CSI 5138: Homework 4

Narmadha Sambandam, 300098713 Dharna Shukla, 300051861

Nov 14, 2019

Variational Autoencoders:

This section discusses the results of the implementation of the Variational Autoencoders (VAE) model trained on MNIST and CIFAR10 datasets using reference [?]. The two key hyper-parameters, latent space dimension as well as model complexity were investigated. In the context of VAE, the model complexity is defined as the intermediate dimension representing the dimension of the dense layer receiving the input.

Varying latent space dimension:

The first case tested was with the intermediate dimension remaining fixed at 512 with varying latent dimensions 2, 5, and 10. It was noticed that this did not contribute to any improvement in training time since it took about the same run time per epoch regardless of the latent dimensions used.

Given below is the plot of the loss function for VAE with intermediate dimension at 512 with varying latent dimensions 2, 5, and 10 for MNIST and CEFAR-10. It can be observed both training and test loss decreases as the latent space increases.

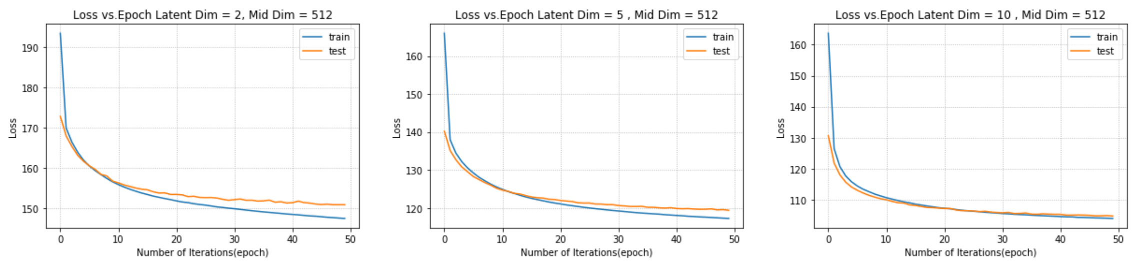


Figure 1: Plot of the loss function for VAE for MNIST

Compared to MNIST, the losses are much higher for CIFAR-10 where the loss are in the thousands vs. the hundreds on MNIST. A possible reason for this is that CIFAR-10 is more complex dataset than MNIST. Note, the erratic nature of the test curve could be a result of dataset allocating only 10,000 sample data for the test set as opposed to 50,000 for the train set.

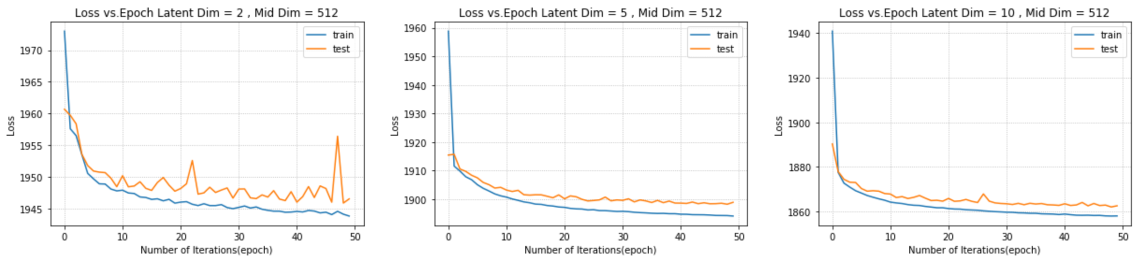
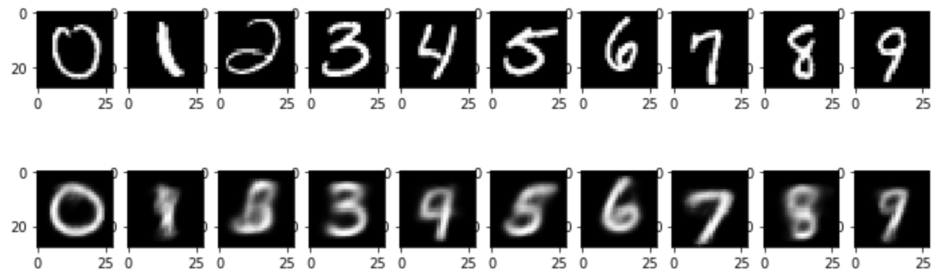
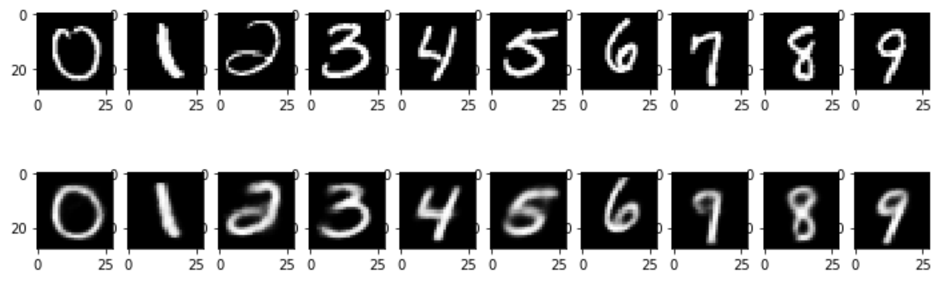


