13/4/2023

# Deep Learning Assignment-4

PCA & Autoencoders


Presented by:

1. R. Ramakrishna Kashyap – T22101
2. Anuj Kumar Shukla – T22103
3. Nishant Gupta- T22221

# Index

## 1. Introduction

Principal Component Analysis (PCA) and Autoencoders are two commonly used techniques in the field of machine learning and data analysis for dimensionality reduction.

- PCA is a linear technique used to reduce the dimensionality of a dataset by projecting it onto a lower-dimensional space while preserving as much of the original information as possible. The idea is to find the directions in the original feature space where the data varies the most, which are called the principal components. These components are then used to transform the original data into a new space with fewer dimensions.
- Autoencoders are a type of neural network that are designed to learn a compressed representation of the input data, which can be used for tasks such as data compression, image denoising, image classification etc. They consist of an encoder and a decoder network, which are trained to reconstruct the input data from the compressed representation. The compressed representation is often referred to as the bottleneck layer or latent space, which has a lower dimensionality than the input data.

Both PCA and autoencoders can be used for dimensionality reduction, but they differ in their approach and the types of datasets they are best suited for. PCA is a linear method and is best suited for datasets with linear relationships between the features, while autoencoders are non-linear and can capture more complex relationships between the features. However, autoencoders can be more computationally expensive and may require more data to train effectively.

## 2. Working of Principal Component Analysis

PCA works by transforming a dataset into a new coordinate system such that the greatest variances in the data are captured along these coordinates. These coordinates are called principal components and they all are orthogonal to each other.

For performing the PCA, following steps are taken:

- **Standardize the data:** PCA assumes that the variables in the dataset have the same scale. Therefore, it is important to standardize the data before performing PCA.
- **Calculate the covariance matrix:** The covariance matrix represents the relationship between variables in the dataset. PCA uses this matrix to find the principal components.
- **Find the eigenvalues and eigenvectors of the covariance matrix:** The eigenvalues represent the amount of variance captured by each principal component, while the eigenvectors represent the direction of the principal components.
- **Select the principal components:** The principal components are selected based on the eigenvalues. Typically, only the top k principal components are selected, where k is a number that is chosen based on the amount of variance that needs to be retained.
- **Transform the data:** Finally, the dataset is transformed into the new coordinate system, which is formed by the selected principal components.

## 3. Working of Autoencoders

An autoencoder is an unsupervised machine learning algorithm that can learn a compressed representation of a dataset. It works by encoding the input data into a lower dimensional representation and then reconstructing it back to its original form. The encoding and decoding process is learned through training the model on the input data.

These are the steps we use for Autoencoders:

- **Pre-processing:** First, the input data is pre-processed to make it suitable for training. This may include normalizing the data, scaling it, or applying other transformations.
- **Design the network:** Next, the architecture of the autoencoder is designed. An autoencoder consists of two main parts: an encoder and a decoder. The encoder takes the input data and maps it to a lower dimensional space, while the decoder takes the encoded data and reconstructs it back to the original dimension.
- **Train the model:** The autoencoder is trained on the input data using an optimization algorithm, such as stochastic gradient descent. During training, the encoder and decoder are adjusted to minimize the difference between the input and the reconstructed output. This is typically done by minimizing the mean squared error (MSE) between the input and the output.
- **Validation:** After training, the performance of the autoencoder is evaluated on a validation set. This is done to ensure that the model is not overfitting to the training data.
- **Use the model:** Once the model is trained, it can be used to encode new data into a lower dimensional representation. This can be useful for data compression or for feature extraction.

Here is a more detailed explanatio of how the encoding and decoding process works:

- **Encoding:** The input data is fed into the encoder network, which typically consists of one or more layers of neural networks. Each layer applies a linear transformation to the input data followed by a non-linear activation function, such as the sigmoid or ReLU function. The output of the final layer represents the encoded representation of the input data.
- **Decoding:** The encoded representation is fed into the decoder network, which also consists of one or more layers of neural networks. Each layer applies a linear transformation to the encoded representation followed by a non-linear activation function. The output of the final layer represents the reconstructed output of the original input data.

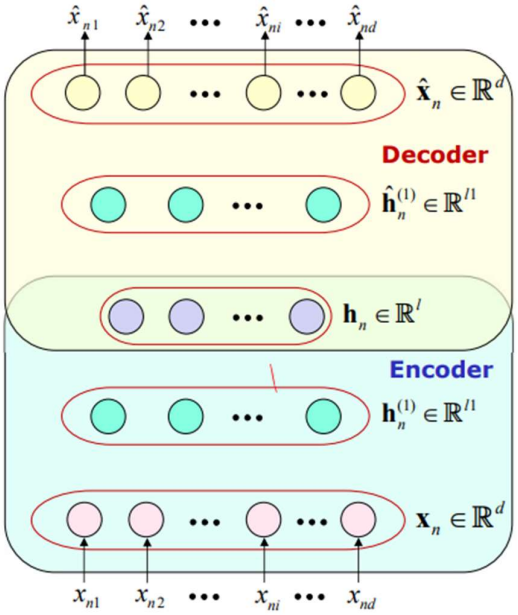



**Figure 1: Autoencoder Architecture**

**Image source:** Lecture slides (CS671: Deep learning and Applications, Dr. Dileep A.D.)

## 4. Link between PCA and Autoencoder

The encoder part of 1-hidden layer autoencoder will be equivalent to PCA if these conditions are satisfied:

