EE6132 Assignment No. 3: Report

-Akash Reddy A, EE17B001

October 15, 2019

Introductory Notes

PyTorch has been leveraged to construct all the models in this assignment and answer the sub-questions as well. `torchvision.datasets` and `torchvision.transforms` have been used to extract the MNIST datasets directly from the website and convert them into Tensors. The code has been written into a Google Colab notebook, linked here.

1 Comparing PCA and Autoencoders

PCA has been implemented using `torch.svd()`, which produces a right singular vector matrix **V** containing the principal directions. The comparison of PCA generated image and Autoencoder output is shown below.

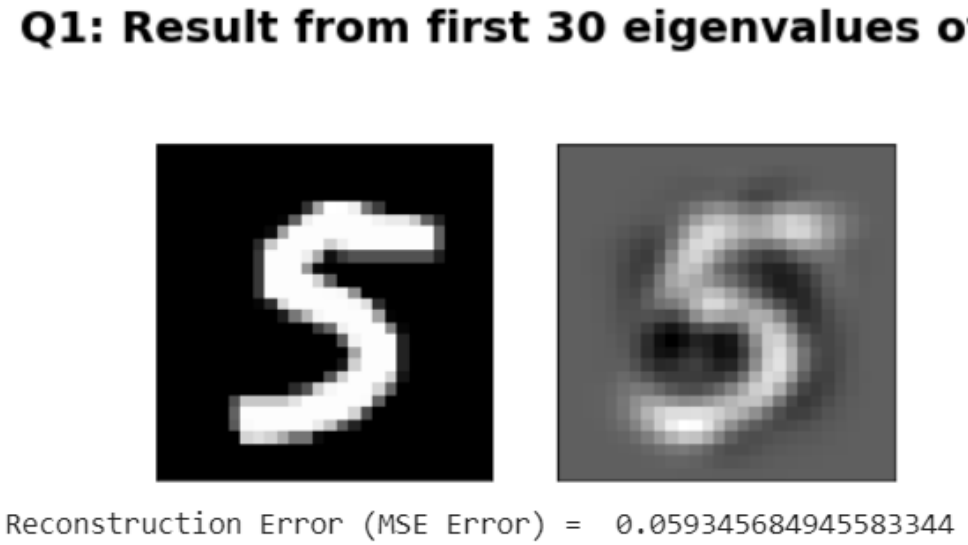

Figure 1: PCA Output

PCA has higher reconstruction error (and hence lower reconstruction accuracy) than autoencoders.
We observe that PCA images are very fuzzy compared to the autoencoder one. Besides this, the background of digits in PCA are grey, and not black like the original images. This is because taking only the first 30 components of PCA captures the 30 most important directions of information, ignoring the other insignificant differentiators such as background which looks the same for all digits.

