# Auto Encoders

EE6132 - Lab 3

Anirudh R

November 20, 2021

## Contents

## 1 Introduction

In this assignment, we look to analyse Auto Encoders. We will be using the MNIST dataset to train the Auto Encoder models. We will use the PyTorch library to build, train and test the models. We will also perform Principal Component Analysis to compare it with the results of an Auto Encoder model. We will also build Sparse, De-noising and Convolutional Auto Encoders to compare their performance.

The following table shows the **Standard Configuration** of Auto Encoder we will look at.

| **Parameter** | **Value** |
|---|---|
| Dataset | MNIST |
| Loss Function | Mean Squared Error |
| Optimizer | Adam |
| Mini-Batch Size | 100 |
| No. of Epochs | 8 |
| No. of Hidden Layers | 1 |
| No. of MLP Layers | 2 (1 hidden & 1 output) |
| FC layer sizes | 256 (or 64 or 128), 784 |
| Hidden Layer Activation | ReLU |
| Output Layer Activation | ReLU |

## 2 Concepts

### 2.1 Dataset

We make use of the **MNIST**(Modified National Institute of Standards and Technology) Dataset here. This is a dataset of hand-written digits and the corresponding labels. This contains **60,000** training data points and **10,000** test data points. The digits have been size-normalized and centered in a fixed-size image (28×28 grayscale image).

### 2.2 Auto Encoders (AE)

In the previous assignments, we saw that feature extraction from raw image data can be very advantageous in classification and other machine learning tasks. Extracting the core underlying features out of raw data can be useful in training more efficient/leaner networks which will have to only process lesser dimensional input.

The Auto Encoder architecture consists of 2 major parts, **Encoder** & **Decoder**. The encoder and decoder can be made up of Multi-Layer Perceptron(MLP) or Convolutional layers. The working of an AE is as follows:

- The encoder takes in input data, **X** and produces an encoded representation, **h**.
- The decoder then uses the encoded representation, **h** alone to reconstruct an estimation of the original input. This estimate/output produced is called **Xhat**.

So, the whole training of an AE will only involve the input data. There is no class label etc. that will be used in the training. So, this is an **Unsupervised Technique**.

The Auto Encoder architecture and Mean Squared Error loss function make it seem to be a rather complicated neural network to just get an identity mapping. Even though this

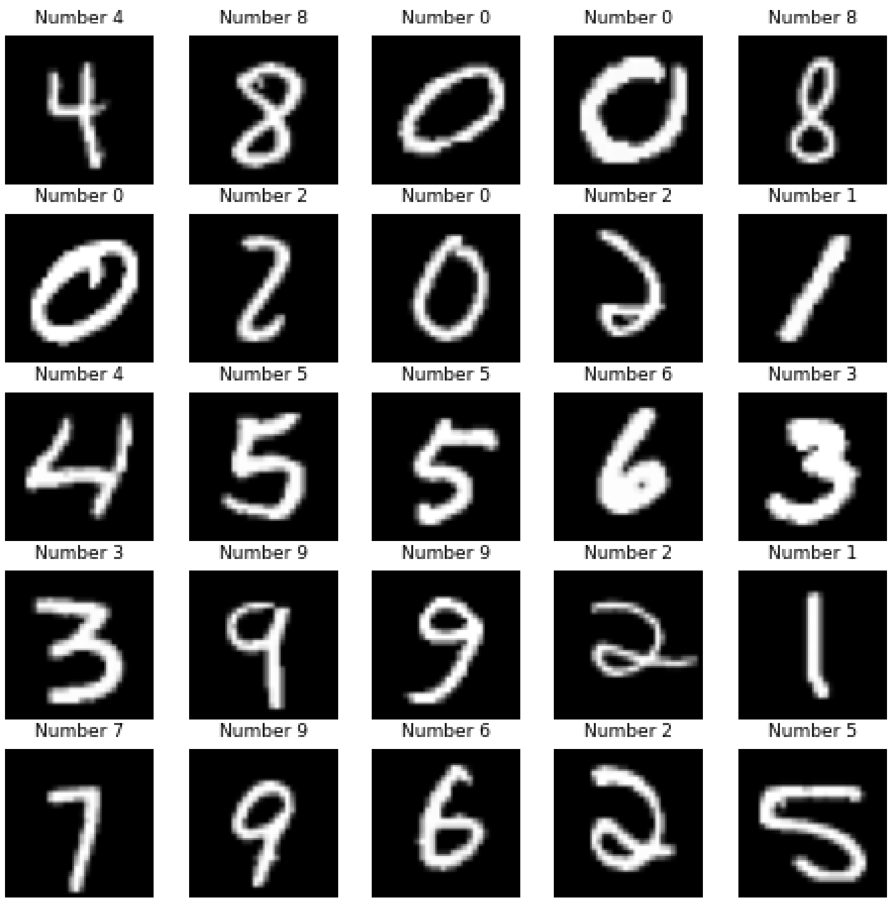

Figure 1: Samples from the MNIST Dataset

may seem to be the external appearance, with regularization and/or restricting size of **h** appropriately, we can make the AE learn fundamental underlying features in the data instead of just learning an identity mapping. As per the **Manifold Hypothesis**, any structured data set in the high dimensional space will likely lie on some low dimensional manifold in the high dimensional space. So, as part of the training, we want the AE to learn the manifold that represents the data set.

