ELEC 475 Lab 1
Matthew Li - 20217346

Model Details

This model is a simple auto-encoder composed of four layers; an input, output and two hidden layers. The general model architecture is described as follows:

1. Input layer: expects tensor of size 28*28 (size of an MNIST image) and outputs it to a tensor half of that size (392)
    a. *Pass tensor through ReLU activation function*
2. first hidden layer: expects a tensor of size 392 and casts it down to the specified bottleneck size. This layer is supposed to capture the important features within that image and store it as a tensor.
    a. *Pass tensor through ReLU activation function*
3. second hidden layer: expects a tensor of size bottle neck, this layer is supposed to reconstruct the input image (to a degree) from the bottleneck tensor, returning a tensor of size 392.
    a. *Pass tensor through ReLU activation function*
4. Output layer: expects tensor of size 392, outputs tensor of size 784.
    a. *Pass tensor through Sigmoid activation function*

Training Details

Per the source code, the model is initially trained with a batch size of 2048 images over 50 epochs. An Adaptive Moment (ADAM) optimizer is used, initialized with a learning rate of 1e-3, weight decay of 1e-5. A learning rate scheduler is attached to this optimizer, set to decrease learning rate when model performance begins to plateau (torch.optim.lr_scheduler). The loss function is specified as Mean Square Error (MSE).

Results

I trained at 3 batch sizes (4, 8, 16) to assess model performance for interpolation and de-noising. The models have otherwise identical hyper parameters. The loss plots are as follows:

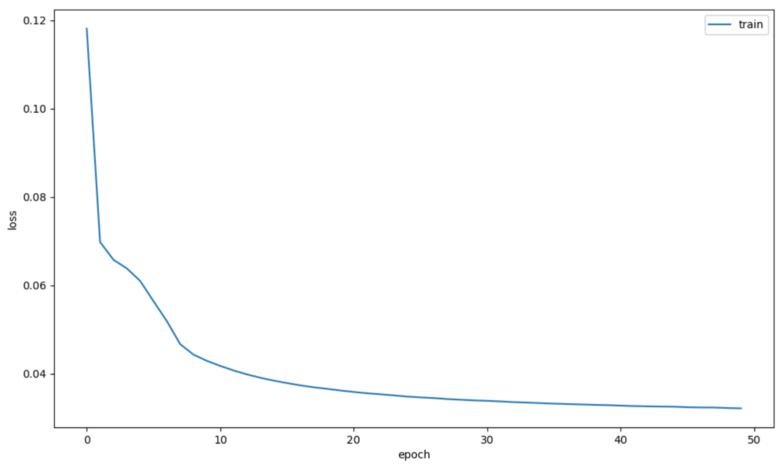

Figure 1: Loss plot (bottleneck = 4)

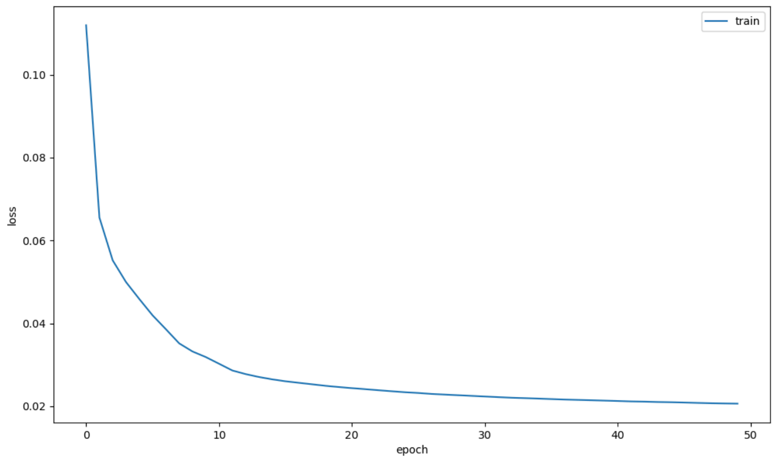

Figure 2: Loss plot (bottleneck = 8)

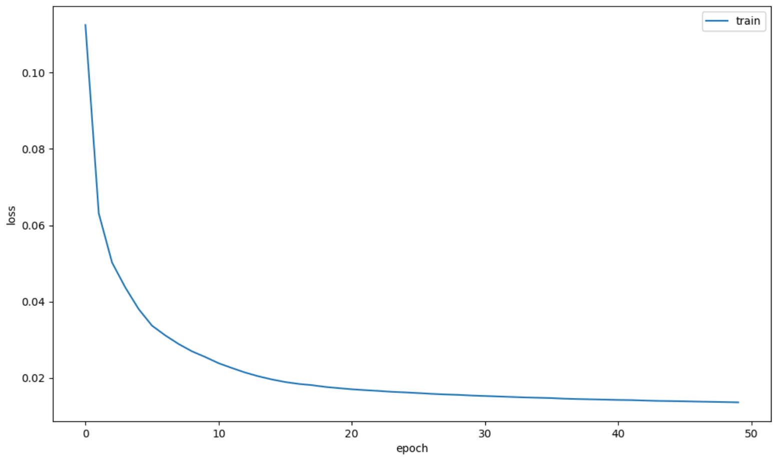

Figure 3: Loss plot (bottleneck = 16)

From the plots, the loss converges at a very similar rate for bottlenecks 8 and 16. In Figure 1, we see that loss begins to plateau a bit earlier for bottleneck = 4 and the final loss point plotted is greater than that of the others.

Bottleneck 8 and 16 performed better than 4 when it came to reconstructing images from the encoded representation, reinforcing the findings from the loss charts.

For the denoising task, I expected that the models at lower bottleneck should have performed better, and this was the case. However, the results produced from the models generally did not look like the pre-noise input. With higher level bottlenecks, it was hard to make out the original input character and the characters tended to be outside the domain (randomness vs an actual number) [Figure 4].

I think this behavior could be due to:

1. Training too long. Perhaps 50 epochs is too long and the model is memorizing rather than generalizing.
2. Too many trainable parameters on such a (relatively) small dataset.

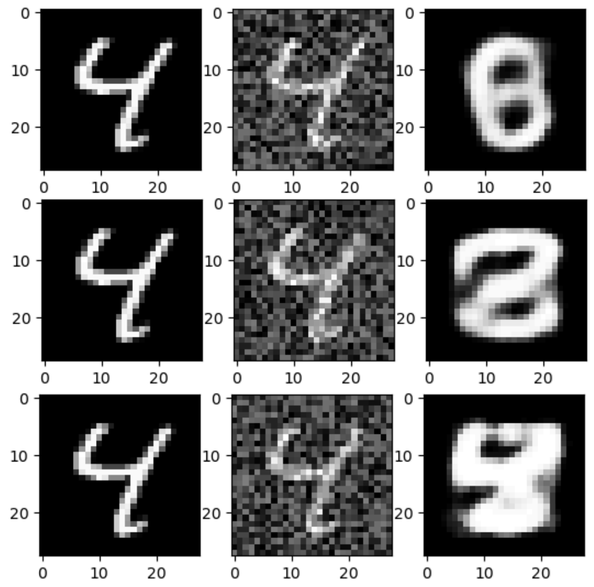

Figure 4: Denoising at bottlenecks [4, 8, 16], top to bottom. Remaining hyperparameters are the same. Note how de-noised characters become less distinguishable as an MNIST number with increasing bottleneck size.

The interpolations worked as expected at all bottlenecks.