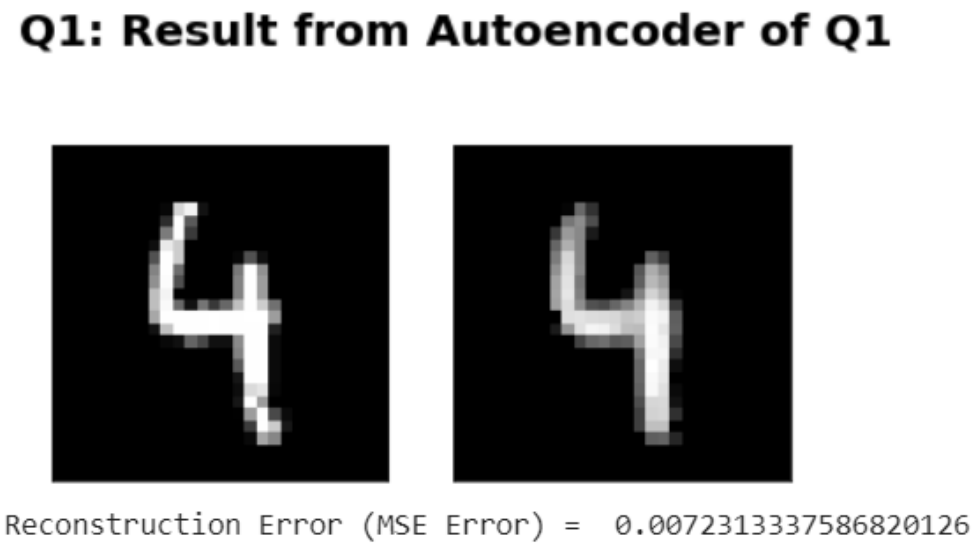

Figure 2: Autoencoder Output

2 Experimenting with hidden units of various sizes

The comparison for x = 64, 128, and 256 respectively is shown below.

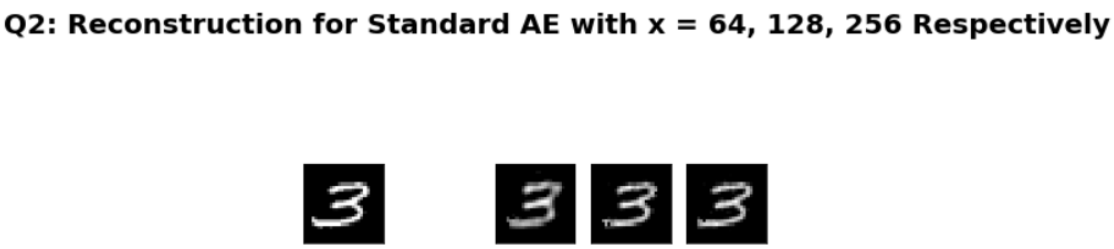

Figure 3: Standard AE outputs for varying hidden layer sizes

Clearly, the reconstruction quality for x = 256 is the most superior and for x = 64 is the least (the latter has some gaps and broken areas).

If a noisy/non-digit image is passed, the autoencoder produces images which have bright pixels towards the centre of the image. This is because it has learnt the important features in the digit images which are closer to the centre.

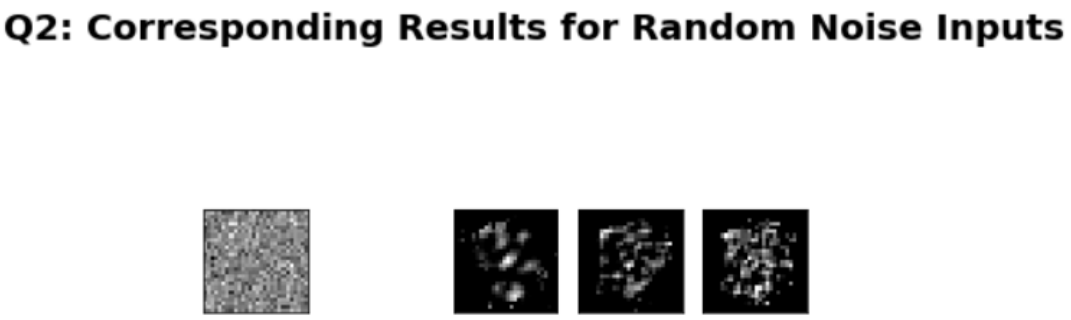

Figure 4: Standard AE outputs for input noise

3 Sparse Autoencoders

The output from a Sparse AE is shown below.

