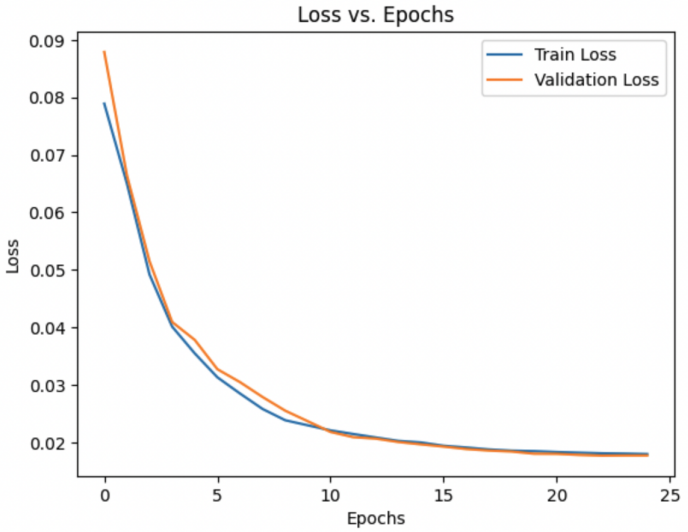
1. PCA vs. Autoencoders

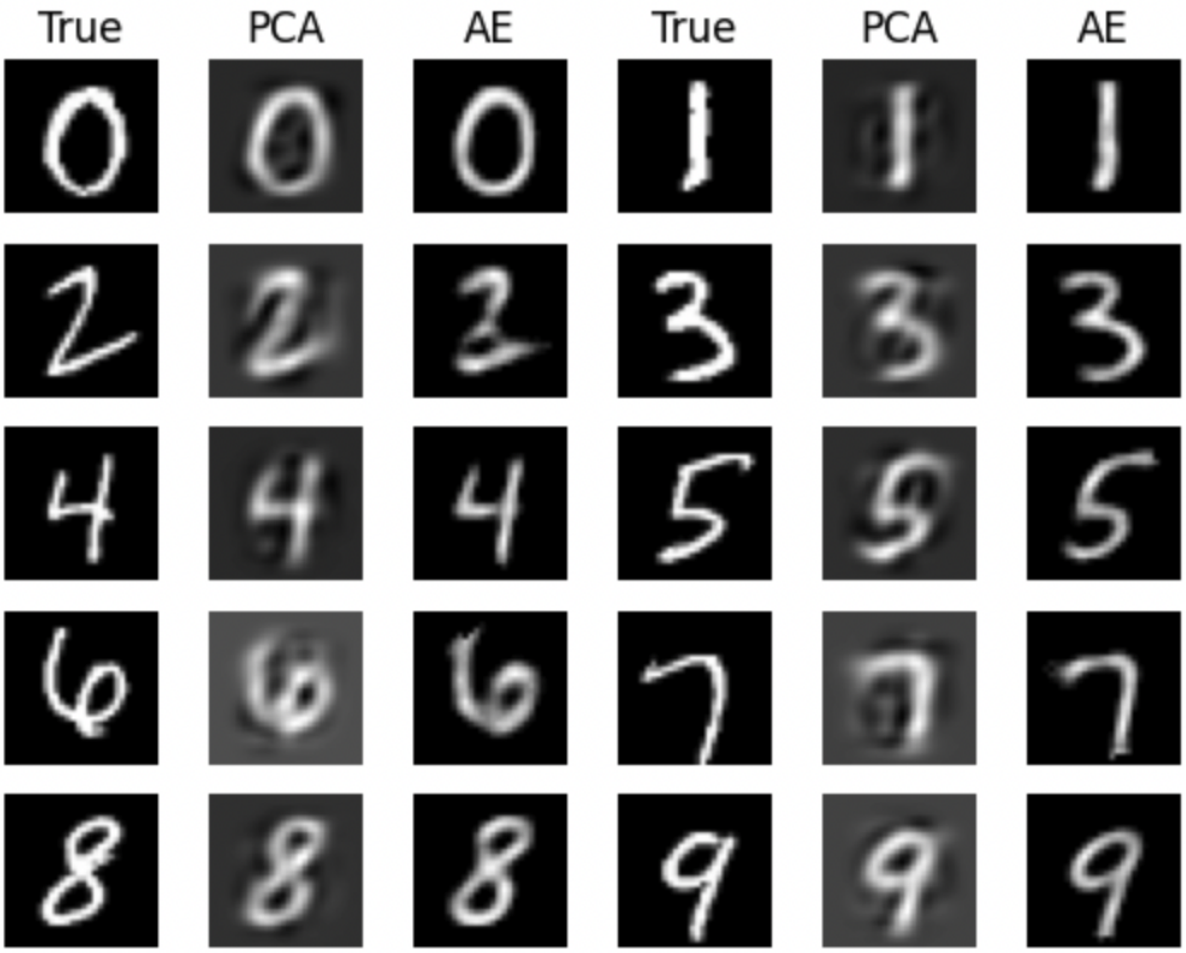
Reconstruction Accuracies

- Using PCA, got a reconstruction accuracy(MSE) of 0.018 for test data.
- Using AE, got a reconstruction accuracy(MSE) of 0.0177 for test data.
- Both the reconstruction accuracies are similar, with AE holding a slight edge.

