Assignment 4

Mitchell Chatterjee
Carleton University (101141206)

The intent of this assignment was to use different generative models and compare their performance in generating real data. This was done using a VAE, a classic GAN, and a Wasserstein GAN (WGAN). In order to measure their performance a different evaluation metric was used for each model corresponding to its unique properties. These included: Reconstruction Loss, Jensen Shannon Divergence, and Earth Mover Distances. The datasets used were the MNIST and CIFAR-10 datasets.

Variational Autoencoder (VAE)

Model

```
VariationalEncoder(
  (linear1): Linear(in_features=784, out_features=512, bias=True)
  (linear2): Linear(in_features=512, out_features=50, bias=True)
  (linear3): Linear(in_features=512, out_features=50, bias=True)
)
Decoder(
  (linear1): Linear(in_features=50, out_features=512, bias=True)
  (linear2): Linear(in_features=512, out_features=784, bias=True)
)
```

Figure: VAE (Model 1)

```
VariationalEncoder(
  (linear1): Linear(in_features=784, out_features=512, bias=True)
  (linear2): Linear(in_features=512, out_features=256, bias=True)
  (linear3): Linear(in_features=256, out_features=100, bias=True)
  (linear4): Linear(in_features=256, out_features=100, bias=True)
)
Decoder(
  (linear1): Linear(in_features=100, out_features=256, bias=True)
  (linear2): Linear(in_features=256, out_features=512, bias=True)
  (linear3): Linear(in_features=512, out_features=784, bias=True)
)
```

Figure: VAE (Model 2)

```
VariationalEncoder(
  (linear1): Linear(in_features=3072, out_features=1536, bias=True)
  (linear2): Linear(in_features=1536, out_features=768, bias=True)
  (linear3): Linear(in_features=768, out_features=512, bias=True)
  (linear4): Linear(in_features=512, out_features=100, bias=True)
  (linear5): Linear(in_features=512, out_features=100, bias=True)
)
Decoder(
  (linear1): Linear(in_features=100, out_features=512, bias=True)
  (linear2): Linear(in_features=512, out_features=768, bias=True)
  (linear3): Linear(in_features=768, out_features=1536, bias=True)
  (linear4): Linear(in_features=1536, out_features=3072, bias=True)
)
```

Figure: VAE (Model 3)

Here we have three different models for the VAE. The first two are used for the MNIST dataset. The second is a more complex model than the first introducing additional layers. The third model is meant for the CIFAR 10 dataset.

Unfortunately, the model used was not optimized for the CIFAR 10 dataset and therefore did not perform well on the more complex data. In the future it would be interesting to explore other models and how they generalize to this task. Particularly more complex convolutional models, which were unable to be explored as the computational resources required to use these models were simply unavailable.

Results

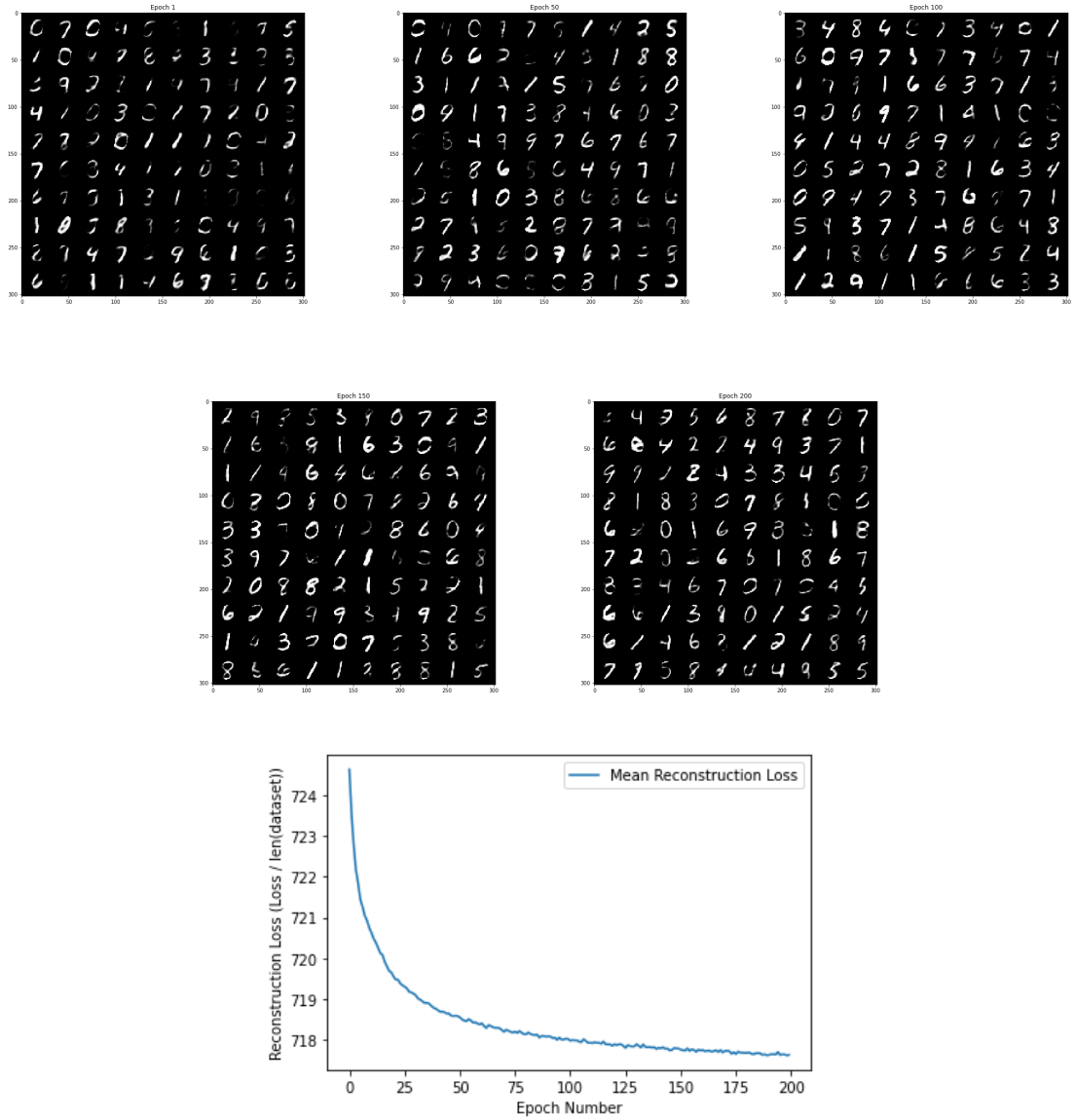

Figure 1: Reconstruction loss and sample images on various epochs from VAE (Model 1) with latent dimension of 50.

