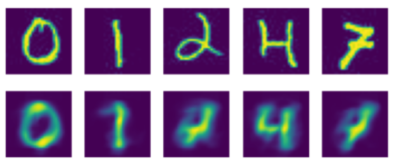
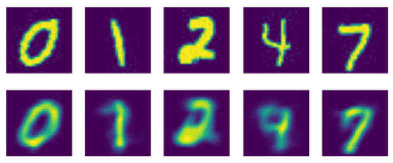
Autoencoders for reconstructing the images and Classification.

➢ **Single Hidden Layer**

Original Images V/S Reconstructed Images

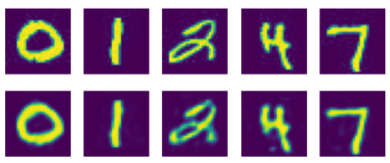
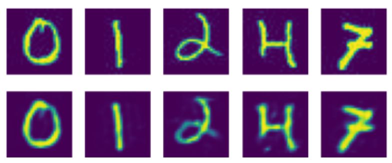
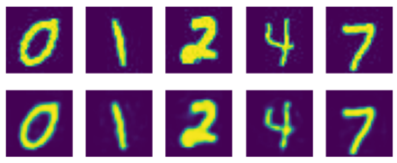
- **32 Dimension Reduction**

Input layer: 784 linear neuron | Hidden layer: 32 logistic neuron | Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

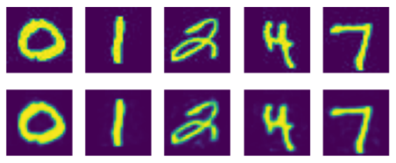
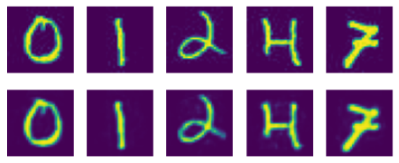
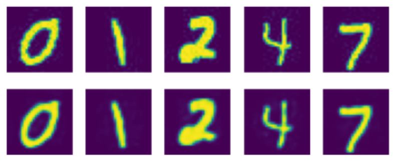
- **64 Dimension Reduction**

Input layer: 784 linear neuron | Hidden layer: 64 logistic neuron | Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

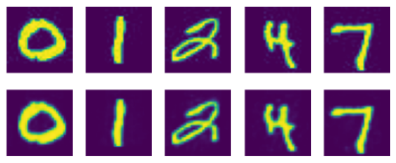
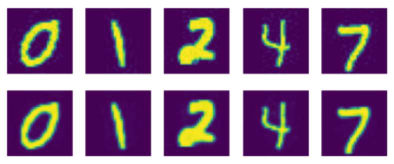
- **128 Dimension Reduction**

Input layer: 784 linear neuron | Hidden layer: 128 logistic neuron | Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

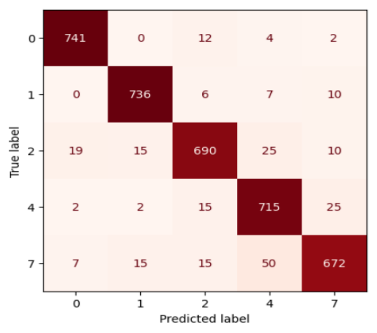
- **256 Dimension Reduction**

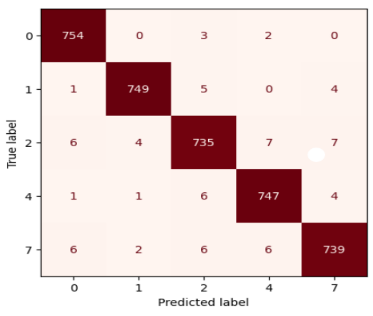
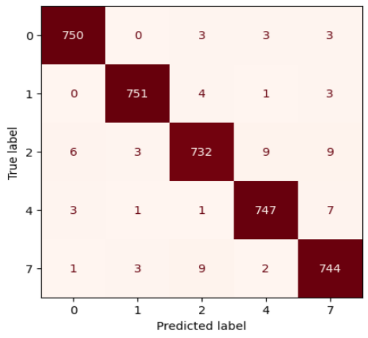
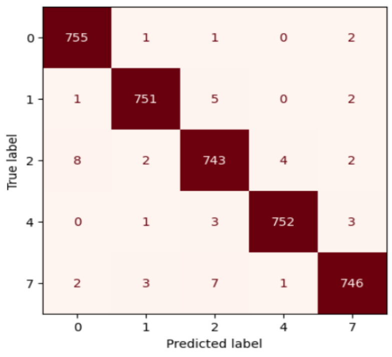
Input layer: 784 linear neuron | Hidden layer: 256 logistic neuron | Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

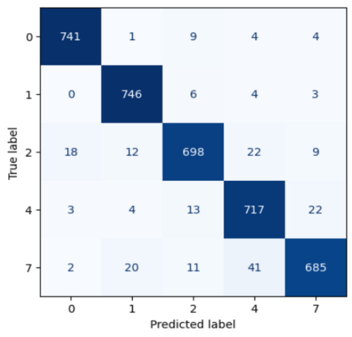
Classification using the reduced dimension image as input for various architectures.