- Encoder is linear
- Decoder is linear
- Loss function is squared error loss function
- Inputs are normalised to:

$$x'_{ni} = \frac{1}{\sqrt{N}}\left( x_{ni} - \frac{1}{N}\sum_{n=1}^{N} x_{ni} \right)$$

$N$: Number of examples in a dataset (training example)

## 5. 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 $i^{th}$ input value($x_{ni}$)

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

With a probability (1-q), input is retained as it is.

This is done to avoid overfitting. It helps because our objective is still to reconstruct the original input ($x_n$).

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

## 6. Details of the tasks and data

There are six tasks to be done.

**Task 1:** Dimension reduction using PCA (Reduce the dimensions to 32, 64, 128 and 256 dimensions.)

**Task 2:** Autoencoders for reconstructing the images

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

**Task 4:** Classification using the compressed representation from the 2-hidden layer autoencoder.

**Task 5:** Denoising autoencoders for reconstructing the images.

**Task 6:** Weight visualization.

We have been given images of size 28*28 of numbers 0, 1, 2, 3, 7 with various orientations for training, validation, and testing.

| Numbers | Training | Validation | Testing |
|---|---|---|---|
| 0 | 2277 | 759 | 759 |
| 1 | 2277 | 759 | 759 |
| 2 | 2277 | 759 | 759 |
| 3 | 2277 | 759 | 759 |
| 7 | 2277 | 759 | 759 |

## 7. Parameters used

- Same initial random values of weights are used for each architecture using each of the optimizers.
- Stopping criteria is used as absolute difference between average error of successive epochs falls below a threshold of 0.0001.
- Learning rate(η) is used as 0.001 for all the optimizers.
- For Adam, $\beta 1 = 0.9$, $\beta 2 = 0.999$ and $\varepsilon = 10^{-8}$ are used.
- Tan hyperbolic activation function was used in hidden layers while Linear action function was used at output layer.

## 8. Architectures Used

We have used three architectures for all cases of dimensions.

| Architecture | Hidden layer-1 | Hidden layer-2 | Hidden layer-3 | Output layer |
|---|---|---|---|---|
| **I** | 64 | 32 | 16 | 5 |
| **II** | 32 | 16 | 8 | 5 |
| **III** | 128 | 64 | 32 | 5 |

**Table-1: Number of Neurons in all layers**

## 9. Results and Inferences

## Task-1

### ➢ Classification Accuracies on validation set

Classification accuracies on validation set for all the architectures.

| Architecture | 32 | 64 | 128 | 256 |
|---|---|---|---|---|
| I | 98.39 | 98.18 | 98.23 | 98.05 |
| II | 97.97 | 97.87 | 97.84 | 97.60 |
| III | 98.37 | 98.13 | 98.23 | 98.10 |

**Table-2: Classification accuracies**

It can be concluded from the validation accuracies that architecture-3 is the best among three architectures.

### ➢ Accuracy and Confusion matrix for test data

Reduced dimension to 32

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 740 | 0 | 12 | 3 | 4 |
| | Class 2 (1) | 0 | 751 | 4 | 3 | 1 |
| | Class 3 (2) | 4 | 2 | 738 | 8 | 7 |
| | Class 4 (3) | 7 | 1 | 6 | 741 | 4 |
| | Class 5 (7) | 0 | 5 | 14 | 4 | 736 |

**Table-3: Confusion matrix of reduced dimension to 32 for Arch. III**

Test accuracy – 97.65%

Reduced dimension to 64

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 750 | 0 | 6 | 1 | 2 |
| | Class 2 (1) | 0 | 753 | 3 | 0 | 3 |
| | Class 3 (2) | 3 | 4 | 730 | 13 | 10 |
| | Class 4 (3) | 3 | 4 | 12 | 733 | 7 |
| | Class 5 (7) | 1 | 5 | 6 | 6 | 741 |

**Table-4: Confusion matrix of reduced dimension to 64 for Arch. III**

Test accuracy – 97.68%

Reduced dimension to 128

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 752 | 0 | 2 | 2 | 3 |
| | Class 2 (1) | 0 | 750 | 4 | 3 | 2 |
| | Class 3 (2) | 7 | 2 | 737 | 7 | 6 |
| | Class 4 (3) | 3 | 5 | 12 | 736 | 3 |
| | Class 5 (7) | 1 | 7 | 12 | 5 | 734 |