Figure 2: Plot of the loss function for VAE for CEFAR-10

Given below are the generated images for VAE with intermediate dimension at 512 with varying latent dimensions 2, 5, and 10 for MNIST and CEFAR-10. Overall, larger latent dimensions yielded in a better quality for the generated images. The generated images in (c) resulted in the least blurry images with a corresponding latent dimension 10. The reason for the reconstruction being blurry is because the input is compressed at the bottleneck (latent space) layer which is controlled by its dimension. Hence, it is sensible that a larger latent dimension yields a higher quality.


(a) Original (top) vs generated images for VAE for MNIST with intermediate dimension at 512 with latent dimension 2 (bottom)


(b) Original (top) vs generated images for VAE for MNIST with intermediate dimension at 512 with latent dimension 5 (bottom)

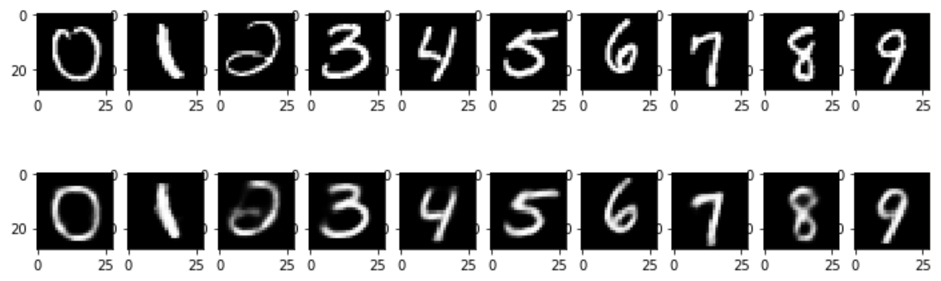
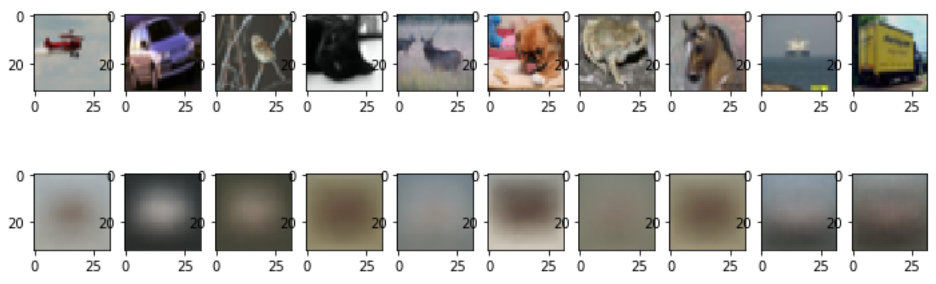
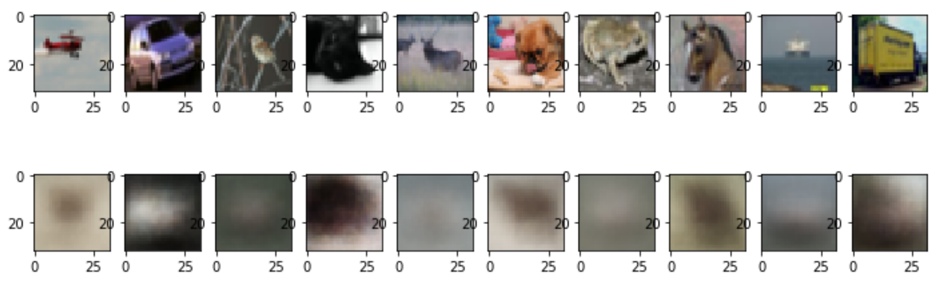