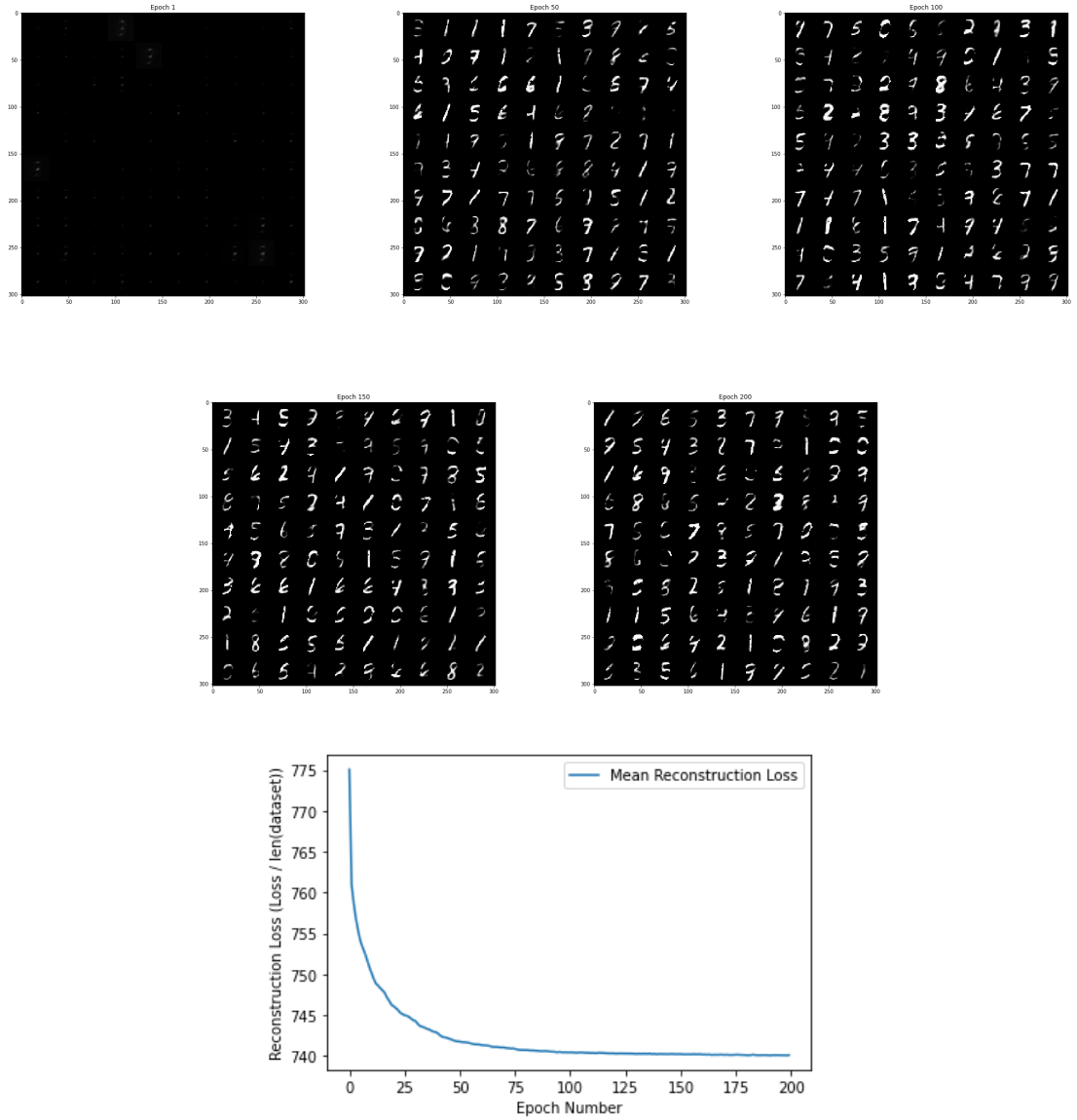

Figure: Reconstruction loss and sample images on various epochs from VAE (Model 1) with latent dimension of 100.