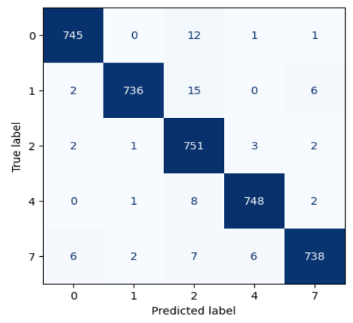
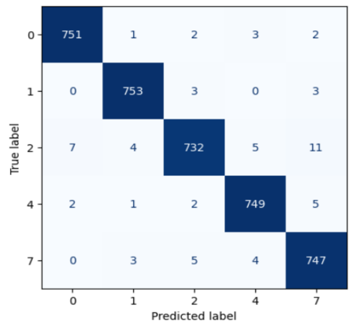
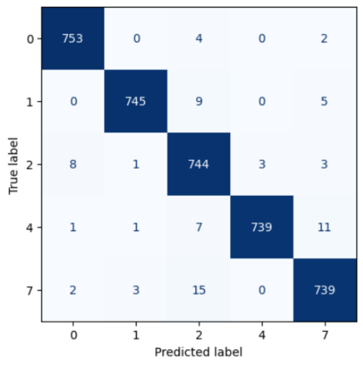
<u>Architecture 1 (512-256-256)</u>

Input Dim.	Reconstruction Error			Confusion Matrix	Accuracy	
	Train	Val	Test		Val	Test
32	0.0365	0.0366	0.0367		94.84%	93.65%

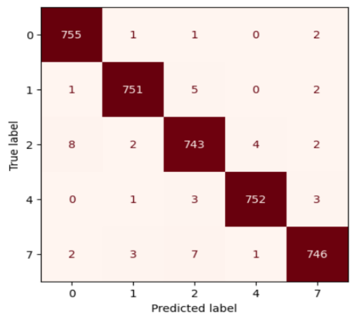
64	**0.0041**	**0.0047**	**0.0048**		**98.63%**	**98.13%**
128	**0.0021**	**0.0026**	**0.027**		**98.60%**	**98.13%**
256	**0.0014**	**0.0019**	**0.0019**		**98.97%**	**98.74%**

Architecture 2 (512-512-512)

Input Dim.	**Reconstruction Error**			**Confusion Matrix**	**Accuracy**	
	Train	**Val**	**Test**		**Val**	**Test**
32	**0.0365**	**0.0366**	**0.0367**		**92.96%**	**92.65%**

64	**0.0041**	**0.0047**	**0.0048**		**97.60%**	**97.20%**
128	**0.0021**	**0.0026**	**0.027**		**97.73%**	**97.05%**
256	**0.0014**	**0.0019**	**0.0019**		**98.42%**	**98.18%**

Best Architecture based on validation accuracy

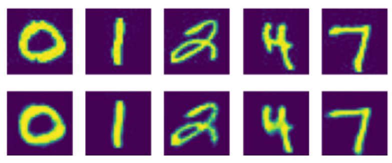
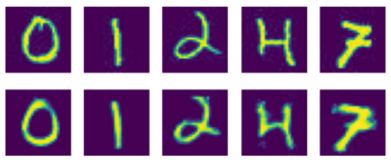
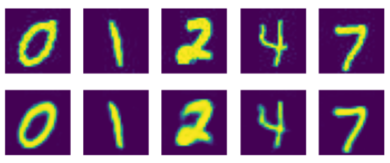
Architecture	Confusion Matrix	Val Accuracy	Test Accuracy
256-512-256-256		**98.97%**	**98.74%**

➢ Three Hidden Layers

Original Images V/S Reconstructed Images

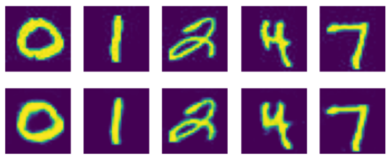
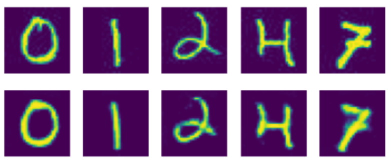
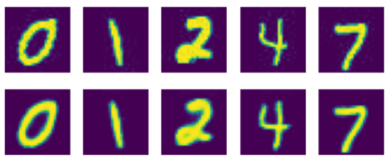
- **32 Dimension Reduction**

