EE6132: Assignment 3

Auto Encoders

November 19, 2021

Contents

1 Brief Description of the Code

Before starting off describing the assignment, here's a brief note.
There were too many plots in this assignment (440 in number to be precise), hence for convenience only one plot per example has been shown in the report. The complete set of plots can be viewed here.

1.1 Libraries Used

The libraries used to run the python code are described below:

- numpy
- matplotlib.pyplot

- tqdm
- time
- os
- sklearn
- torch
- torchvision
- mnist
- PIL
- skimage

1.2 Functions Defined

Each Autoencoder network is defined as a separate class. Thus, the different AE classes are as follows:

- Class AE for the autoencoder in the first question.
- Class AE 1h for an autoencoder with one hidden layer whose size is taken as a parameter.
- Class AE manifold for the autoencoder designed for analyzing the manifold.
- Class conv AE unpool for the convolutional autoencoder with just unpooling.
- Class conv AE deconv for the convolutional autoencoder with just deconvolution.
- Class conv AE deconv unpool for the convolutional autoencoder with both unpooling and deconvolution operations.

There are different functions defined for the various parts of the assignment.These functions are listed below.

- **Train:** function to train the AEs, returns the training loss and performs backprop appropriately.
- **Test:** function to test the AEs, returns the test loss.
- **average act:** function to find the average activation values across the test dataset.
- **Test Image:** function to plot the reconstruction of a test image.
- **Visualize activations:** function used to visualize the hidden layer activations.
- **Visualize filters:** function used to visualize the weights of a normal(Non-convolutional) autoencoder.
- **Add Noise:** function used to add noise to the image input.
- **Manifold Analysis:** function used to perform the manifold analysis.
- **Visualize Decoder Weights:** function used to visualize the decoder weights for a convolutional autoencoder.
- **Run AE:** a function which encompasses all the above functions and runs them appropriately based on a series of flags.
- **Run Assignment:** a wrapper function which runs the entire assignment based on a series of user inputs.

2 Comparing PCA and AE

2.1 PCA

Principal Component Analysis was done by using 30 principle components.
The image inputs were scaled to [0,1] before performing PCA.
The PCA actually performed better if there wasn't any standardization or transformation done externally.
The resulting mean square error for PCA was approximately 0.018.

2.1.1 Actual MNIST Images

As mentioned in the disclaimer, for convenience just one true image is taken.
The results corresponding to the other digits can be found in the aforementioned drive link.
The digit 3 is taken for all experiments for the sake of representation and observation in this report.

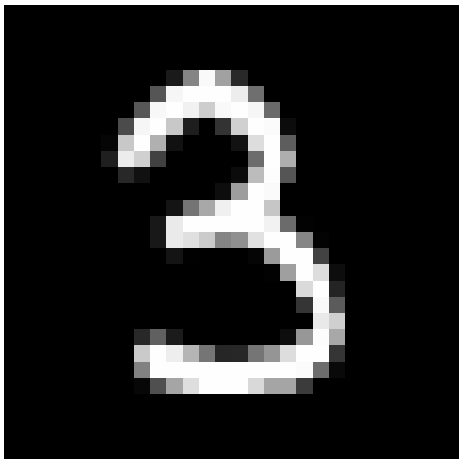

Figure 1: The digit 3 from the MNIST dataset

2.1.2 Reconstructed Images

Figure 2: The digit 3 reconstructed using PCA

2.2 AE

2.2.1 Architecture

The architecture of the AE is as follows is given below:
Encoder:Flattened Image (784) → fc(512) → ReLU → fc(256) → ReLU → fc(128) → ReLU → fc(30) → hidden representation →
Decoder: hidden representation → fc(128) →ReLU → fc(256) → ReLU → fc(784) → ReLU → → fc(784) →ReLU → reconstructed image (784)
The loss function was a MSE loss function and the AE was trained using the ADAM optimizer.
The batch size was 64 and a learning rate of 10^{-3} was employed and the AE was trained for 10 epochs.

2.2.2 Loss plot and accuracy

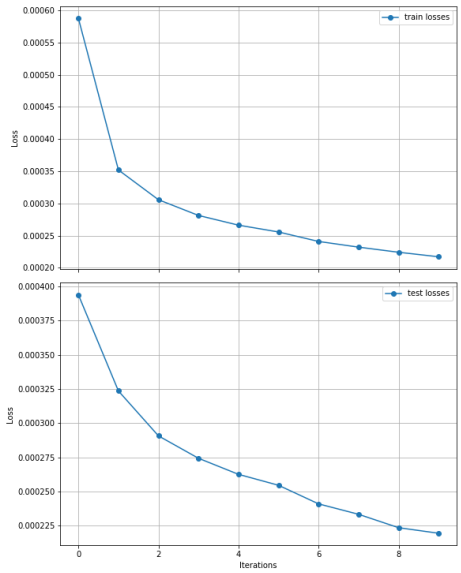

Figure 3: The Loss plot of the AE described above

The test reconstruction MSE per image was 0.00021954145631752908.