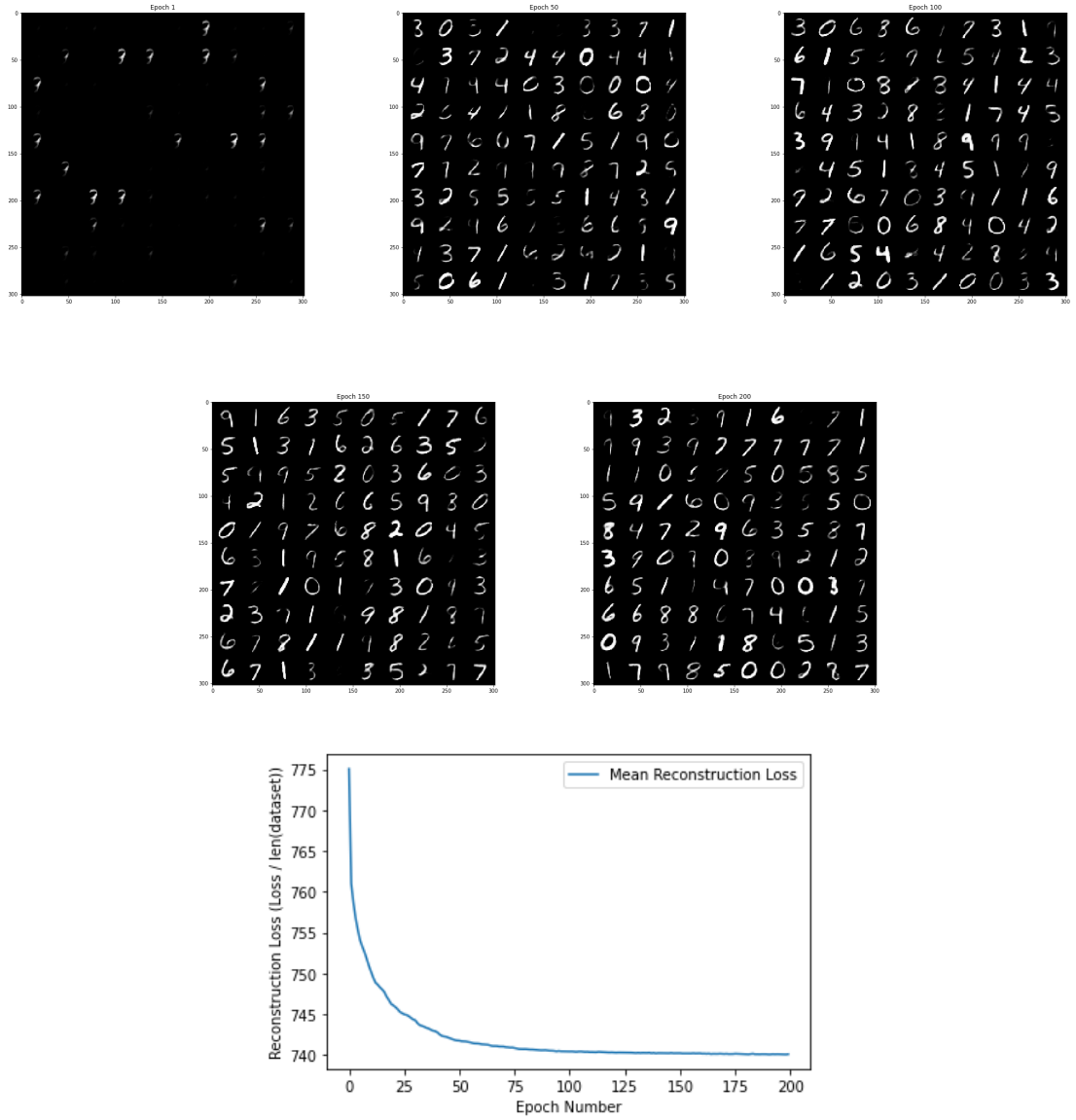

Figure: Reconstruction loss and sample images on various epochs from VAE (Model 2) with latent dimension of 100.

As we can see from the above figures. Increasing the size of the latent dimension increases the reconstruction loss as the space from which we draw the latent variables becomes larger. Thus there is more variation in the types of images the network produces.

When increasing the model complexity along with the size of the latent dimension we observe sharper images as a result of the increased capacity of the network. We also observe that the more complex model is able to handle the larger latent space. Obtaining a similar reconstruction loss to the less complex model on the same latent space.

Generative Adversarial Network (GAN)

Model

```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
```

Figure: GAN (Model 1)