Input layer: 784 linear neuron • Hidden layer 1: 400 logistic neuron • Hidden layer 2: 32 logistic neuron • Hidden layer 3: 400 logistic neurons – Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

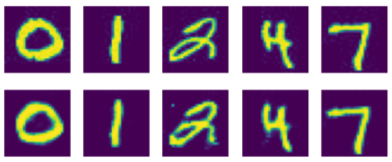
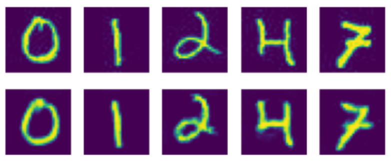
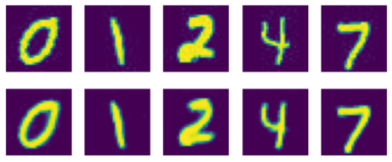
- **64 Dimension Reduction**

Input layer: 784 linear neuron • Hidden layer 1: 400 logistic neuron • Hidden layer 2: 64 logistic neuron • Hidden layer 3: 400 logistic neurons – Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

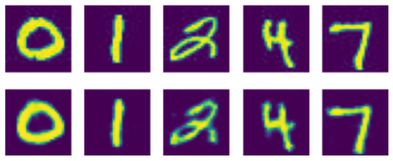
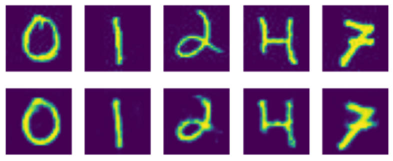
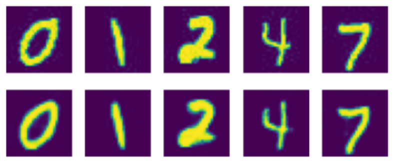
- **128 Dimension Reduction**

Input layer: 784 linear neuron • Hidden layer 1: 400 logistic neuron • Hidden layer 2: 128 logistic neuron • Hidden layer 3: 400 logistic neurons – Output layer: 784 logistic neuron

Training Data	Validation Data	Test Data
		

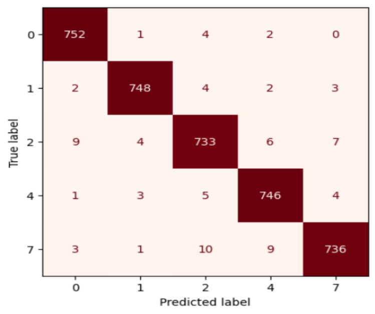
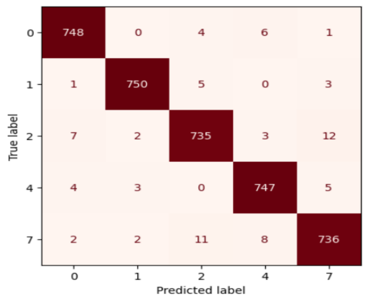
- **256 Dimension Reduction**

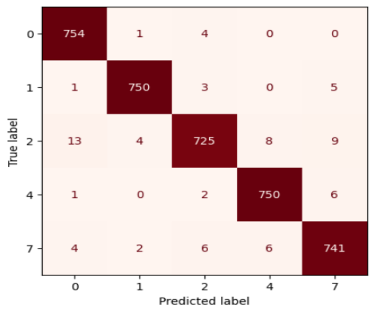
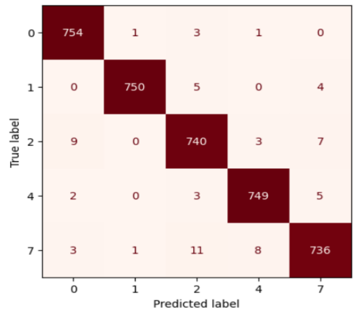
Input layer: 784 linear neuron • Hidden layer 1: 400 logistic neuron • Hidden layer 2: 256 logistic neuron • Hidden layer 3: 400 logistic neurons – Output layer: 784 logistic neuron

Training Data	**Validation Data**	**Test Data**
		

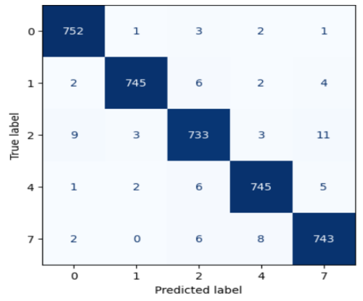
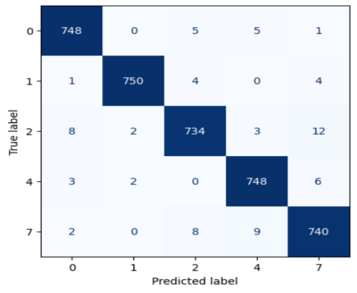
Classification using the reduced dimension image as input for various architectures.

