Instructions

- For all questions that are not graded only on the answer, show your work! Any problem without work shown will get no marks regardless of the correctness of the final answer.
- Please try to use a document preparation system such as LaTeX. *If you write your answers by hand, note that you risk losing marks if your writing is illegible without any possibility of regrade, at the discretion of the grader.*
- Submit your answers electronically via the course GradeScope. *Incorrectly assigned answers can be given 0 automatically at the discretion of the grader.* To assign answers properly, please:
  - Make sure that the top of the first assigned page is the question being graded.
  - Do not include any part of answer to any other questions within the assigned pages.
  - Assigned pages need to be placed in order.
  - For questions with multiple parts, the answers should be written in order of the parts within the question.
- In the code, each part to fill is referenced by a TODO and 'Not Implemented Error'
- Questions requiring written responses should be short and concise when necessary.Unnecessary wordiness will be penalized at the grader's discretion.
- Please sign the agreement below.
- It is your responsibility to follow updates to the assignment after release. All changes will be visible on Overleaf and Piazza.
- Any questions should be directed towards the TAs for this assignment: *Vitória Barin Pacela, Philippe Martin.*

For this assignment, the GitHub link is the following: https://github.com/philmar1/Teaching$_IFT6135---Assignment-3---H25$

I acknowledge I have read the above instructions and will abide by them throughout this assignment. I further acknowledge that any assignment submitted without the following form completed will result in no marks being given for this portion of the assignment..

Signature: E. Kreuzer

Name: Ethan Kreuzer

UdeM Student ID: 20221553

6135B-Assignment 3-Report

Ethan Kreuzer

March 2025

Question 1.6

```python
def validate():
    model.eval()
    validation_loss = 0
    with torch.no_grad():
        for data, _ in test_loader:
            data = data.to(device)
            recon_batch, mu, logvar = model(data)
            loss = loss_function(recon_batch, data, mu, logvar)
            validation_loss += loss.item()
    avg_val_loss = validation_loss / len(test_loader.dataset)
    print('====> Validation Loss: {:.4f}'.format(avg_val_loss))
    return avg_val_loss


if __name__ == "__main__":
    train_losses = []
    val_losses = []

    for epoch in range(1, args.epochs + 1):
        train_loss = train(epoch)
        val_loss = validate()
        train_losses.append(train_loss)
        val_losses.append(val_loss)

    final_val_loss = val_losses[-1]
    print('Final Validation Loss: {:.4f}'.format(final_val_loss))

    plt.figure()
    epochs = range(1, args.epochs + 1)
    plt.plot(epochs, train_losses, label='Training Loss')
    plt.plot(epochs, val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Validation Losses')
    plt.legend()

    save_path = "/home/ethan/IFT6135/IFT6135-2025/Teaching_IFT6135-Assignment3-H25/plots/q1_6"
    plt.savefig(save_path)
    print("Plot saved to:", save_path)
    torch.save(model.state_dict(), 'model.pt')
```

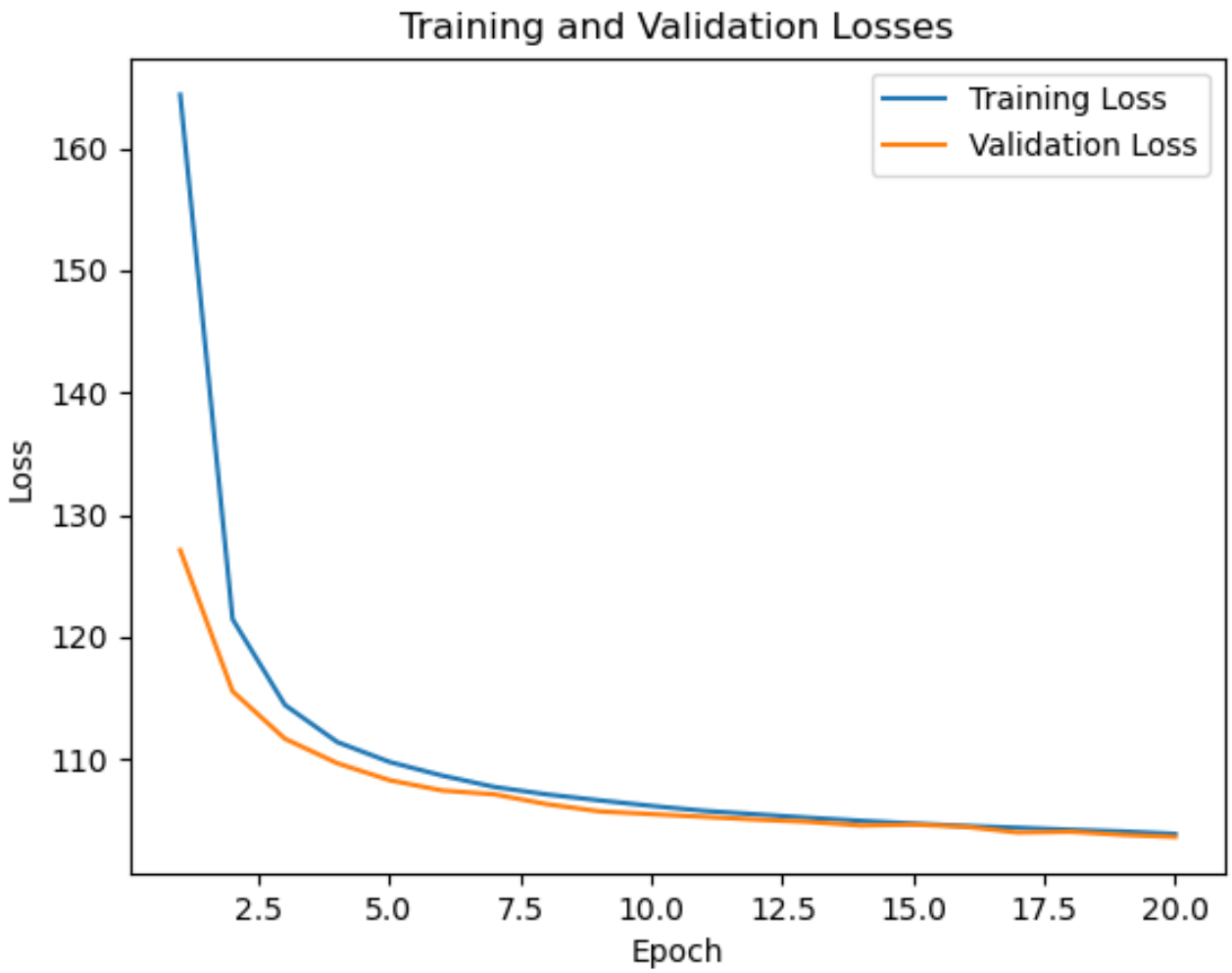

Figure 1: VAE Losses

The final Validation Loss was 103.6456

Question 1.7

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = VAE().to(device)
model.load_state_dict(torch.load("model.pt", map_location=device))
model.eval()


n_samples = 16
latent_dim = 20


with torch.no_grad():
    z = torch.randn(n_samples, latent_dim).to(device)
    generated = model.decode(z).cpu()


generated_images = generated.view(n_samples, 28, 28)