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=768, bias=True)
    (9): BatchNorm1d(768, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=768, out_features=1024, bias=True)
    (12): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (13): LeakyReLU(negative_slope=0.2, inplace=True)
    (14): Linear(in_features=1024, out_features=784, bias=True)
    (15): Tanh()
  )
)
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=128, bias=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): Linear(in_features=128, out_features=1, bias=True)
    (7): Sigmoid()
  )
)
```

Figure: GAN (Model 2)

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=3072, bias=True)
    (12): Tanh()
  )
)
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=3072, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

Figure: GAN (Model 3)

Here we have three different models for the GAN. The first two are used for the MNIST dataset. The second is a more complex model than the first introducing additional layers. The third model is meant for the CIFAR 10 dataset.

Results

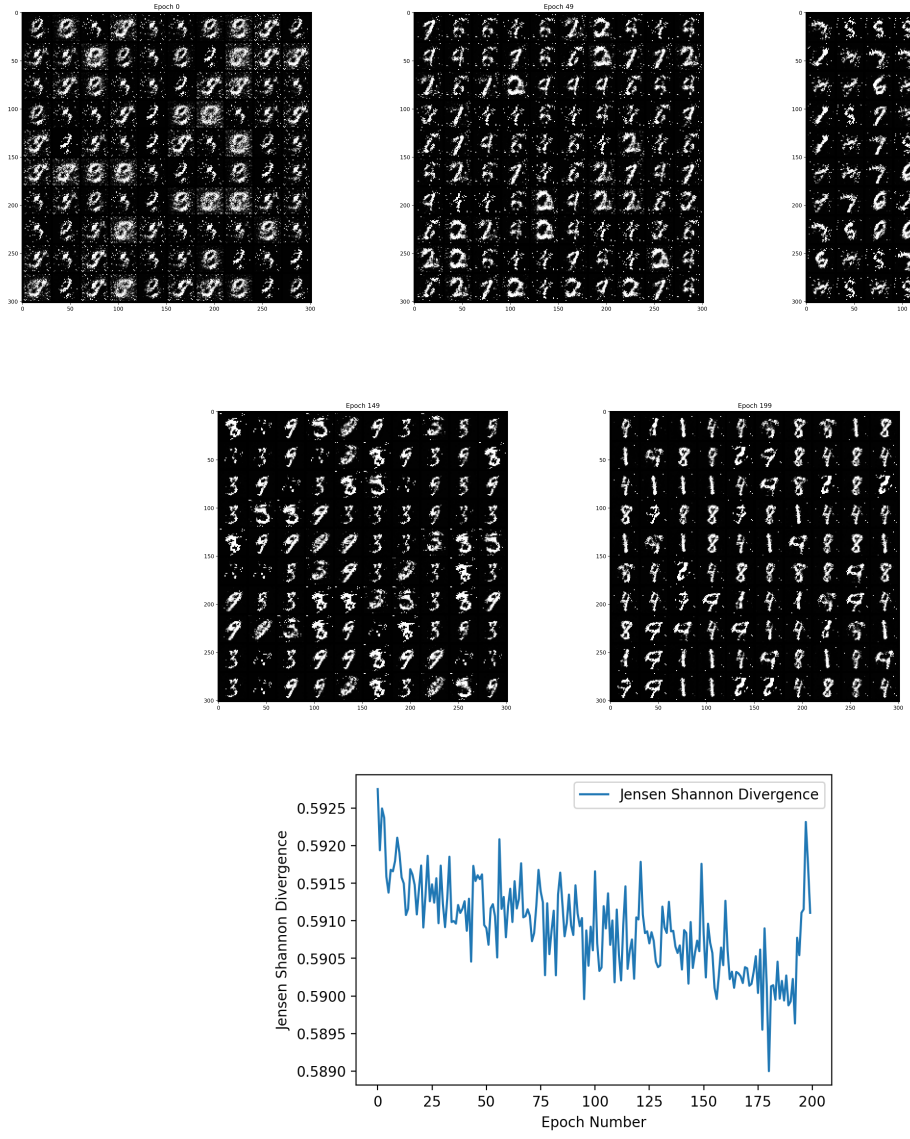

Figure: JSD and sample images on various epochs from GAN (Model 1) with latent dimension of 100.

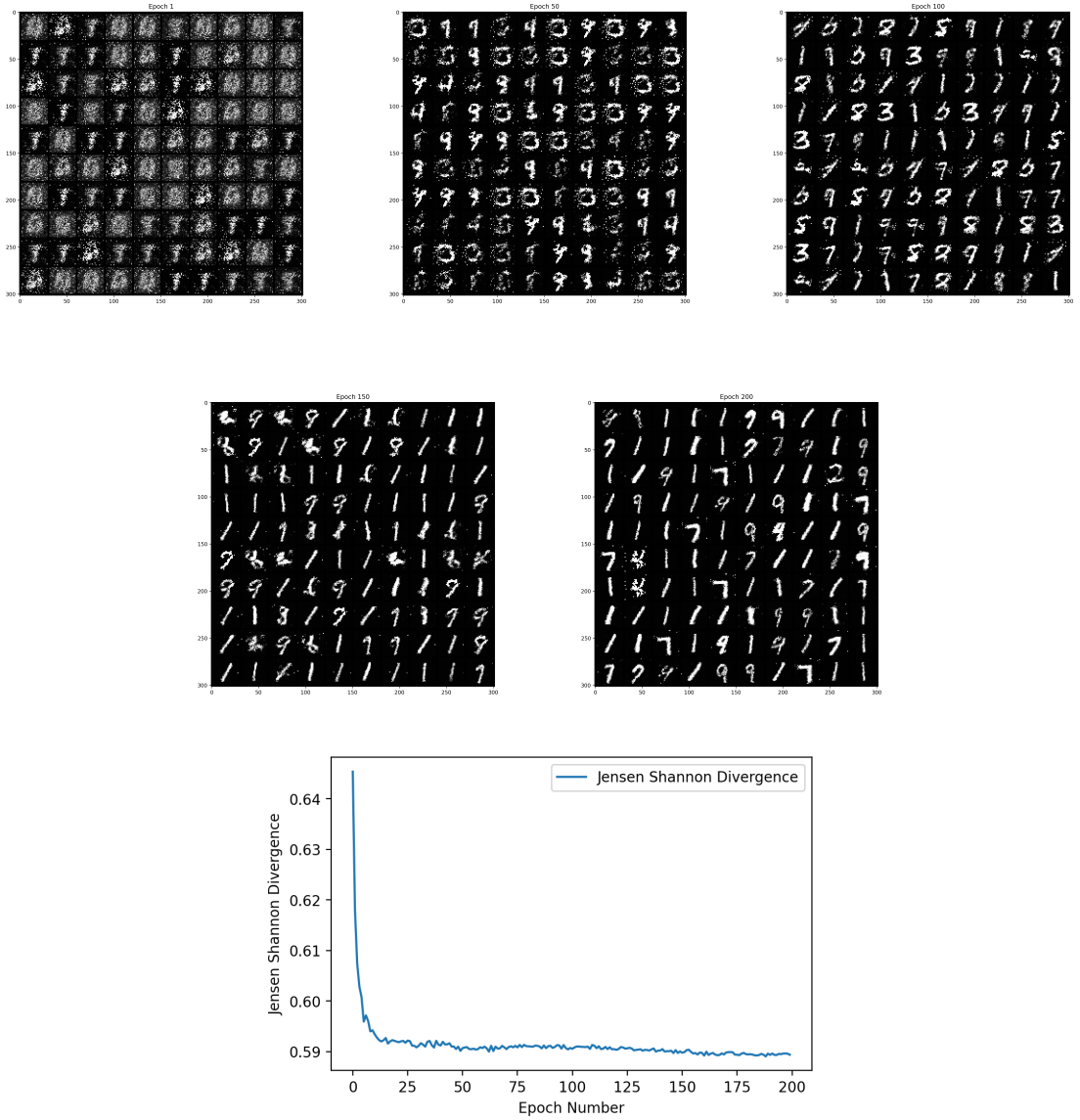

Figure: JSD and sample images on various epochs from GAN (Model 1) with latent dimension of 200.