(c) Original (top) vs generated images for VAE for MNIST with intermediate dimension at 512 with latent dimension 10 (bottom)

Figure 3: Plot of the generated images for VAE for MNIST

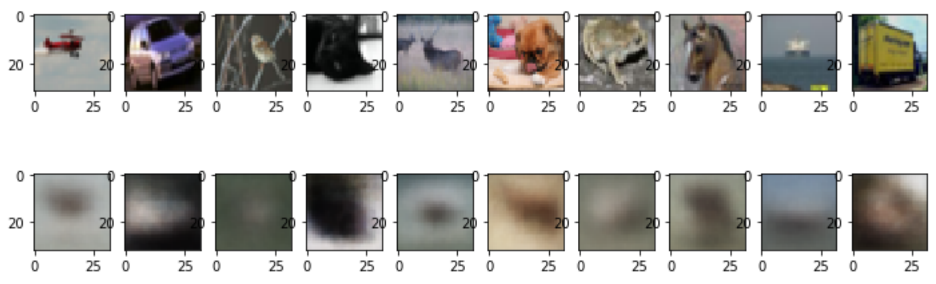
The higher losses of the CEFAR-10 dataset is indicative of the poorly generated images which are barely legible. Although the objects are too blurry to identify in the images, the larger latent dimension makes it possible to relate the generated images to the original based on the colour scheme and the low visibility of the outline of the object.



(a) Original (top) vs generated images for VAE for CEFAR-10 with intermediate dimension at 512 with latent dimension 2 (bottom)



(b) Original (top) vs generated images for VAE for CEFAR-10 with intermediate dimension at 512 with latent dimension 5 (bottom)



(c) Original (top) vs generated images for VAE for CEFAR-10 with intermediate dimension at 512 with latent dimension 10 (bottom)

Figure 4: Plot of the generated images for VAE for CEFAR-10

Varying model complexity:

While keeping the latent dimension fixed at 2, the model complexity was varied by modifying the intermediate dimension to 128, 256 and 512. As expected, the run time was much faster per epoch for the smaller intermediate dimensions.

Given below is the plot of the loss function for VAE with latent dimension at 2 with varying intermediate dimensions 128, 256 and 512 for MNIST and CEFAR-10. It can be observed that loss converges more quickly with larger intermediate dimensions. However, the divergence in loss between the train and test sets increase with larger dimensions.