Loss vs Epochs Plot

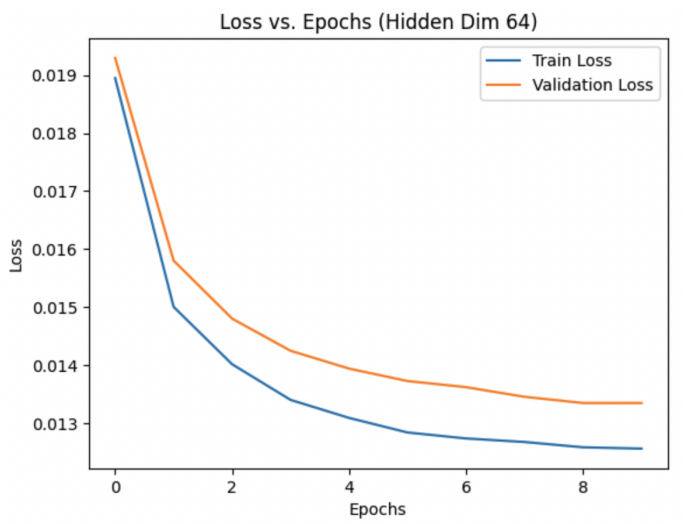


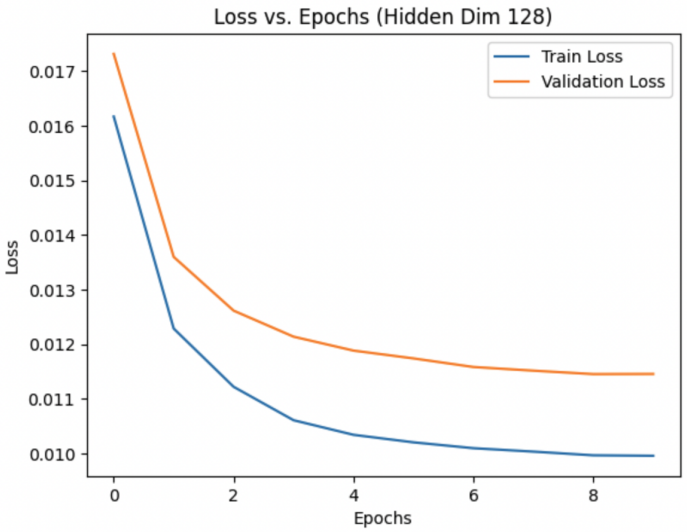
Visual Comparison

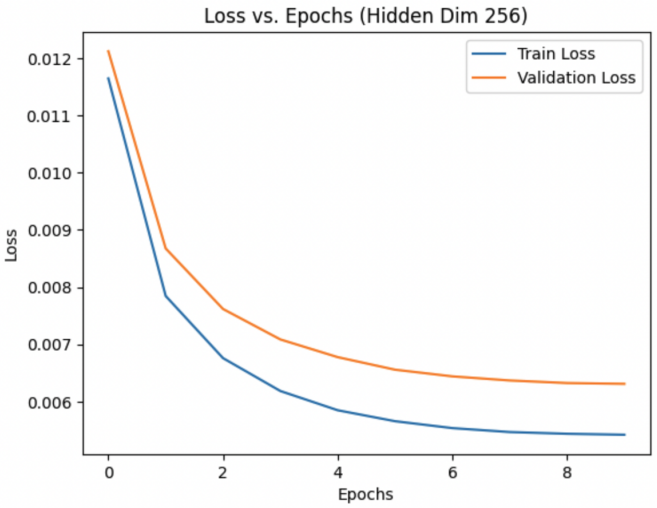


2. Varying Hidden Unit Size

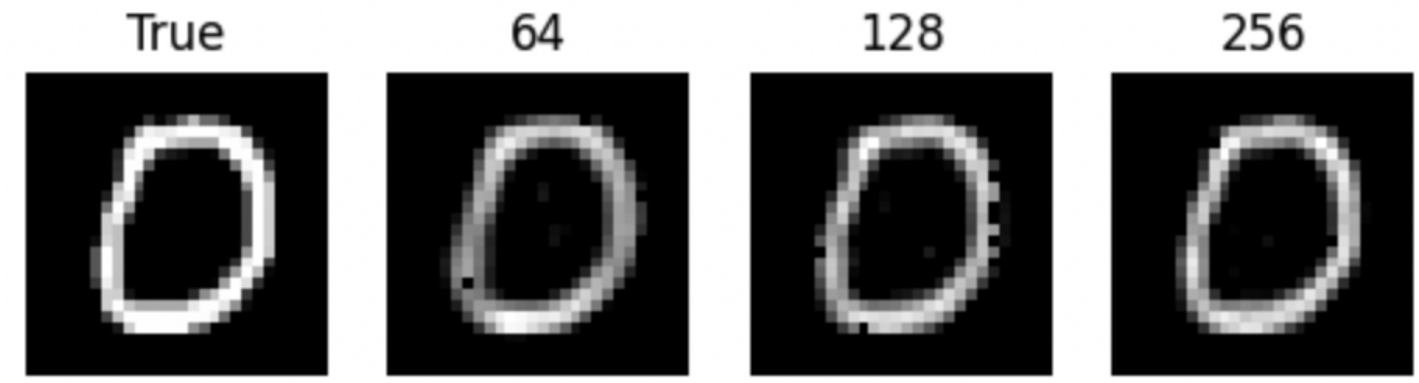
Plots







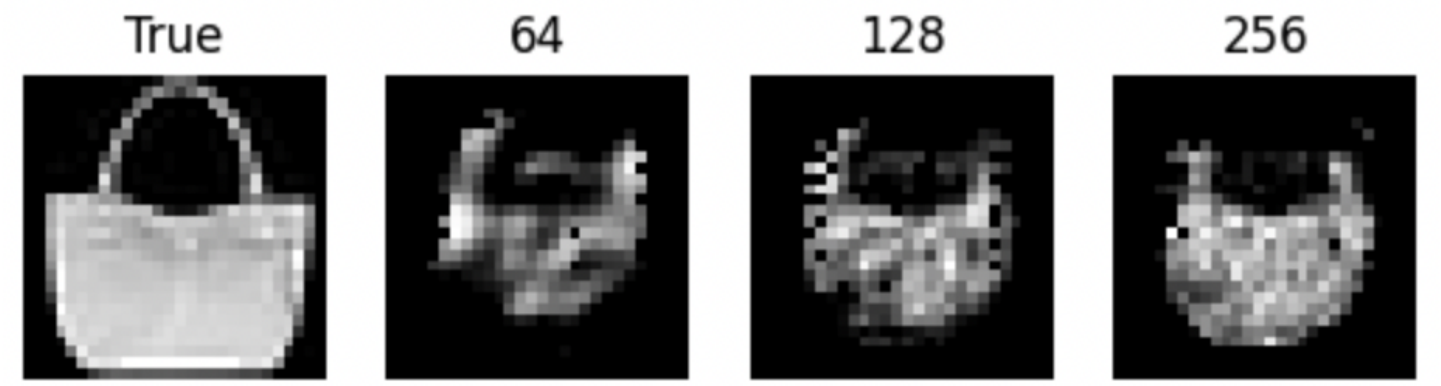
Comparison



For Hidden Dimension = 64, the image is blurry. For 128, the image became somewhat sharp, but some pixels were missing. For 256, the image looks almost as good as the true image.