2.2.3 Reconstructed Images

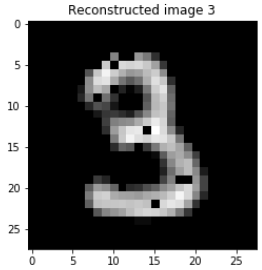

Figure 4: The digit 3 reconstructed using AE

2.3 Observations

- We see that the reconstruction accuracy of AE is much much lower than PCA when brought to the same scale which is what we'd expect.
- We see that the PCA jmage is somewhat all over the place, as it essentially does the reconstruction across 30 directions with the largest variance which is also not a lot for the 784 dimensional space. However, that said, even the PCA leads to a visually perceptible image which is appreciable.
- The AE does a much better job at reconstructing the digit. The reconstructed digit is almost a copy of the original test digit but with a few missing pixels here and there. The contour of the digit also seems to be smoother and larger.
- The AE performs much better than the PCA as it makes use of non-linear activation functions whereas PCA is a linear scheme.

3 Hidden Units of Different Sizes

3.1 Architecture

This section involves autoencoders of the following architecture:
Encoder:Flattened Image (784) → fc(hidden layer) → ReLU → hidden representation →
Decoder: hidden representation → fc(hidden layer) →ReLU → fc(784) →ReLU →reconstructed image (784)

3.2 Hidden Unit of size 64

3.2.1 Loss Plot and Accuracy

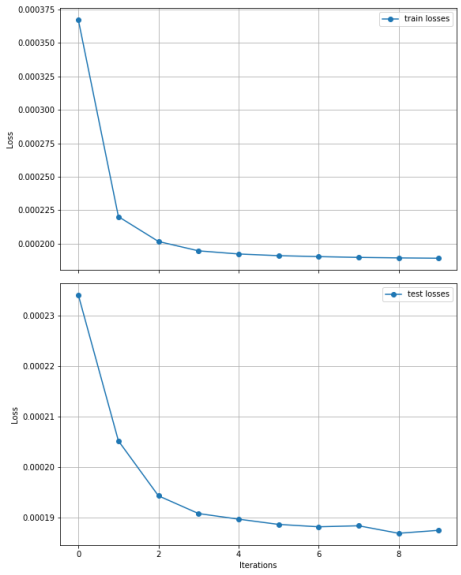

Figure 5: The loss plot of the AE described above

The test reconstruction MSE per image was 0.000187423822353594.

3.2.2 Reconstructed Images

3.2.3 Loss Plot and Accuracy

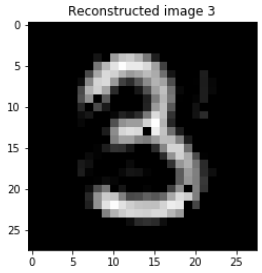

Figure 6: The digit 3 reconstructed using an AE with hidden layer size of 64 neurons

3.3 Hidden Unit of size 128

3.3.1 Loss Plot and Accuracy

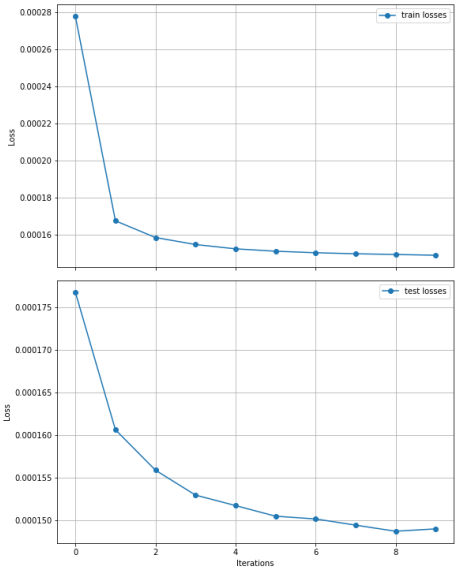

Figure 7: The loss plot of the AE described above

The test reconstruction MSE per image was 0.00014897370419930667.

3.3.2 Reconstructed Images

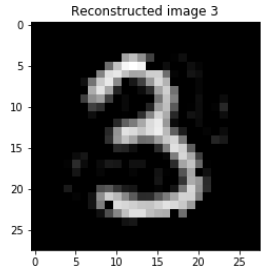

Figure 8: The digit 3 reconstructed using an AE with hidden layer size of 128 neurons

3.4 Hidden Unit of size 256

3.4.1 Loss Plot and Accuracy

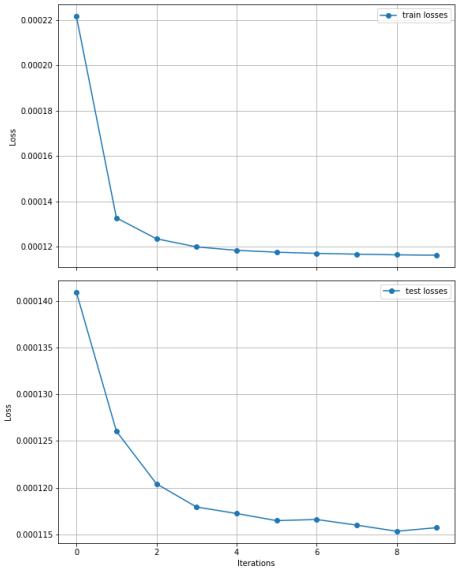

Figure 9: The loss plot of the AE described above

The test reconstruction MSE per image was 0.00011571851064218208.

3.4.2 Reconstructed Images

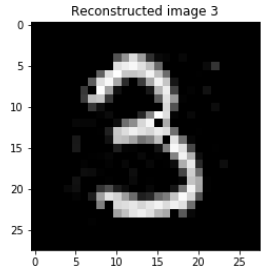

Figure 10: The digit 3 reconstructed using an AE with hidden layer size of 256 neurons

3.5 Output for a non-digit image

The above autoencoder with a hidden layer of 256 neurons was used for this experiment.
An image of Lena (shown below),

Figure 11: Lena

Was converted to gray scale

Figure 12: Lena Grey Scale

and reduced to 28x28 pixels. This was sent as the input to the Auto Encoder.