fig, axes = plt.subplots(4, 4, figsize=(8, 8))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(generated_images[i], cmap="gray")
    ax.axis("off")
plt.suptitle("Generated MNIST Samples", fontsize=16)
plt.tight_layout()
plt.show()
```

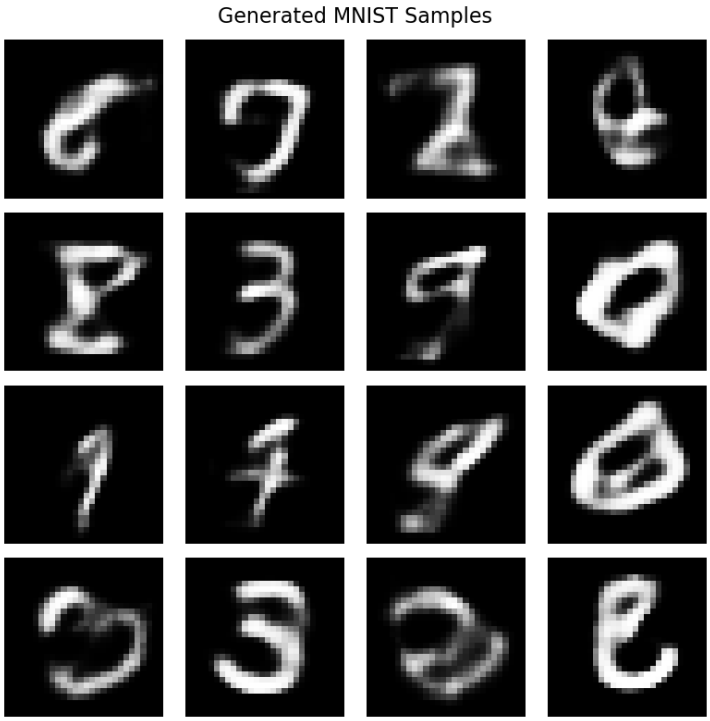

Figure 2: VAE Losses

The generated images are relatively realistic. They are somewhat blurry and also tend to have faded patches in the images, but some of the images are definitely identifiable as numbers while others are "close" to actual numbers. One can identify in these generated images the numbers: 1 and 3. Once can also identify images that resemble the numbers 0, 6, 8, 9 and 7.

Question 1.8

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = VAE().to(device)
model.load_state_dict(torch.load("model.pt", map_location=device))
model.eval()
latent_dim = 20
n_traversals = 5
epsilon_values = [0, -4, -2, 2, 4]

z_base = torch.randn(1, latent_dim).to(device)


def plot_latent_traversals(dim_range, title):
    n_rows = len(dim_range)
    fig, axes = plt.subplots(n_rows, n_traversals, figsize=(10, 20))
    for i, dim in enumerate(dim_range):
        for j, eps in enumerate(epsilon_values):

            z_modified = z_base.clone()
            z_modified[0, dim] += eps
            with torch.no_grad():
                generated = model.decode(z_modified)

            image = generated.view(28, 28).cpu().numpy()
            axes[i, j].imshow(image, cmap="gray")
            axes[i, j].axis("off")

    plt.suptitle(title, fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

plot_latent_traversals(range(0, 10), "Latent Traversals (Latent 1-10)")
plot_latent_traversals(range(10, 20), "Latent Traversals (Latent 11-20)")
```

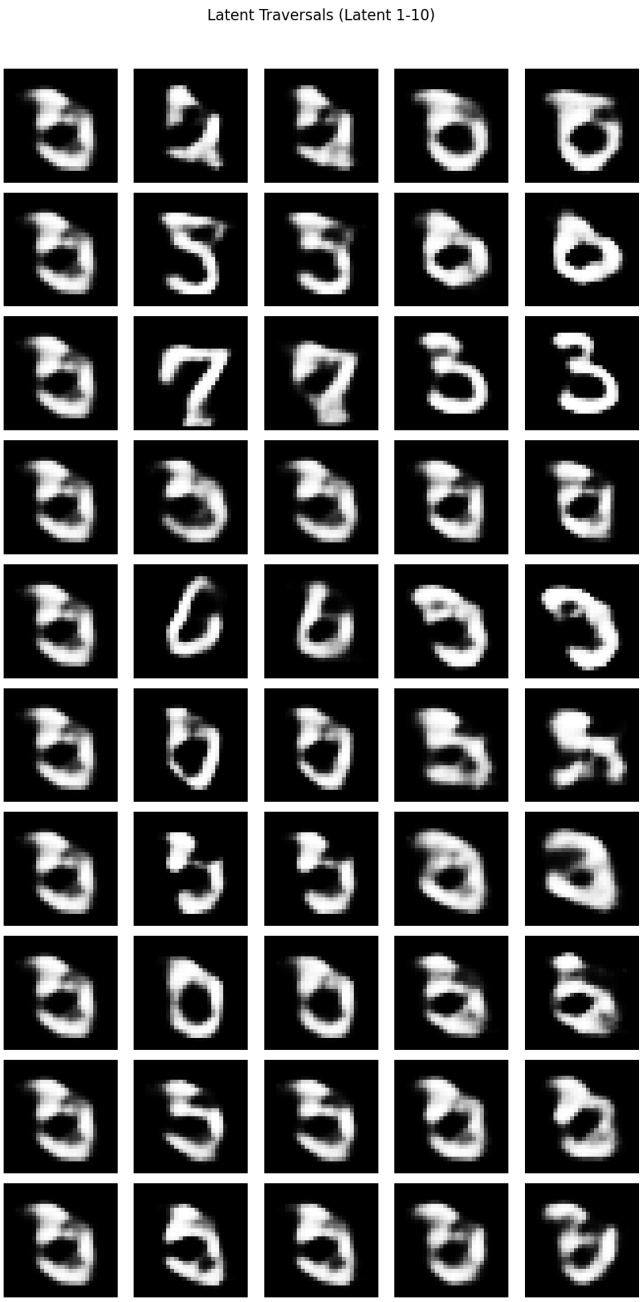

Figure 3: Latent 1-10

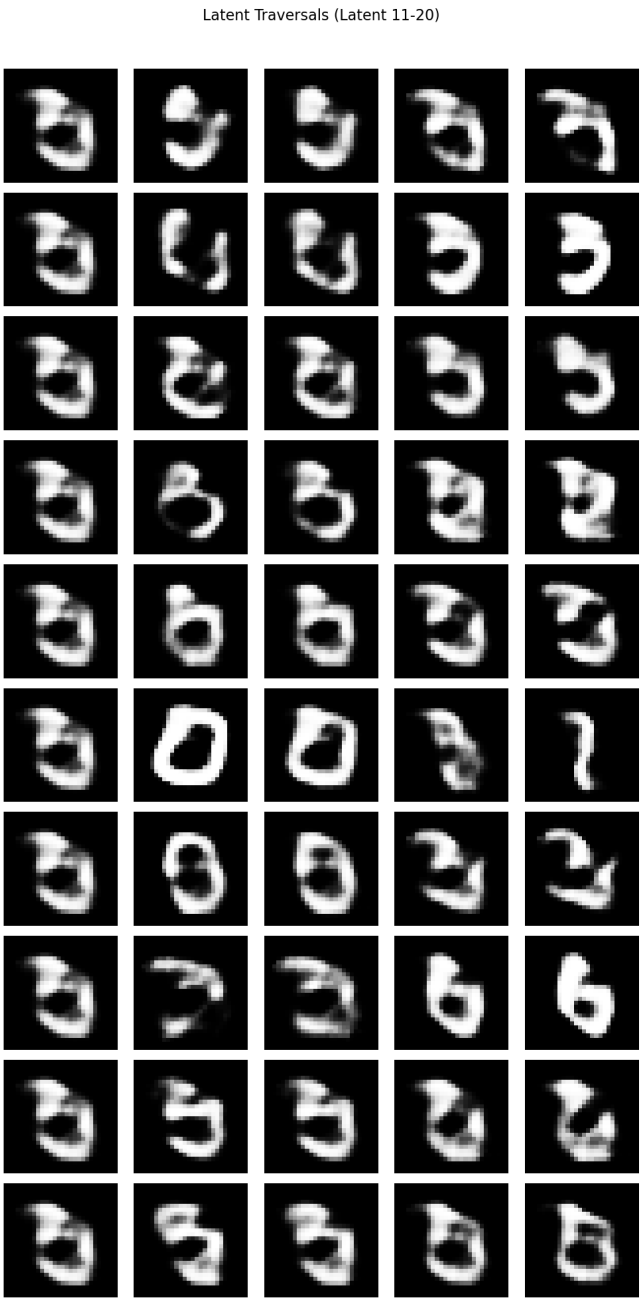

Figure 4: Latent 11-20

Figure 5: Dimension perturbations

The left-most image is a ground truth "control" image, resulting from an $\epsilon = 0$ dimension perturbation, while the four images to the right result from $\epsilon \in \{-4, -2, 2, 4\}$ perturbations. With regards to the disentangling of latent representations, these images convey that we do have entangled features. We can see that most of these dimensions affect the shape of the digit, as we can see digits like 3 and 6 appear pretty distinctly as some dimensions are perturbed, but we also see other aspects vary. For example, we often see the "thickness" and "blurriness" of the digit vary as the digit shape changes.