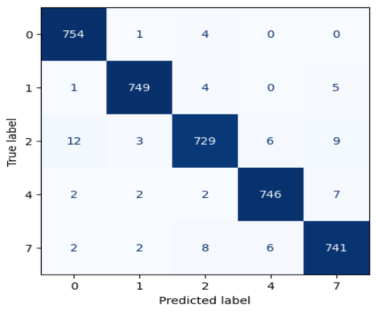
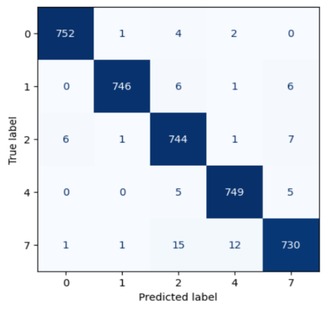
<u>Architecture 1 (512-256-256)</u>

Input Dim.	**Reconstruction Error**			**Confusion Matrix**	**Accuracy**	
	Train	**Val**	**Test**		**Val**	**Test**
32	**0.0078**	**0.0103**	**0.0105**		**94.38%**	**93.70%**
64	**0.0078**	**0.0102**	**0.0104**		**96.97%**	**95.84%**

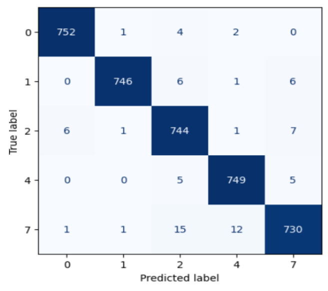
128	**0.007**	**0.0105**	**0.0106**		**97.76%**	**97.73%**
256	**0.0076**	**0.0103**	**0.0104**		**97.63%**	**97.44%**

<u>Architecture 2 (512-512-512)</u>

Input Dim.	**Reconstruction Error**			**Confusion Matrix**	**Accuracy**	
	Train	**Val**	**Test**		**Val**	**Test**
32	**0.0078**	**0.0103**	**0.0105**		**95.84%**	**95.65%**
64	**0.0078**	**0.0102**	**0.0104**		**96.97%**	**95.84%**

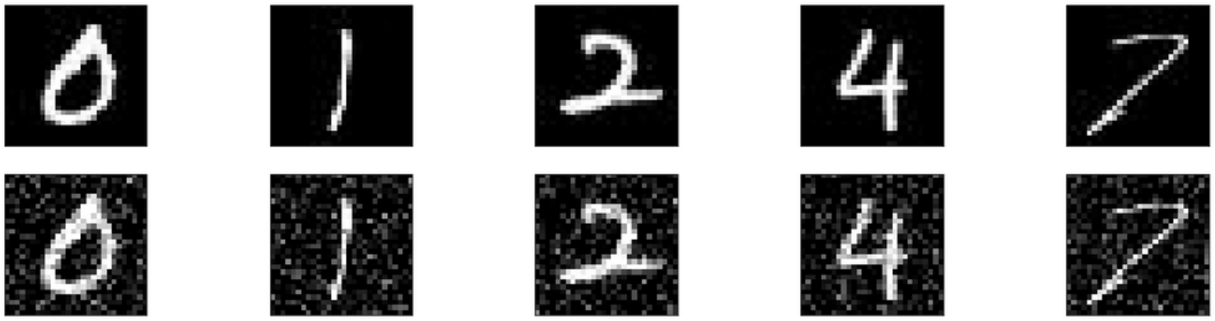
128	**0.007**	**0.0105**	**0.0106**		**97.05%**	**96.68%**
256	**0.0076**	**0.0103**	**0.0104**		**98.42%**	**98.05%**

Best Architecture based on validation accuracy

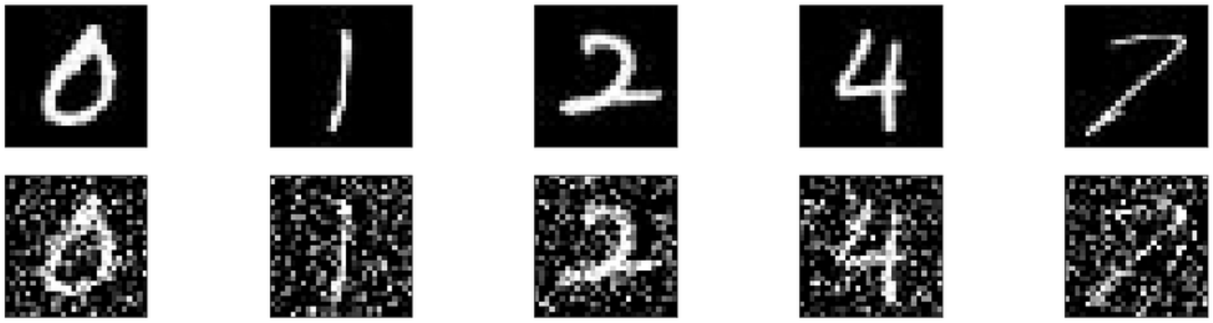
Architecture	Confusion Matrix	Val Accuracy	Test Accuracy
256-512-512-512		**98.42%**	**98.04%**

Denoising autoencoders for reconstructing images.