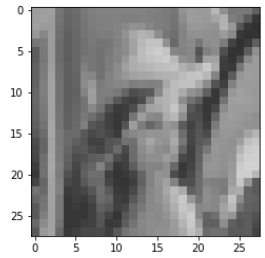

Figure 13: Input to the Auto Encoder

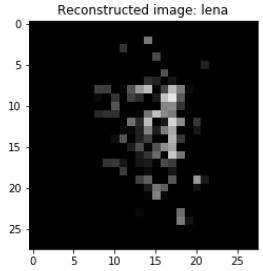

Figure 14: Reconstructed Image of Lena through the Auto Encoder

3.6 Observations

- We see that the performance of the AE improves with the number of neurons in the hidden layer.
- This is substantiated by the reconstructed image too as the resulting image is a lot crisper with lesser background noise as we progress towards an auto encoder with more hidden layer neurons.
- We see that the improvement in the MSE is not a lot as we move from a hidden layer of 128 neurons to 256 neurons. This could be because we are approaching the size of the manifold space and hence there is minimal change in the performance.
- On observing the output of the reconstructed image of Lena, we see that the AE has learnt a manifold which has a dark background. This is expected as the training data is also of the same form.
- We see that a lot of the background is dark for the 256 neuron hidden layer case. On performing this experiment for the other two auto encoders, there is more and more portions visible as the hidden layer size goes lower. This could mean a lesser understanding of the manifold space by the autoencoder.
- Lastly, to summarize, the output reconstructed image is garbage as clearly lena is not a digit which is a testament to the existence of a manifold space.
- The test reconstruction accuracy of all the three autoencoders were much lower than the previous one which tells us that the accuracy is highest when the hidden layer dimension is closer to the manifold's dimension.

4 Sparse Auto Encoders

The architecture of the sparse autoencoders is the same as the previous section but with a hidden layer size of 900 neurons thus making it overcomplete. An L1 regularization was imposed on the activations of the hidden layer with a regularization constant which was varied in steps of 10.

4.1 Regularization of 0.001

4.1.1 Loss plots and accuracy

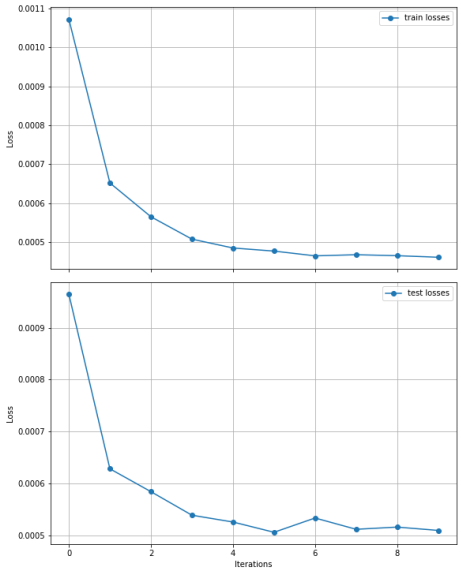

Figure 15: Loss plot for the sparse AE with a regularization of 0.001

The test reconstruction MSE per image was 0.0005093217478133738.

4.1.2 Reconstructed Image

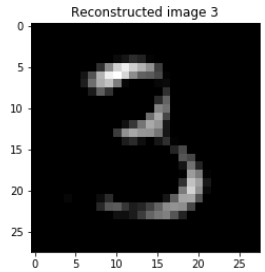

Figure 16: Reconstructed Image of the digit 3

4.1.3 Activations for the digit

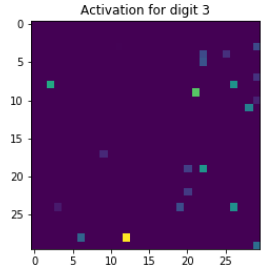

Figure 17: Hidden Layer Activation of the AE for the digit 3

The average activation for this AE was 0.0012347785599529744 per image over the test dataset.

4.1.4 Filter Plots

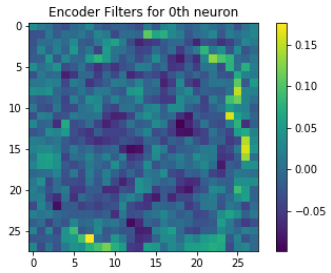

Figure 18: The encoder filters

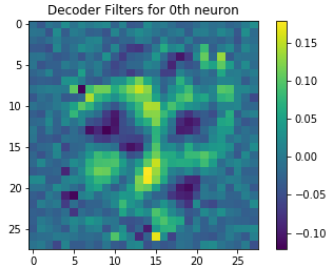

Figure 19: The decoder filters

4.2 Regularization of 0.1

4.2.1 Loss plots and accuracy

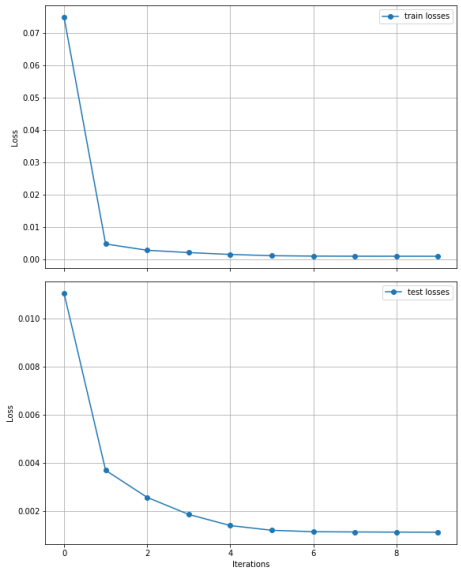

Figure 20: Loss plot for the sparse AE with a regularization of 0.1

The test reconstruction MSE per image was 0.0011195661500096321.