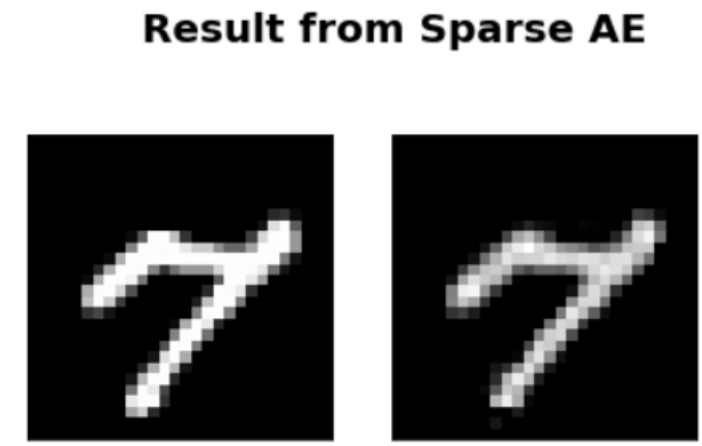

Figure 5: Output of Sparse AE

The average hidden layer activations for the Sparse AE are much smaller in magnitude than any of the standard AEs. This is in line with what is expected, due to the L1 penalty imposed on the activations in the Sparse AE.

The learned filters for 3 hidden layer neurons are shown below, along with filters of the standard AEs. These are more varying from pixel to pixel as opposed to the standard AE weights; this is explained by how very local features are learnt

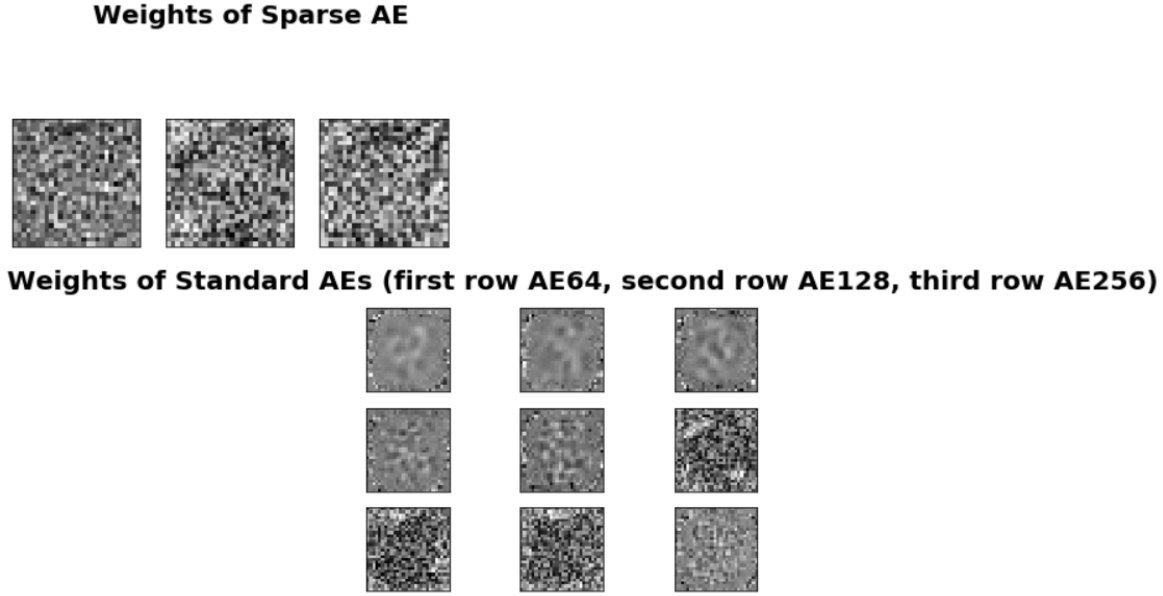

Figure 6: Weights of Sparse AE

4 Denoising Autoencoders

When corrupted images are passed into the previouly trained standard AEs, we get reconstructed images which are also corrupted. This is because the standard AEs are not trained to denoise, i.e, their loss function does not compare the output of a *noisy* image with a clean image during training, hence they are not used to noisy images.

When noise levels are increased (0.3, 0.5, 0.8, 0.9), the output produced by standard AEs grow increasingly corrupted as well, as shown below.