Question 1.9

```
1
2  z0 = torch.randn(1, 20).to(device)
3  z1 = torch.randn(1, 20).to(device)
4  alphas = np.linspace(0, 1, 11)
5
6  with torch.no_grad():
7      x0 = model.decode(z0).view(28, 28).cpu().numpy()
8      x1 = model.decode(z1).view(28, 28).cpu().numpy()
9
10 latent_interpolated_images = []
11
12 for alpha in alphas:
13
14     z_alpha = alpha*z0 + (1-alpha)*z1
15     with torch.no_grad():
16         x_alpha = model.decode(z_alpha).view(28, 28).cpu().numpy()
17     latent_interpolated_images.append(x_alpha)
18
19
20 data_interpolated_images = []
21 for alpha in alphas:
22     x_hat = alpha*x0 + (1-alpha)*x1
23     data_interpolated_images.append(x_hat)
24
25
26 fig, axes = plt.subplots(1, len(alphas), figsize=(15, 2))
27 for idx, ax in enumerate(axes):
28     ax.imshow(latent_interpolated_images[idx], cmap="gray")
29     ax.axis("off")
30     ax.set_title(f"  ={alphas[idx]:.1f}")
31 plt.suptitle("Interpolation in Latent Space")
32 plt.tight_layout()
33 plt.show()
34
35 fig, axes = plt.subplots(1, len(alphas), figsize=(15, 2))
36 for idx, ax in enumerate(axes):
37     ax.imshow(data_interpolated_images[idx], cmap="gray")
38     ax.axis("off")
39     ax.set_title(f"  ={alphas[idx]:.1f}")
40 plt.suptitle("Interpolation in Data Space")
41 plt.tight_layout()
42 plt.show()
```

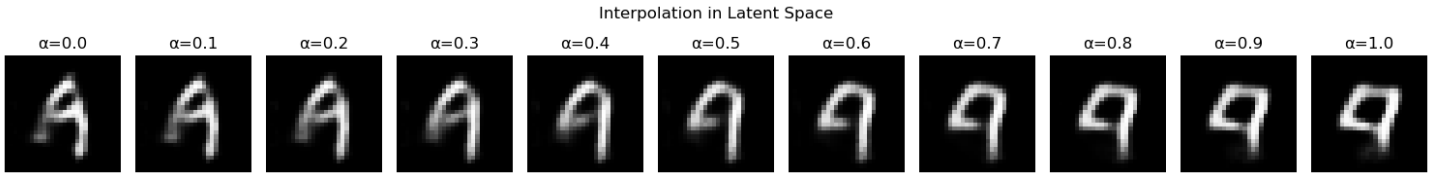

Figure 6: Interpolating in Latent Space

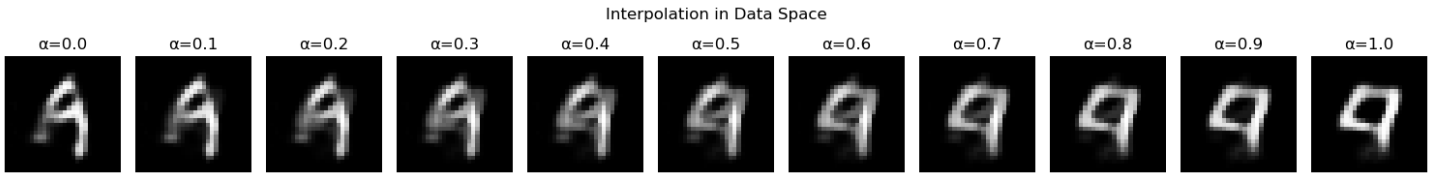

Figure 7: Interpolating in Data Space

In setting a) we are moving from the decoded z_1 to decoded z_0 by doing the computations in latent space and then decoding. In setting b) we are immediately decoding to our data space and then moving from the decoded z_1 to decoded z_0 in the data space.

We can see interpolating in latent space results in smoother transitions between images, likely due to the latent space being smoother than pixel space.