**Table-5: Confusion matrix of reduced dimension to 128 for Arch. III**

Test accuracy – 97.73%

Reduced dimension to 256

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 751 | 0 | 4 | 3 | 1 |
| | Class 2 (1) | 0 | 752 | 3 | 1 | 3 |
| | Class 3 (2) | 8 | 5 | 729 | 12 | 5 |
| | Class 4 (3) | 5 | 4 | 10 | 731 | 9 |
| | Class 5 (7) | 3 | 3 | 8 | 4 | 741 |

**Table-6: Confusion matrix of reduced dimension to 256 for Arch. III**

Test accuracy – 97.60%

From the above observations we can conclude that, one has to consider the trade-off between reducing the dimensionality of the data and maintaining sufficient information for accurate classification. As the dimensionality of the data decreases, some information is inevitably lost, but to preserve the most important features of the data we have to increase the reduced dimensions.

From our results it appears that reducing the dimensionality to 128 strikes a good balance between preserving relevant information and eliminating noise. We got similar accuracy for less complex architecture compare to more complex architectures.

## ➢ Comparison of the result with best result from Assignment- 3

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 | Class 2 | Class 3 | Class 4 | Class 5 |
| | Class 1 | 742 | 0 | 3 | 13 | 1 |
| | Class 2 | 0 | 737 | 7 | 13 | 2 |
| | Class 3 | 14 | 0 | 701 | 39 | 5 |
| | Class 4 | 6 | 4 | 14 | 725 | 10 |
| | Class 5 | 3 | 8 | 16 | 11 | 721 |

**Test Confusion matrix**

In assignment -3, based on validation accuracy 512-256-128 architecture was performing better compared to other architectures considered, It gave an accuracy of 95.5% on test data. In case of PCA we observed an average of 97% test accuracy over all the reduced dimensions.

## Task-2

- **Average reconstruction error for 1 hidden layer architecture**

| Reduced Dimension | Training | Validation | Testing |
|---|---|---|---|
| **32** | 0.015524892 | 0.015637841 | 0.015739448 |
| **64** | 0.008880058 | 0.009039381 | 0.009072194 |
| **128** | 0.004591397 | 0.004748563 | 0.004760539 |
| **256** | 0.001868027 | 0.001991396 | 0.002005240 |

**Table-7: Avg. recon. Error for 1 hidden layer**

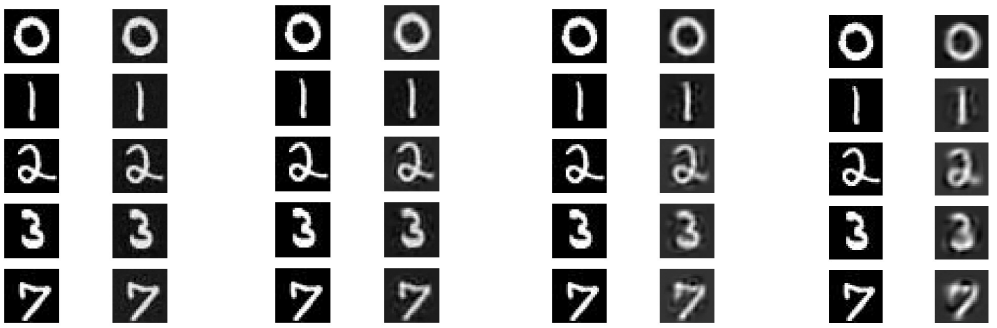
- **Average reconstruction error for 3 hidden layer architecture**

| Reduced Dimension | Training | Validation | Testing |
|---|---|---|---|
| **32** | 0.015733834 | 0.015835140 | 0.015912112 |
| **64** | 0.009226938 | 0.009360145 | 0.009384766 |
| **128** | 0.004951029 | 0.005110173 | 0.005126679 |
| **256** | 0.002803486 | 0.002938116 | 0.002953617 |

**Table-8: Avg. recon. Error for 3 hidden layers**

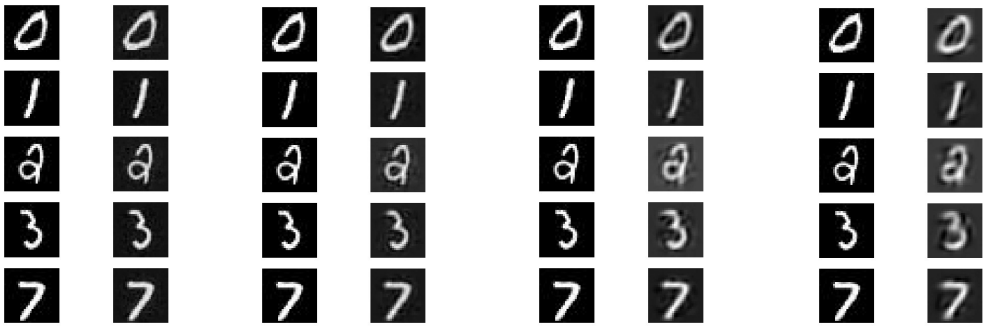
Reconstructed images for each class and each dimension is shown below.