Figure 7: Output of Standard AE to Noise = 0.3

Figure 8: Output of Standard AE to Noise = 0.5

Figure 9: Output of Standard AE to Noise = 0.8

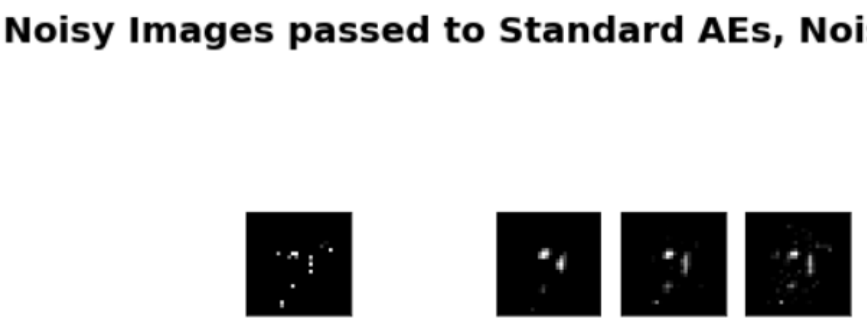

Figure 10: Output of Standard AE to Noise = 0.9

The same inputs to the trained denoising AE (trained with noise = 0.5), shows significant denoising ability to make the image cleaner again. This is shown below.

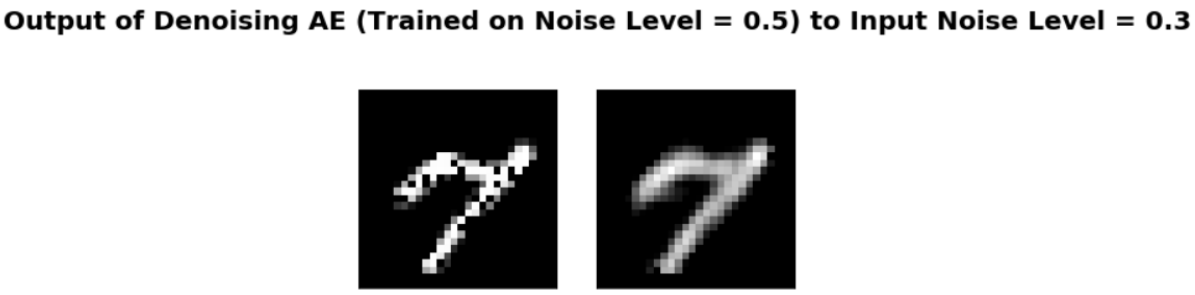

Figure 11: Output of Denoising AE to Noise = 0.3

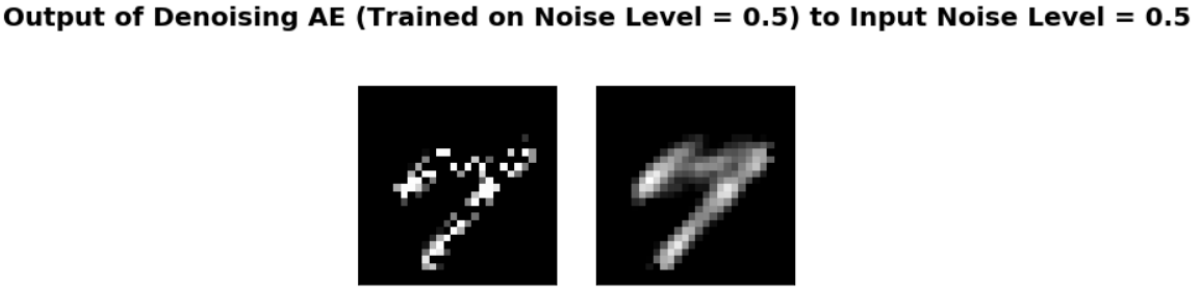

Figure 12: Output of Denoising AE to Noise = 0.5

Figure 13: Output of Denoising AE to Noise = 0.8

Figure 14: Output of Denoising AE to Noise = 0.9

The learned encoding filters for 3 of the hidden layer neurons in Denoising AEs are shown below, along with the same for standard AEs. The denoising AE's weights seem to try and capture the general shape of input digits (some vague general digit shapes can be spotted in the weights).

