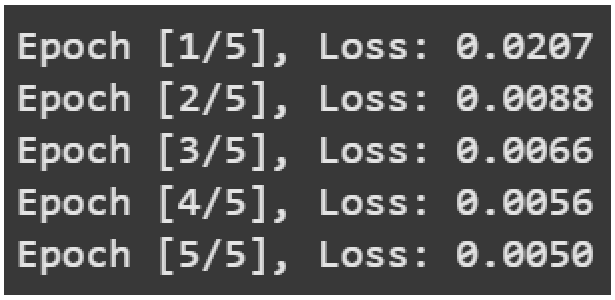
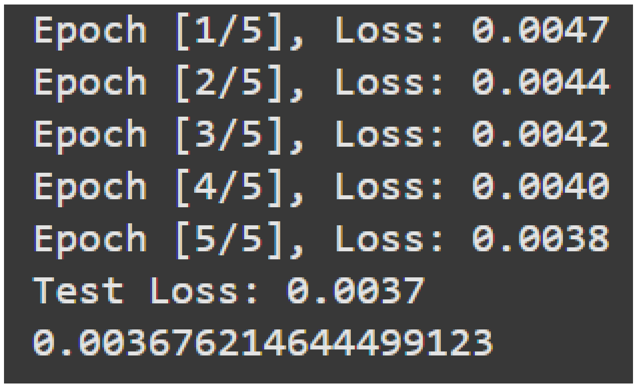
1.4 Deciding Loss Function

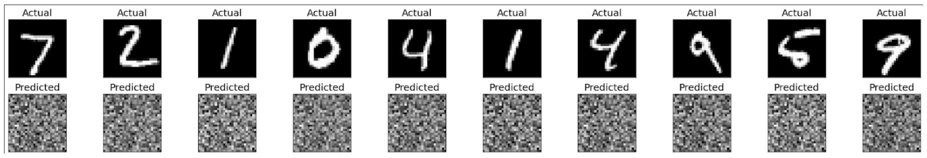
The loss functions that I tried on my model are mean squared error, L1 Loss, and BCE Loss. The most suitable loss function to use for this model is BCE Loss. Out of the loss functions I tried, this is the loss function that produced the most accurate and clear images that construct the handwritten numbers. Despite the higher loss values, I believe the images generated using BCE Loss are the most clear so it is the best loss function for this situation.

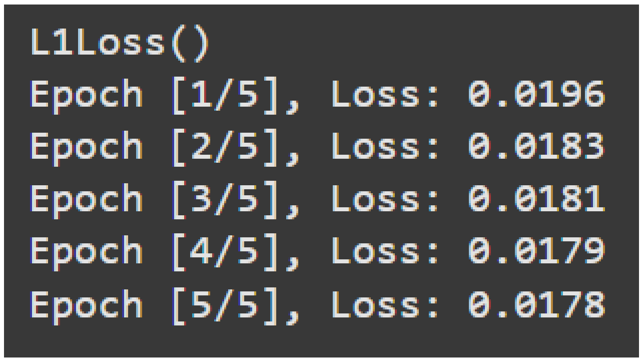
MSE Train Loss: 

```
Epoch [1/5], Loss: 0.0207
Epoch [2/5], Loss: 0.0088
Epoch [3/5], Loss: 0.0066
Epoch [4/5], Loss: 0.0056
Epoch [5/5], Loss: 0.0050
```

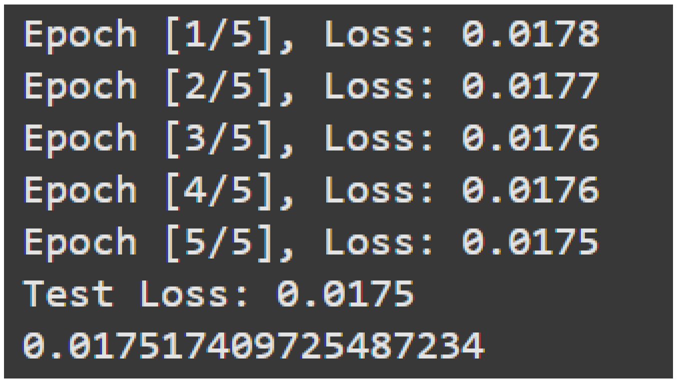
MSE Test Loss: 

```
Epoch [1/5], Loss: 0.0047
Epoch [2/5], Loss: 0.0044
Epoch [3/5], Loss: 0.0042
Epoch [4/5], Loss: 0.0040
Epoch [5/5], Loss: 0.0038
Test Loss: 0.0037
0.003676214644499123
```