4.2.2 Reconstructed Image

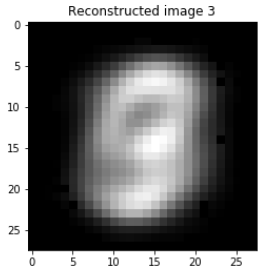

Figure 21: Reconstructed Image of the digit 3

4.2.3 Activations for the digit

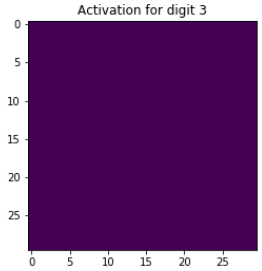

Figure 22: Hidden Layer Activation of the AE for the digit 3

The average activation for this AE was 0.001188675931096077 per image over the test dataset.

4.2.4 Filter Plots

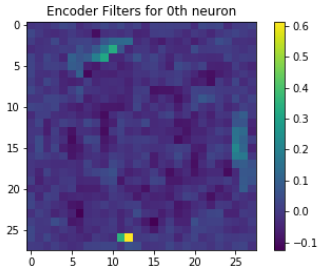

Figure 23: The encoder filters

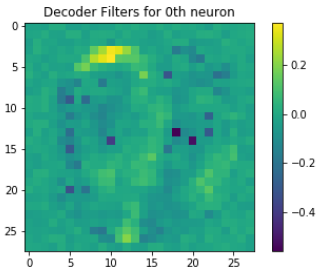

Figure 24: The decoder filters

4.3 Regularization of 1

4.3.1 Loss plots and accuracy

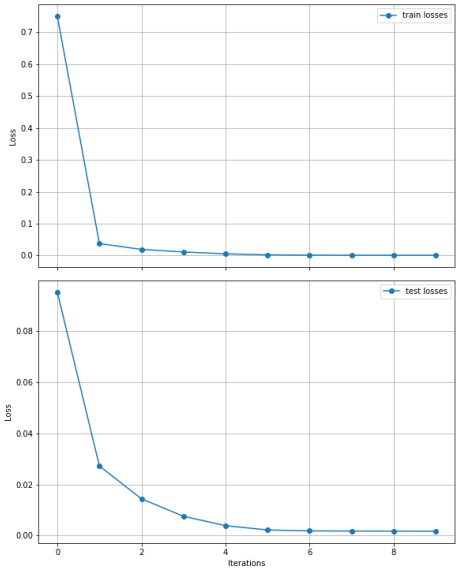

Figure 25: Loss plot for the sparse AE with a regularization of 1

The test reconstruction MSE per image was 0.0017272192053496838.

4.3.2 Reconstructed Image

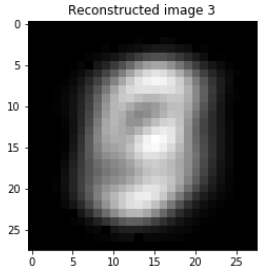

Figure 26: Reconstructed Image of the digit 3

4.3.3 Activations for the digit

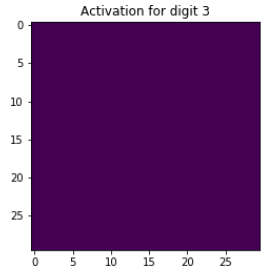

Figure 27: Hidden Layer Activation of the AE for the digit 3

The average activation for this AE was 0.0011713464580476284 per image over the test dataset.

4.3.4 Filter Plots

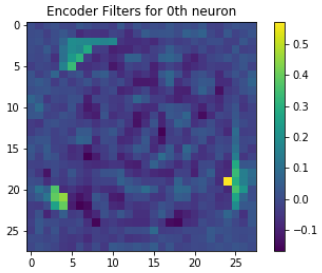

Figure 28: The encoder filters

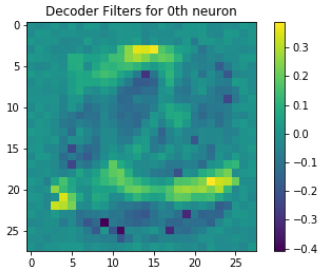

Figure 29: The decoder filters

4.4 Plots for the Standard AE

4.4.1 Activations for the digit

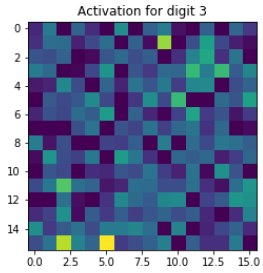

Figure 30: Hidden Layer Activation of the AE for the digit 3

The average activation for this AE was 0.0012687933064997195 per image over the test dataset.