- **Original v/s Noisy images (with 20% Noise factor)**

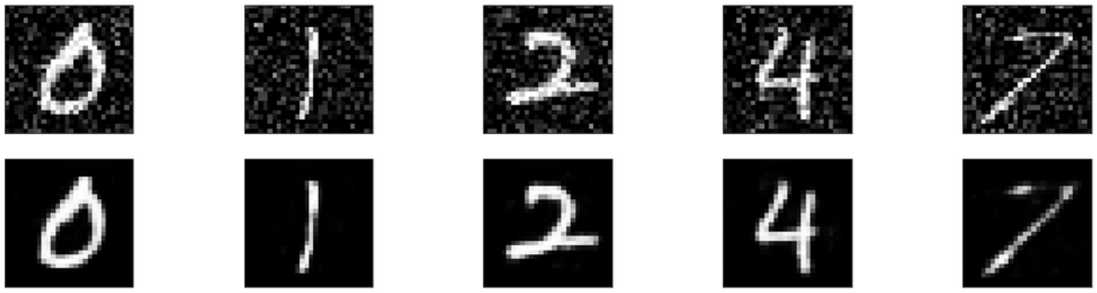


- **Original v/s Noisy images (with 40% Noise factor)**

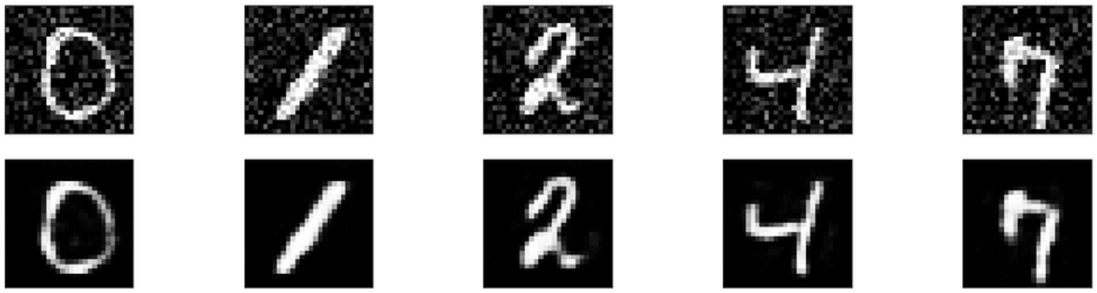


Denoising Autoencoders for reconstructing the images with 20% Noise

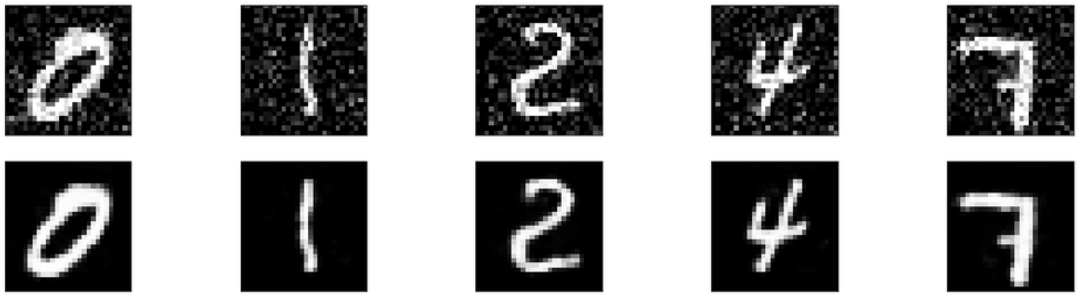
- **Training Dataset**



- **Validation Set**



- **Test Dataset**



<u>**Comparison of Average Reconstruction Error**</u>

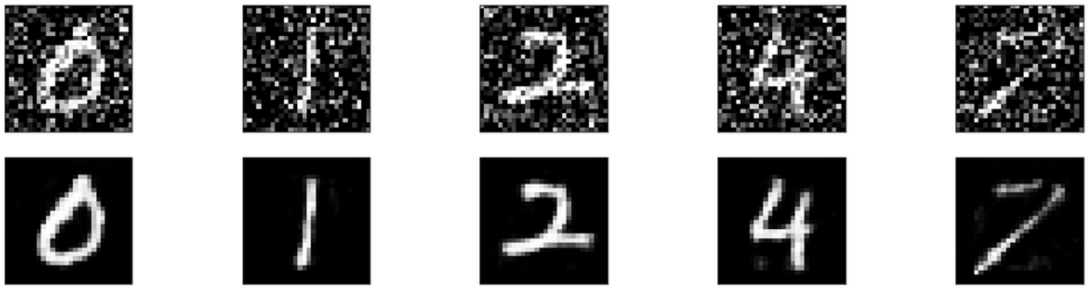
Data	Average Reconstruction Error
Training Data	0.0029
Validation Data	0.0047
Test Data	0.0047

<u>Classification accuracy on the validation and test set for the different architectures for the best reduced dimension input.</u>