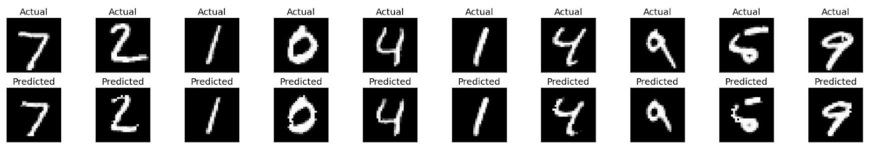
MSE Image Generation 

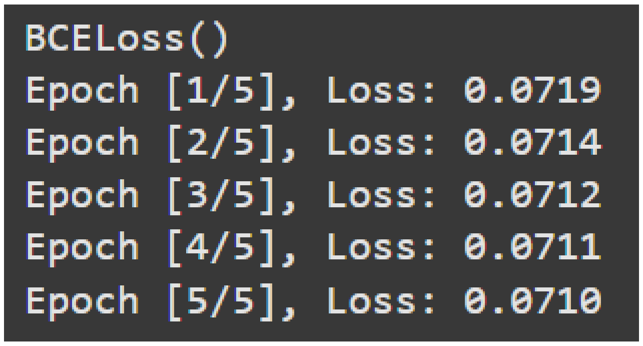
L1Loss Train Loss: 

```
L1Loss()
Epoch [1/5], Loss: 0.0196
Epoch [2/5], Loss: 0.0183
Epoch [3/5], Loss: 0.0181
Epoch [4/5], Loss: 0.0179
Epoch [5/5], Loss: 0.0178
```

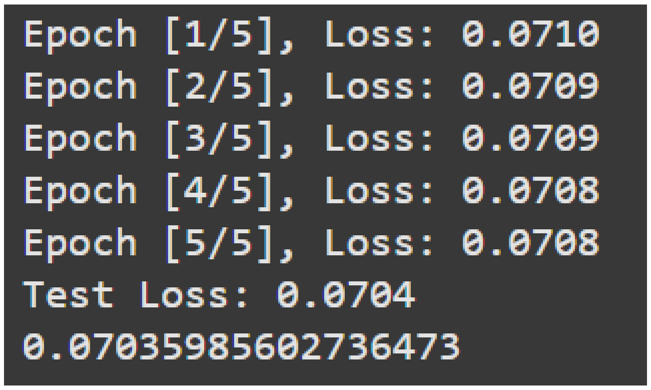
L1Loss Test Loss: 

```
Epoch [1/5], Loss: 0.0178
Epoch [2/5], Loss: 0.0177
Epoch [3/5], Loss: 0.0176
Epoch [4/5], Loss: 0.0176
Epoch [5/5], Loss: 0.0175
Test Loss: 0.0175
0.017517409725487234
```

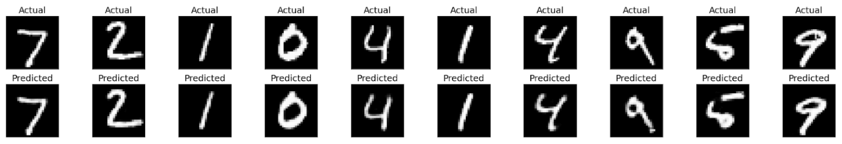
L1Loss Image Generation: 

BCELoss Train Loss: 

```
BCELoss()
Epoch [1/5], Loss: 0.0719
Epoch [2/5], Loss: 0.0714
Epoch [3/5], Loss: 0.0712
Epoch [4/5], Loss: 0.0711
Epoch [5/5], Loss: 0.0710
```

BCELoss Test Loss: 

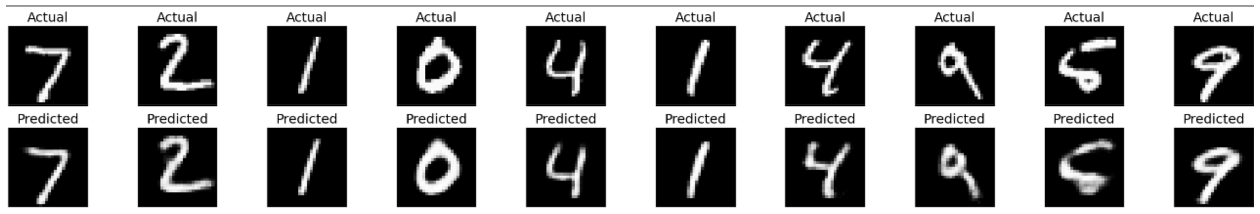
```
Epoch [1/5], Loss: 0.0710
Epoch [2/5], Loss: 0.0709
Epoch [3/5], Loss: 0.0709
Epoch [4/5], Loss: 0.0708
Epoch [5/5], Loss: 0.0708
Test Loss: 0.0704
0.07035985602736473
```

BCELoss Image Generation: 