**Training Images**



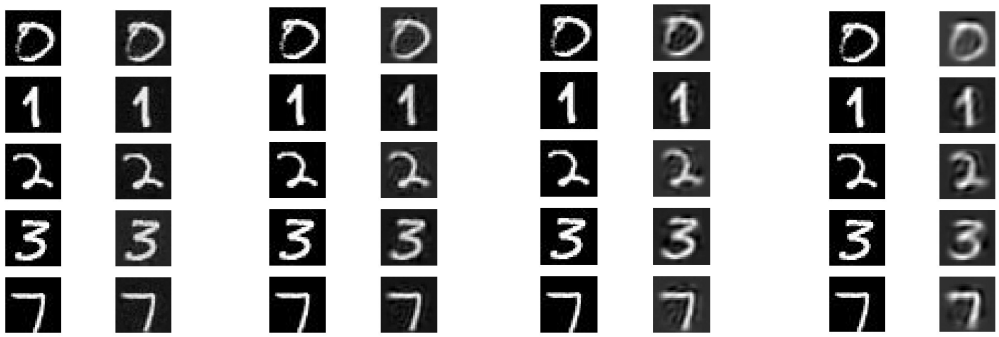
256 128 64 32

**Validation Images**



256 128 64 32

**Test Images**

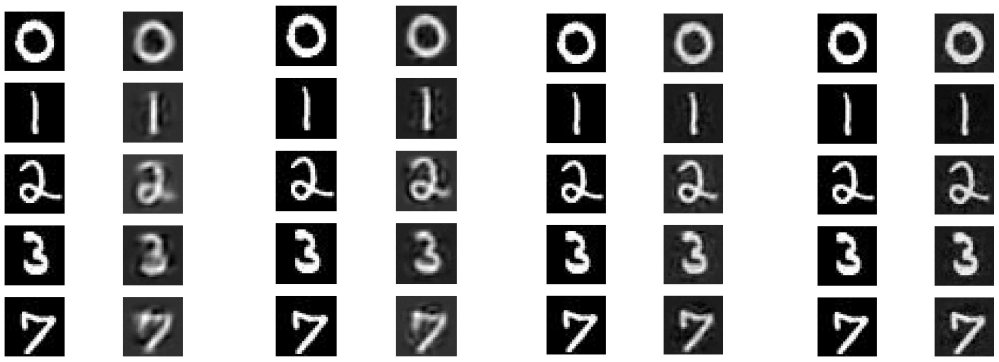


256 128 64 32

**Figure 2: Reconstructed images for 1 hidden layer**

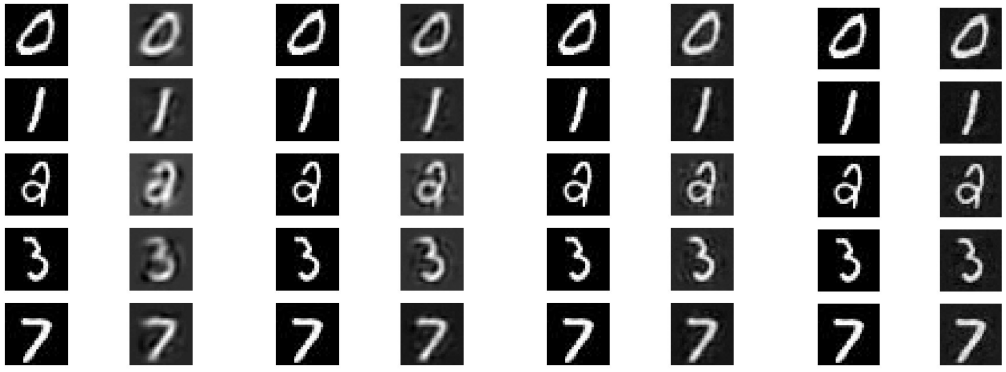
As we are increasing the neurons in the bottleneck layer the reconstructed images are becoming much clearer.

