CS671: Deep Learning and Applications
Programming Assignment 4

RAUNAV GHOSH(M.Tech-CSP), T22104

PRASHANT DHANANJAY KULKARNI(M.Tech-CSE), T22058

SACHIN BAHULEYAN(M.Tech-CSE), T22060

Contents

1 Theory

1.1 PCA

Principal Component Analysis (PCA) is a technique used in machine learning and data analysis to reduce the dimensionality of a dataset. It works by finding the principal components, which are the directions in the data that have the largest variance. By projecting the data onto these principal components, it is possible to reduce the dimensionality while retaining most of the information.
Using PCA to reduce the dimensionality of a Fully Connected Neural Network (FCNN) can be particularly useful in scenarios where the number of input features is very large. In such cases, the FCNN may suffer from the curse of dimensionality, where the high dimensionality makes it difficult to train the model accurately. By using PCA to reduce the dimensionality, we can reduce the number of input features, which can help to make the FCNN more efficient and effective. Additionally, reducing the dimensionality can also help to prevent overfitting, which occurs when the model becomes too complex and starts to memorize the training data instead of generalizing to new data. Overall, using PCA to reduce the dimensionality of an FCNN can be an effective way to improve its performance and make it more scalable.

1.2 Autoencoder

Autoencoders are a type of neural network that can be used for unsupervised learning tasks such as data compression, feature extraction, and anomaly detection. They work by learning a compressed representation of the input data, and then reconstructing the original data from the compressed representation. The network consists of an encoder, which compresses the input data into a lower-dimensional representation, and a decoder, which reconstructs the original data from the compressed representation. The loss function used during training is usually based on the difference between the original and reconstructed data, with the goal of minimizing this difference. Autoencoders can be used for a variety of applications, such as reducing the dimensionality of high-dimensional data, generating new data that is similar to the input data, or detecting anomalies in the input data. They are a powerful tool for unsupervised learning, as they can learn useful representations of the data without requiring labeled data for training. Figure 1 shows a generic structure of an autoencoder.

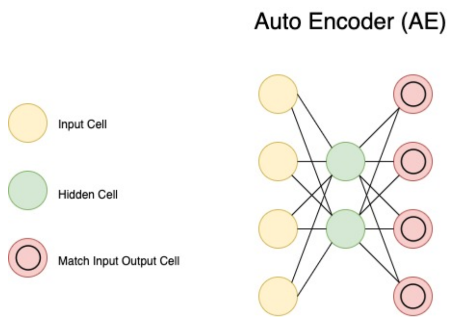


Figure 1: AutoEncoder architechture, Source: https://github.com/kennethleungty/Neural-Network-Architecture-Diagrams/blob/main/Auto%20Encoder%20(AE).jpg

2 Problem statement

This assignment consists of 6 tasks which are as listed below:

1. Task-1: Dimension reduction using PCA.
2. Task-2: Autoencoders for reconstructing the images.
3. Task-3: Classification using the compressed representation from the 1-hidden layer autoencoder.
4. Task-4: Classification using the compressed representation from the 2-hidden layer autoencoder.
5. Task-5: Denoising autoencoders for reconstructing the images.
6. Task-6: Weight visualization.

3 Methodology

3.1 Dataset preprocessing

We were given a subset of *MNIST dataset* for the assignment. Each of the image in the dataset was as grayscale image of 28×28. The dataset for training, testing and validation are summeraised as below,

Set	**Size**	**Labels**
Training	11385	*0, 1, 2, 6, 7*
Testing	3795	*0, 1, 2, 6, 7*
Validation	3795	*0, 1, 2, 6, 7*

3.2 Task-1: Dimension reduction using PCA.

For this task we started with preprocessing the data by computing the mean from the training set and subtracting that from the training, testing and validation set. This centers the dataset about the origin so that we can perform PCA on the sets and project along the orthogonal components.
Next we feed the reduced training set to an FCNN based multiclass classification model for training. For this we tested with two architectures, and the obtained results as given in Table 5.
The best architecture was decided based on the validation accuracy which was of *0.9887*.

3.3 Task-2: Autoencoders for reconstructing the images

For this task we start with designing two neural networks for two autoencoders, one with *one hidden layer* and another with *three hidden layer*. Total of 8 such autoencoder were built with 32, 64, 128 and 256 nodes in the bottleneck layer.

Tables 1, 2, 3, 4 show all the architectures designed for this task. The average reconstruction error of the images from all the the sets are summaried in Table 8.