So, AE can lead to very potent, compact representations(as **h**) of the data that we can use for other purposes. The decoder on its own can also be used as some sort of *generator*. *Eg.* If we train an AE on image data, by passing different values of **h** to the decoder, we will be producing different estimates.

But, an important point to note is that this endeavour to produce a potent representation of the data will be successful only if we make the network struggle to reconstruct the data. This will force the network to learn the most fundamental features in the data and the manifold represented by the data. If we make it too easy for it to reconstruct, it will just end up emulating the identity mapping which is useless.

## 2.3 PCA & AE

The **Principal Component Analysis(PCA)** is a *linear, data-dependent* way of analyzing data and potentially producing more compact representations of the data set in hand. This is a widely used linear dimensionality reduction technique.

Auto Encoders also usually help in providing much lower dimensional representations of data sets. But, Auto Encoders with their non-linear activations are a non-linear data-dependent dimensionality reduction technique. An interesting connection between PCA and AE is that if we train a **linear** AE (with no non-linear activations) with **Mean Squared Error loss**, we will end up with an AE that exactly represents performing PCA on the data set.

## 2.4 Types of Auto Encoders

There are various types of Auto Encoders based on structure, regularization used etc. If we use a more implicit regularization technique such as adding noise to input while training, we call it a **De-noising Auto Encoder**. **Sparse Auto Encoders**, on the other hand, explicitly add a regularization term to the loss while training to force sparsity. This could be an L1 norm calculated on $\mathbf{h}$ that is added to the Mean Squared Error loss. When convolutional layers are used to make the AE, we have a **Convolutional Auto Encoder**.

## 3 Experiments & Observations

### 3.1 Comparing PCA & Auto Encoders

The architecture mentioned in the assignment sheet was trained for a total of 12 epochs. The following reconstruction error was obtained for the Auto Encoder and PCA (with 30 eigenvalues):

- **PCA:**
  - Train Data: 14.47081, Test Data: 14.12896
- **AE:**
  - Train Data: 15.06312, Test Data: 15.24986

We see that the Reconstruction errors on the Train and Test data of PCA and AE are pretty close but the PCA seems to be performing slightly better here (as of 12 epochs of training of the AE).

Considering that we allow only 30 values in the encoded representation for both PCA & AE, any difference in performance isn't affected by this. We are using non-linear activation for AE which should allow it to perform better than the linear method, PCA. Perhaps, the lower reconstruction performance of AE is due to not enough epochs of training. It may also just be the case that throughout the network, the pre-activation rarely goes $< 0$. This would mean that the non-linear ReLU activation functions basically as just a linear activation. A linear activation with MSE loss for the AE would explain how both PCA and AE have very similar performance (if we attribute the difference in performance purely to not enough training of AE).

### 3.2 Experimenting with hidden units of varying sizes

The Standard Auto Encoders were trained for 8 epochs.

**Passing a Test Image:** The image shows the original MNIST image on the first column and the reconstructed images formed by each of the 3 configurations on the second column. We see from the captions that the 256 neuron hidden layer AE performed the best while the 64 neuron hidden layer AE performed the worst. The trend among these 3 sizes seems to be that, *the more the hidden neurons, better the reconstruction performance.*

We also see that there are missing pixels in the reconstructed images and the number of missing pixels reduces with increase in number of hidden neurons. So, overall, the quality of reconstruction improves in terms of missing pixels and the error with increase in hidden layer size.

**Passing a Non-Digit/Random Noise Image:** Here, again, the reconstruction performance follows the same trend for the 3 hidden layer sizes. The output here is not a reproduction of the random noise input at all(shown by the huge error values). The output is mostly blacked out near the edges and only has non-zero value near the centre. This is perhaps because of how the non-zero pixels in our data set was always in the centre and pixels were zeroed out near the edges.