4.5 Filter Plots

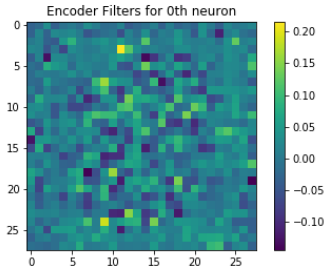

Figure 31: The encoder filters

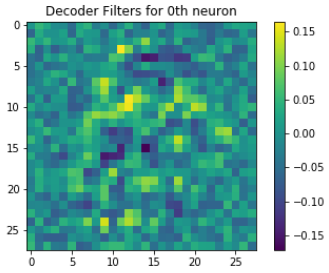

Figure 32: The decoder filters

4.6 Observations

- Firstly, we see that the losses on the regularized sparse autoencoders are much higher compared to their standard AE counterparts.
- We also see that lower the regularization, better is the reconstruction which is what we'd expect as the regularization penalizes the activation.
- The reconstructions for the last two higher regularizations just resemble blobs which look like 3. This demonstrates that higher regularization kills the reconstruction.

- The reconstructed image for the 0.001 regularization looks much sharper than the standard AE despite having a larger reconstruction error. This is because every last bit of performance is squeezed out of the neurons as they are forced to function properly due to the regularization.
- We see that the average activation values are lower for a higher regularization and much higher for the standard AE. (note: difference may appear small, but it is scaled by an order of 10^4).
- We see that the difference in the average activations for the last two regularizations are not a lot which is because both of them are high and they influence the loss function in a similar manner thus leading to a similar result.
- When we look at the activations for the sparse AE, we see that very few of them fire whereas for the standard AE almost all the neurons fire.
- Fewer and fewer neurons fire as the regularization goes higher and higher.
- We see that the encoder and decoder filters resemble digits for the sparse AE implying that each neuron caters towards a particular digit and fires only when they appear.
- For the standard AE no such digit can be seen.

5 Denoising Auto Encoders

The architecture of the sparse autoencoders is the same as the Standard AE but with a hidden layer size of 256 neurons.Noisy images are input to the AE while the loss is taken with respect to the actual image forcing the AE to denoise.
This experiment was performed for three different values of noise.

5.1 Noise = 0.1

5.1.1 Loss Plots and accuracy

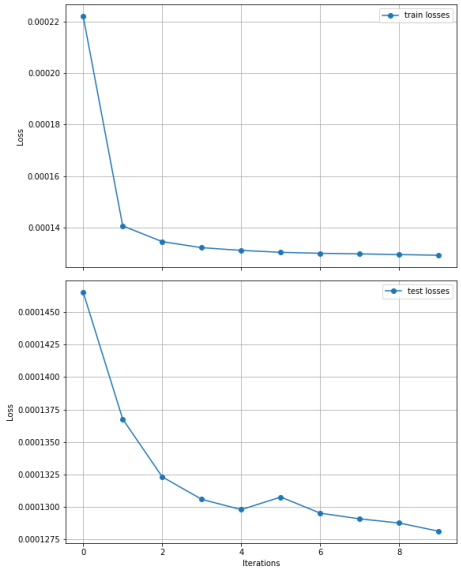

Figure 33: Loss plot for the denoising AE

The test reconstruction MSE per image was 0.00012817009701393545.

5.1.2 Noisy Image

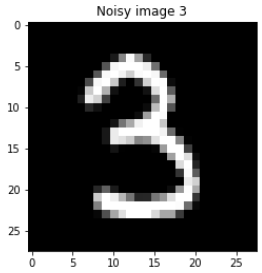

Figure 34: Noisy Digit

5.1.3 Reconstructed Image

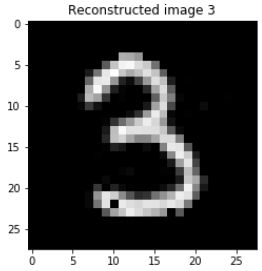

Figure 35: Reconstructed Digit for the denoising AE

5.1.4 Filter plots

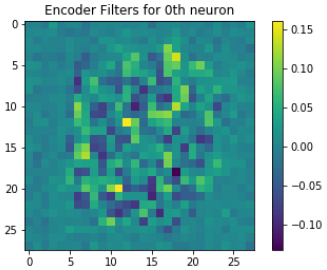

Figure 36: The encoder filters

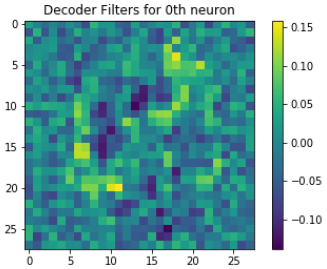

Figure 37: The decoder filters

5.2 Noise = 0.3

5.2.1 Loss Plots and accuracy

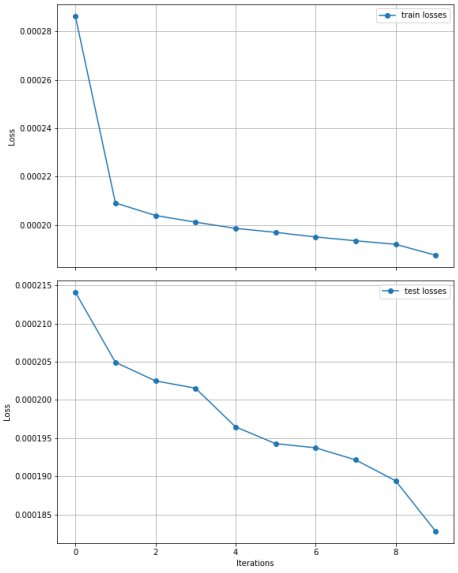

Figure 38: Loss plot for the denoising AE

The test reconstruction MSE per image was 0.00018255253962706774.