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	32	*Hyperbolic Tangent*
Output layer	784	*Linear*

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	400	*Hyperbolic Tangent*
Hidden layer 2	32	*Hyperbolic Tangent*
Hidden layer 3	400	*Hyperbolic Tangent*
Output layer	784	*Linear*

Table 1: Auto-encoders with 32 nodes in bottleneck layer

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	64	*Hyperbolic Tangent*
Output layer	784	*Linear*

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	400	*Hyperbolic Tangent*
Hidden layer 2	64	*Hyperbolic Tangent*
Hidden layer 3	400	*Hyperbolic Tangent*
Output layer	784	*Linear*

Table 2: Auto-encoders with 64 nodes in bottleneck layer

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	128	*Hyperbolic Tangent*
Output layer	784	*Linear*

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	400	*Hyperbolic Tangent*
Hidden layer 2	128	*Hyperbolic Tangent*
Hidden layer 3	400	*Hyperbolic Tangent*
Output layer	784	*Linear*

Table 3: Auto-encoders with 128 nodes in bottleneck layer

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	256	*Hyperbolic Tangent*
Output layer	784	*Linear*

Layer	**Nodes**	**Activation function**
Input	784	*Linear*
Hidden layer 1	400	*Hyperbolic Tangent*
Hidden layer 2	256	*Hyperbolic Tangent*
Hidden layer 3	400	*Hyperbolic Tangent*
Output layer	784	*Linear*

Table 4: Auto-encoders with 256 nodes in bottleneck layer

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

Using all the autoencoder designed with different bottleneck layers, we obtain the output of the bottleneck layer corresponding to each of the autoencoder such that we can feed a FCNN based multiclass classifier to classify the data. The reduced representation of each of the encoder acts as non-linear principal component of the data which enables us to make the classifier learn and perform with much lower computational complexity as compared to what would have been in the case of a full dimensional representation.
The architectures tested for the task are summarised in Table 10. The best architecture was chosen based on the validation accuracy which was *0.9855*. The best architecture is given in Table 11.

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

Similar to the autoencoders with one hidden layer, we take the output of the bottleneck layer from the autoencoder with 3 hidden layer and feed the output to FCNN for classification.
The architectures tested for the task are summarised in Table 13. The best architecture was chosen based on the validation accuracy which was *0.9887*. The best architecture is given in Table 14.

3.6 Task-5: Denoising autoencoders for reconstructing the images

For this task, we build train two autoencoders with the same architecture as that of the best performing autoencoder in Task-2. We train these networks with training data corrupted with 20% and 40% noise respectively. The results of the trained denoising autoencoders can be viewed in Table 16 and 17.

3.7 Task-6: Weight visualization

For these we pick random nodes from the hidden bottleneck layer of the autoencoder, and view the image input weights that activates that particular node. The results of can be seen in Figure 2.

4 Results

4.1 Task-1: Dimension reduction using PCA

Classification accuracy on the training set, validation set and testing set for different architectures of FCNN classification model. The hidden layers have *Sigmoidal tanh activation function.*

Reduced dimension	Architecture	Train acc.	Validation acc.	Test acc.
32	[32,16,8,5]	0.9987	0.9816	0.9783
	[32,256,128,64,32,5]	0.9959	0.9823	0.9823
64	[64,16,8,5]	0.9989	0.9821	0.9757
	[64,256,128,64,32,5]	0.9994	0.9887	0.9857
128	[128,16,8,5]	0.9998	0.9829	0.9760
	[128,256,128,64,32,5]	0.9971	0.9826	0.9812
256	[256,16,8,5]	0.9996	0.9771	0.9754
	[256,256,128,64,32,5]	0.9976	0.9831	0.9805

Table 5: Summary of classifiers trained using PCA reduced data as input

Best architecture
The accuracy and confusion matrix on the test data for the best architecture selected based on validation accuracy

Layer	Nodes	Activation function
Input	64	*Linear*
Hidden layer 1	256	*Sigmoidal Hyperbolic Tangent*
Hidden layer 2	128	*Sigmoidal Hyperbolic Tangent*
Hidden layer 3	64	*Sigmoidal Hyperbolic Tangent*
Hidden layer 4	32	*Sigmoidal Hyperbolic Tangent*
Output layer	5	*Softmax*

Table 6: Best architecture

	Pred *0*	**Pred 1**	**Pred 2**	**Pred 6**	**Pred 7**
Actual 0	751	1	3	2	2
Actual 1	1	752	3	1	2
Actual 2	6	3	740	1	9
Actual 6	6	2	4	747	0
Actual 7	0	2	6	0	751