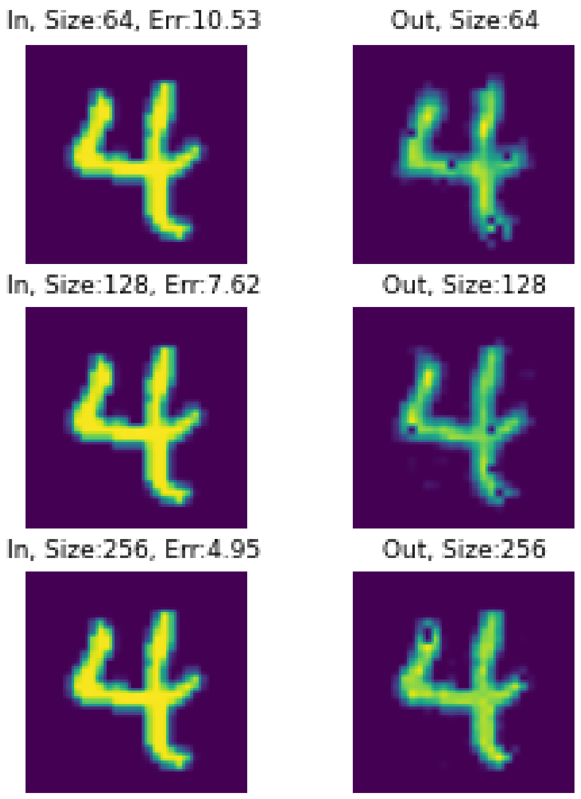

Figure 2: Test Image passed through each of the standard AE configurations

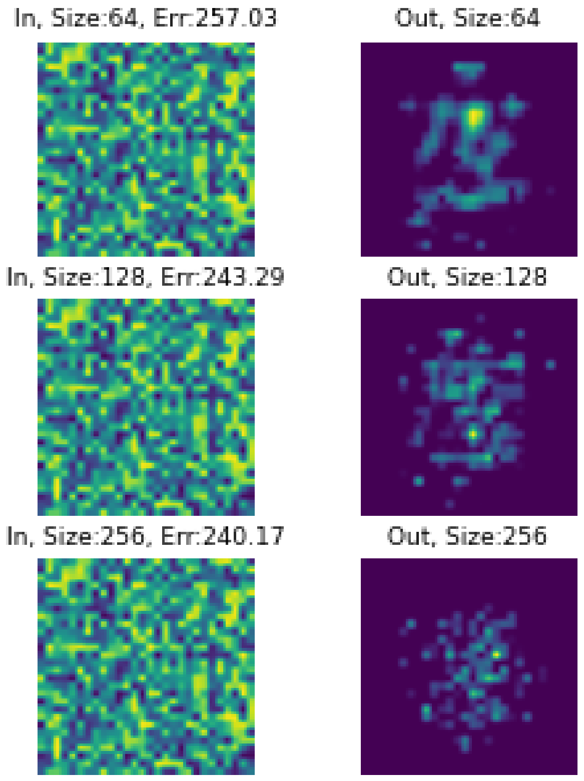

Figure 3: Random Noise passed through each of the standard AE configurations

The important point to note is that the AE works to reproduce the input at the output when we pass images from the data set used to train the AE. When input images that don't resemble the data set are passed, the AE does a bad job at reproducing the input at the output. This shows how the encoding and decoding are very data set specific. The AE will not lead to good reconstruction for any input.

## 3.3 Sparse Auto Encoders

An over-complete Auto Encoder with 1 hidden (1296 neurons) and 1 output layer was used here. The L1 norm of **h** regularization term was added to the MSE loss during training to force sparsity in the encoded representation. It is important to choose the regularization parameter with care as adding too much of the L1 norm term to the loss will lead to the network significantly compromising on reconstruction to ensure sparsity of encoded representation. Here, the regularization parameter was chosen to be 0.1.

**Average Hidden Layer Activation:** The standard AE mentioned in the table here is the 256 hidden neuron variant. Moving forward, standard AE will always refer to the 256 hidden neuron variant.

| Parameter | Value |
|---|---|
| Average L1 norm of hidden layer activation(sparse) | 17.49965 |
| Average L1 norm of hidden layer activation(standard) | 188.67882 |
| Average L1 norm of hidden layer activation(sparse) /neuron | 0.01350 |
| Average L1 norm of hidden layer activation(standard) /neuron | 0.73702 |

Here, we see that, even though the Sparse AE has more neurons in the hidden layer (**about 5x**), average L1 norm of **h** is more than **11 times lower** compared to the standard AE. This is due to the sparsity inducing regularization term we added to the loss while training. Since this term wasn't used in the standard AE, the training didn't force sparsity of **h** like training did in this sparse AE. The difference is much more drastic on a per neuron basis **(about 56x)**.

**Visualizing Filters:** Now, we visualize the weights of the hidden layer as images by reshaping them. If the network has N hidden neurons, then the hidden layer weights are of dimension Nx784. This is a basic shape difference between weights of the 2 networks. Here, we take rows of the weights and plot it as 28x28 images.