Question 2

0.1 Sampled images generated by each epoch

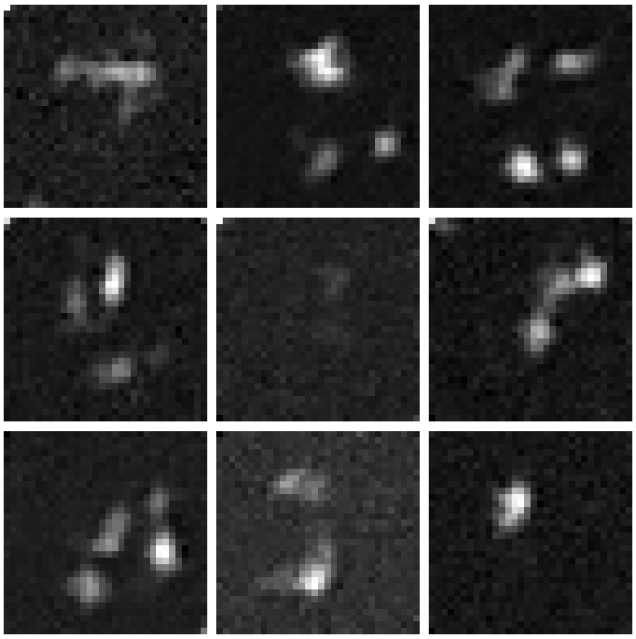

Figure 8: Epoch 0

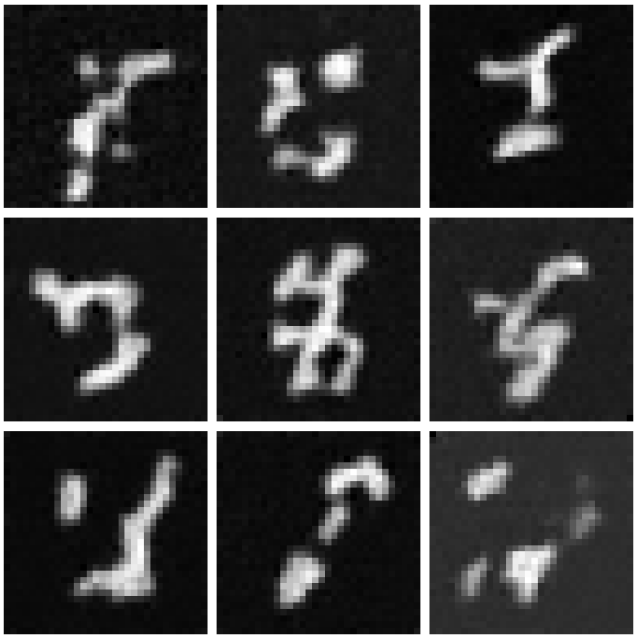

Figure 9: Epoch 2

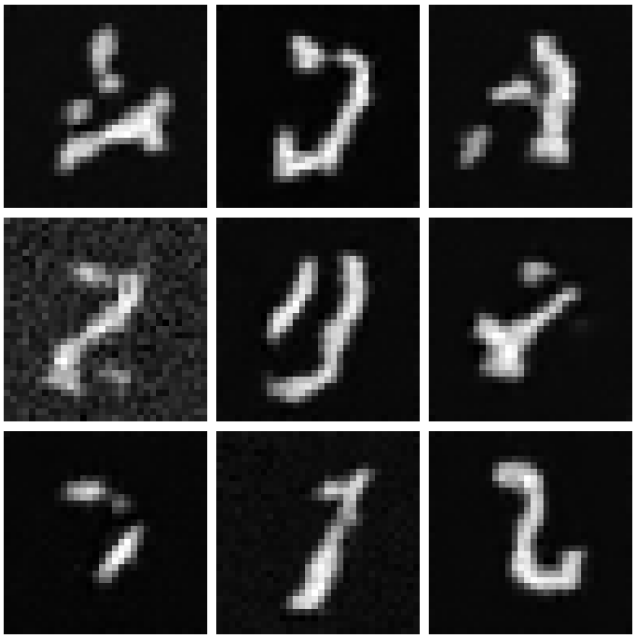

Figure 10: Epoch 4

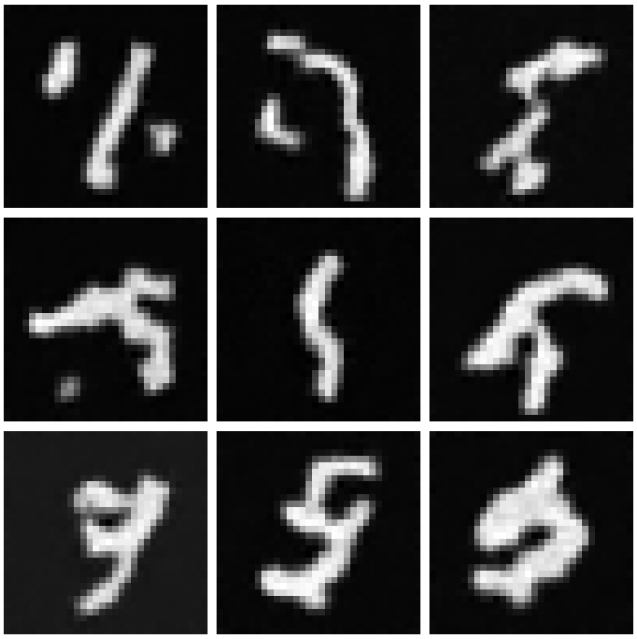

Figure 11: Epoch 6

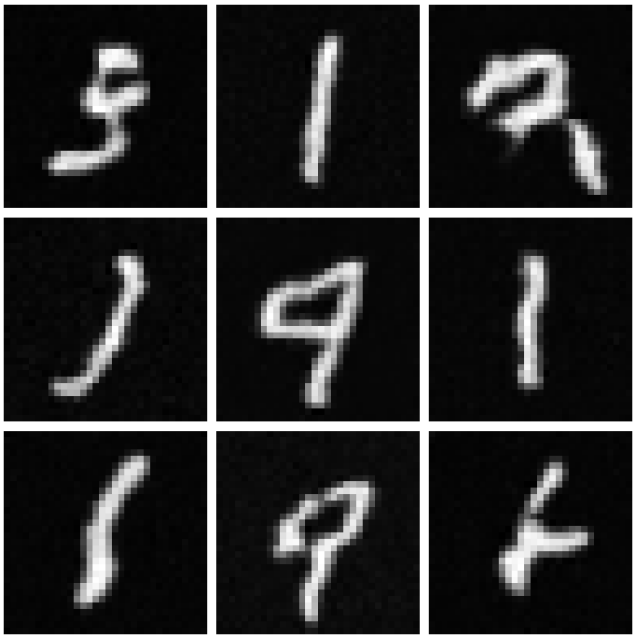

Figure 12: Epoch 8

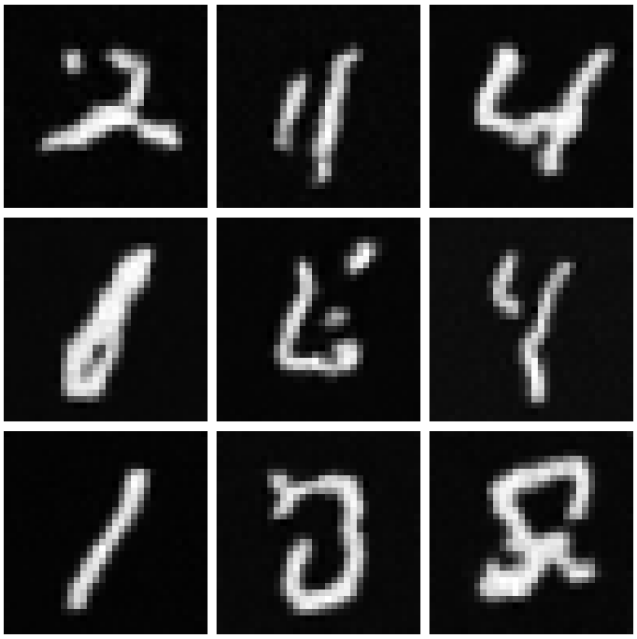

Figure 13: Epoch 10

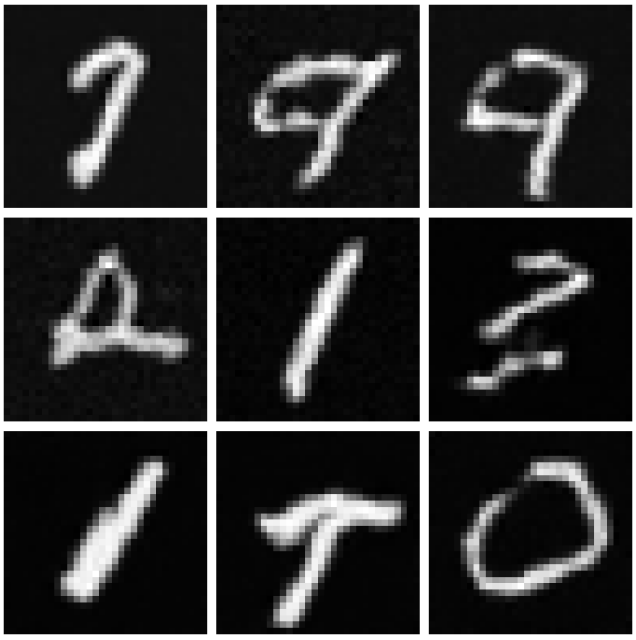

Figure 14: Epoch 12

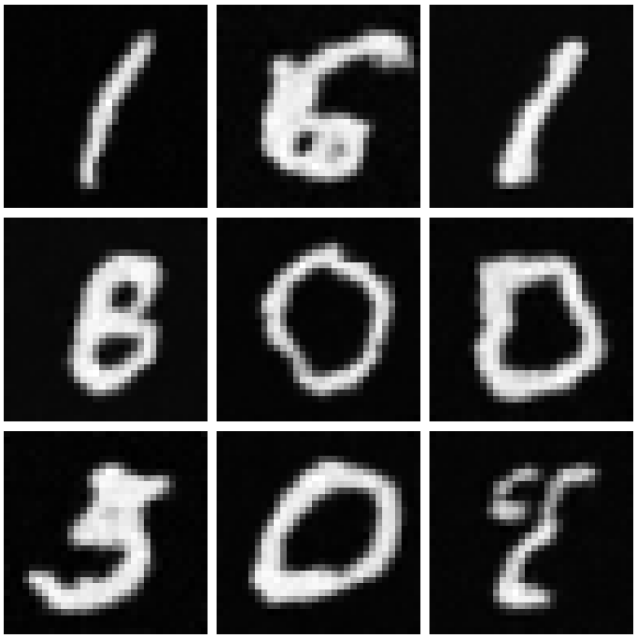

Figure 15: Epoch 14

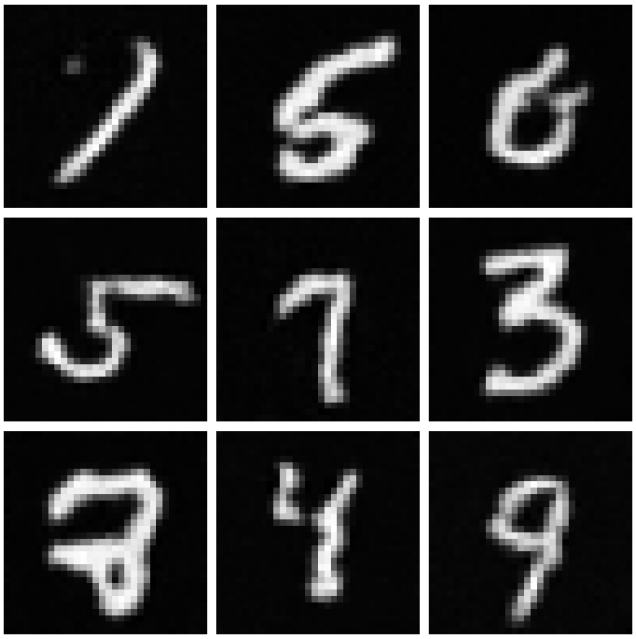

Figure 16: Epoch 16

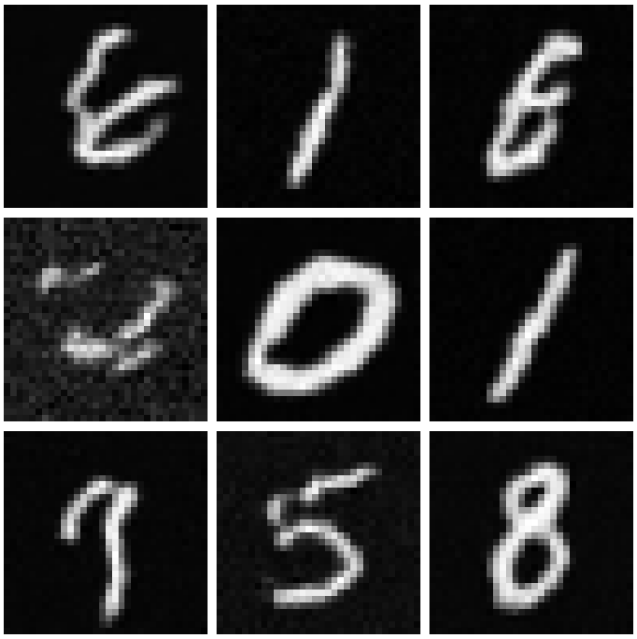

Figure 17: Epoch 18

We can see in the early epochs that the generated images are pretty uninterpretable. The model starts by producing "blobs" that do not at all resemble digits. Once we get to around epoch 8, the images start to resemble digits, but are still far from clean images. The generated images continue to improve as training progresses. Once we get to the end, the model is doing a decent job of generating digits. Some of the images are clearly identifiable as digits, like 1, 0, 5 and 8, but some of the images are still quite ambiguous.

A couple of possible improvements. A simple potential improvement could be to tune the actual diffusion steps. Currently we set 1000 steps but maybe we could do better with more steps. More involved solutions could be to change the scheduler. In this implementation we use a beta schedule with "self.beta = torch.linspace(0.0001, 0.02, n steps).to(device)", but we could use a different scheduler