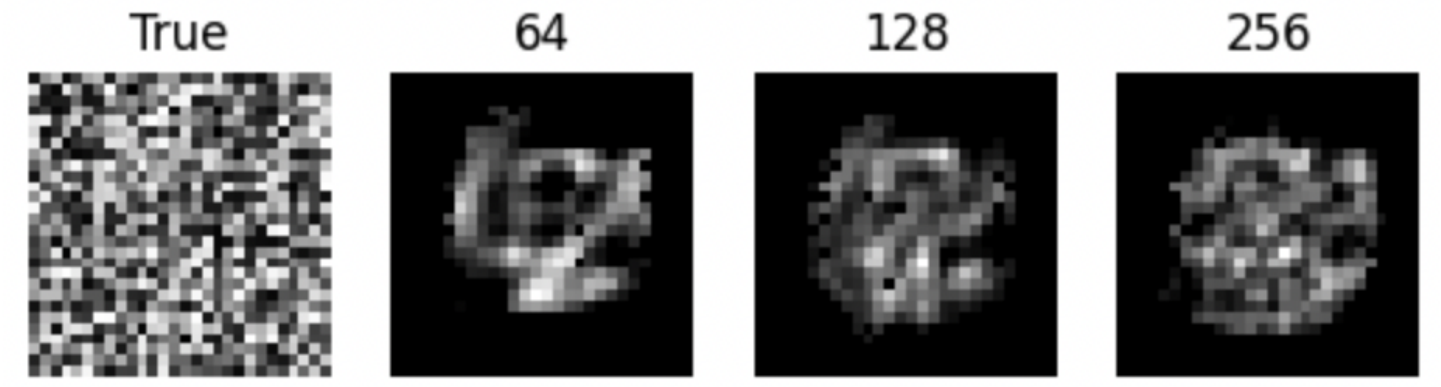
Other Data Distribution

Here is the model output on an MNIST Fashion dataset



Some part of the bag, like handles, is missing even with hidden size 256.

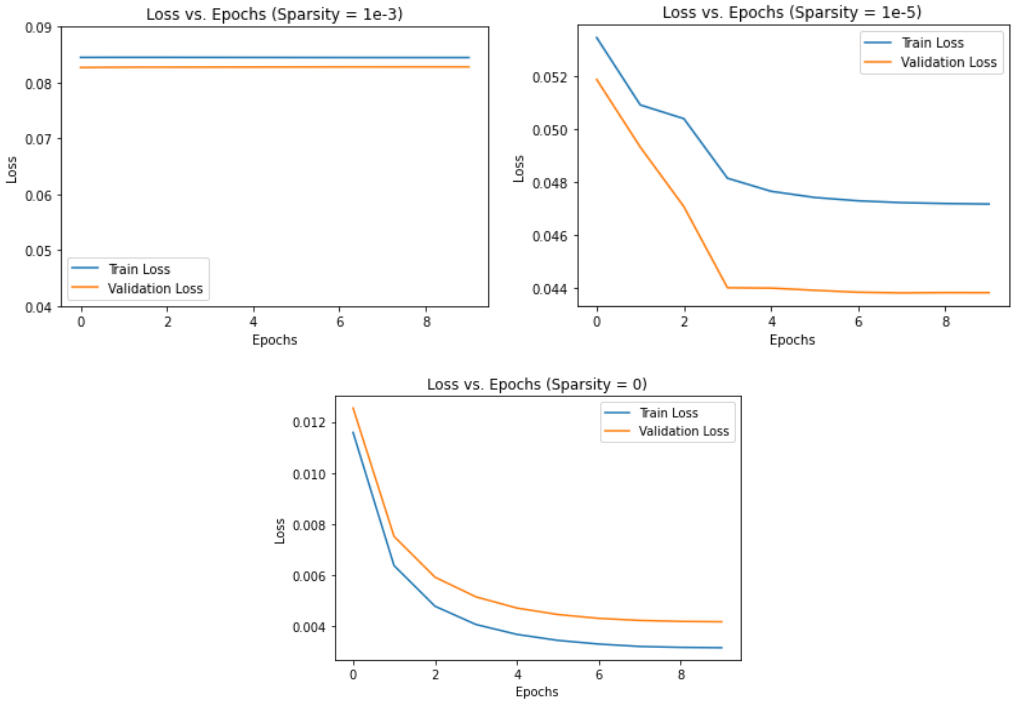
Here is the model output for Random Noise



The model makes the image's outer parts zero, which may be because the MNIST digits are present only in the centre of the frame.