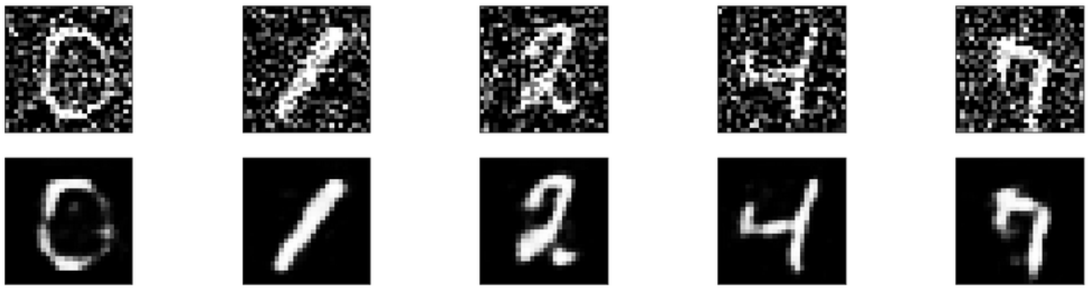
Architectures	Val Accuracy	Test Accuracy
512-256-256	98.55%	97.84%
512-512-512	98.05%	97.58%
512-256-128-64	97.44%	97.44%

<u>Denoising Autoencoders for reconstructing the images with 40% Noise</u>

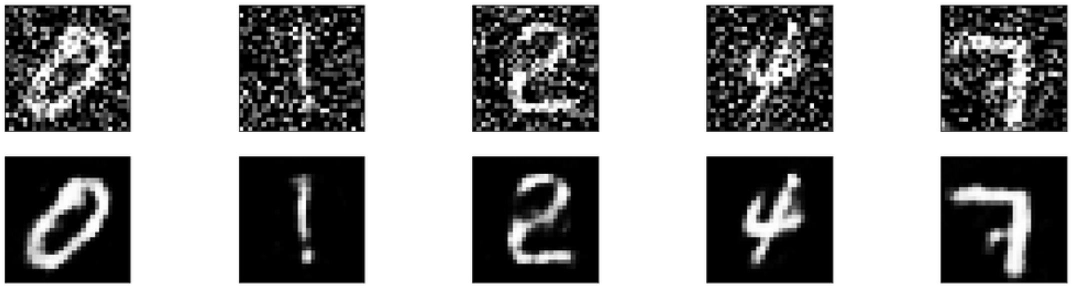
- **Training Dataset**



- **Validation Dataset**



- **Test Dataset**



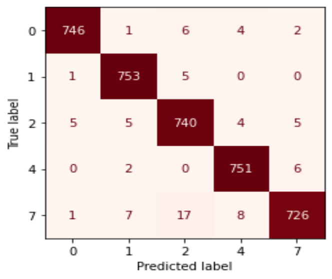
Comparison of Average Reconstruction Error

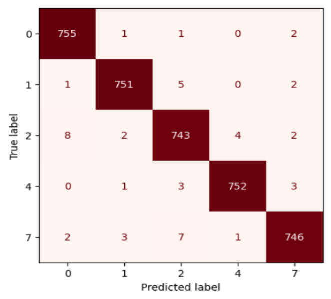
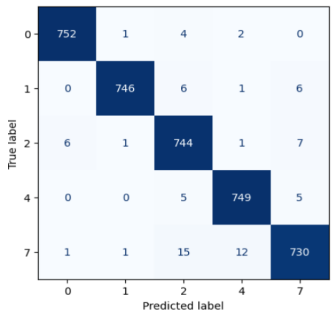
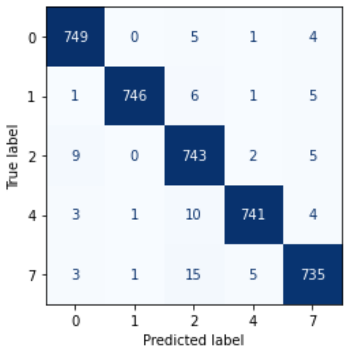
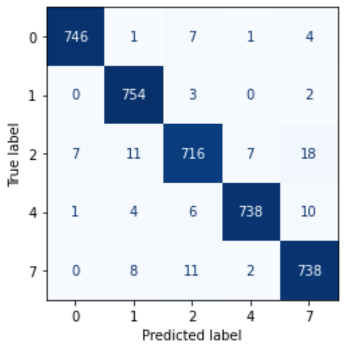
Data	Average Reconstruction Error
Training Data	0.0051
Validation Data	0.0113
Test Data	0.0114