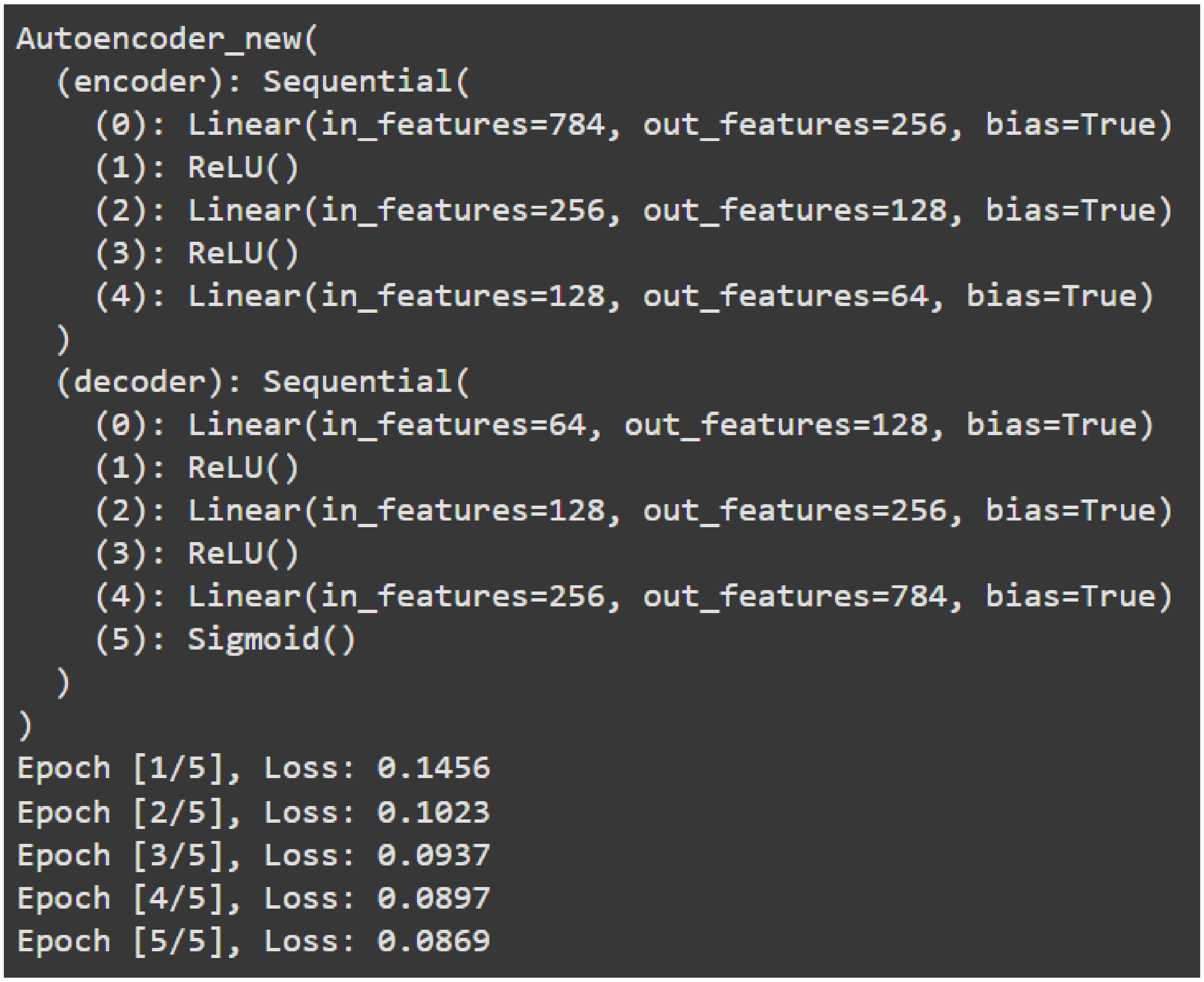
1.5 Adding Layers to Autoencoder

I developed three different models to test the autoencoder's performance. The first model I trained had an extra Linear layer to both the encoder and decoder with 256 intermediate features and an extra ReLU layer after the final linear layer in the encoder. This first model didn't result in a significant improvement in performance as the decoded images were more fuzzy. The second model I trained removed the extra ReLU layer that I added in the first model. This second architecture performed worse than the first model resulting in more fuzzy images. The third architecture I tried had a middle layer with 512 features, but also provided an output that was fuzzier than the original model. My hypothesis is that adding extra layers to the encoder/decoder overcomplicated the model and made it more difficult to result in strong model performance without additional epochs of training.

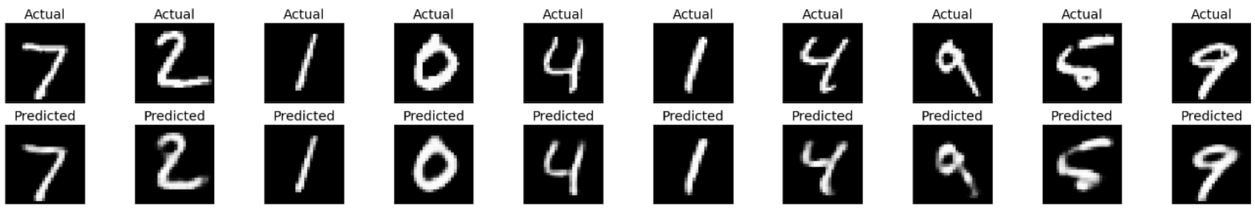
Architecture one:



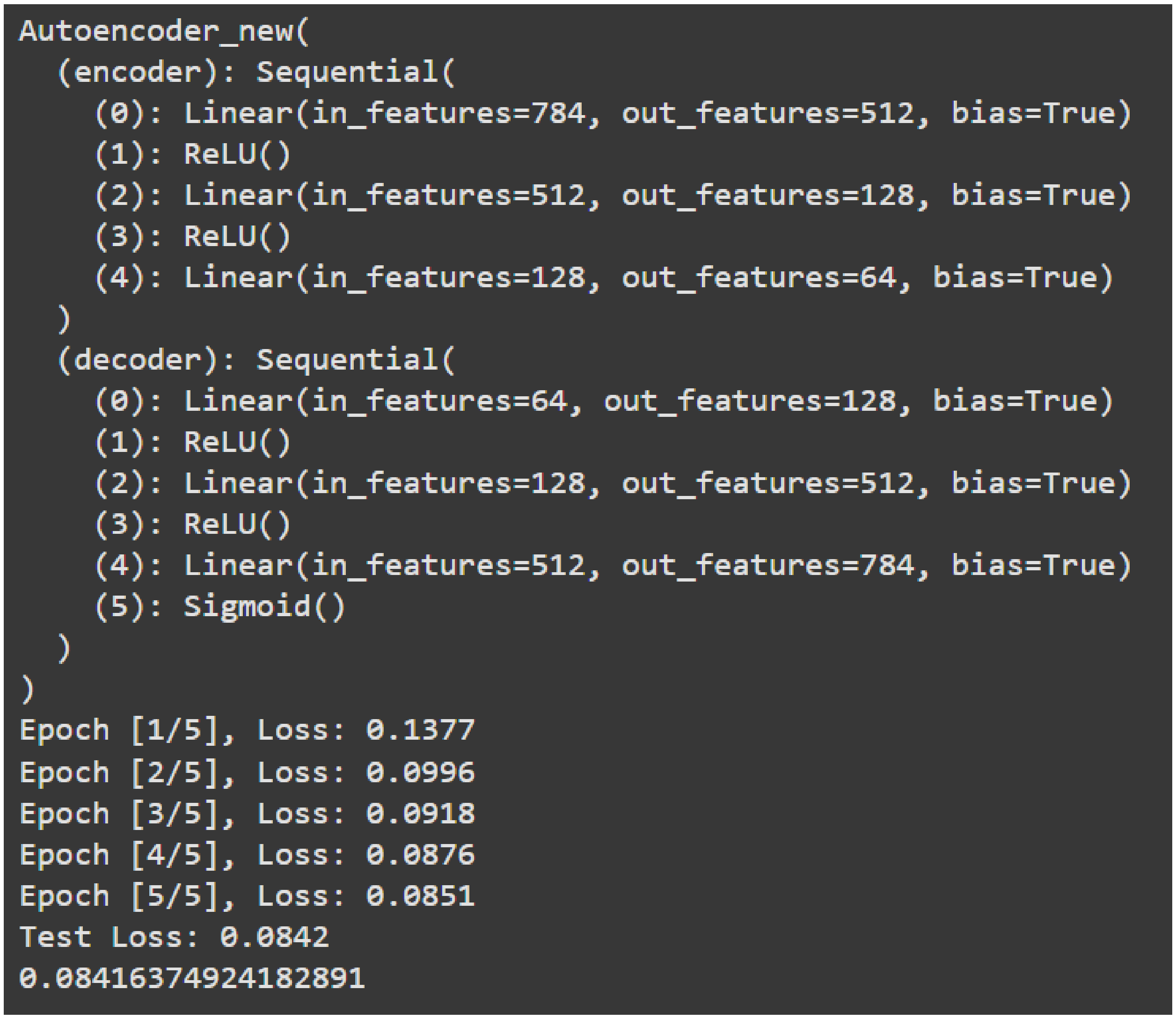
Architecture two:



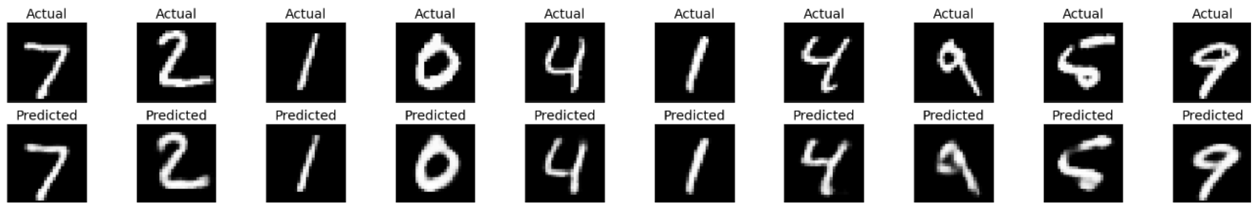
```
Autoencoder_new(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=64, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=784, bias=True)
    (5): Sigmoid()
  )
)
Epoch [1/5], Loss: 0.1456
Epoch [2/5], Loss: 0.1023
Epoch [3/5], Loss: 0.0937
Epoch [4/5], Loss: 0.0897
Epoch [5/5], Loss: 0.0869
```



Architecture three:

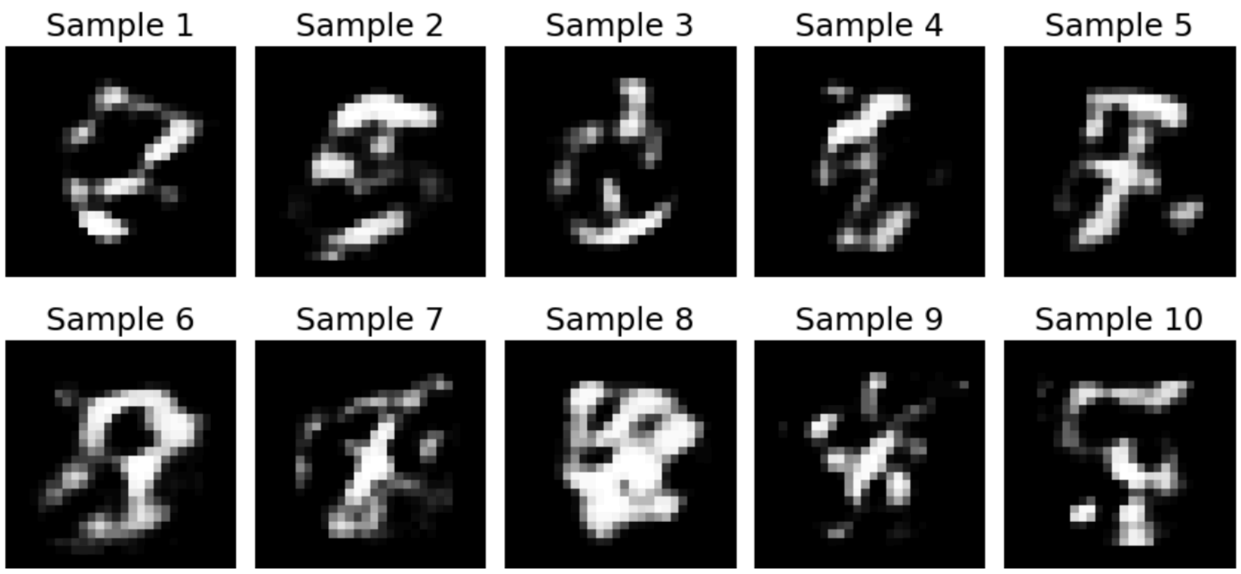


```
Autoencoder_new(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=64, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=784, bias=True)
    (5): Sigmoid()
  )
)
Epoch [1/5], Loss: 0.1377
Epoch [2/5], Loss: 0.0996
Epoch [3/5], Loss: 0.0918
Epoch [4/5], Loss: 0.0876
Epoch [5/5], Loss: 0.0851
Test Loss: 0.0842
0.08416374924182891
```



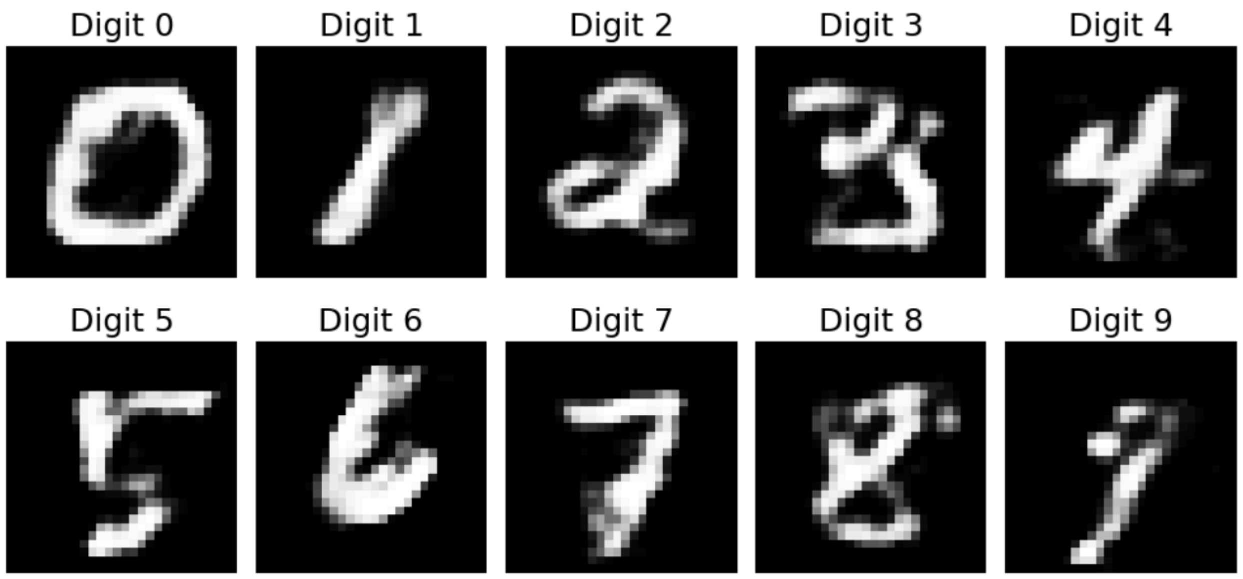
1.6 Decoding Training Data

After determining the mean and covariances of all the samples from our training data and generating 10 random samples following the calculated normal distribution, the outputs do not look like the images of hand drawn digits. While some samples vaguely look like digits (like sample 2 looking like a 5, sample 5 looking like a 7, and sample 6 looking like an 8), many other samples look blurry and unclear. This is likely due to the fact that the samples are generated from the means and covariances of all samples rather than from the individual distributions for each digit.