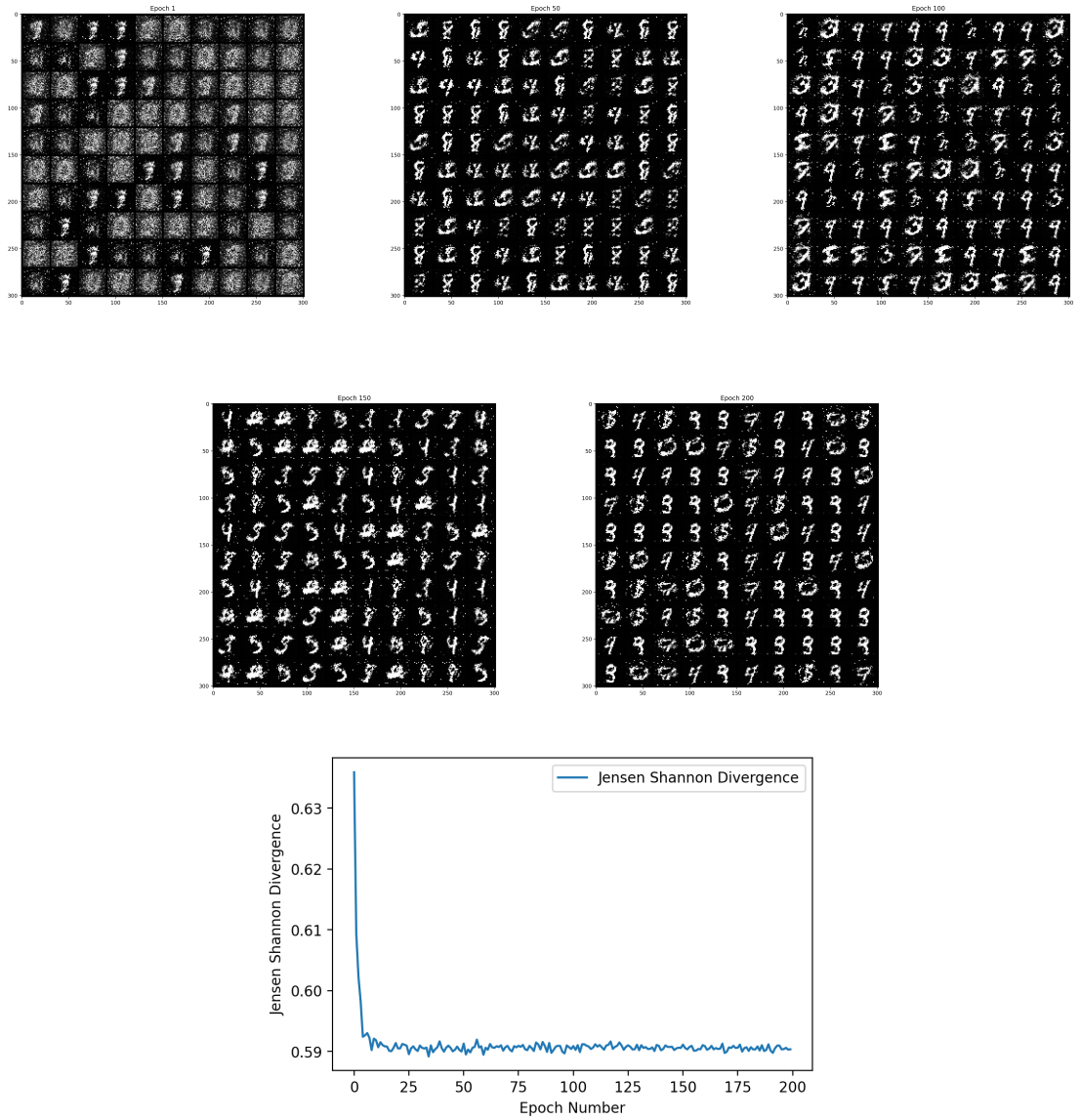

Figure: JSD and sample images on various epochs from GAN (Model 2) with latent dimension of 200.

As we can see from the above figures. Increasing the size of the latent dimension from 100 to 200 stabilized the Jensen Shannon Divergence and generated much sharper images. In all cases we can see the effects of mode collapse as the model collapses to 3 or 4 different classes.

When increasing the model complexity along with the size of the latent dimension we observe that the images become less distinct. This is likely a result of increased training requirements for a more complex model. Though the model has a greater capacity to represent the data, it requires more training in order to converge to an acceptable accuracy.

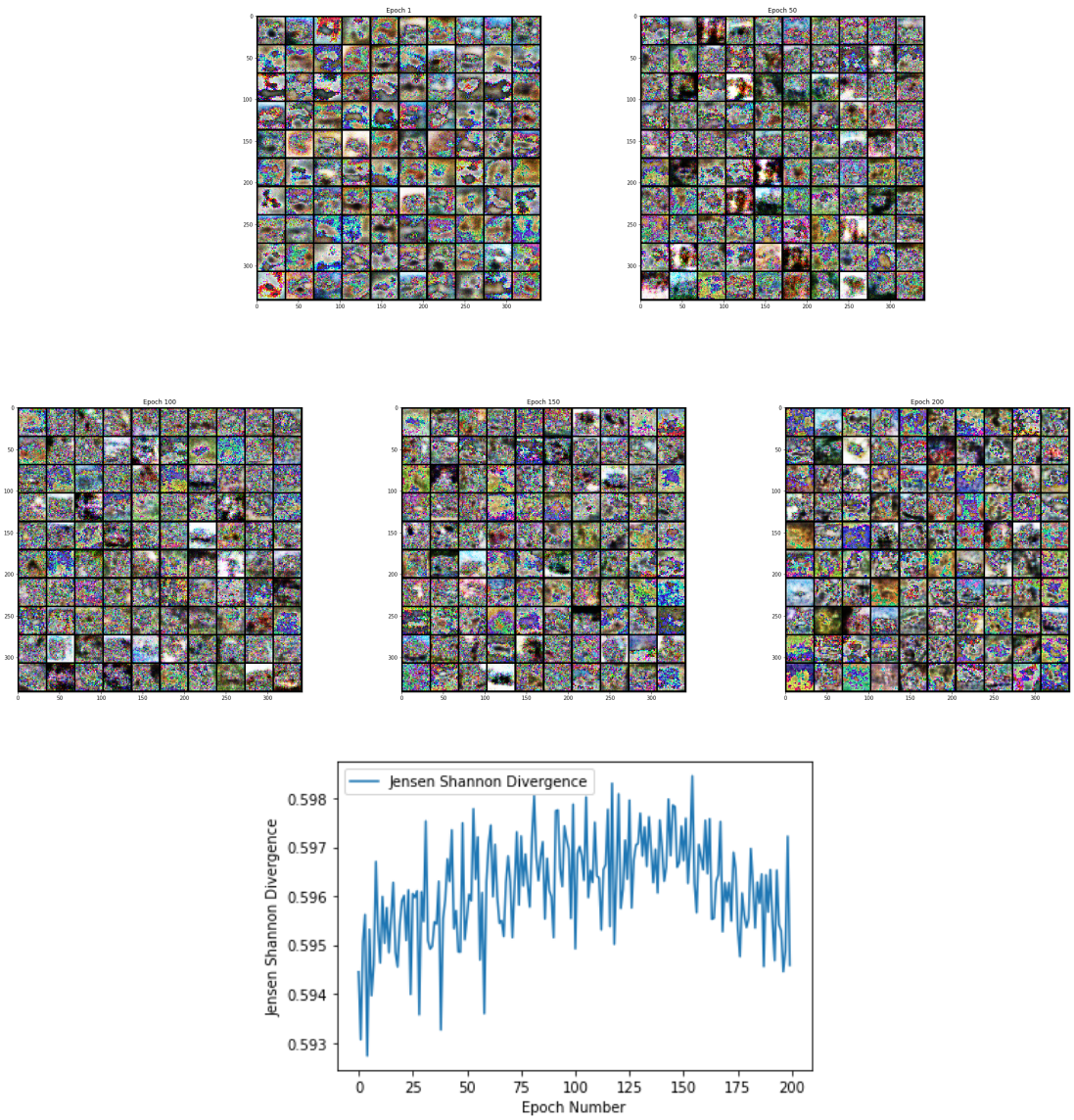

Figure: JSD and sample images on various epochs from GAN (Model 3) with latent dimension of 100 on CIFAR 10 dataset.

Unfortunately, the model used was not optimized for the CIFAR 10 dataset and therefore did not perform well on the more complex data. In the future it would be interesting to explore other models and how they generalize to this task. Particularly more complex convolutional models, which were unable to be explored as the computational resources required to use these models were simply unavailable.

Wasserstein GAN (WGAN)

Model