3. Sparse Autoencoders

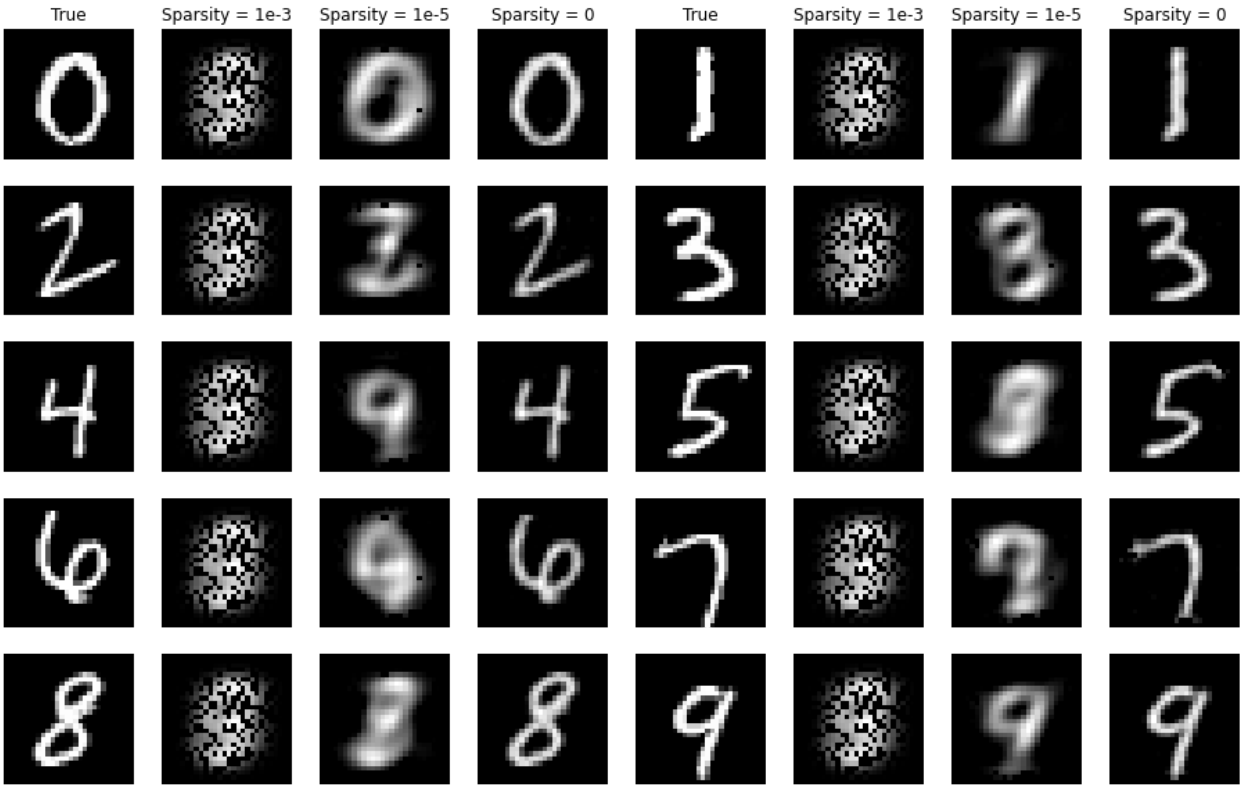
Plots



Average Hidden Layer Activations

Standard AE = 6.835
Sparse AE (Sparsity = 1e-5) = 0.05

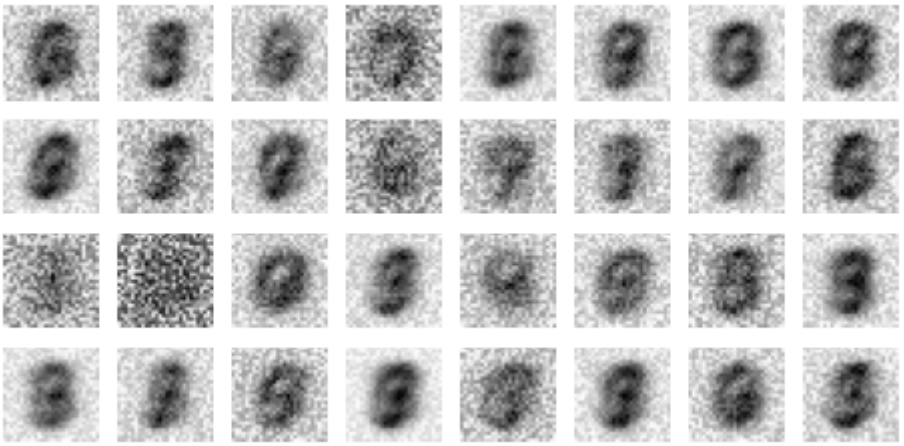
Visual Comparison



As the sparsity decreased, images were reconstructed in a better fashion. This is because now the hidden layer can pass on more information to the decoder. For sparsity 0 we got very good reconstruction, for sparsity 1e-3 the hidden layer activations became zero that's why got the same image irrespective of input. For sparsity 1e-5 the hidden layer activations are not high and not low so the reconstructed images are not that great.