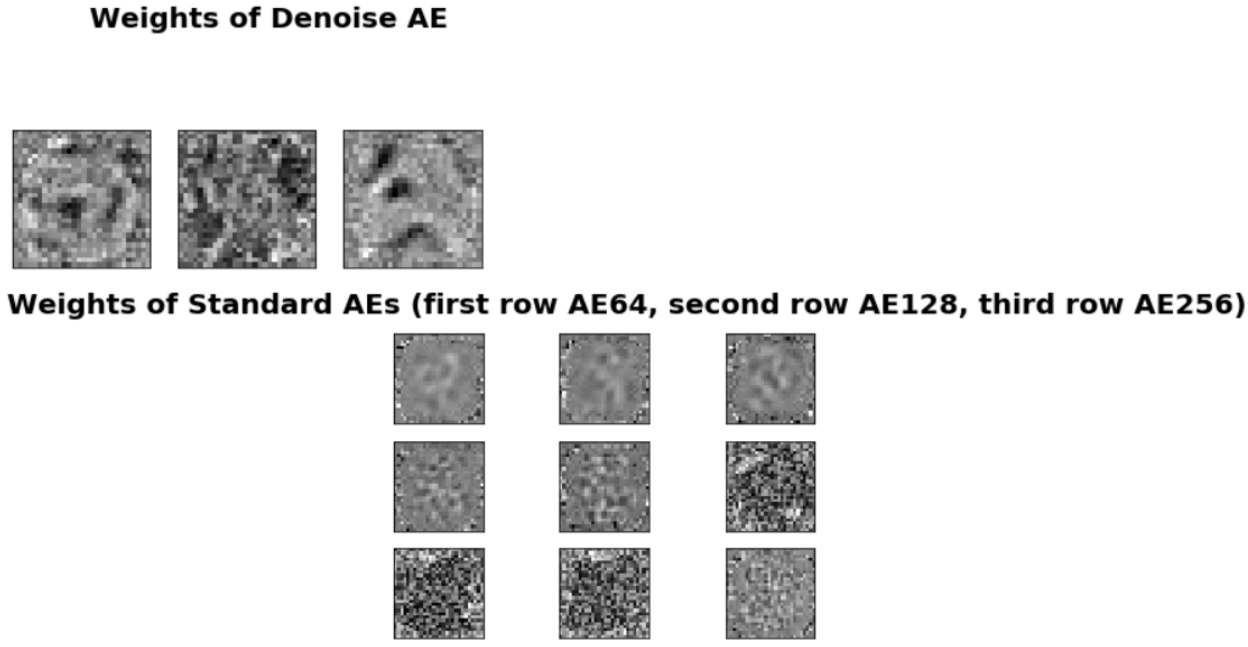

Figure 15: Output of Denoising AE to Noise = 0.9

5 Manifold Learning

When an input data from MNIST is taken and the pixel values are moved in random directions, we are moving randomly in a 784-dimensional space. Therefore, each pixel can end up taking any random value. This is why we get random noise which isn't a valid digit.