5.2.2 Noisy Image

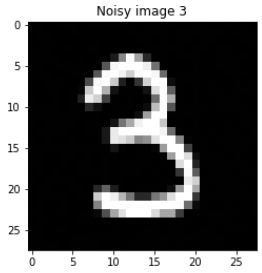

Figure 39: Noisy Digit

5.2.3 Reconstructed Image

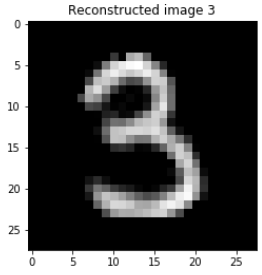

Figure 40: Reconstructed Digit for the denoising AE

5.2.4 Filter plots

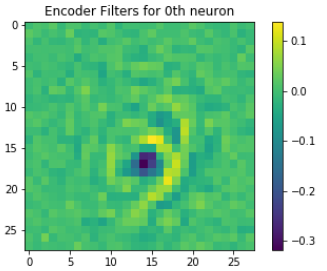

Figure 41: The encoder filters

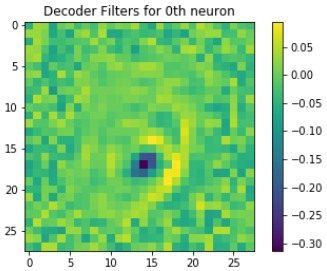

Figure 42: The decoder filters

5.3 Noise = 0.7

5.3.1 Loss Plots and accuracy

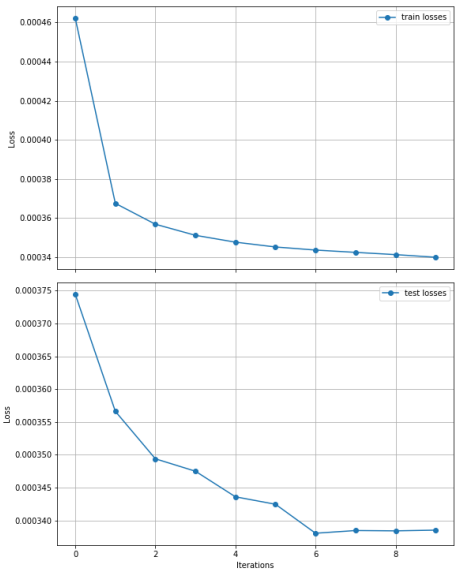

Figure 43: Loss plot for the denoising AE

The test reconstruction MSE per image was 0.00033803473343141377.

5.3.2 Noisy Image

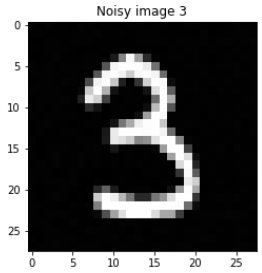

Figure 44: Noisy Digit

5.3.3 Reconstructed Image

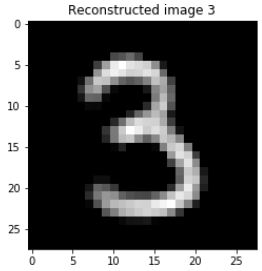

Figure 45: Reconstructed Digit for the denoising AE

5.3.4 Filter plots

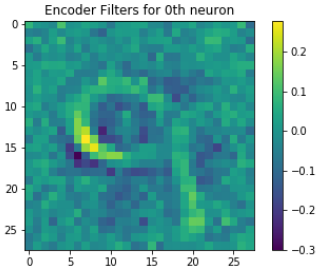

Figure 46: The encoder filters

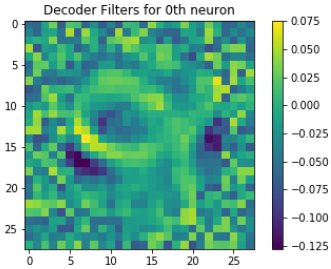

Figure 47: The decoder filters

5.4 Noisy Image to Standard AE

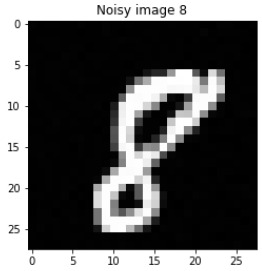

Figure 48: Noisy Image Input

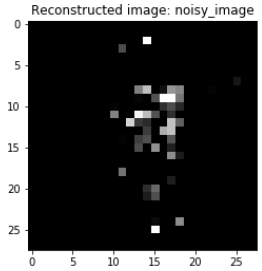

Figure 49: Reconstruction of Noisy Image

5.5 Observations

- Note: Although the noisy images look like they're absent of noise, the truth is that noise is added to the images. It's just that since the noise is very little and is scaled Gaussian noise, it is not present clearly in the image.
- We see that the reconstruction error increases with the increase in noise added which makes sense as this is similar to the effect of increasing regularization in the previous section.
- As in the previous case, we see that the reconstructions look much better in the case of lesser regularization/noise.
- We see that the filters resemble digits with the increase of noise which is what we'd expect.
- We see that the while the denoising AEs function perfectly, the standard AE is not resilient to higher values of noise as shown by the power reconstruction of the digit eight.