Of the visualized weights, most of standard AE's filters seem to be just random noise near the centre and smudged out near the edges of the image. The filters of sparse AE seem to vary much more. Filters 1 and 2 look like just random noise. Filter 3 seems fully smudged out but there seems to be some pattern in the filter(looks like a 7). Filter 4 clearly has 2 in it. Filter 5 too has some pattern on it. Overall, the filters seem to be much more uniform for the standard case while the filters vary more for sparse and also have patterns on them occasionally.

## 3.4 De-noising Auto Encoders

Here, we just use the same architecture as the Standard AE and train it with implicit regularization (using noise added input while training).

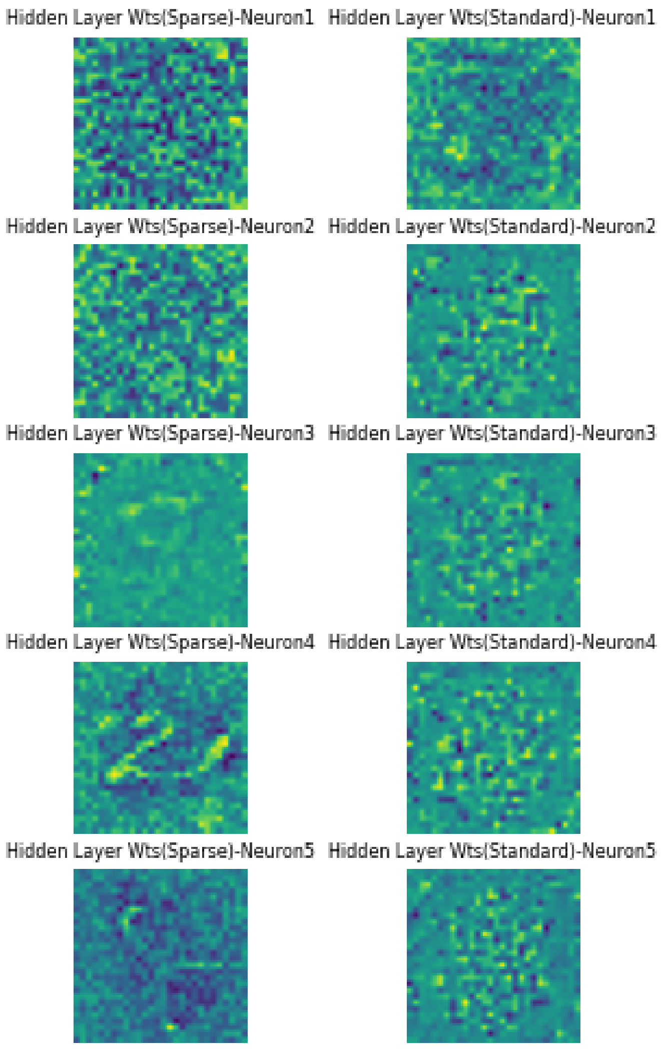

Figure 4: Weights of the Hidden Layer of Standard and Sparse AEs

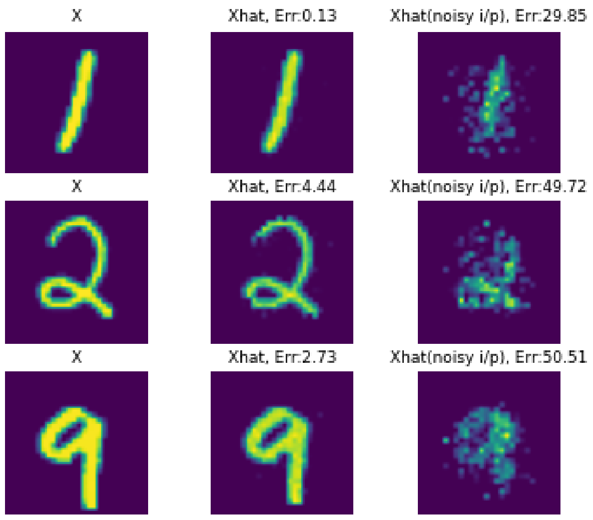

Figure 5: Passing Corrupted Images to Standard AE

**Passing Corrupted Images to Standard AE:** Here, the first column shows the original image and the second column shows the image reconstructed by the network when passed without noise. The third column shows the output of the network when the input image is corrupted with noise.

We see from the error values that, once we add the noise to input, reconstruction becomes really bad. So, the ability to reconstruct of the standard AE is pretty sensitive to noise added to input image. But, overall, we are sort of still able to discern what number the input image might have been even with the noisy input reconstruction.