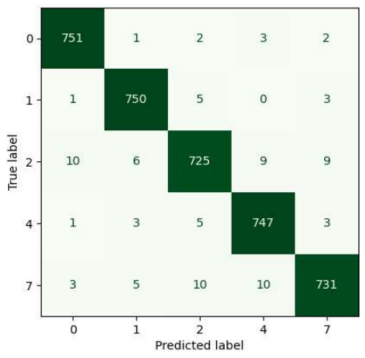
Classification accuracy on the validation and test set for the different architectures for the best reduced dimension input.

Architectures	Val Accuracy	Test Accuracy
512-256-256	97.94%	97.29%
512-512-512	97.92%	97.55%
512-256-128-64	97.34%	97.18%

Comparison of PCA , Autoencoder (Single layer) , Autoencoder (3 Layers) , Denoising(20%), Denoising(40%) and Optimizers(Assignment3).

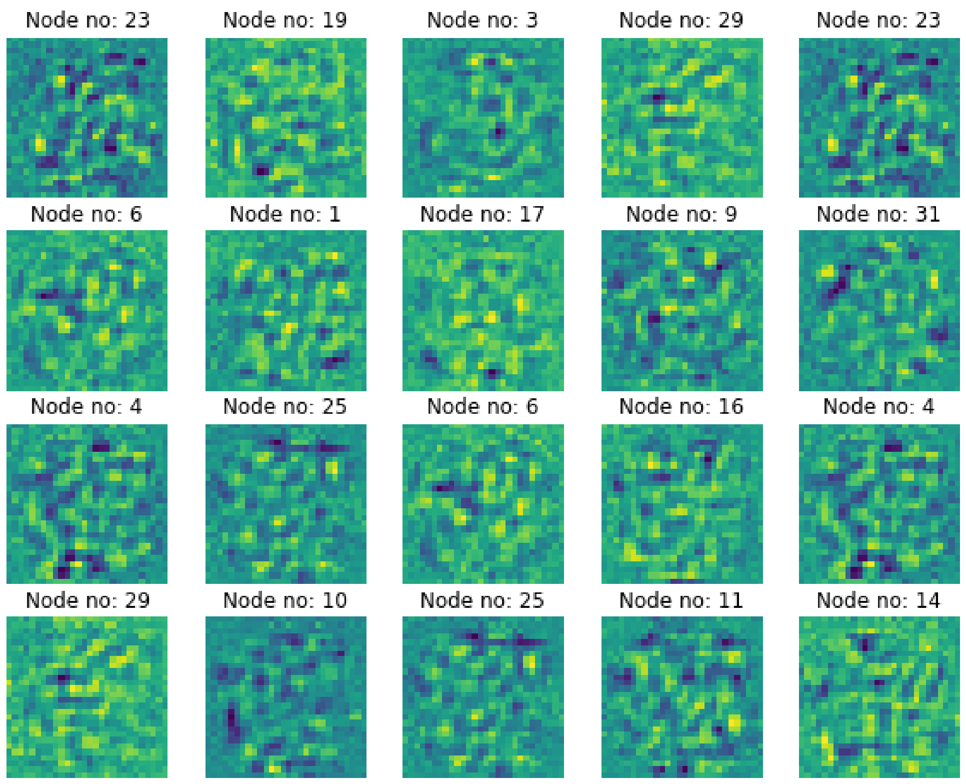
Method	Confusion Matrix	Val Accuracy	Test Accuracy
PCA		**98.96%**	**97.97%**

Autoencoder (Single Layer)		**98.97%**	**98.74%**
Autoencoder (3 Layers)		**98.42%**	**98.05%**
Denoising Autoencoder (20%)		**98.55%**	**97.84%**
Denoising Autoencoder (40%)		**97.54%**	**97.29%**