6 Manifold Learning

This section involves autoencoders of the following architecture:
Encoder:Flattened Image (784) → fc(64) → ReLU → fc(8) →ReLU →hidden representation →
Decoder: hidden representation → fc(64) →ReLU →fc(784) →ReLU → reconstructed image (784)

6.1 Hidden Unit of size 64

6.1.1 Loss Plot and Accuracy

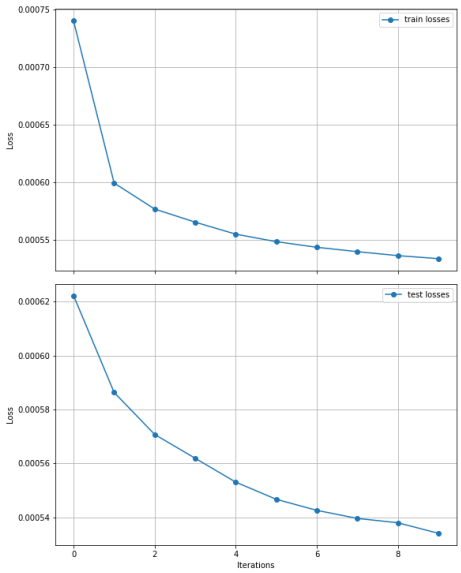

Figure 50: The loss plot of the AE described above

The test reconstruction MSE per image was 0.000534076418261975.

6.1.2 Reconstructed Images

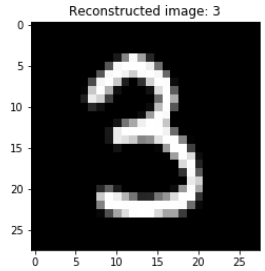

Figure 51: The digit 3 reconstructed after applying random noise of 0.1 at the hidden activations

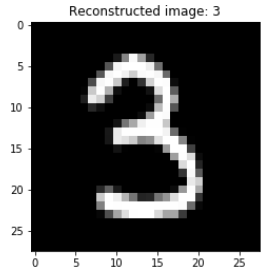

Figure 52: The digit 3 reconstructed after applying random noise of 0.3 at the hidden activations

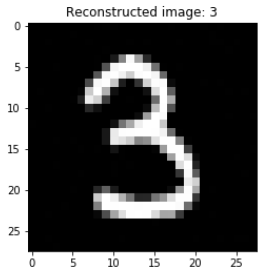

Figure 53: The digit 3 reconstructed after applying random noise of 0.7 at the hidden activations

6.2 Observations

- As we saw in the previous section, input image with noise doesn't result in a good output for a standard AE.
- However, the manifold as seen here tolerates variable amount of noise changes and produces a picture perfect reconstruction of the image.
- Increase in noise results to larger amount of grey spots in the image.

7 Convolutional Auto Encoders

7.1 CAE with unpooling

7.1.1 Architecture

The architecture is shown below:

```
    conv_AE_unpool(
  (encoder_conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_conv3): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (decoder_conv1): Sequential(
    (0): Identity()
  )
  (decoder_conv2): Sequential(
    (0): Conv2d(16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (decoder_conv3): Sequential(
    (0): Conv2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (unpool): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
)
```

Since pure unpooling is not very meaningful, 2 3x3 filters were learnt at the decoder end. The first "conv" filter at the decoder is made identity to make the decoder more reliant on the unpooling operation.
The unpooling operation is performed before every decoder layer shown above.

7.1.2 Loss Plots

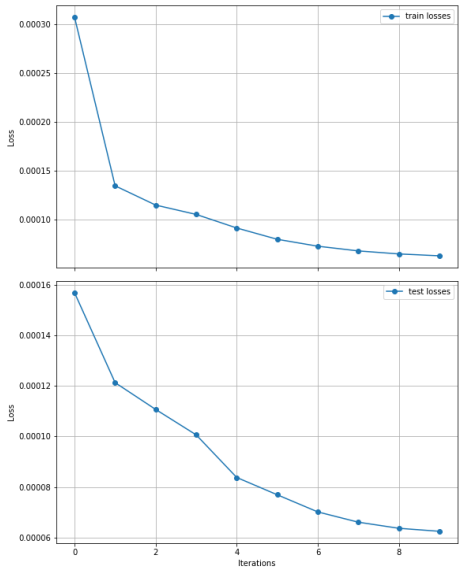

Figure 54: The loss plot of the CAE described above

7.1.3 Reconstructed Image

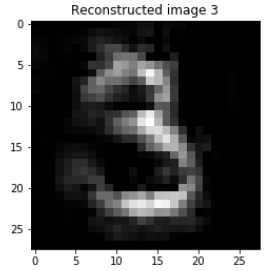

Figure 55: Reconstructed Image from the CAE

7.1.4 Decoder Filters

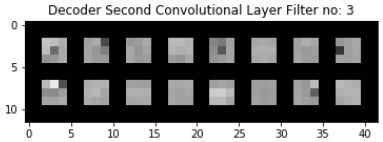

Figure 56: A second layer decoder filter

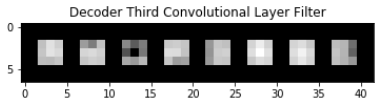

Figure 57: A third layer decoder filter

7.2 CAE with deconvolution

7.2.1 Architecture

```
    conv_AE_deconv(
  (encoder_conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_conv3): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (decoder_conv1): Sequential(
    (0): ConvTranspose2d(16, 16, kernel_size=(3, 3), stride=(2, 2))
    (1): ReLU()
  )
  (decoder_conv2): Sequential(
    (0): ConvTranspose2d(16, 8, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
  )
  (decoder_conv3): Sequential(
    (0): ConvTranspose2d(8, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
  )
)
```

Although several different filter sizes could be employed, this particular combination of filter sizes have been chosen as they result in the best MSE accuracy. They also have fewer parameters to train since the strides are larger.