```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
```

Figure: WGAN (Model 1)

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=768, bias=True)
    (9): BatchNorm1d(768, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=768, out_features=1024, bias=True)
    (12): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (13): LeakyReLU(negative_slope=0.2, inplace=True)
    (14): Linear(in_features=1024, out_features=784, bias=True)
    (15): Tanh()
  )
)
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=128, bias=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): Linear(in_features=128, out_features=1, bias=True)
  )
)
```

Figure: WGAN (Model 2)

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=3072, bias=True)
    (12): Tanh()
  )
)
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=3072, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

Figure: WGAN (Model 3)

Here we have three different models for the WGAN. The first two are used for the MNIST dataset. The second is a more complex model than the first introducing additional layers. The third model is meant for the CIFAR 10 dataset.

Results

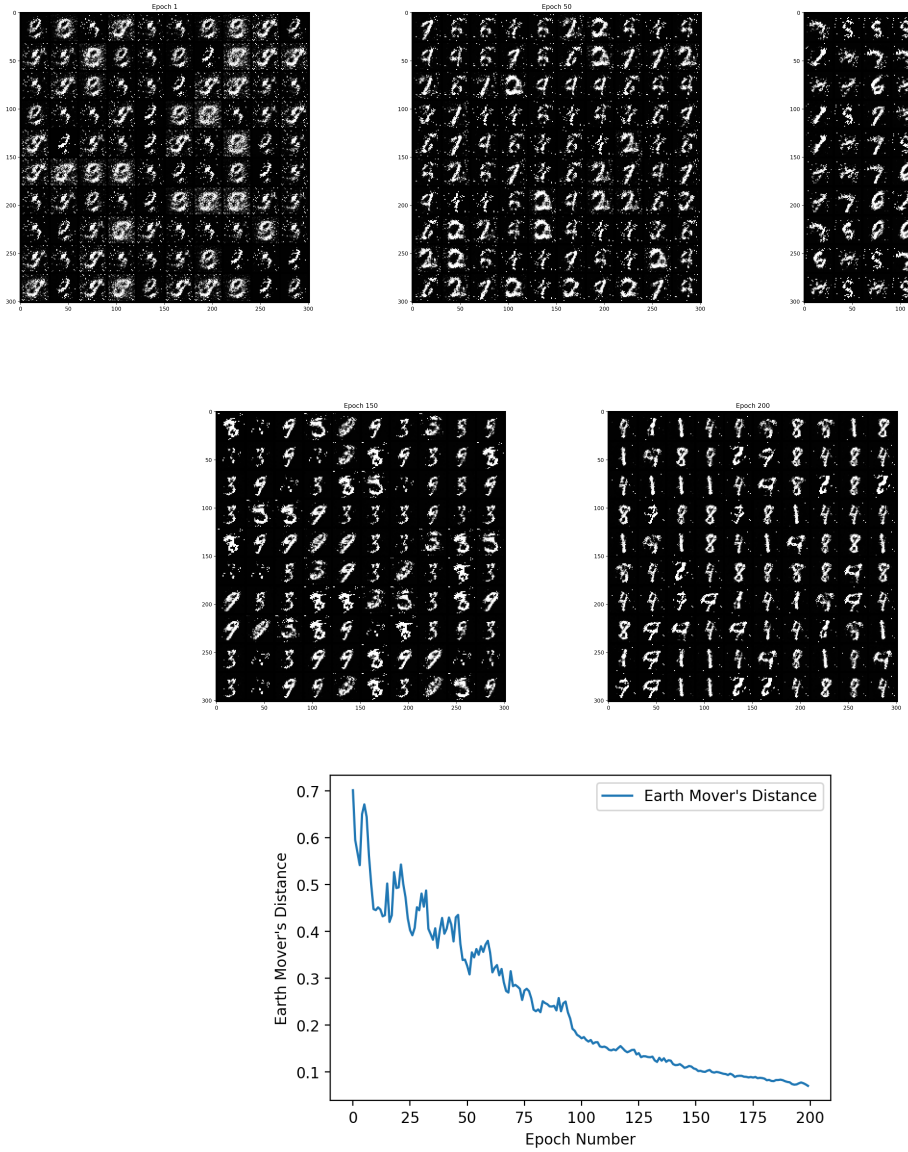

Figure: EMD and sample images on various epochs from WGAN (Model 1) with latent dimension of 100.