**Training Images**



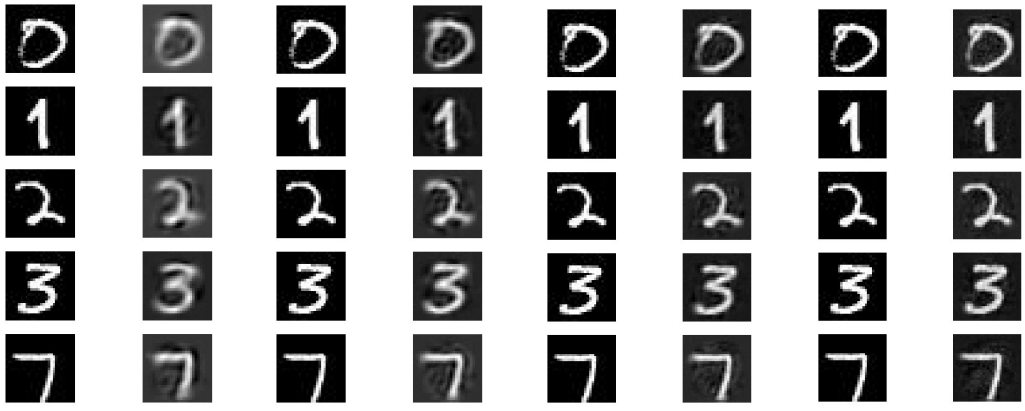
32 64 128 256

**Validation Images**



32 64 128 256

**Testing Images**



32 64 128 256

**Figure 3: Reconstructed images for 3 hidden layers**

As we are increasing the neurons in the bottleneck layer the reconstructed images are becoming much clearer. We can conclude that as the number for neurons in the bottleneck layer increases, the average reconstruction error decreases, this is because if we use a greater number of neurons, it will learn more components from the input data.

## Task-3

### ➢ Classification Accuracies on validation set

Classification accuracies on validation set for all the architectures.

**I: 64-32-16-5**

**II: 32-16-8-5**

**III: 128-64-32-5**

**1-Hidden layer**

| Architecture | 32 | 64 | 128 | 256 |
|---|---|---|---|---|
| I | 98.23 | 98.55 | 98.39 | 98.13 |
| II | 97.99 | 98.18 | 97.86 | 97.73 |
| III | 98.03 | 98.36 | 98.41 | 97.89 |

**Table-9: Classification accuracies**

It can be concluded from the validation accuracies that architecture-1 is best among three architectures.

### ➢ Accuracy and Confusion matrix for test data

Reduced dimension to 32

**Table-10: Confusion matrix of reduced dimension to 32 for Arch. I**

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 745 | 0 | 10 | 3 | 1 |
| | Class 2 (1) | 1 | 750 | 4 | 1 | 3 |
| | Class 3 (2) | 8 | 3 | 731 | 10 | 7 |
| | Class 4 (3) | 2 | 3 | 8 | 736 | 10 |
| | Class 5 (7) | 2 | 1 | 8 | 1 | 747 |

Test accuracy – 97.68%

Reduced dimension to 64

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | **Class 1 (0)** | **Class2 (1)** | **Class 3 (2)** | **Class 4 (3)** | **Class 5 (7)** |
| | **Class 1 (0)** | 751 | 0 | 3 | 3 | 2 |
| | **Class 2 (1)** | 1 | 748 | 2 | 3 | 5 |
| | **Class 3 (2)** | 8 | 1 | 735 | 7 | 8 |
| | **Class 4 (3)** | 6 | 2 | 12 | 732 | 7 |
| | **Class 5 (7)** | 1 | 2 | 8 | 5 | 743 |

**Table-11: Confusion matrix of reduced dimension to 64 for Arch. I**

Test accuracy – 97.65%

Reduced dimension to 128

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | **Class 1 (0)** | **Class2 (1)** | **Class 3 (2)** | **Class 4 (3)** | **Class 5 (7)** |
| | **Class 1 (0)** | 754 | 0 | 3 | 1 | 1 |
| | **Class 2 (1)** | 2 | 750 | 2 | 3 | 2 |
| | **Class 3 (2)** | 18 | 2 | 710 | 22 | 7 |
| | **Class 4 (3)** | 4 | 1 | 8 | 740 | 6 |
| | **Class 5 (7)** | 2 | 2 | 6 | 6 | 743 |

**Table-12: Confusion matrix of reduced dimension to 128 for Arch. I**

Test accuracy – 97.73%

Reduced dimension to 256

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 755 | 0 | 1 | 2 | 1 |
| | Class 2 (1) | 1 | 747 | 2 | 2 | 7 |
| | Class 3 (2) | 11 | 3 | 722 | 16 | 7 |
| | Class 4 (3) | 6 | 0 | 7 | 741 | 5 |
| | Class 5 (7) | 4 | 3 | 9 | 7 | 736 |

**Table-13: Confusion matrix of reduced dimension to 256 for Arch. I**

## Test accuracy – 97.52%

From our results it appears that reducing the dimensionality to 128 strikes a good balance between preserving relevant information and eliminating noise.

In assignment -3, based on validation accuracy 512-256-128 architecture was performing better compared to other architectures considered. It gave an accuracy of 95.5% on test data. In case of PCA we observed an average of 97% test accuracy over all the reduced dimensions. In autoencoder we didn't see much improvement in test accuracies (all of them were approximately 97.5%) as compared to PCA.