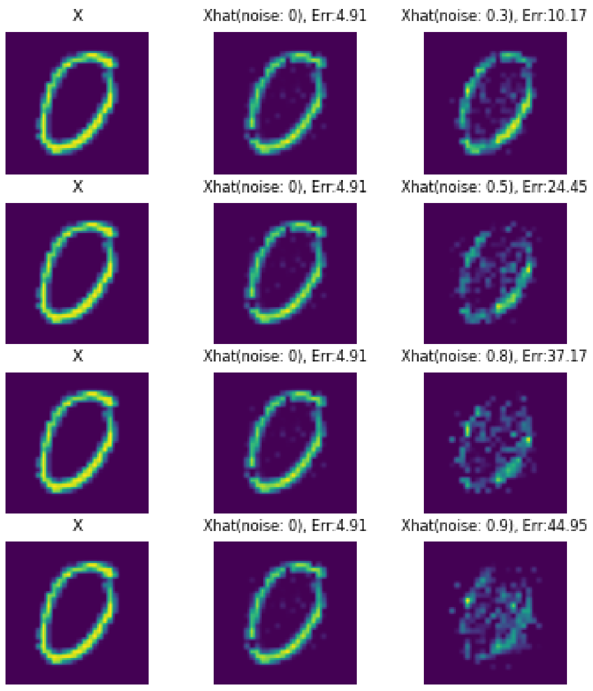

Figure 6: Passing Corrupted Images of Varied Noise Values to Standard AE

**Variation of Noise Levels:** Obviously, just as the previous case, the reconstruction error increases when noise is added. We also see that more the noise that is added, higher the reconstruction error and lower the reconstruction performance. But, overall, we are sort of still able to discern what number the input image might have been for the first 2 cases and maybe even the 3rd. But, not for the case of 0.9 noise. For the 4th case, the reconstruction could have even been 7 or 2 etc.

**Visualizing Filters:** Now, we visualize the weights of the hidden layer as images by reshaping them. The hidden layer weights are of dimension 256x784 for both AEs. So, here, we have no shape difference like we had for sparse AE. Here, we take rows of the weights and plot it as 28x28 images.

Of the visualized weights, most of standard AE's filters seem to be just random noise near the centre and smudged out near the edges of the image. Filters 1,2 and 5 of the de-noising AE seem to be similar to this. But, the smudging near the edges is much more *smoother* and the variation of pixel values only seem to happen near the centre (as in, near the edges, pixels all seem to be of same value). Filters 3 and 4 of de-noising AE seem to be just random noise throughout with no smudging near the edges. Also, the

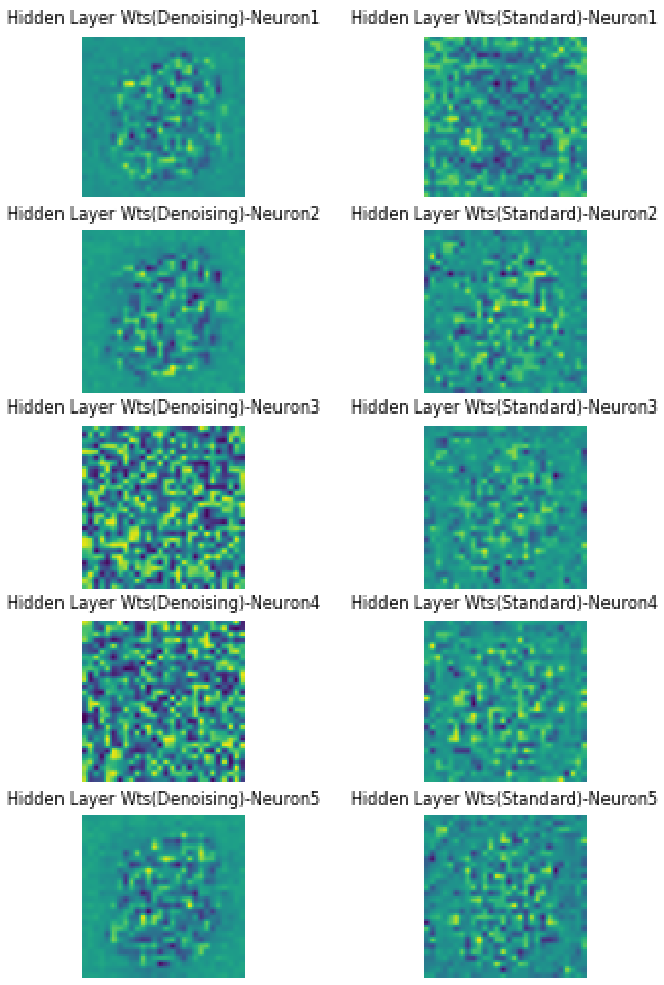

Figure 7: Visualizing De-noising Auto Encoder Filters

contrast between the colours of these 2 filters is also high which means the values held are either close to higher/lower limit and not much in the middle. The filters of the Standard AE, in general, seem to be much more smoother overall compared to filters 3 and 4 of de-noising AE.