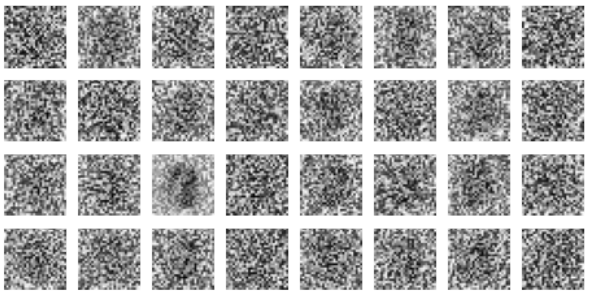
Filter comparision

Sparsity = 1e-3

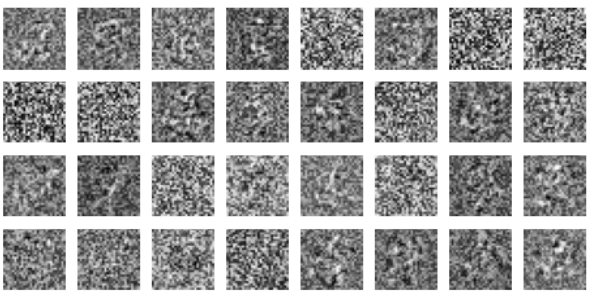


This is very sparse i.e., hidden layer activations are zero for all the images, that's why you can see the filters resemble the digits. Pixels in the place of digits are black and other are white. That means learned to completely ignore the digit and consider only the background.

Sparsity = 1e-5



Standard AE

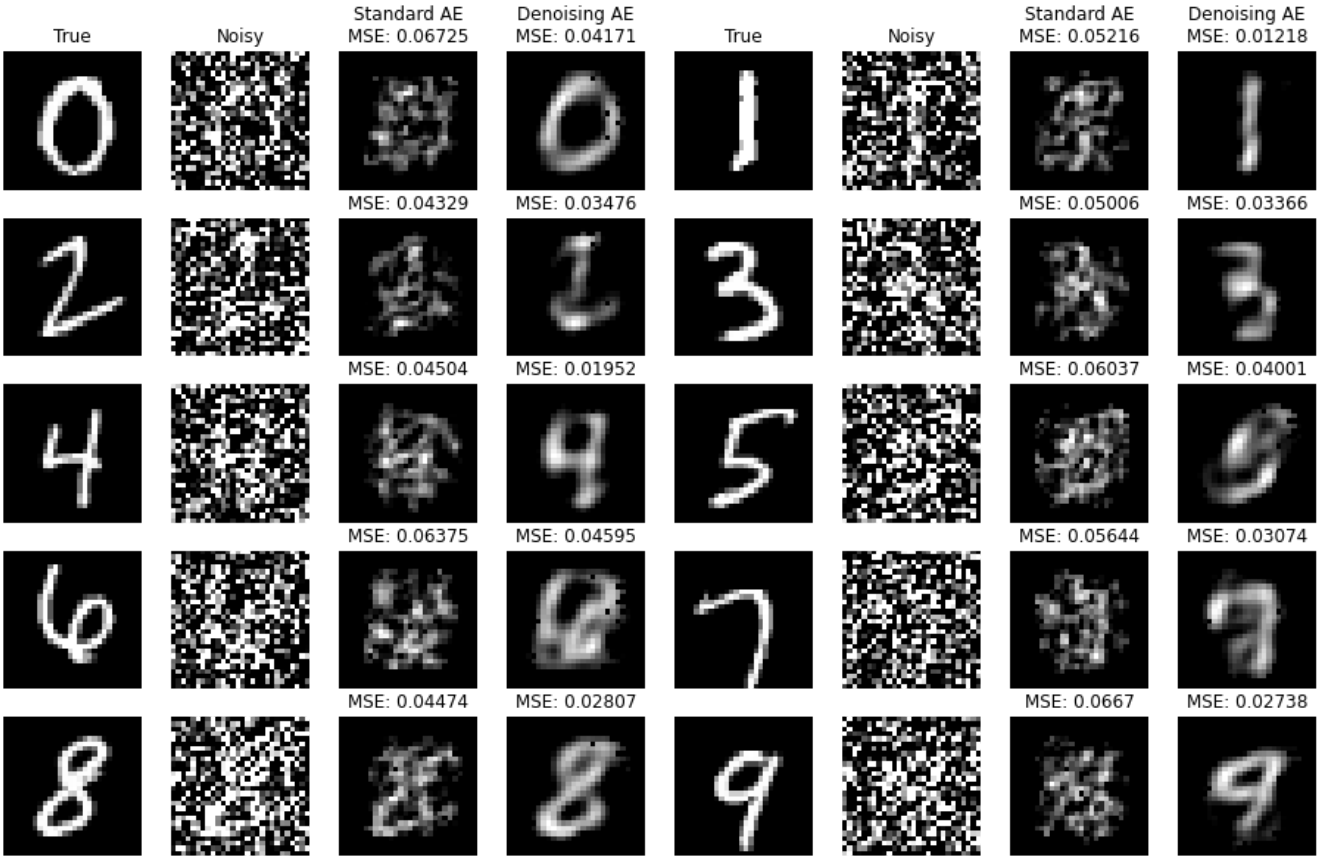


There are little blobs like structures in Standard AE filters, for sparsity 1e-5 it resembles like some random noise. Theoretically the filters should be less abstract as the Sparsity increases, but it is difficult to observe in the above filters.