of noise which could yield better results, like a cosine schedule. We could also use a different loss metric, such as reweighting the loss based on the noise level. By reweighting the loss, the model is incentivized to pay more attention to accurately predicting noise in stages where errors are more critical and less in stages where the impact is less critical, which could lead to better images.

0.2 Images generated at different time steps of diffusion by trained model

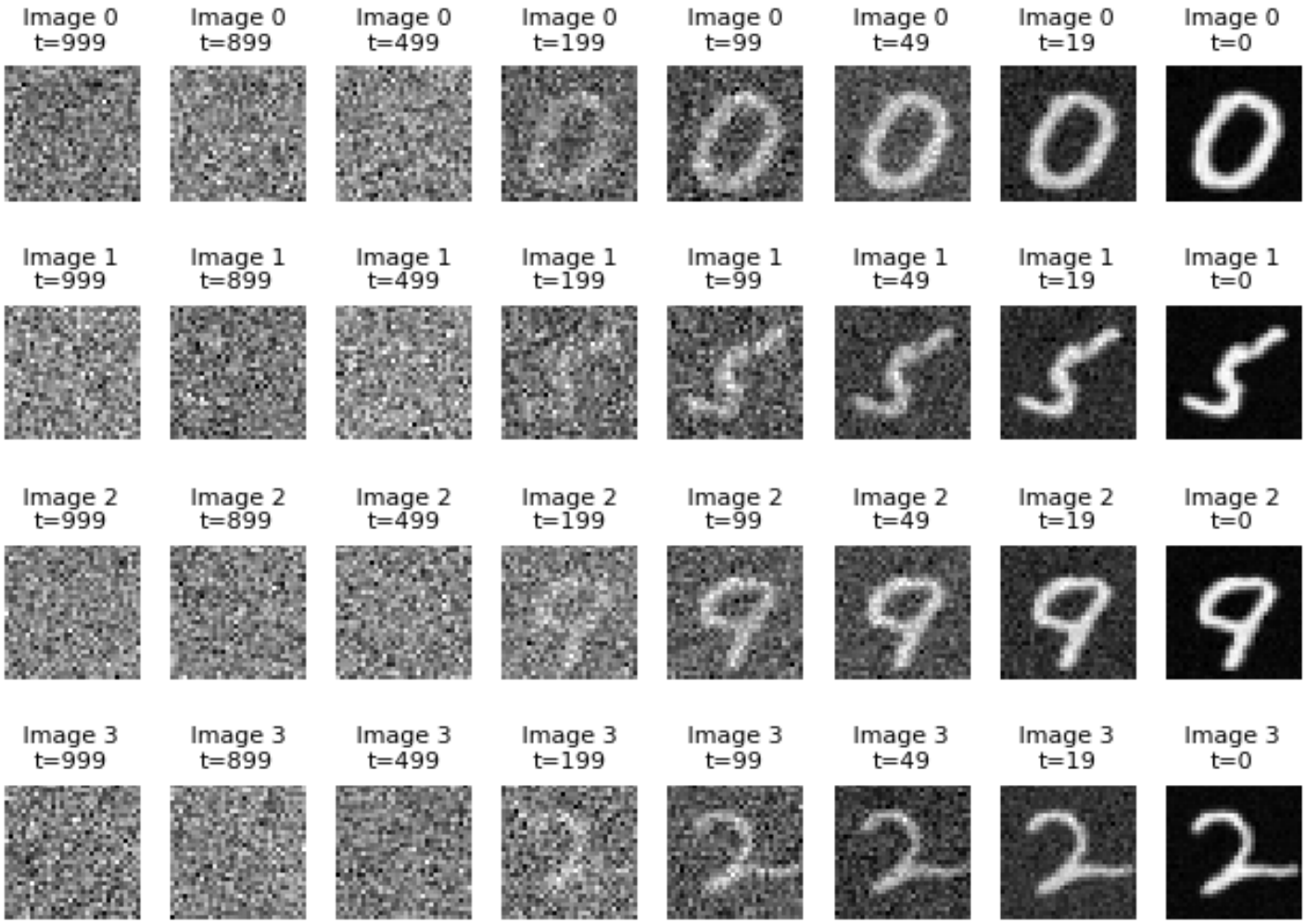

Figure 18: Images throughout reverse diffusion

We see at first that images are pure noise and the image remains super noisy for the majority of the diffusion process. Once we get to time step 200 out of 1000, we begin to be able to make out the form of the digits. It is really the last 100 steps that we start to be able to clearly identify the digit shapes and at the end the digits are clearly identifiable.

Question 3

0.3 Explain why is the model called Classifier Free and why Guidance

It's called "classifier-free" because the model doesn't rely on a separate classifier to steer the generation process. Instead, the authors jointly train a model that works both conditionally and unconditionally, where we condition on a class label corresponding to the class we want to generate. The "guidance" part comes from how the conditional and unconditional score estimates are combined to direct the sampling process.

0.4 According to the paper, what would be an alternative of classifier free ? Explain how would the loss change in this alternative compared to the original DDPM loss?