### 3.5 Manifold Learning:

**Adding Noise to Images:** Here, we add noise to images and see how they look. A 28x28 image is nothing but a point in the 784 dimensional space and adding noise to it is simply like moving some distance in a random direction to a new point in the 784 dimensional space.

Here, we think of a "valid" image as an image that looks like it is part of the data set. Any image that doesn't *look* like it is from the data set will not be considered valid even if the number written in the image is still discernible.

We see that even though the digits are discernible in the 2nd column, they definitely don't look like images from our data set and therefore aren't valid images. This is because according to **Manifold Hypothesis**, our data set will form a low dimensional manifold in the high dimensional(784 dimensional) space[Refer here].

Any image of the data set will be on this manifold. When we add a random noise to

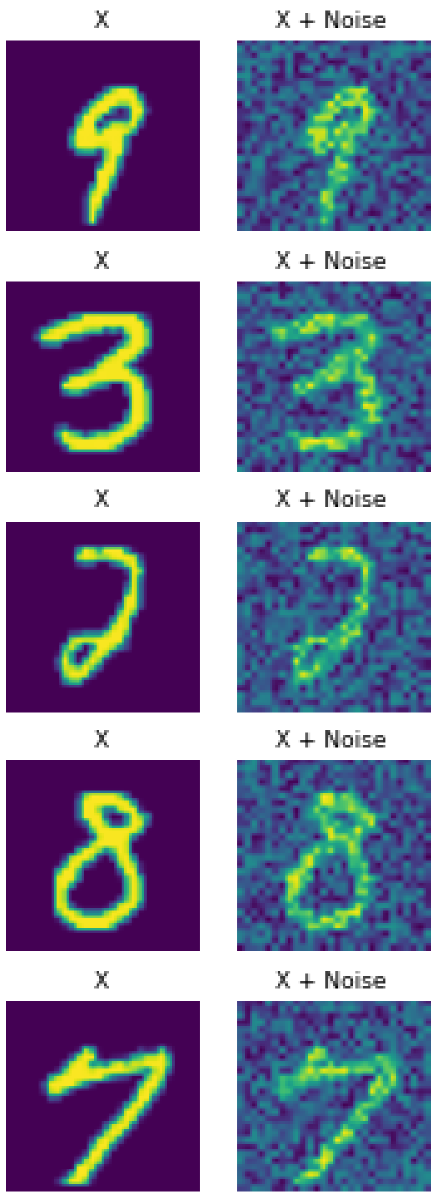

Figure 8: Images alongside Images+Noise

the image, the random noise is very likely to take us out of the manifold as the manifold is a fairly low dimensional structure in the 784 dimensional space. The chances that we end up adding noise that lands us again on the manifold is low. This leads to us getting non-valid images when adding noise to our images.

**Training and Testing AE:** Here, due to the very small latent representation that we allow to the network, the direct reconstruction itself is not great. We end up getting a significant error. When noise is added to the latent representation and then reconstructed, we see that the recognizability of the number has reduced but the error hasn't changed by a lot considering the significant error that we had in the pure reconstruction.

This is because the hidden representation is a very low dimensional representation of the data that comes close to representing the actual low dimensional manifold that the data set resides on. So, when we add noise to this latent representation, all we

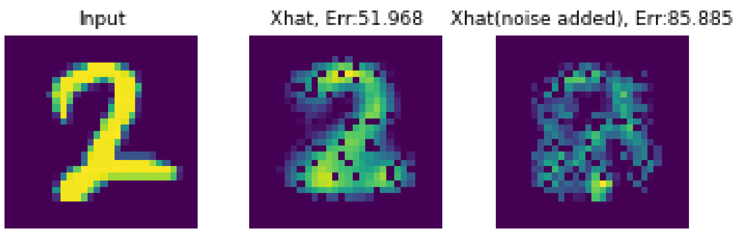

Figure 9: Manifold Learning

end up doing is slightly moving **on** the manifold. We don't go out of the manifold as easily as we did before. This is assuming we actually managed to learn the exact low dimensional manifold of this data set. This may not even be completely possible if the manifold is more than 8 dimensional. In that case, the learned manifold would only be an approximation(which perhaps explains the bad reconstruction error).

**An example:** Suppose we have a 3 dimensional data set that only contains points on the x axis. We see that the low dimensional manifold we are looking for here is nothing but the 1 dimensional x axis. In the 3 dimensional space, when we add noise, we are very likely to not end up on the x axis. But, if we manage to get a 1 dimensional latent representation to represent the x axis value of the data set, then, adding random noise to this latent representation will only again land us on the x axis (which is within the manifold).

All of this reasoning is of course assuming that the AE has actually managed to learn the manifold instead of just modelling an identity mapping.