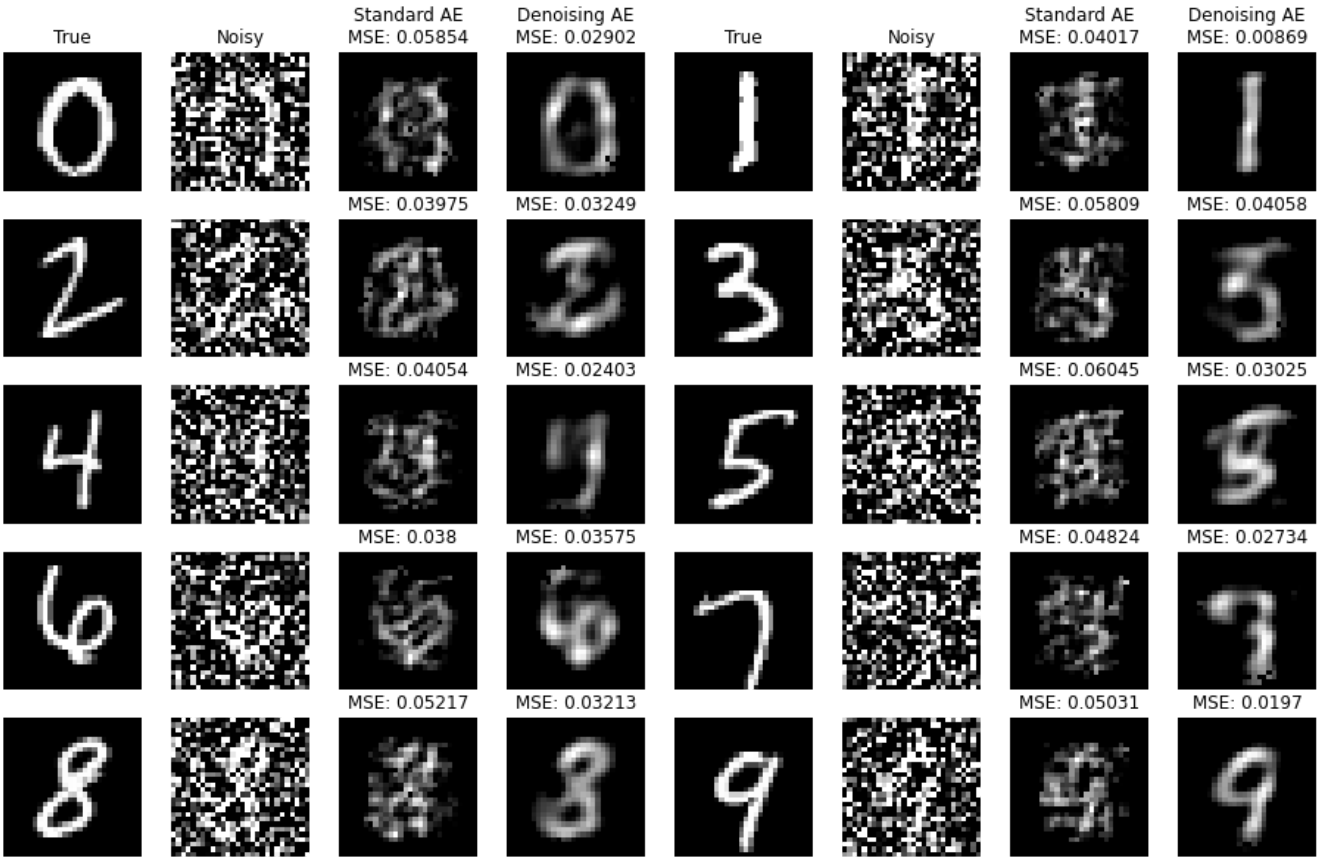
4. Denoising Autoencoders

Noise image to Standard AE vs Denoising AE

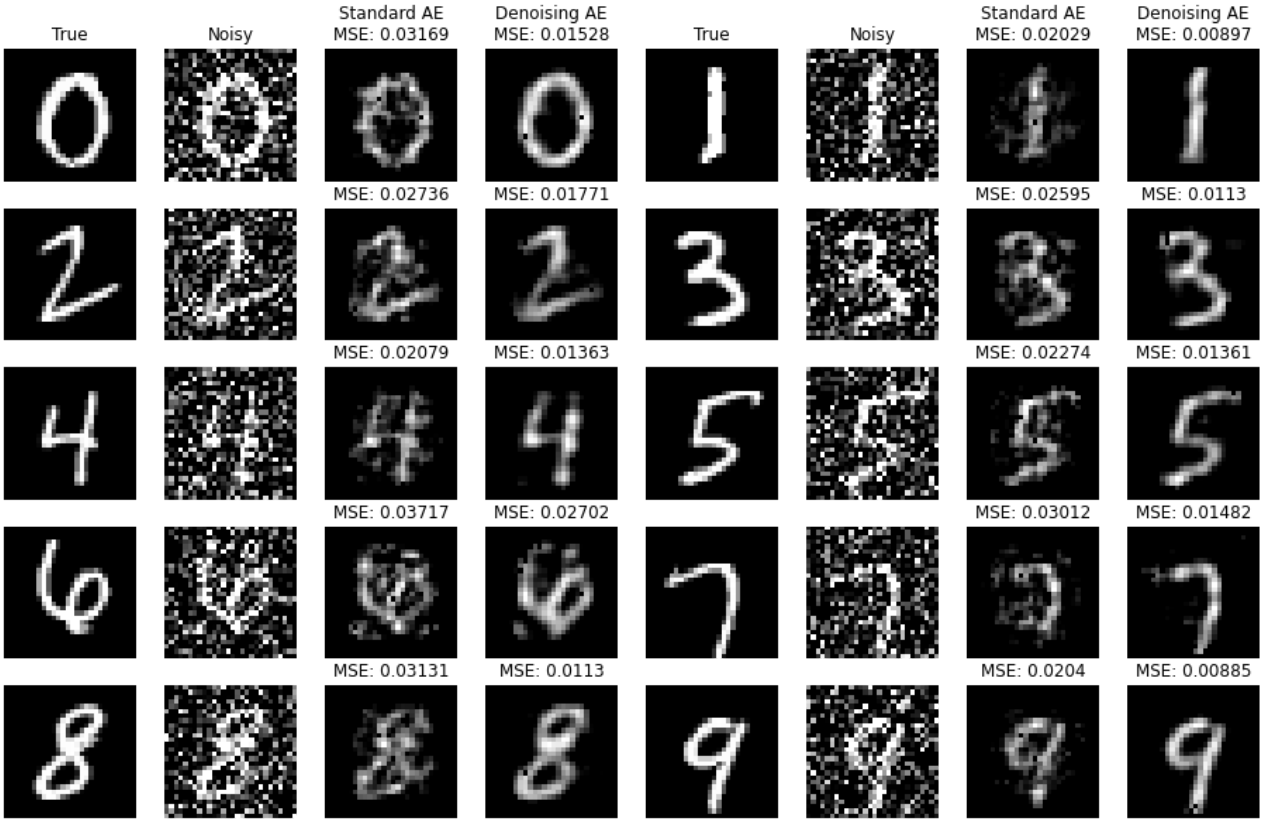
Noise 0.9



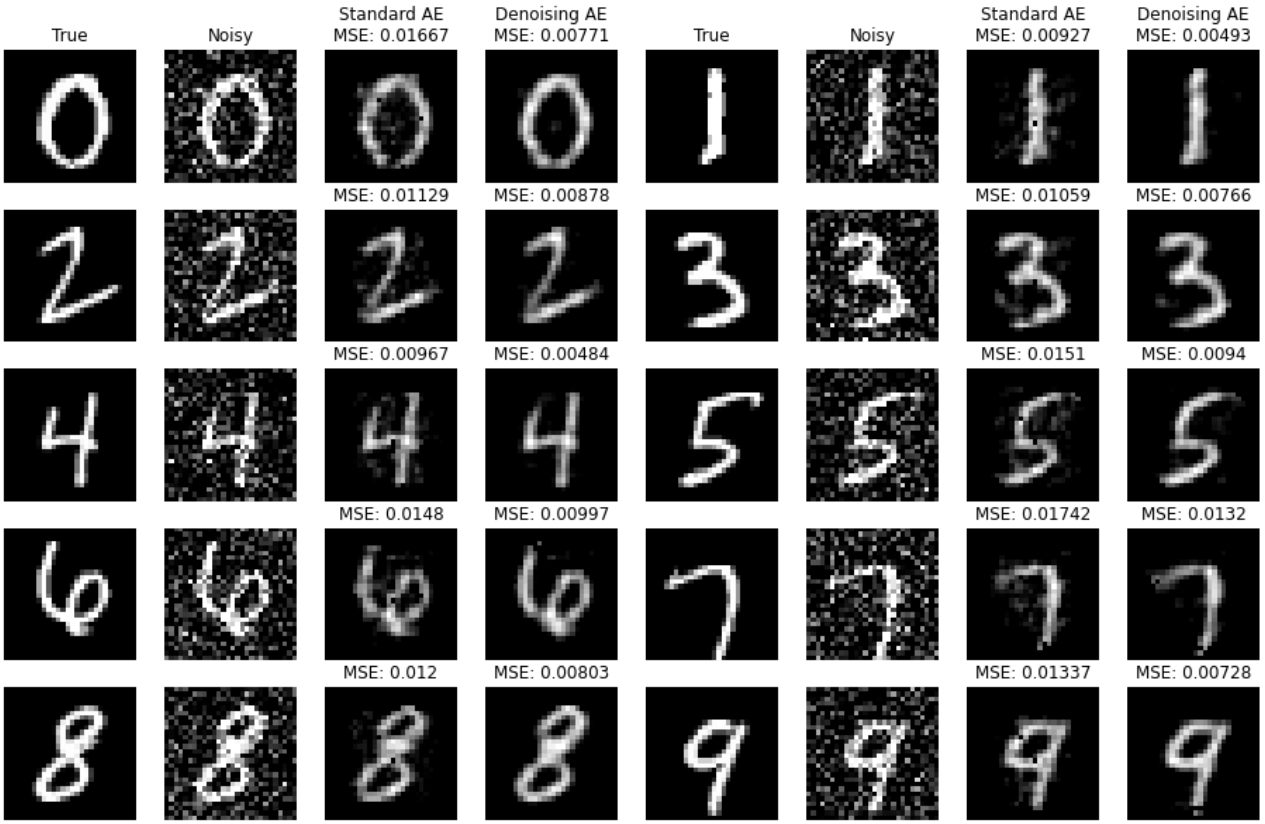
Noise 0.8



Noise 0.5

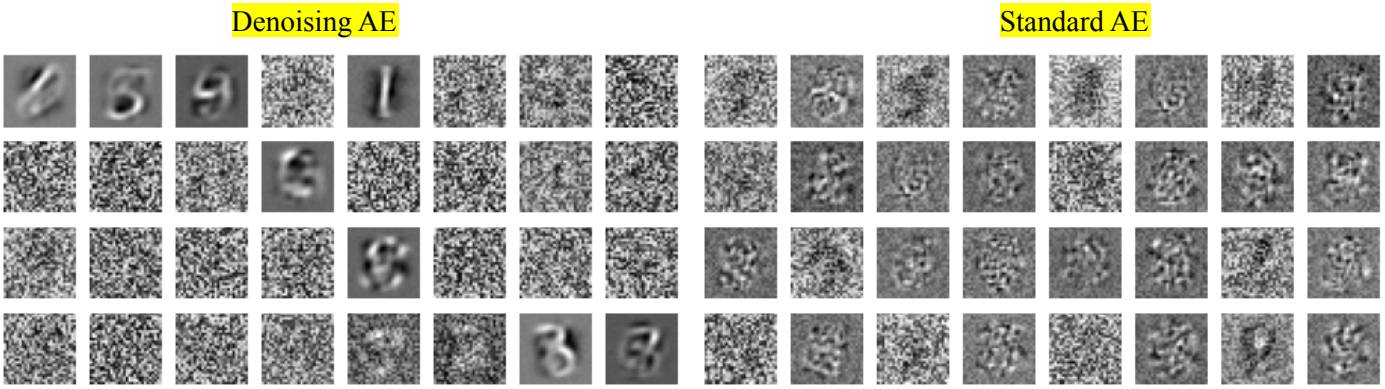


Noise 0.3



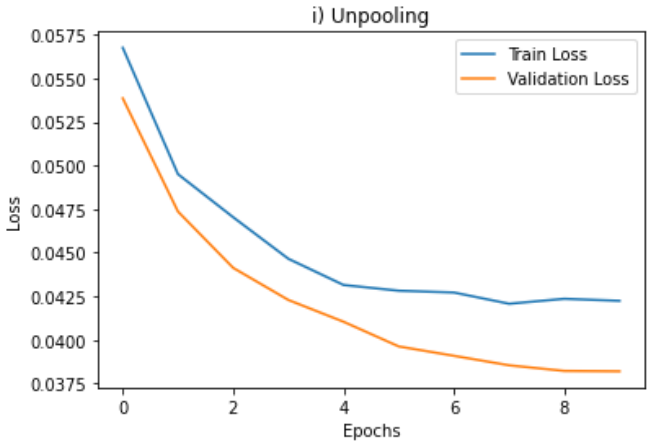
As the noise value decreased standard AE started reconstructing better images (both visually and MSE), same is the case with denoising AE. 0.9 Noise is too much we can't even identify the image. But irrespective of the noise value denoising AE is better than standard AE (both visually and MSE).

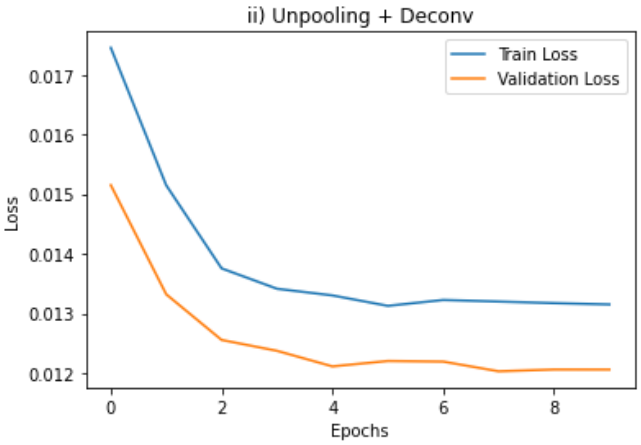
Filter Visualisation

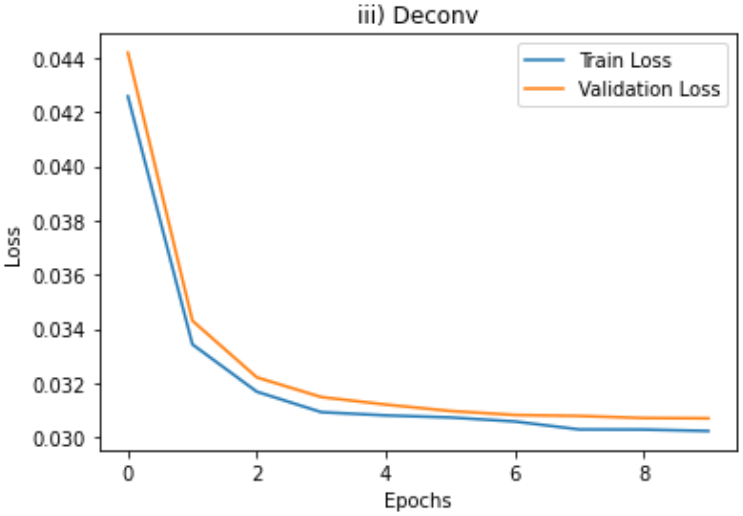