AdaGrad Optimizer (Assignment 3)		**98.81%**	**97.59%**

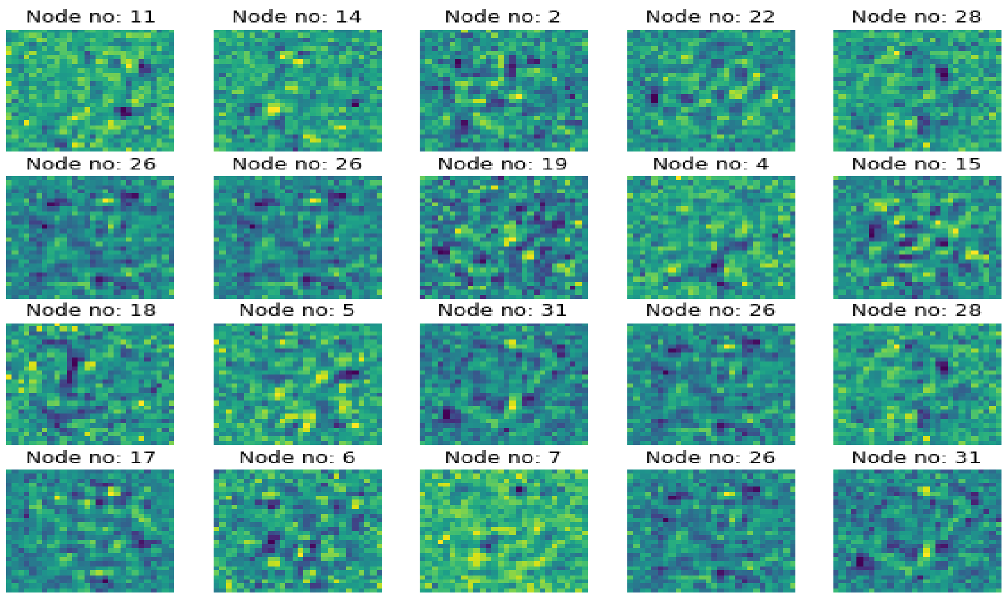
Autoencoder 256 dimension Weight Visualization

- ➢ Autoencoder learns some meaningful patterns
- ➢ The hidden neurons seem to act like pen-stroke detectors

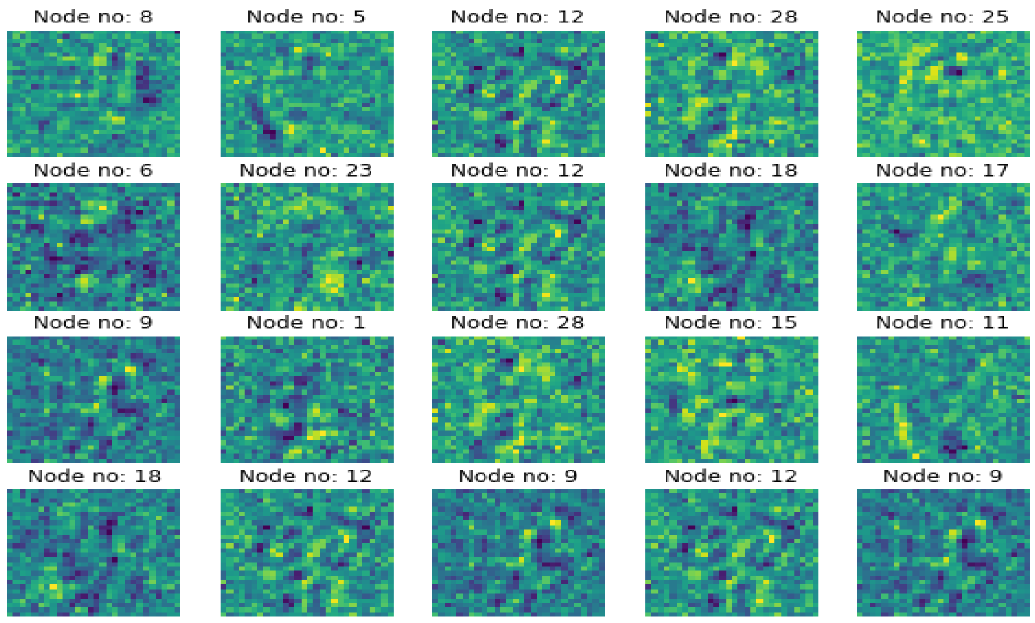


Like in node 3 we can see a pattern resembling the digit ‘2’.

- **20% Noise Weight Visualization**



- **40% Noise Weight Visualization**



Comparing the weights of the autoencoder with the weights of denoising autoencoders , it can be observed that without noise it is possible to see the patterns (penstrokes) clearly and there is a high chance of the model memorizing those patterns. With noise, there are lots of irregularities and here too pattern is visible, but avoids overfitting by not letting the model to memorize the noise.