## 3.6 Convolutional Auto Encoders

For this, 3 different convolutional networks were trained with different methods of up-sampling used in the decoder(namely De-Convolutional+MaxUnpooling, De-Convolutional and MaxUnpooling). The exact architectures can be looked at in the code 4. All networks were trained for a total of 8 epochs. All other parameters used for the 3 networks are the same too.

**Reconstruction Error:**

- **Deconv+MaxUnpool:**
  - Train Data: 2.43547, Test Data: 2.42588
- **MaxUnpool:**
  - Train Data: 1.91074, Test Data: 1.89393
- **DeConv:**
  - Train Data: 8.06247, Test Data: 7.82189

Overall, we see that MaxUnpool performs better than MaxUnpool+DeConv which performs better than DeConv for the same number of epochs of training.

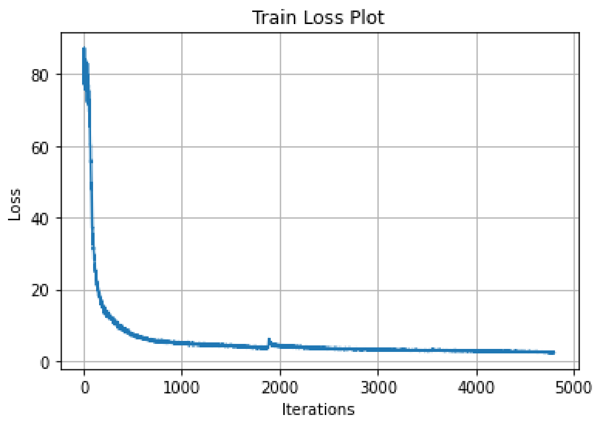

Figure 10: DeConv+MaxUnpool Train Loss

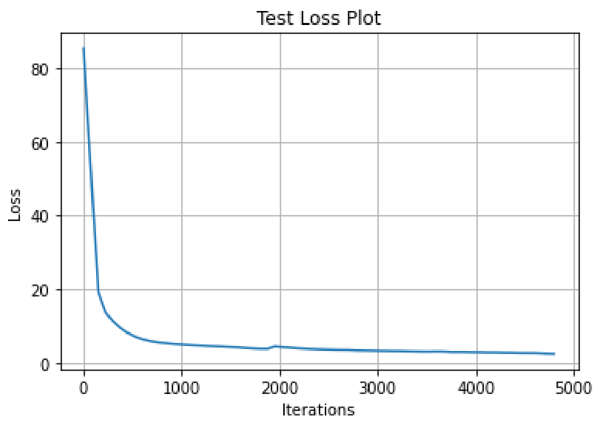

Figure 11: DeConv+MaxUnpool Test Loss

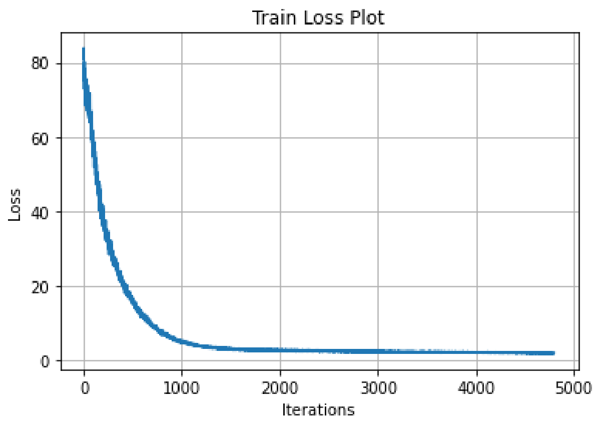

Figure 12: MaxUnpool Train Loss

**Convergence:** We plot the Training and Test loss plots for the 3 networks.

- The loss plots for DeConv alone seems to be not smooth. The rest 2 cases have a smooth loss transition.
- The loss takes a sharp dip initially but then the rate of decrease slows down abruptly in DeConv.