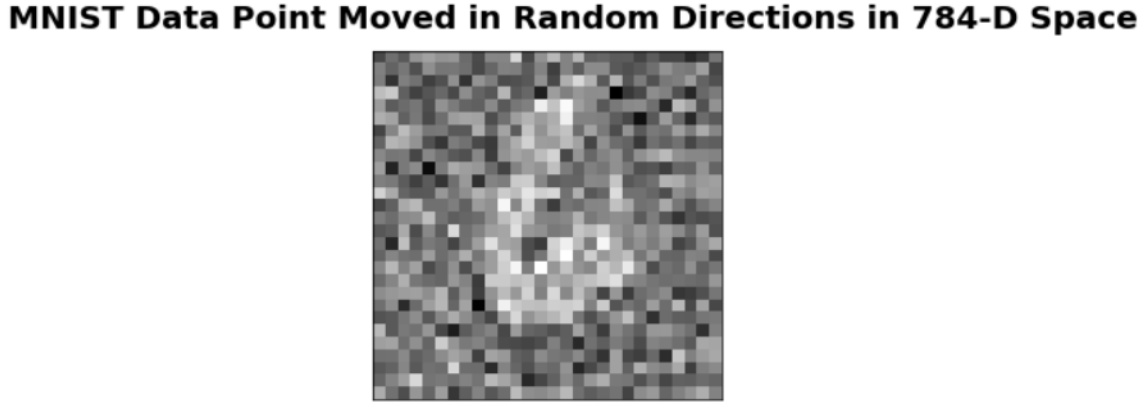

Figure 16: MNIST Data Point Moved Randomly in 784-D Space

However, when we add noise only to the hidden layer representation of the autoencoder, this means that we are moving in a random direction **along the manifold**. The manifold looks like a smaller dimensional space locally, and hence, any random movement along this manifold will mean that our image now looks like a slightly different version of the same digit, or even another digit which lies close to this point on the manifold.

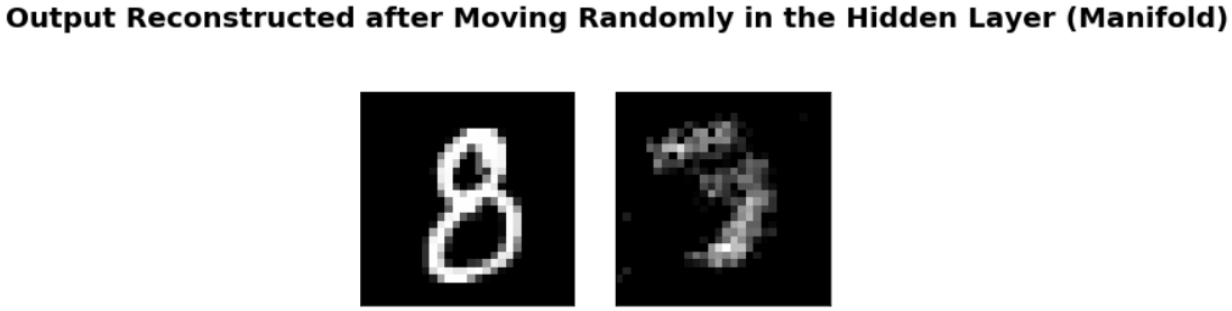

Figure 17: MNIST Data Point Moved Along Manifold

Our output after adding noise to representation looks more like a digit than the previous noise added to 784-dimensional space. This verifies that AEs learn a manifold. (In the above particular example, the input image is 8 whereas the output looks more like a 3. This is expected because 3's lie very close to 8's on the manifold, due to lots of common features!)

6 Convolutional Autoencoders

The reconstructions, errors, convergence, and filters for all three types of upsampling are shown below. Each unpooling layer has a convolution layer in tandem with it. Unpooling and Unpooling + Deconvolution seem to do nearly equally well. Applying deconvolution alone for upsampling performs badly.