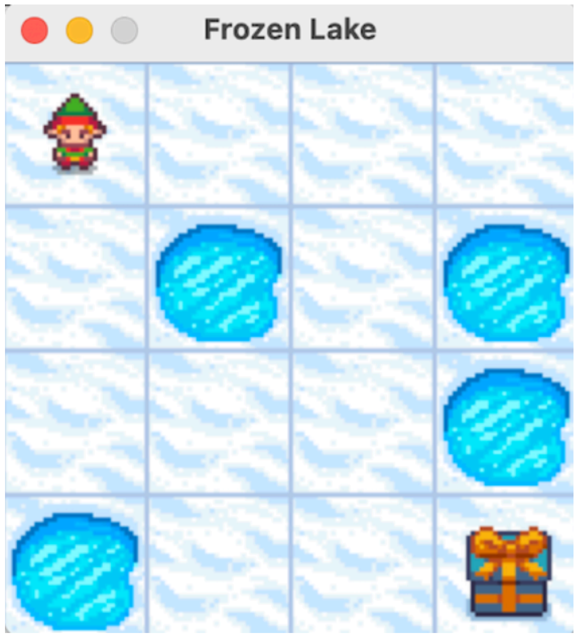
1.7 Decoding Using Mixture of Normals

After generating a new mixture distribution that contains individual distributions of every individual digit, the resulting output is significantly better than the previous output with decoded images that actually look like identifiable digits. This is a result that I expected because by creating individual distributions for each digit, the multivariate distribution better represents the relationship between pixels for an individual readable digit instead of modeling the distribution of all digits across the dataset.



2.1 Frozen Lake Environment

There are 4 possible actions that can be taken by the agent, representing traveling in each of the cardinal directions (UP, DOWN, LEFT, RIGHT). There are 4 possible states in this game that represent the space that the agent is standing on. These states are S (representing the starting point), G (the goal point), F (a frozen space, where it is safe to walk), and H (a hole in the ice, where it is not safe to walk). The terminal states are G (if the agent successfully reaches the goal) and H (if the agent falls into a hole). The minimum possible length of a reward sequence is 2 (if the agent moves down then right or right and then down) where the agent falls into the closest hole resulting in a reward of 0 with a sequence of 2 actions. The longest possible length reward sequence is infinite because the agent could never fall into a hole or reach the goal. However, due to the Frozen Lake Environment's implementation, there is a maximum reward sequence length of 100 for the 4x4 environment.



2.6 Q Table Initialization

Observing the results of training the Q-Learning algorithm, it seems like the Q table initialized with 0 performs extremely poorly when there are only 500 training episodes. During training, it was only able to have a positive reward in a single training episode, and that single success wasn't enough to set the Q table values to appropriately direct the agent toward the goal successfully resulting in a win rate of 0%. However, as the number of training episodes increased (especially at 5000 and 10000), Q-Learning was able to teach the agent to reach the goal successfully with a win rate of 63%. Here are the results of the 0 initialization Q-table (random initialization below results):

```
-------------------- init_method = zeros --------------------
number of wins: 0.0 out of 10000 total episodes

Q table we learned for init_method=zeros, n_episodes=500:
Left    Down    Right   Up      State   best_action
0       0.0     0.0     0.0     0.0     S       Left
1       0.0     0.0     0.0     0.0     F       Left
2       0.0     0.0     0.0     0.0     F       Left
3       0.0     0.0     0.0     0.0     F       Left
4       0.0     0.0     0.0     0.0     F       Left
5       0.0     0.0     0.0     0.0     H
6       0.0     0.0     0.0     0.0     F       Left
7       0.0     0.0     0.0     0.0     H
8       0.0     0.0     0.0     0.0     F       Left
9       0.0     0.0     0.0     0.0     F       Left
10      0.0     0.0     0.0     0.0     F       Left
11      0.0     0.0     0.0     0.0     H
12      0.0     0.0     0.0     0.0     H
13      0.0     0.0     0.0     0.0     F       Left
14      0.0     0.0     0.5     0.0     F       Right
15      0.0     0.0     0.0     0.0     G

winrate, optimal numsteps = (0.0, nan)
-------------------

number of wins: 0.0 out of 10000 total episodes
Q table we learned for init_method=zeros, n_episodes=1000:
Left    Down    Right   Up      State   best_action
0       0.00    0.0     0.00    0.0     S       Left
1       0.00    0.0     0.00    0.0     F       Left
2       0.00    0.0     0.00    0.0     F       Left
3       0.00    0.0     0.00    0.0     F       Left
4       0.00    0.0     0.00    0.0     F       Left
```

```
5       0.00    0.0     0.00    0.0     H
6       0.00    0.0     0.00    0.0     F       Left
7       0.00    0.0     0.00    0.0     H
8       0.00    0.0     0.00    0.0     F       Left
9       0.00    0.0     0.00    0.0     F       Left
10      0.24    0.0     0.00    0.0     F       Left
11      0.00    0.0     0.00    0.0     H
12      0.00    0.0     0.00    0.0     H
13      0.00    0.0     0.00    0.0     F       Left
14      0.00    0.0     0.75    0.0     F       Right
15      0.00    0.0     0.00    0.0     G
winrate, optimal numsteps = (0.0, nan)
-------------------

number of wins: 6351.0 out of 10000 total episodes
Q table we learned for init_method=zeros, n_episodes=5000:
Left    Down    Right   Up      State   best_action
0       0.27    0.18    0.14    0.10    S       Left
1       0.04    0.08    0.08    0.10    F       Up
2       0.07    0.08    0.09    0.06    F       Right
3       0.06    0.05    0.02    0.07    F       Up
4       0.30    0.27    0.13    0.17    F       Left
5       0.00    0.00    0.00    0.00    H
6       0.01    0.01    0.13    0.01    F       Right
7       0.00    0.00    0.00    0.00    H
8       0.21    0.13    0.14    0.39    F       Up
9       0.23    0.37    0.12    0.21    F       Down
10      0.24    0.17    0.09    0.12    F       Left
11      0.00    0.00    0.00    0.00    H
12      0.00    0.00    0.00    0.00    H
13      0.14    0.04    0.63    0.31    F       Right
14      0.41    0.45    0.79    0.44    F       Right
15      0.00    0.00    0.00    0.00    G
winrate, optimal numsteps = (0.6351, 39.27145331443867)
-------------------

number of wins: 6295.0 out of 10000 total episodes
Q table we learned for init_method=zeros, n_episodes=10000:
Left    Down    Right   Up      State   best_action
0       0.18    0.14    0.09    0.10    S       Left
1       0.09    0.11    0.02    0.11    F       Down
2       0.11    0.09    0.07    0.07    F       Left
3       0.01    0.02    0.06    0.06    F       Right
4       0.26    0.13    0.12    0.05    F       Left
```

```
5       0.00    0.00    0.00    0.00    H
6       0.10    0.00    0.07    0.05    F       Left
7       0.00    0.00    0.00    0.00    H
8       0.05    0.10    0.02    0.36    F       Up
9       0.24    0.30    0.07    0.04    F       Down
10      0.14    0.31    0.10    0.06    F       Down
11      0.00    0.00    0.00    0.00    H
12      0.00    0.00    0.00    0.00    H
13      0.12    0.18    0.67    0.30    F       Right
14      0.57    0.67    0.51    0.49    F       Down
15      0.00    0.00    0.00    0.00    G
winrate, optimal numsteps = (0.6295, 38.5208895949166)
-------------------
```