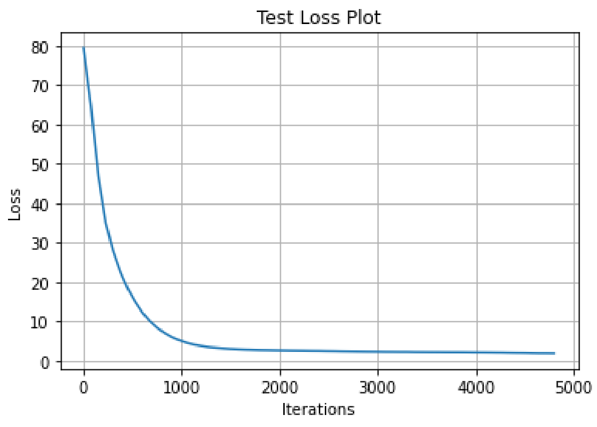

Figure 13: MaxUnpool Test Loss

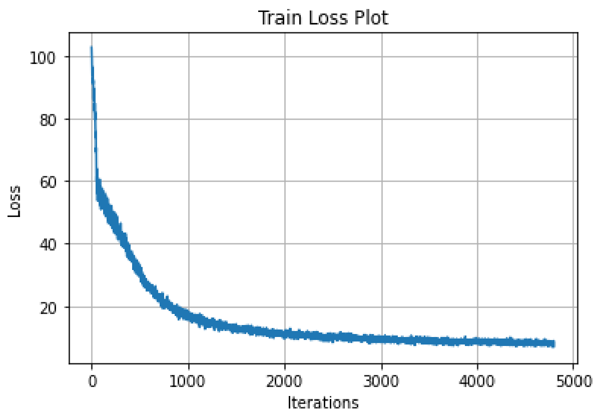

Figure 14: DeConv Train Loss

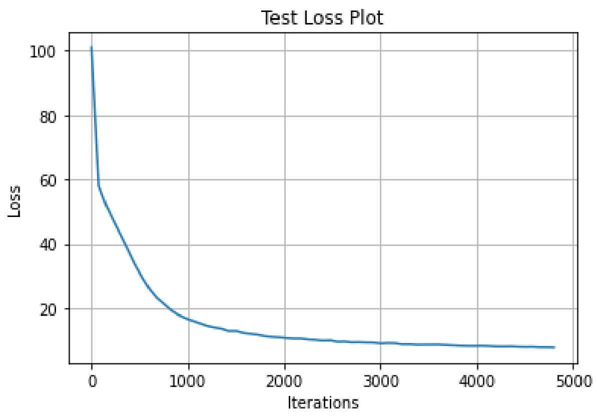

Figure 15: DeConv Test Loss

- Overall, at 1000 iterations, DeConv+MaxUnpool and MaxUnpool seem to have reached the same loss but DeConv still has a higher loss.
- Within 1000 iterations, DeConv+MaxUnpool falls much more rapidly initially compared to MaxUnpool. So, if we want to get to lowest loss in a very limited number of iterations, we can use DeConv+MaxUnpool.
- At the end, with convergence, we already saw that MaxUnpool performs the best.

- In terms of both loss transition initially and final performance, DeConv seems to perform the worst. DeConv+MaxUnpool performs better initially within 1000 iterations but MaxUnpool performs better at the end.

**Decoder Weights:** The decoder weights have not been plotted for the MaxUnpool case as it has no learnable parameters.

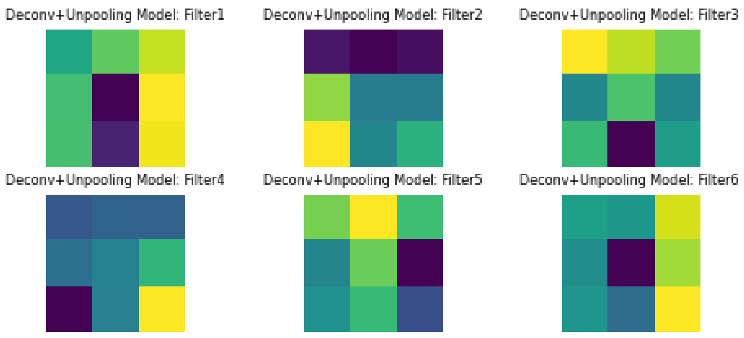

Figure 16: DeConv+MaxUnpool Filters

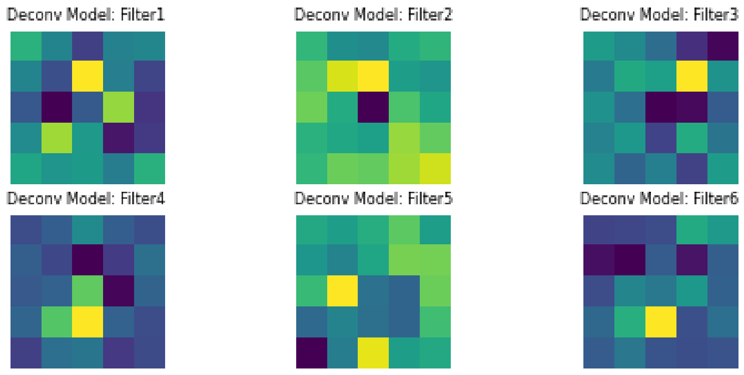

Figure 17: DeConv Filters

There doesn't seem to be any well recognizable pattern in the filters. One point to note is that the filters are of size 3x3 and 5x5 for the 2 cases. This is due to the decoder architecture chosen.

## 4 Code

The implementation of all the experiments used in this report can be found in the following colab notebook: Link

## 5 Conclusion

We have thus built various AE models and analyzed them & their connection to manifolds and PCA.