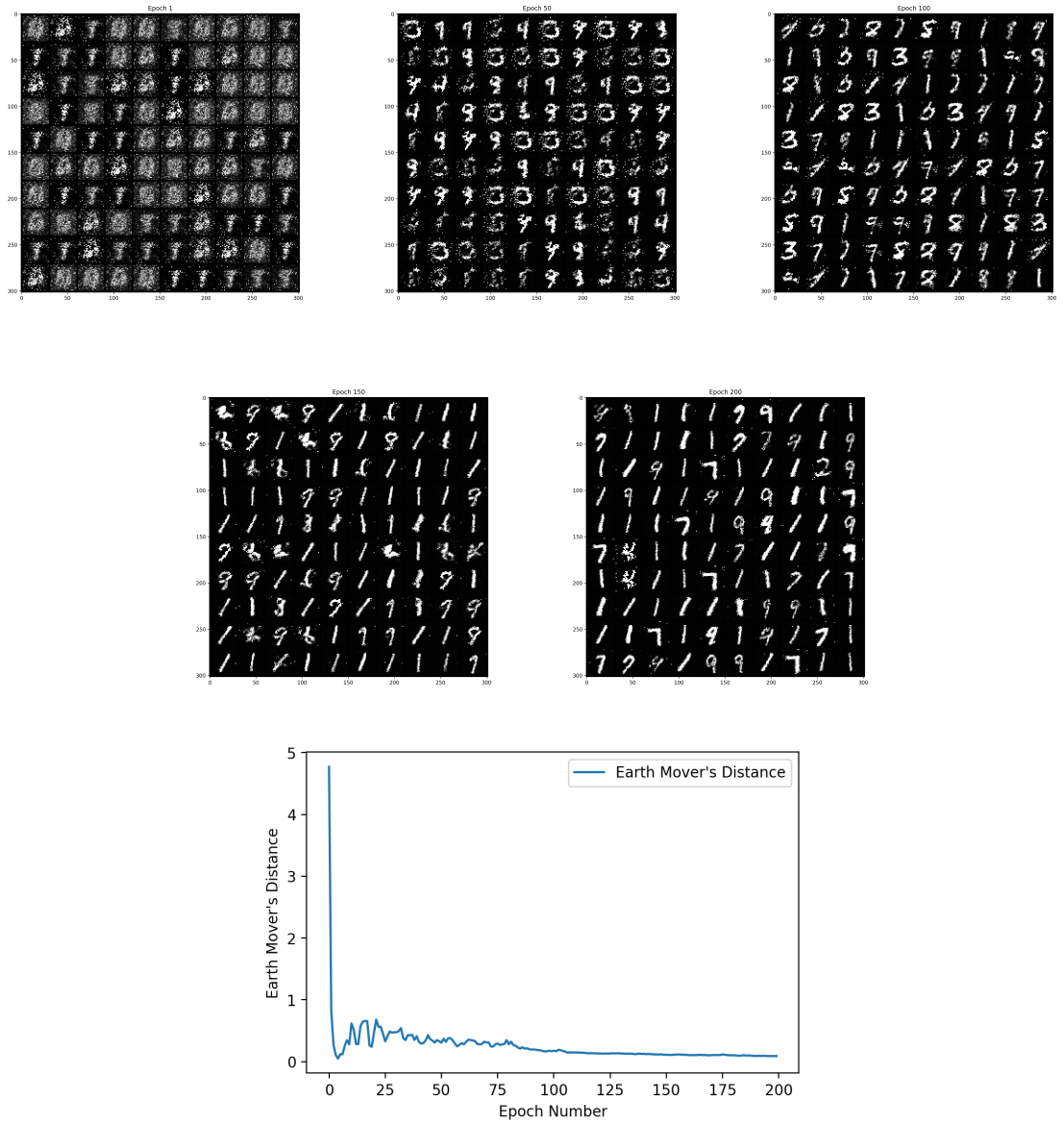

Figure: EMD and sample images on various epochs from WGAN (Model 1) with latent dimension of 200.