## Task-4

### ➢ Classification Accuracies on validation set

Classification accuracies on validation set for all the architectures.

**I: 64-32-16-5**
**II: 32-16-8-5**
**III: 128-64-32-5**

**3-Hidden layer**

| Architecture | 32 | 64 | 128 | 256 |
|---|---|---|---|---|
| I | 98.23 | 98.55 | 98.39 | 98.13 |
| II | 98.16 | 98.88 | 98.35 | 97.89 |
| III | 98.18 | 98.47 | 97.97 | 97.89 |

**Table-14: Classification accuracies**

It can be concluded from the validation accuracies that architecture-1 is best among three architectures.

### ➢ Accuracy and Confusion matrix for test data

Reduced dimension to 32

**Table-15: Confusion matrix of reduced dimension to 32 for Arch. I**

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 753 | 0 | 2 | 2 | 2 |
| | Class 2 (1) | 0 | 748 | 5 | 1 | 5 |
| | Class 3 (2) | 9 | 6 | 728 | 3 | 13 |
| | Class 4 (3) | 7 | 2 | 12 | 730 | 8 |
| | Class 5 (7) | 2 | 1 | 7 | 1 | 748 |

Test accuracy – 97.25%

Reduced dimension to 64

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 752 | 0 | 3 | 2 | 2 |
| | Class 2 (1) | 1 | 748 | 3 | 2 | 5 |
| | Class 3 (2) | 12 | 1 | 726 | 11 | 9 |
| | Class 4 (3) | 8 | 1 | 13 | 728 | 9 |
| | Class 5 (7) | 2 | 1 | 7 | 1 | 748 |

**Table-16: Confusion matrix of reduced dimension to 64 for Arch. I**

Test accuracy – 97.39%

Reduced dimension to 128

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | Class 1 (0) | Class2 (1) | Class 3 (2) | Class 4 (3) | Class 5 (7) |
| | Class 1 (0) | 753 | 0 | 4 | 1 | 1 |
| | Class 2 (1) | 1 | 750 | 3 | 2 | 3 |
| | Class 3 (2) | 8 | 7 | 724 | 12 | 8 |
| | Class 4 (3) | 12 | 6 | 7 | 727 | 7 |
| | Class 5 (7) | 2 | 3 | 9 | 8 | 737 |

**Table-17: Confusion matrix of reduced dimension to 128 for Arch. I**

Test accuracy – 97.46%

Reduced dimension to 256

| | Actual Class | | | | | |
|---|---|---|---|---|---|---|
| Predicted Class | | **Class 1 (0)** | **Class2 (1)** | **Class 3 (2)** | **Class 4 (3)** | **Class 5 (7)** |
| | **Class 1 (0)** | 751 | 0 | 4 | 1 | 3 |
| | **Class 2 (1)** | 2 | 747 | 2 | 3 | 5 |
| | **Class 3 (2)** | 8 | 4 | 729 | 9 | 9 |
| | **Class 4 (3)** | 5 | 5 | 15 | 728 | 6 |
| | **Class 5 (7)** | 3 | 3 | 8 | 10 | 735 |

**Table-18: Confusion matrix of reduced dimension to 256 for Arch. I**

Test accuracy – 97.23%

From our results it appears that reducing the dimensionality to 128 strikes a good balance between preserving relevant information and eliminating noise.
In assignment -3, based on validation accuracy 512-256-128 architecture was performing better compared to other architectures considered. It gave an accuracy of 95.5% on test data. In case of PCA we observed an average of 97% test accuracy over all the reduced dimensions. In 3 Hidden layer autoencoder we didn't see much improvement in test accuracies (all of them were approximately 97.5%) as compared to PCA.

## Task-5

Based on Test accuracy, best architecture of 1 Hidden layer Autoencoder was found to be the one with 64 neurons in hidden layer.

Gaussian distribution was used to add noise at the input.

### 20% noise in Input

| Average Reconstruction Error | |
|---|---|
| **Data set** | **Error** |
| Training Data | 0.0089 |
| Validation Data | 0.0090 |
| Testing | 0.0090 |