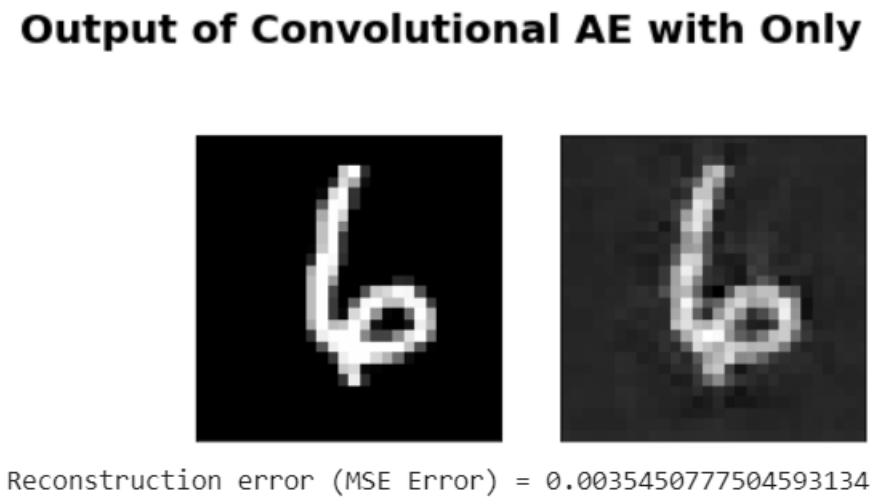

Figure 18: Reconstruction and Error for Unpool Upsampling

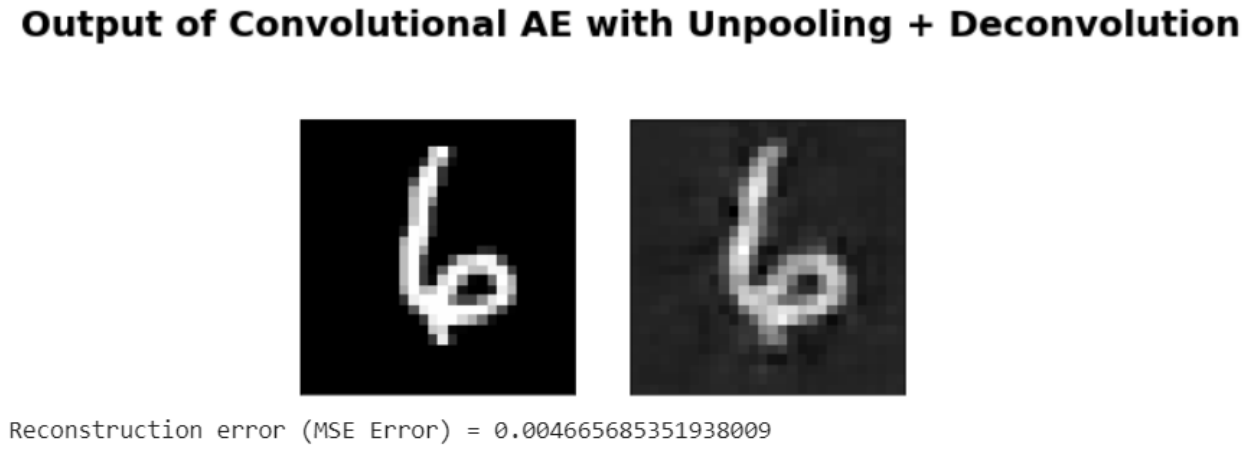

Figure 19: Reconstruction and Error for Unpool+Deconv Upsampling

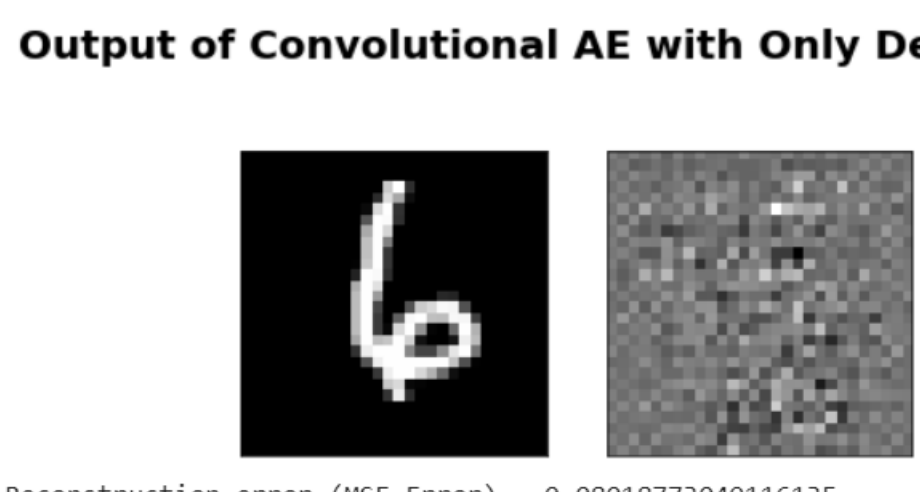

Figure 20: Reconstruction and Error for Deconv Upsampling

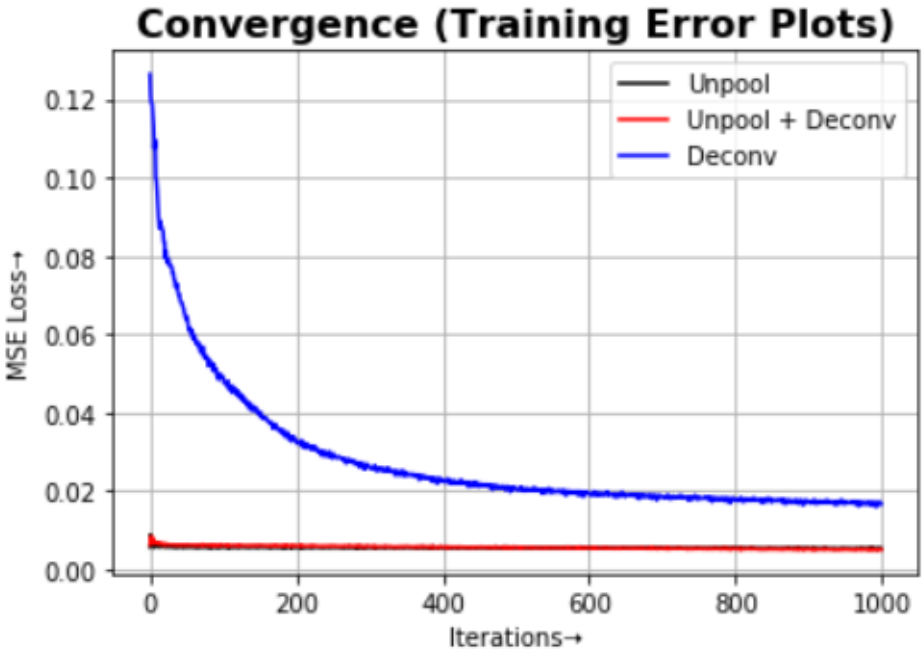