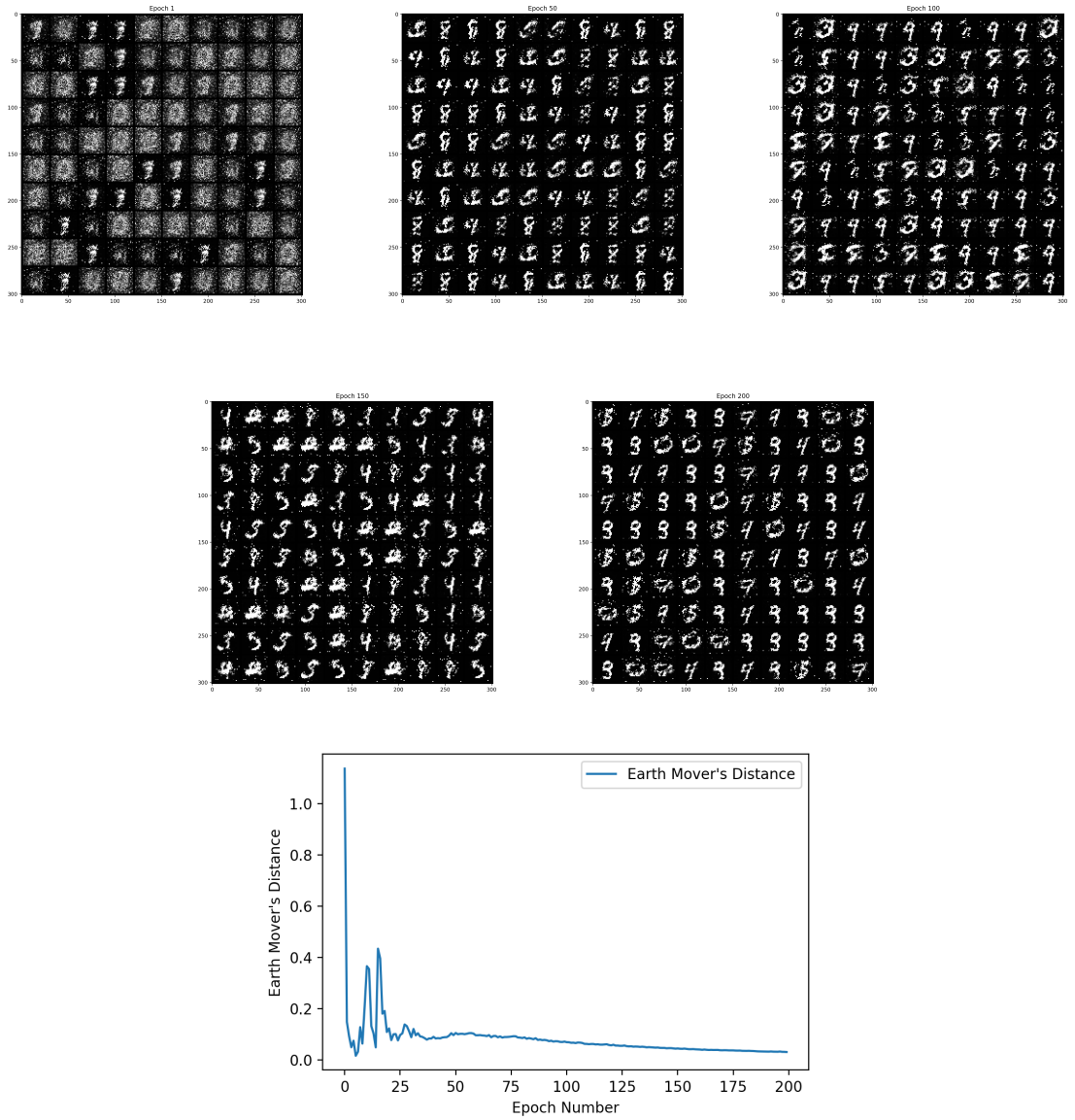

Figure: EMD and sample images on various epochs from WGAN (Model 2) with latent dimension of 200.

As we can see from the above figures, increasing the size of the latent dimension once again results in sharper images.

When increasing the model complexity along with the size of the latent dimension we observe that the images become less distinct. This is likely a result of increased training requirements for a more complex model. Though the model has a greater capacity to represent the data, it requires more training in order to converge to an acceptable accuracy.

In all cases the EMD steadily decreases with the number of epochs.

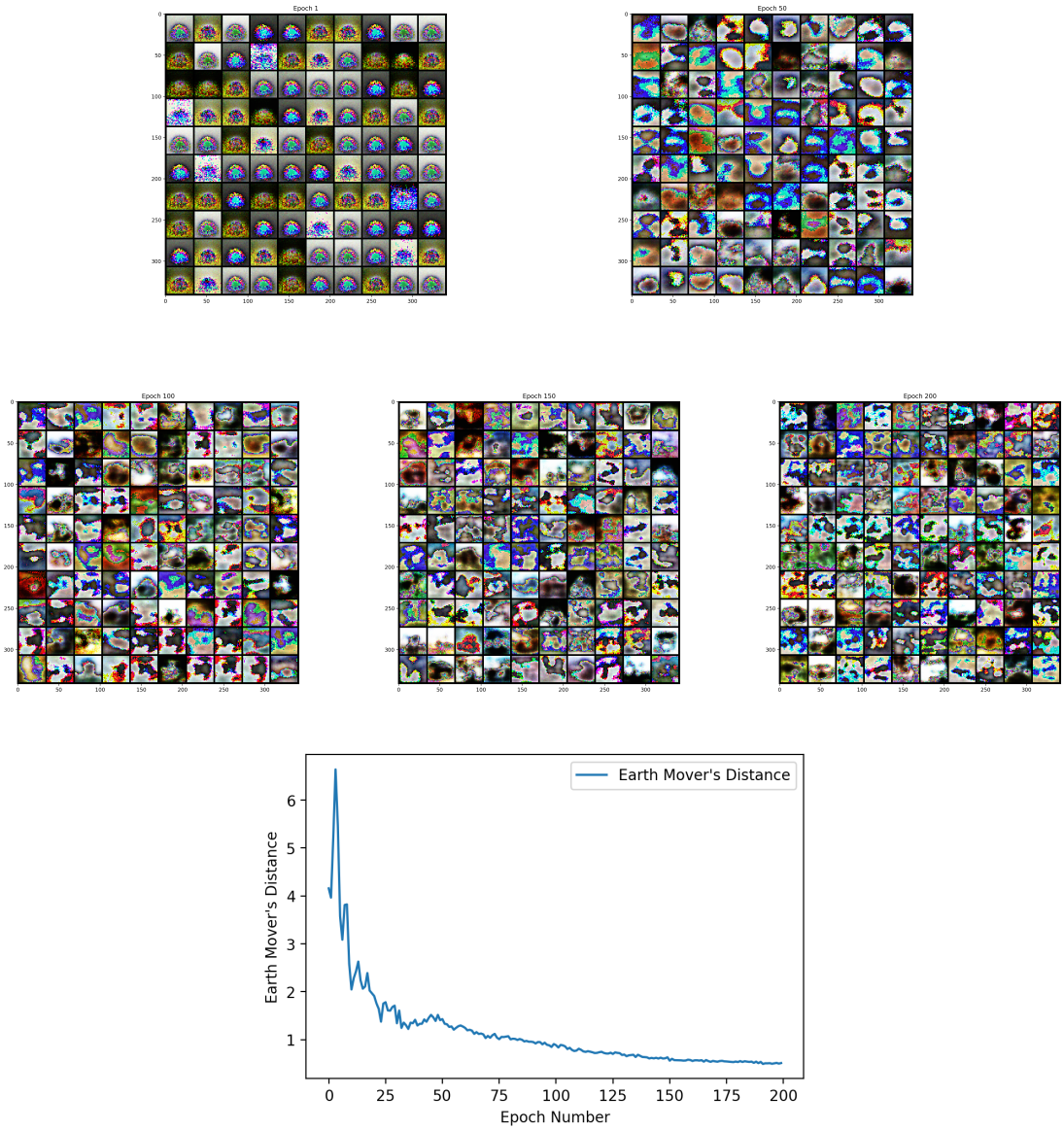

Figure: EMD and sample images on various epochs from WGAN (Model 3) with latent dimension of 200 on CIFAR 10 dataset.

Unfortunately, the model used was not optimized for the CIFAR 10 dataset and therefore did not perform well on the more complex data. In the future it would be interesting to explore other models and how they generalize to this task. Particularly more complex convolutional models, which were unable to be explored as the computational resources required to use these models were simply unavailable.