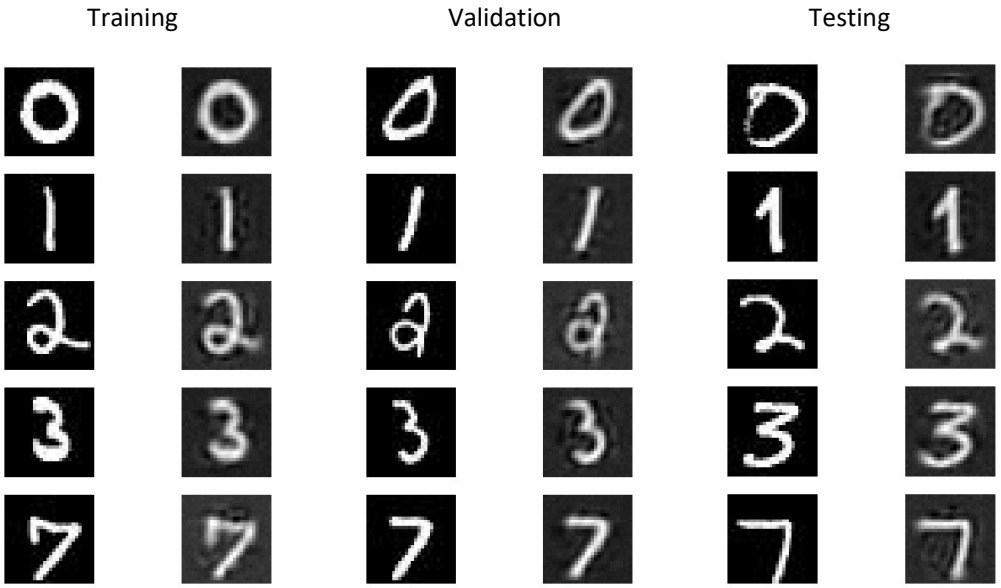



**Figure 4: Reconstructed images for 1 hidden layer with 20% noise**

## Classification accuracy

| Validation Dataset | |
|---|---|
| **Architectures** | **64** |
| 64-32-16 | 98% |
| 32-16-8 | 98.1% |
| 128-64-32 | 98.2% |

| Test Dataset | |
|---|---|
| **Architectures** | **64** |
| 64-32-16 | 97% |
| 32-16-8 | 98.3% |
| 128-64-32 | 97.78% |

## 40% noise in Input

| Average Reconstruction Error | |
|---|---|
| **Data set** | **Error** |
| Training Data | 0.0098 |
| Validation Data | 0.0099 |
| Testing | 0.0100 |