Figure 21: Convergences of Error for the 3 Upsampling Methods

Some Convolutional Filters of Only Unpooling AE (1 from each layer)

Figure 22: Decoder Filters for Unpool Upsampling

Some Deconvolutional Filters of Unpool + Deconv AE (1 from each layer)

Figure 23: Decoder Filters for Unpool + Deconv Upsampling

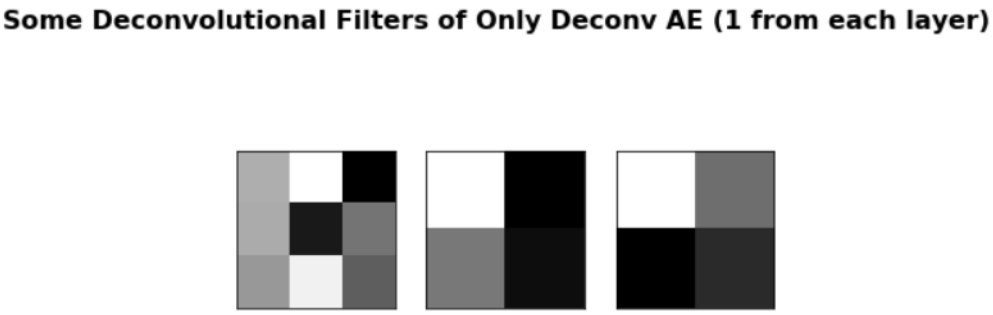

Figure 24: Decoder Filters for Deconv Upsampling