When the Q table was initialized with random values, it was able to achieve significant results much faster than the 0 initialization. In the first training cycle with only 500 training episodes, it was able to achieve a win rate of 50%. As the number of training cycles increased further, the agent performed better and better, achieving the result of a win rate close to 75% after 10000 training episodes. However, it does seem like some agents learned to try to go left from the starting square as the best action, leading to progress only when they "slip" on the ice. This result may be because there isn't a penalty for long paths. The results of the random initialization trials are shown below:

```
-------------------- init_method = random --------------------
number of wins: 5097.0 out of 10000 total episodes
Q table we learned for init_method=random, n_episodes=500:
Left    Down    Right   Up      State   best_action
0       0.19    0.19    0.19    0.16    S       Left
1       0.07    0.02    0.06    0.16    F       Up
2       0.11    0.09    0.09    0.11    F       Left
3       0.07    0.12    0.07    0.12    F       Down
4       0.20    0.17    0.13    0.06    F       Left
5       0.00    0.00    0.00    0.00    H
6       0.29    0.02    0.05    0.01    F       Left
7       0.00    0.00    0.00    0.00    H
8       0.23    0.11    0.16    0.25    F       Up
9       0.18    0.47    0.02    0.15    F       Down
10      0.36    0.14    0.04    0.05    F       Left
11      0.00    0.00    0.00    0.00    H
12      0.00    0.00    0.00    0.00    H
13      0.22    0.32    0.65    0.27    F       Right
14      0.60    0.83    0.61    0.40    F       Down
15      0.00    0.00    0.00    0.00    G
winrate, optimal numsteps = (0.5097, 34.40670982931136)
```

```
-------------------

number of wins: 5969.0 out of 10000 total episodes
Q table we learned for init_method=random, n_episodes=1000:
Left    Down    Right   Up      State   best_action
0       0.14    0.08    0.09    0.12    S       Left
1       0.02    0.05    0.07    0.08    F       Up
2       0.06    0.06    0.07    0.07    F       Right
3       0.04    0.03    0.03    0.06    F       Up
4       0.17    0.03    0.07    0.06    F       Left
5       0.00    0.00    0.00    0.00    H
6       0.09    0.02    0.04    0.03    F       Left
7       0.00    0.00    0.00    0.00    H
8       0.07    0.05    0.10    0.25    F       Up
9       0.11    0.39    0.11    0.12    F       Down
10      0.32    0.06    0.30    0.08    F       Left
11      0.00    0.00    0.00    0.00    H
12      0.00    0.00    0.00    0.00    H
13      0.09    0.30    0.46    0.10    F       Right
14      0.33    0.53    0.41    0.82    F       Up
15      0.00    0.00    0.00    0.00    G
winrate, optimal numsteps = (0.5969, 44.54146423186464)
-------------------

number of wins: 0.0 out of 10000 total episodes

Q table we learned for init_method=random, n_episodes=5000:
Left    Down    Right   Up      State   best_action
0       0.17    0.15    0.16    0.16    S       Left
1       0.08    0.11    0.06    0.11    F       Down
2       0.10    0.09    0.10    0.11    F       Up
3       0.07    0.05    0.08    0.09    F       Up
4       0.19    0.16    0.14    0.02    F       Left
5       0.00    0.00    0.00    0.00    H
6       0.05    0.03    0.02    0.00    F       Left
7       0.00    0.00    0.00    0.00    H
8       0.11    0.07    0.13    0.18    F       Up
9       0.11    0.17    0.17    0.22    F       Up
10      0.09    0.06    0.06    0.17    F       Up
11      0.00    0.00    0.00    0.00    H
12      0.00    0.00    0.00    0.00    H
13      0.42    0.49    0.65    0.50    F       Right
14      0.52    0.87    0.49    0.87    F       Down
15      0.00    0.00    0.00    0.00    G
```

```
winrate, optimal numsteps = (0.0, nan)
-------------------

number of wins: 7498.0 out of 10000 total episodes
Q table we learned for init_method=random, n_episodes=10000:
Left    Down    Right   Up      State   best_action
0       0.22    0.18    0.12    0.14    S       Left
1       0.02    0.02    0.06    0.12    F       Up
2       0.05    0.07    0.09    0.09    F       Right
3       0.05    0.03    0.02    0.10    F       Up
4       0.32    0.18    0.14    0.07    F       Left
5       0.00    0.00    0.00    0.00    H
6       0.04    0.02    0.01    0.02    F       Left
7       0.00    0.00    0.00    0.00    H
8       0.10    0.15    0.04    0.43    F       Up
9       0.19    0.47    0.18    0.11    F       Down
10      0.27    0.15    0.10    0.10    F       Left
11      0.00    0.00    0.00    0.00    H
12      0.00    0.00    0.00    0.00    H
13      0.32    0.16    0.62    0.14    F       Right
14      0.58    0.91    0.59    0.57    F       Down
15      0.00    0.00    0.00    0.00    G
winrate, optimal numsteps = (0.7498, 41.69311816484396)
-------------------
```