Table 7: Confusion matrix on test dataset of best architecture

4.2 Task-2: Autoencoders for reconstructing the images

Autoencoders for reconstructing the images:

Bottleneck	**1 Hidden Layer Average reconstruction Error**			**3 Hidden Layer Average reconstruction Error**		
	Train data	**Val. data**	**Test Data**	**Train data**	**Val. data**	**Test Data**
32	0.01503	0.01527	0.01505	0.01047	0.01099	0.01083
64	0.00850	0.00871	0.00858	0.00838	0.00861	0.00849
128	0.00428	0.00447	0.00441	0.00471	0.00490	0.00484
256	0.00169	0.00182	0.00180	0.00231	0.00248	0.00245

Table 8: Average reconstruction error for autoencoders

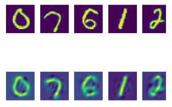
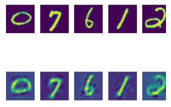
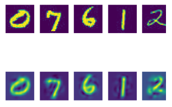
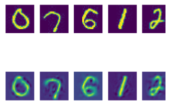
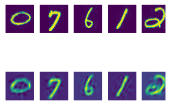
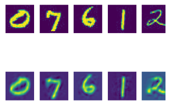
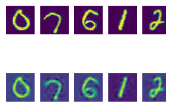
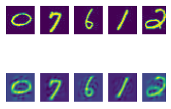
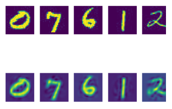
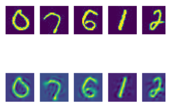
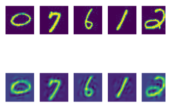
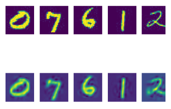
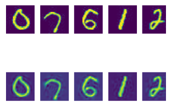
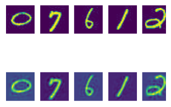
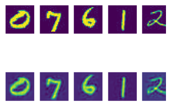
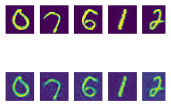
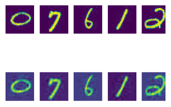
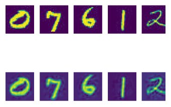
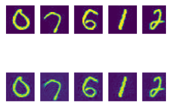
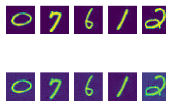
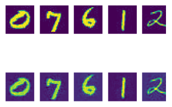
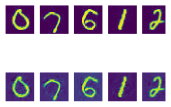
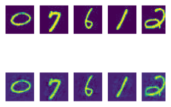
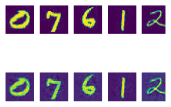
Bottleneck	**1 Hidden Layer Reconstruction**			**3 Hidden Layer Reconstruction**		
	Train data	**Val. data**	**Test Data**	**Train data**	**Val. data**	**Test Data**
32						
64						
128						
256						

Table 9: Reconstruction of images using Autoencoders with different nodes in bottleneck layer. Image on the top are the input to the autoencoder, the images below are corresponding reconstructed images

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

Classification using the compressed representation from the 1-hidden layer autoencoder

Reduced dimension	**Architecture**	**Train acc.**	**Validation acc.**	**Test acc.**
32	[32,16,8,5]	0.9999	0.9802	0.9792
	[32,256,128,64,32,5]	1.0000	0.9868	0.9842
64	[64,16,8,5]	0.9997	0.9829	0.9821
	[64,256,128,64,32,5]	0.9979	0.9842	0.9837
128	[128,16,8,5]	0.9997	0.9837	0.9787
	[128,256,128,64,32,5]	1.0000	0.9847	0.9829
256	[256,16,8,5]	1.0000	0.9826	0.9779
	[256,256,128,64,32,5]	1.0000	0.9866	0.9850

Table 10: Summary of classifiers trained using encoded data as input

Best architecture

The accuracy and confusion matrix on the test data for the best architecture selected based on validation accuracy

Layer	**Nodes**	**Activation function**
Input	256	*Linear*
Hidden layer 1	256	*Sigmoidal Hyperbolic Tangent*
Hidden layer 2	128	*Sigmoidal Hyperbolic Tangent*
Hidden layer 3	64	*Sigmoidal Hyperbolic Tangent*
Hidden layer 4	32	*Sigmoidal Hyperbolic Tangent*
Output layer	5	*Softmax*