7.2.2 Loss Plots

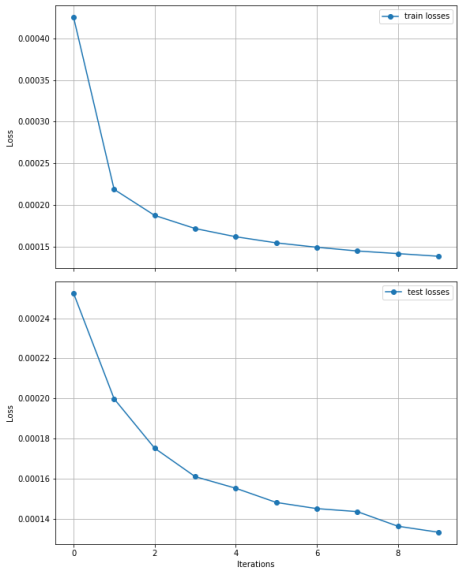

Figure 58: The loss plot of the CAE described above

7.2.3 Reconstructed Image

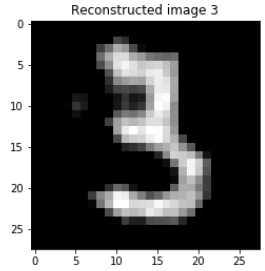

Figure 59: Reconstructed Image from the CAE

7.2.4 Decoder Filters

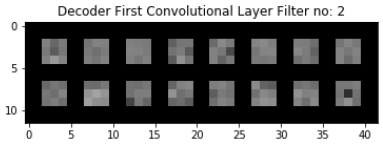

Figure 60: A first layer decoder filter

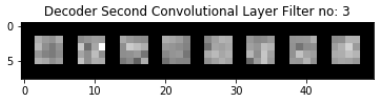

Figure 61: A second layer decoder filter

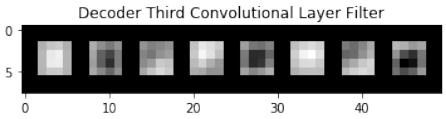

Figure 62: A third layer decoder filter

7.3 CAE with both unpooling and deconvolution

7.3.1 Architecture

```
    conv_AE_deconv_unpool(
  (encoder_conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_conv3): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (decoder_conv1): Sequential(
    (0): ConvTranspose2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (decoder_conv2): Sequential(
    (0): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (decoder_conv3): Sequential(
    (0): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (unpool): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
)
```

Here, the decoder is an exact mirror image of the encoder.

7.3.2 Loss Plots

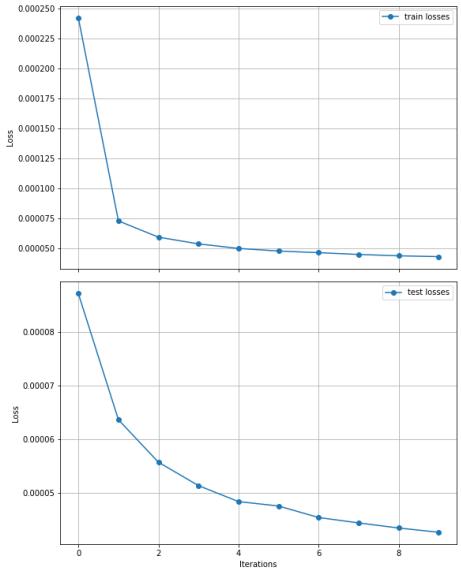

Figure 63: The loss plot of the CAE described above

7.3.3 Reconstructed Image

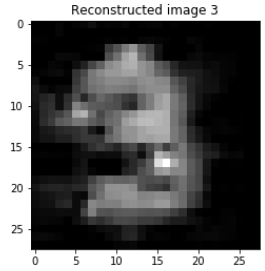

Figure 64: Reconstructed Image from the CAE

7.3.4 Decoder Filters

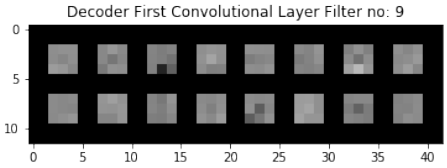

Figure 65: A first layer decoder filter

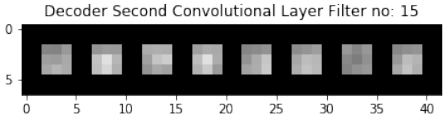

Figure 66: A second layer decoder filter

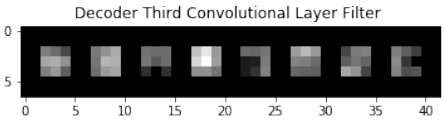

Figure 67: A third layer decoder filter

7.4 Observations

- As far as the order of reconstruction errors are concerned, we see that the trend follows Unpool > Deconvolution > Unpooling and Deconvolution.
- We see that the reconstructed images aren't as great as the MLP images but they are very perceptible.
- We see that the boundary of the digit is smooth for the first and the last reconstruction images.This maybe an effect of the unpooling as it sort of reverses max pooling thus leading to spilling/smoothening of values.
- We see that the third reconstruction image performs marginally better than the second one however both of them perform much worse than the unpooling case. This is because, the 3x3 filters are learned in the unpooling case. In the absence of any 3x3 filters and if we implement a 1x1 convolution instead,we see that the unpooling performs much worse and leads to a "checked" image.
- As seen in the CNN assignment, we see that the lower order filters are primitive and the higher order ones detect complex features.