2.7 Reward

By changing the reward function to include a penalty (and by including a penalty) for wandering around without reaching the goal, the win rate seems to increase dramatically from the Q-Learning model that didn't have penalties for falling in holes or meandering around the frozen lake. It seems like adjusting the penalty does influence the optimal path length, drastically reducing the length of the optimal path when the correct penalty values are chosen. It does seem that at some penalty values, the agent is actually incentivized to try not to move to avoid the penalty from falling into the hole. This means fine tuning for the penalty is required to improve the agent's win rate.

penalty = 0.1
```
winrate, optimal path length = (0.0239, 10.594142259414227)
```

penalty = 0.01
```
winrate, optimal path length = (0.8228, 49.55590666018473)
```

penalty = 0.005
```
winrate, optimal path length = (0.8228, 49.55590666018473)
```

penalty = 0.001
```
winrate, optimal path length = (0.0, nan)
```

penalty = 0.0005
```
winrate, optimal path length = (0.8237, 49.52264173849703)
```

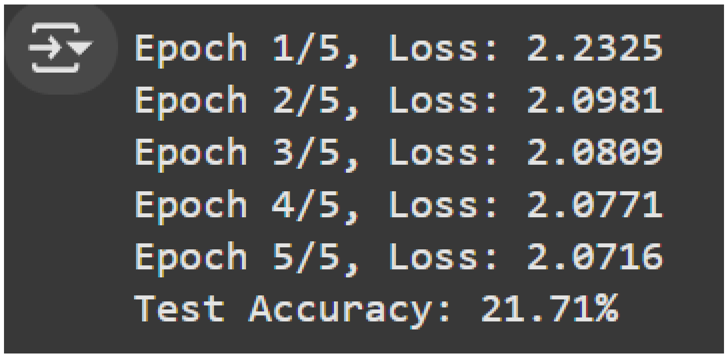
penalty = 0.0001
```
winrate, optimal path length = (0.8228, 49.55590666018473)
```

penalty = 0.00001
```
winrate, optimal path length = (0.8157, 94.1435576805198)
```

3.3 Basic Attention vs Self Attention

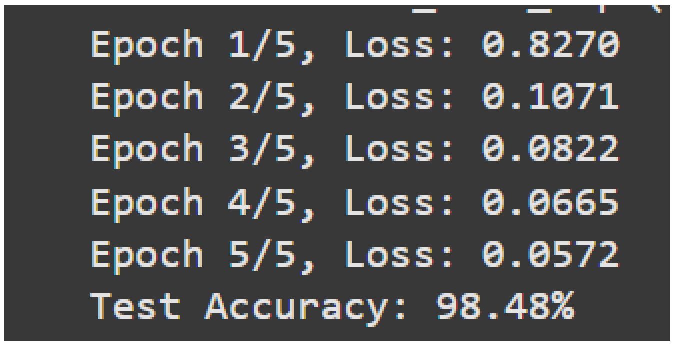
Basic attention achieved a training loss of 2.07 and a test accuracy of 21.71%. The self attention model was able to achieve a training loss of 0.06 and a test accuracy 98.48%. The self attention model clearly demonstrated a stronger performance than the basic attention model. I believe that this is the case because the query inputs in the self-attention model actually come from the input itself, so the self-attention model is able to better generalize to different inputs. The basic attention model on the other hand has to train its query parameters, making it difficult for the basic attention to model to generalize to different inputs from the same distribution.

Basic Attention:



```
Epoch 1/5, Loss: 2.2325
Epoch 2/5, Loss: 2.0981
Epoch 3/5, Loss: 2.0809
Epoch 4/5, Loss: 2.0771
Epoch 5/5, Loss: 2.0716
Test Accuracy: 21.71%
```

Self Attention:



```
Epoch 1/5, Loss: 0.8270
Epoch 2/5, Loss: 0.1071
Epoch 3/5, Loss: 0.0822
Epoch 4/5, Loss: 0.0665
Epoch 5/5, Loss: 0.0572
Test Accuracy: 98.48%
```