Table 11: Best architecture-using encoded input

Accuracy on the test dataset of the best architecture: *0.9847*

	Pred 0	**Pred 1**	**Pred 2**	**Pred 6**	**Pred 7**
Actual 0	749	1	2	4	3
Actual 1	0	753	3	1	2
Actual 2	10	4	737	4	4
Actual 6	6	1	2	749	1
Actual 7	2	3	3	1	750

Table 12: Confusion matrix for test dataset of best architecture

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

Classification using the compressed representation from the 3-hidden layer autoencoder

Reduced dimension	Architecture	Train acc.	Validation acc.	Test acc.
32	[32,16,8,5]	0.9997	0.9821	0.9802
	[32,256,128,64,32,5]	0.9982	0.9831	0.9823
64	[64,16,8,5]	1.0000	0.9808	0.9816
	[64,256,128,64,32,5]	0.9971	0.9810	0.9805
128	[128,16,8,5]	0.9999	0.9829	0.9768
	[128,256,128,64,32,5]	1.0000	0.9887	0.9823
256	[256,16,8,5]	0.9999	0.9794	0.9758
	[256,256,128,64,32,5]	0.9980	0.9855	0.9787

Table 13: Summary of classifiers trained using encoded data as input

Best architecture

The accuracy and confusion matrix on the test data for the best architecture selected based on validation accuracy

Layer	Nodes	Activation function
Input	128	*Linear*
Hidden layer 1	256	*Sigmoidal Hyperbolic Tangent*
Hidden layer 2	128	*Sigmoidal Hyperbolic Tangent*
Hidden layer 3	64	*Sigmoidal Hyperbolic Tangent*
Hidden layer 4	32	*Sigmoidal Hyperbolic Tangent*
Output layer	5	*Softmax*

Table 14: Best architecture-using encoded input

Accuracy on the test dataset of the best architecture: *0.9887*

	Pred 0	**Pred 1**	**Pred 2**	**Pred 6**	**Pred 7**
Actual 0	747	2	4	5	1
Actual 1	1	753	4	1	0
Actual 2	11	6	723	5	14
Actual 6	5	2	1	749	2
Actual 7	1	5	7	2	744

Table 15: Confusion matrix for test dataset of best architecture

4.5 Task-5: Denoising autoencoders for reconstructing the images

Results for denoising autoencoder with one hidden layer of *128* nodes,

	20% noise	**40% noise**
Training data	0.00724	0.01431
Validation data	0.00737	0.01441
Testing data	0.00742	0.01448

Table 16: Average reconstruction error for data from each set

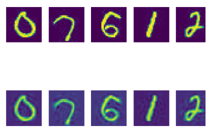
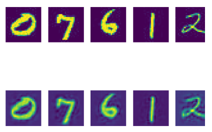
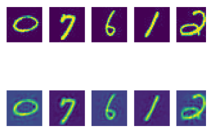
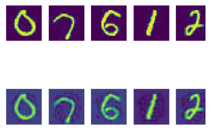
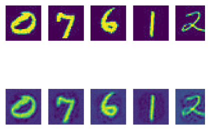
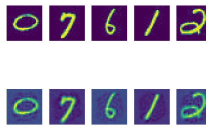
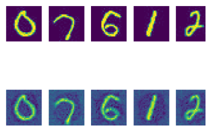
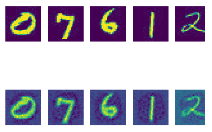
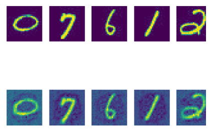
Noise	**Training set**	**Validation set**	**Testing set**
0% Noise			
20% Noise			
40% Noise			

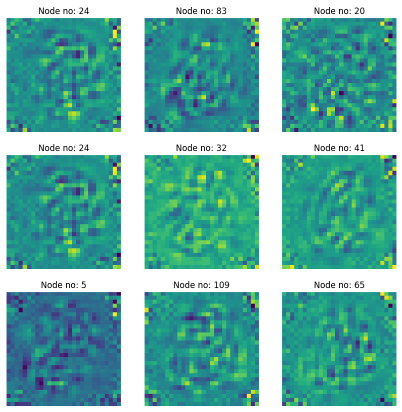
Table 17: Reconstruction of images using denoising auto-encoder