(Noise 0.9) As expected Denoising AE is learning useful features compared to Standard AE. See the bottom right 2 filters in Denoising AE, they look like 8, the top row third column looks like 9, top row third column looks like 1.

5. Convolutional Autoencoders







Reconstruction error (Low to High)
1. Unpooling + Deconv (error: 0.012)
2. Deconv (error: 0.031)
3. Unpooling (error: 0.0375)

Reason: Unpooling operation has to set 3 out of 4 pixels as zero which means it won't be able to capture that much information. One reason for Deconv not performing well is due to lack of symmetry in the encoder-decoder part. In the encoder we are doing max pool which is not being counteracted in Deconv case, so it is difficult for the model to learn the pattern. Unpooling + Deconv is obviously the best performing model among these three.

Convergence

Unpooling is the worst. Unpooling + Deconv & Deconv are quite similar both started converging at around 4 epochs, which might be because both have similar number of paramters to optimize.

Filter Visualization

Hidden - Deconv1 - Deconv2 - Deconv3 - Output

Deconv 3

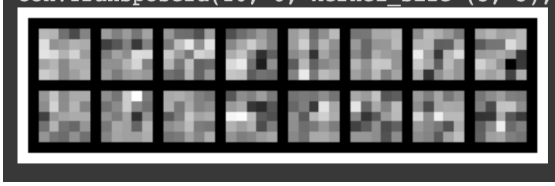
```
Model ii)
ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1))
```

```
Model iii)
ConvTranspose2d(8, 1, kernel_size=(4, 4), stride=(2, 2))
```

Deconv 2

```
Model ii)
ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(1, 1))
```

```
Model iii)
ConvTranspose2d(16, 8, kernel_size=(5, 5), stride=(2, 2))
```



Deconv 1

```
Model ii)
ConvTranspose2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
```

```
Model iii)
ConvTranspose2d(16, 16, kernel_size=(5, 5), stride=(2, 2))
```