An alternative to classifier-free guidance is classifier guidance, which uses an external classifier trained on noisy images to steer the generation process. In classifier guidance, the diffusion model's score is modified at sampling time by subtracting a term proportional to the gradient of the log-probability from the classifier, specifically: $\epsilon_\theta(z_\lambda, c) – w * \sigma_\lambda * \nabla_{z_\lambda} log * p_\theta(c|z_\lambda)$. Meanwhile, the original DDPM loss is a denoising score matching objective that minimizes the squared error between the predicted noise and the true noise, without any classifier term.

0.5 Describe the sampled images generated by each epocs.

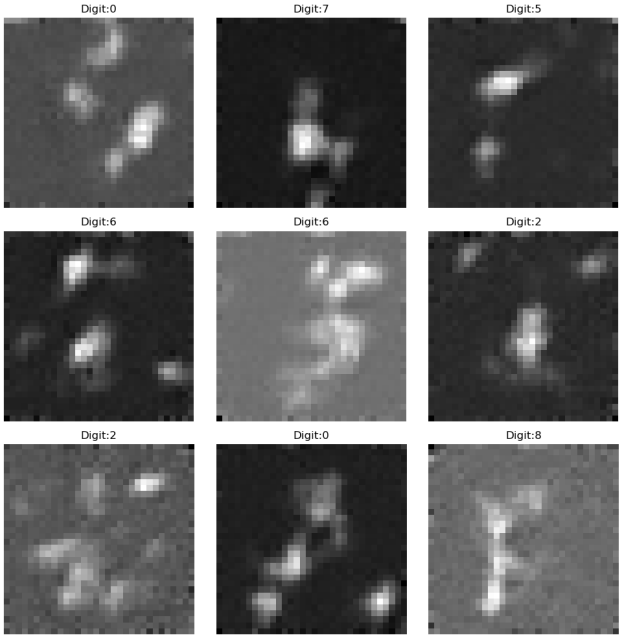