FCNN- We an FCNNs with architecture given in Table 18

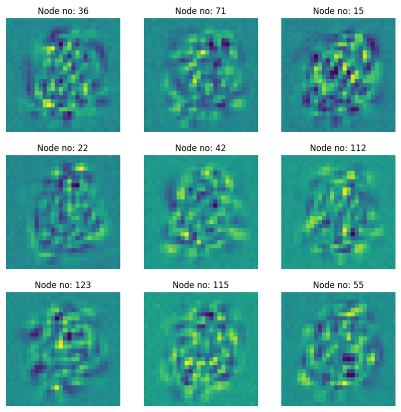
Reduced dimension	**Architecture**	**Train acc.**	**Validation acc.**	**Test acc.**
20% noise	[128,256,128,64,32,5]	1.0000	0.9866	0.9837
40% noise	[128,256,128,64,32,5]	1.0000	0.9866	0.9823

Table 18: Summary of classifiers trained using encoded data as input

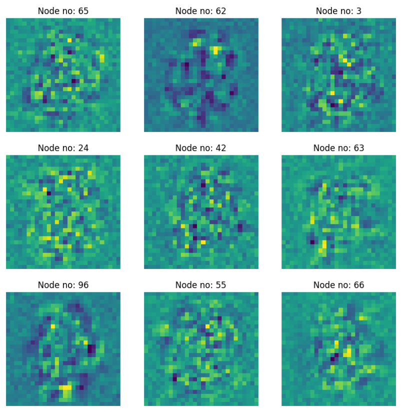
4.6 Task-6: Weight visualization



(a) Autoencoder-hidden layer with 128 nodes



(b) Denoiseing autoencoder(20% noise)



(c) Denoiseing autoencoder(40% noise)

Figure 2: Inputs that activate the given nodes in the hidden layer

5 Inferences

- Using PCA we reduce the dimension of the input data space which makes the operations less computationally expensive with negligible tradeoff with accuracy. This was observed in Task 1 where we used the same architecture from the previous assignment where the input to the FCNN is as 784 dimensional vector, while we achieved similar accuracy with using only 32, 64, 128 and 256 dimensions. We tested with a smaller, less complex architecture which gave similar accuracy, suggesting that the data with principal components only is enough to classify the overall dataset.
- In case of autoencoders, as the number of neurons in bottleneck layer increases, the average reconstruction error on any dataset decreases. This is because, using more neurons is analogous to learning more components from the input data, which gives us better reconstruction.
- Observing the two autoencoder architecture, one hidden layer and three hidden layer, we observe that if lower number of neurons are kept in the bottleneck layer, the three hidden layer model gives a better reconstruction error compared to one hidden layer. Whereas increasing the number of nodes gives better results for one hidden layer compared to three hidden layer.
- Building classifiers using compressed representation from autoencoders are done in Task 3 and 4. Here we get similar observations as was in Task 1 that increasing the number of bottleneck node, i.e., increasing the dimension of representation gives us better performance of the network while trading-off the number of

training parameters for training. Even though we get very similar accuracies from the networks with different input dimension, there is clear increase in accuracy where increasing the dimension of input.

- Comparing the best model in Assignment 3, Task 1, Task 3 and Task 4, we can see that using full representation and using reduced representation has very similar performance. But using reduced representation gives us the advantage of reducing the computation required for training the network. This can be advantageous where computation power is limited.

- For the denoising autoencoders, we observe that the reconstruction error is directly proportional to the amount of noise added to the training data. Therefore, we need to add noise within a threshold amount such that the autoencoder learns the necessary features but also doesn't learn the noise themselves.

- Denoising autoencoder is used to remove noise from data, making it better for analysis. By removing noise from inputs, the model is able to better focus on the relevant features and learn from them more efficiently. This helps reduce the risk of overfitting, which can lead to poor performance when applied to new data. As can be seen in Task 5 and 6, the denoising autoencoder learns to activate the nodes in the hidden layer for certain strokes only. But putting too much noise in the training data leads to the network learning the noise as well. This can be seen in the inputs that activate the nodes for 20% noise denoising autoencoder and 40% nosie denoising autoencoder.

- Visualizing the inputs that activate the nodes of the hidden layer shows that, autoencoders learn the pen strokes. The network is optimized to minimize the difference between the original input image and its reconstructed output. This encourages the network to learn to represent the input image as a set of pen strokes in the latent space, which can be used to reconstruct the image with minimal error.

- In the case of denoising autoencoders, the network is trained to reconstruct the original image from a noisy version of the input image. By learning to remove noise from the input image, the network is forced to learn the underlying pen strokes that are essential to reconstructing the original image.