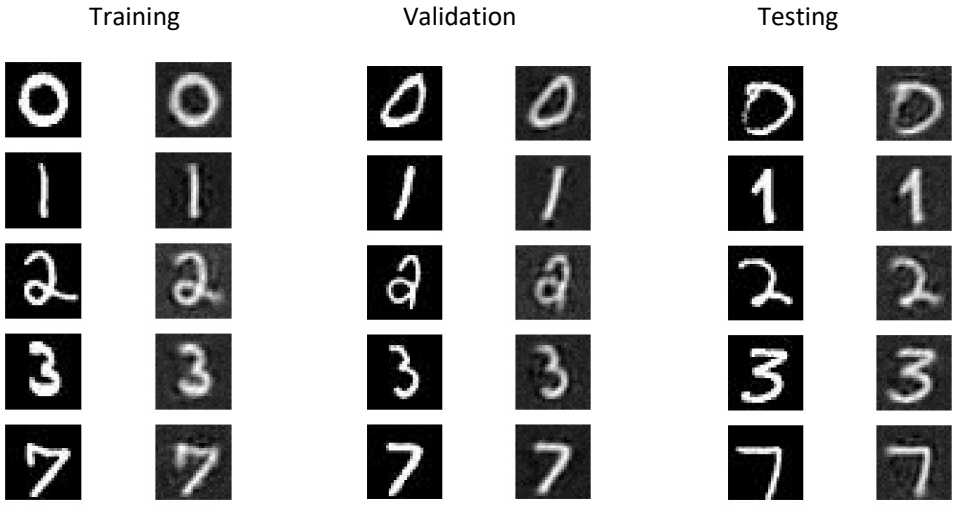


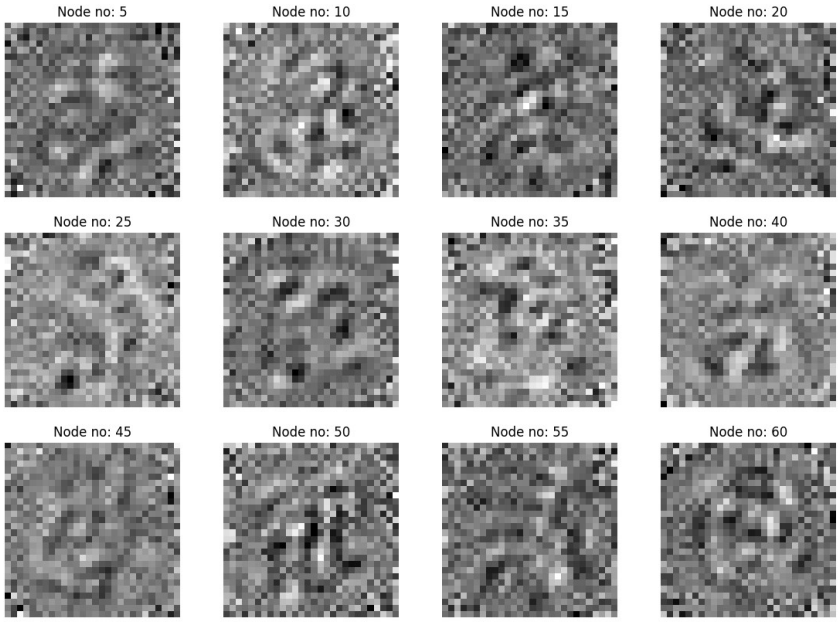

**Figure 5: Reconstructed images for 1 hidden layer with 40% noise**

# Classification accuracy

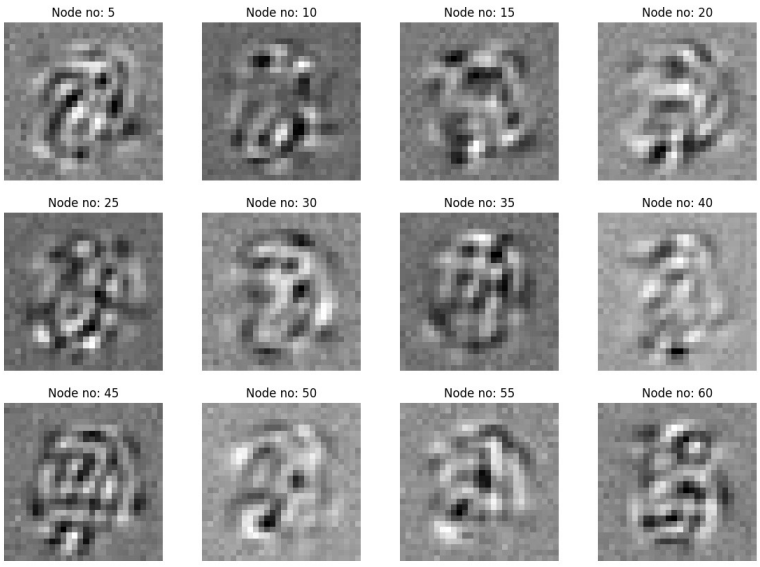
| Validation Dataset | |
|---|---|
| **Architectures** | **64** |
| 64-32-16 | 98.10% |
| 32-16-8 | 97.78% |
| 128-64-32 | 98.34% |

| Test Dataset | |
|---|---|
| **Architectures** | **64** |
| 64-32-16 | 97.54% |
| 32-16-8 | 97.42% |
| 128-64-32 | 97.86% |

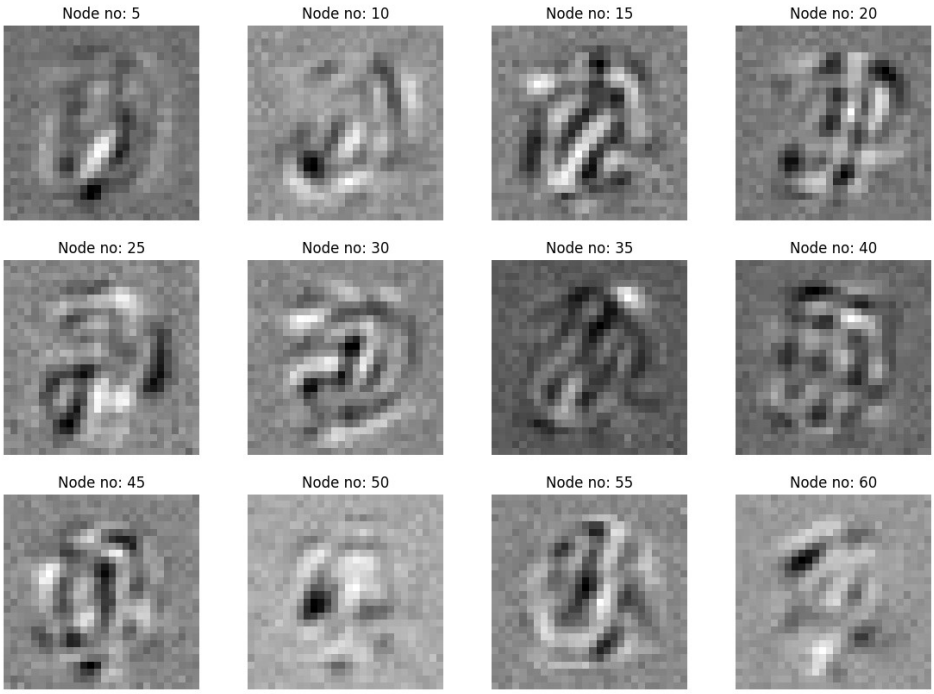
## Task 6 : Weight Visualization




Autoencoder




20% Noisy Input at Denoising Autoencoder




40% Noisy Input at Denoising Autoencoder

Comparing the results of 1 hidden layer Autoencoder and that of denoising autoencoder, we observe that not much meaning patterns are learnt in autoencoder. While denoising autoencoder seems to have detected the pen strokes in images.