Figure 19: Epoch 0

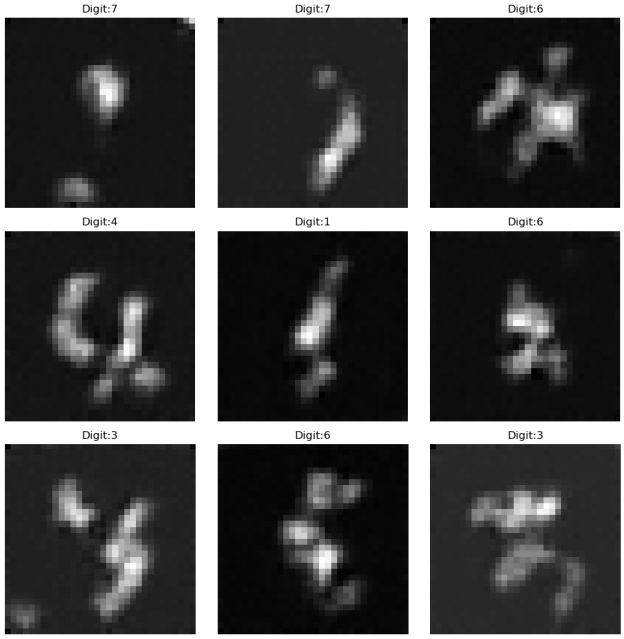

Figure 20: Epoch 1

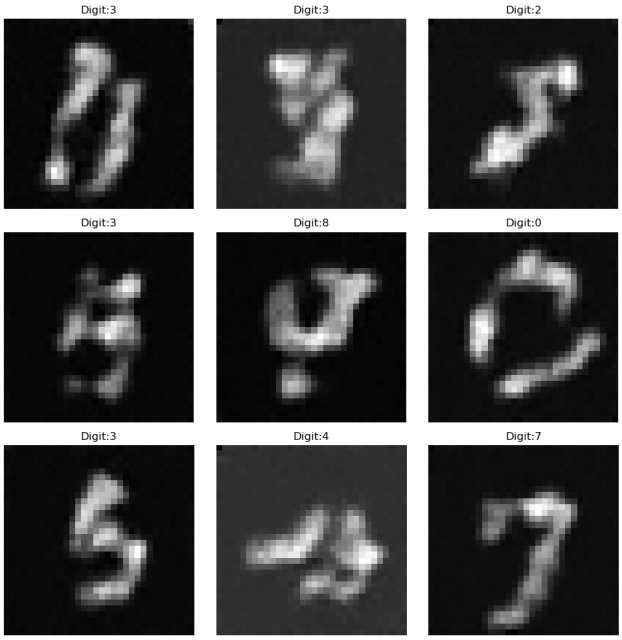

Figure 21: Epoch 2

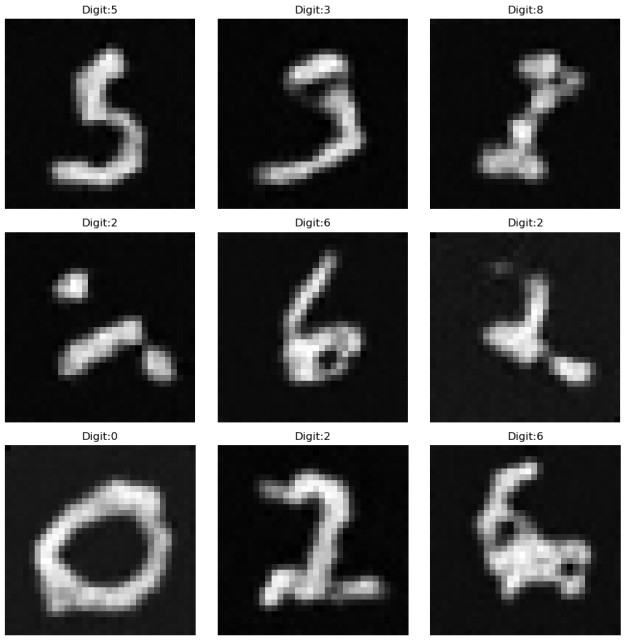

Figure 22: Epoch 3

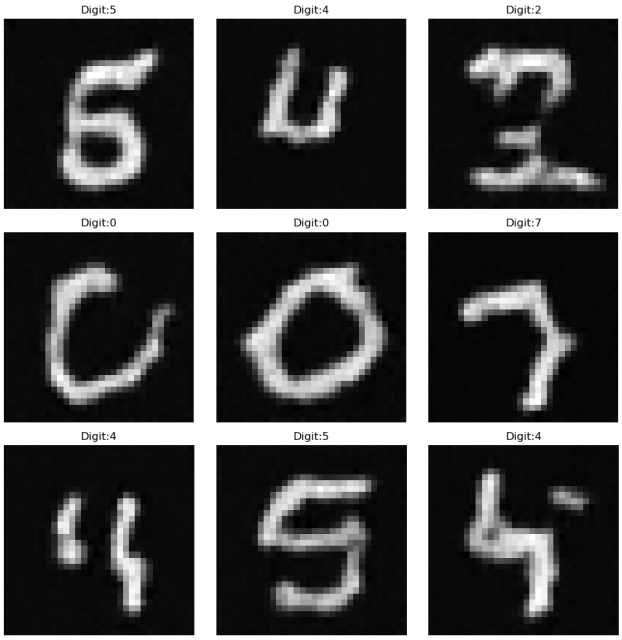

Figure 23: Epoch 4

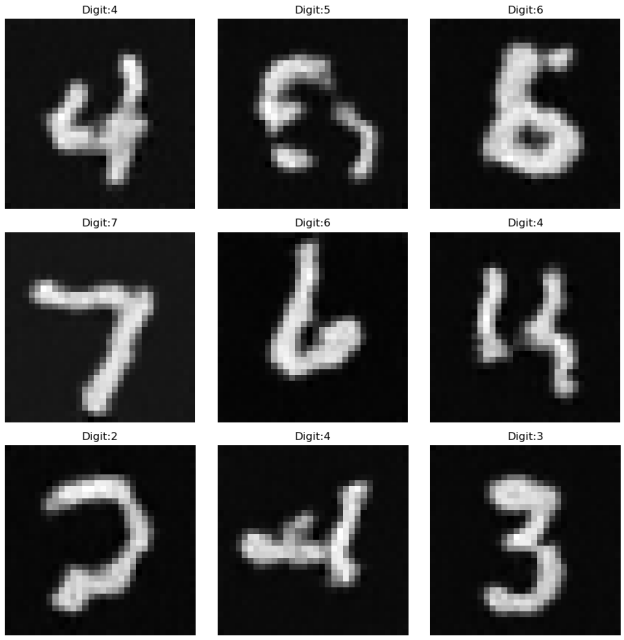

Figure 24: Epoch 5

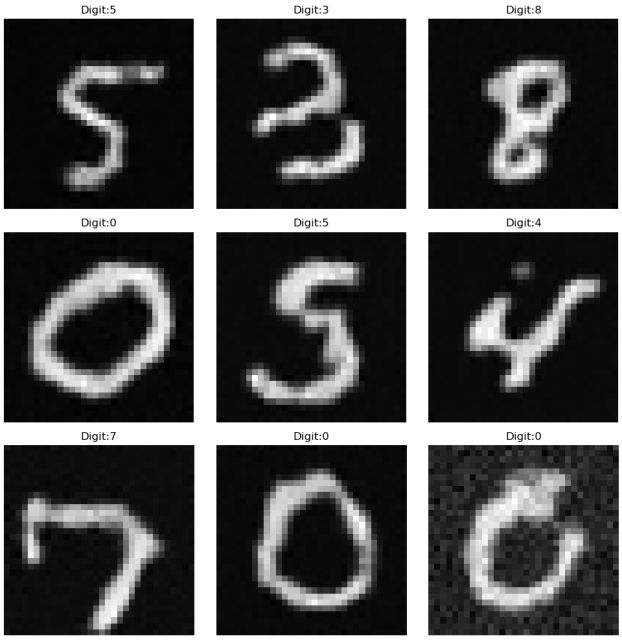

Figure 25: Epoch 6

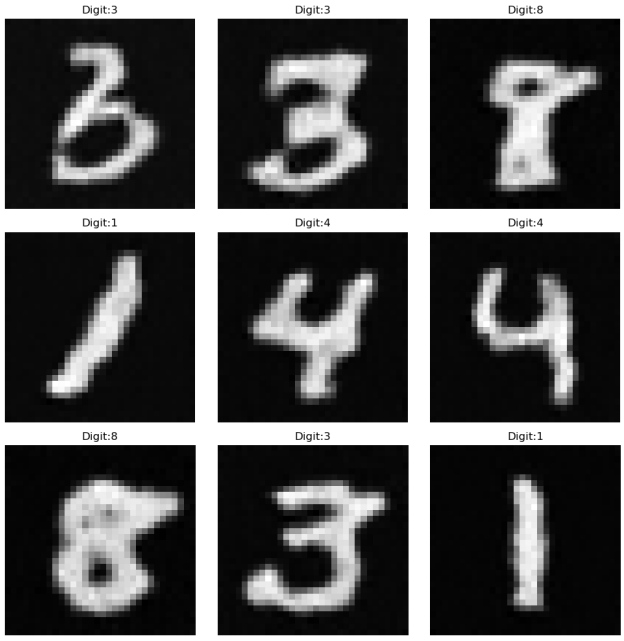

Figure 26: Epoch 7

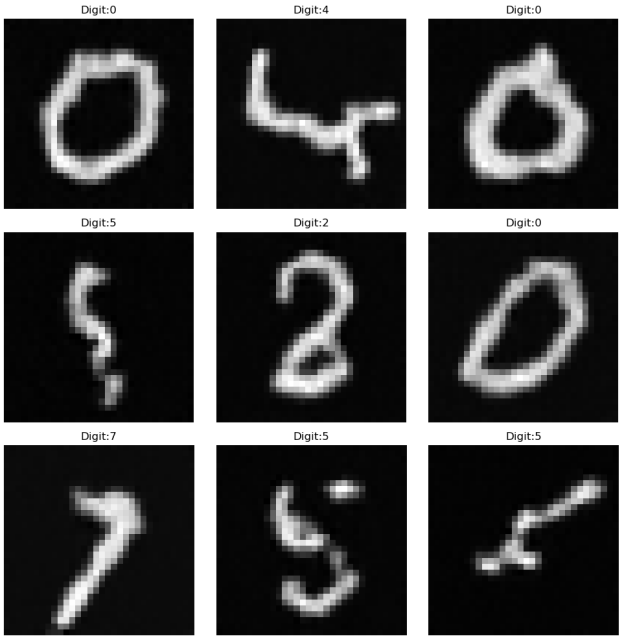

Figure 27: Epoch 8

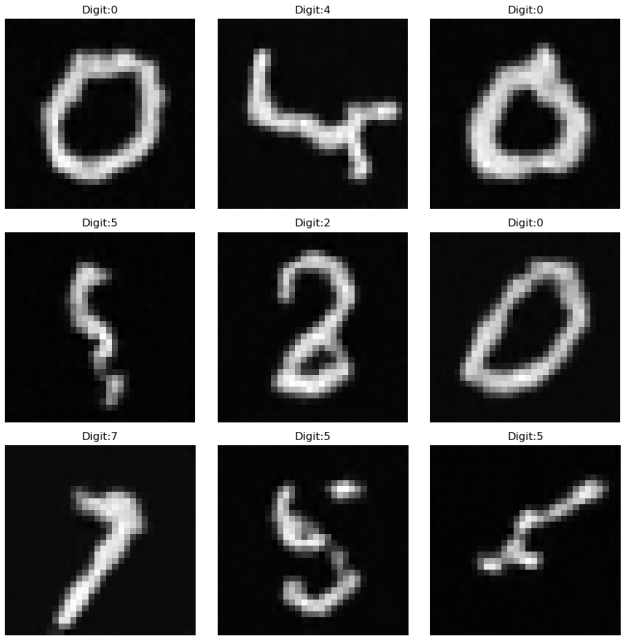

Figure 28: Epoch 8

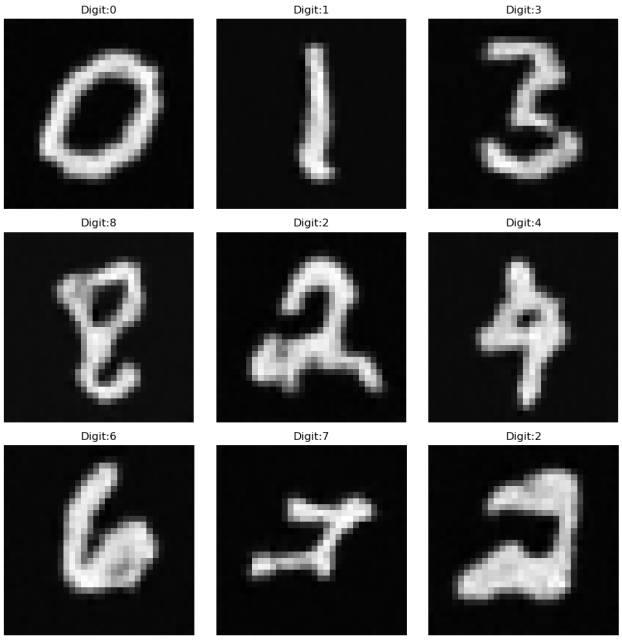

Figure 29: Epoch 9

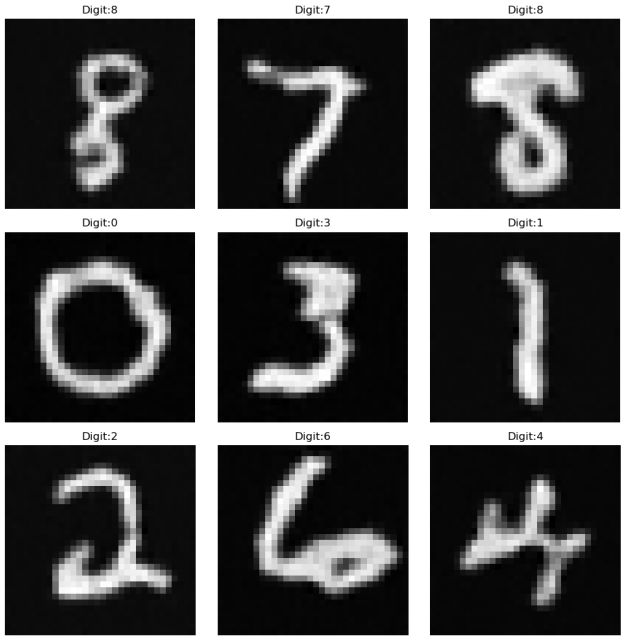

Figure 30: Epoch 10

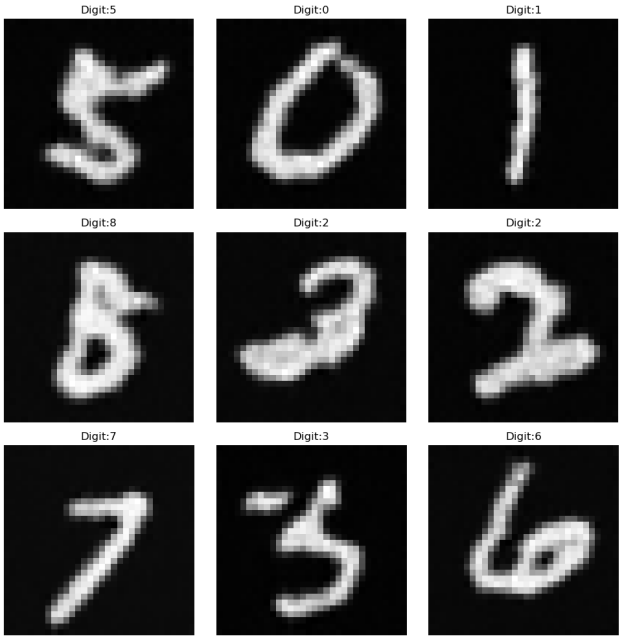

Figure 31: Epoch 12

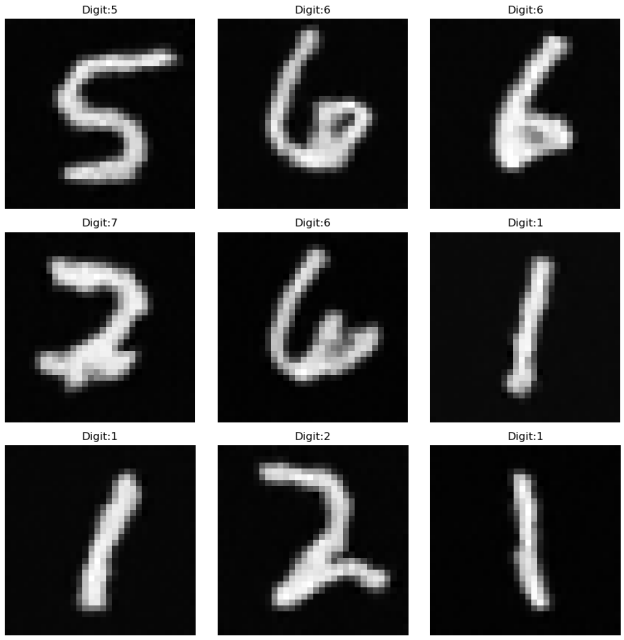

Figure 32: Epoch 14

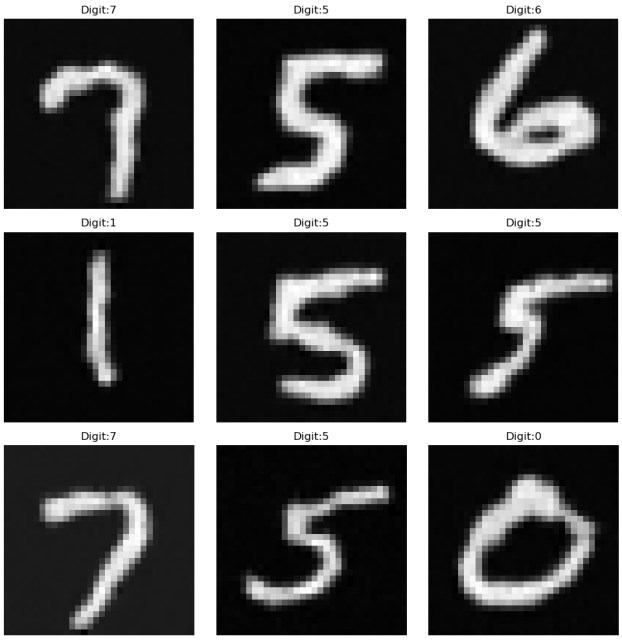

Figure 33: Epoch 16

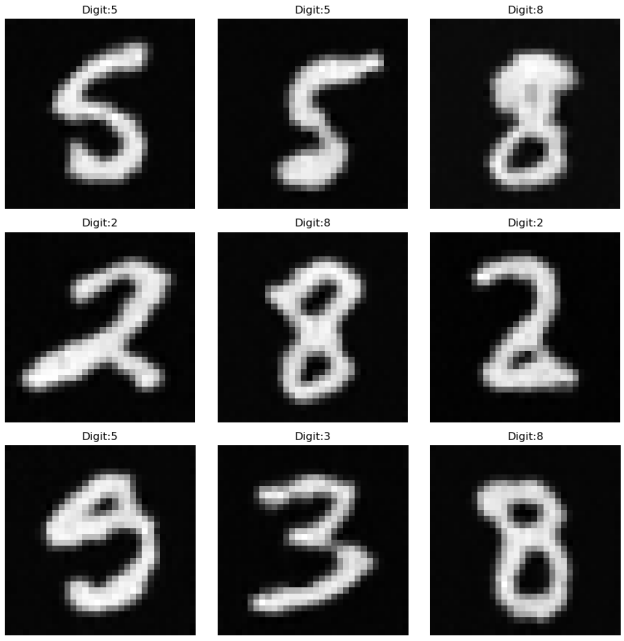

Figure 34: Epoch 18

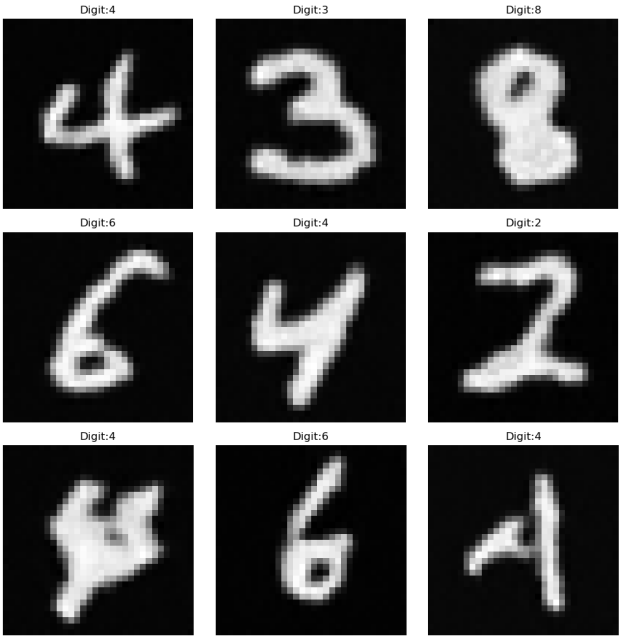

Figure 35: Epoch 19

Again we see in the beginning of the epochs the sampled images are uninterpretable. At epoch 3 the figures begin to resemble digits. Once we get to epoch 12 we can start to reliably identify all the digits. We can see that for the last few epochs the sampled images are quite clear and none of them images are still ambiguous, unlike the DDPM model which still produced ambiguous images even at the very last trianing epoch.