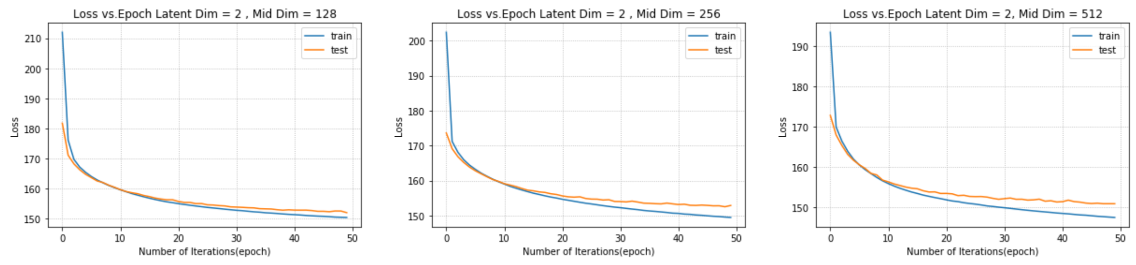


Figure 5: Plot of the loss function for VAE for MNIST

The losses are higher as expected for CEFAR-10 as previously discussed in the test for latent space dimensionality. Similar to MNIST, the losses converge quicker with larger intermediate dimensions. Note, the erratic nature of the test curve could be a result of dataset allocating only 10,000 sample data for the test set as opposed to 50,000 for the train set.

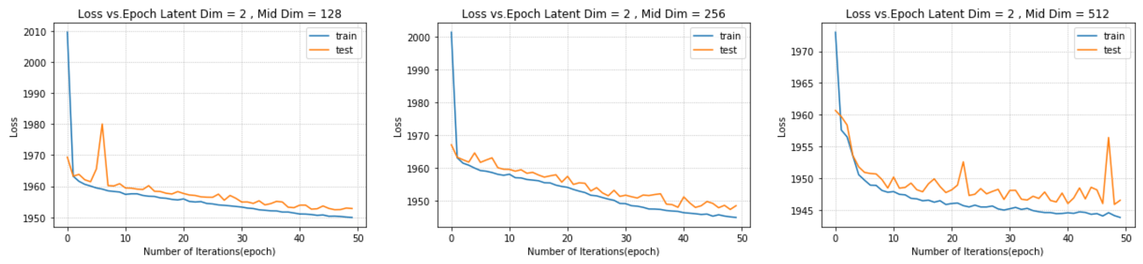
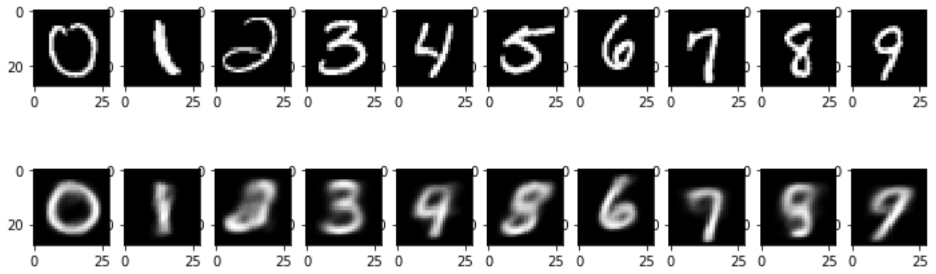
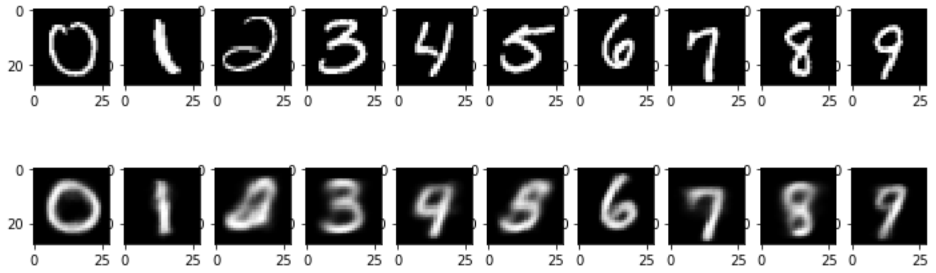


Figure 6: Plot of the loss function for VAE for CEFAR-10

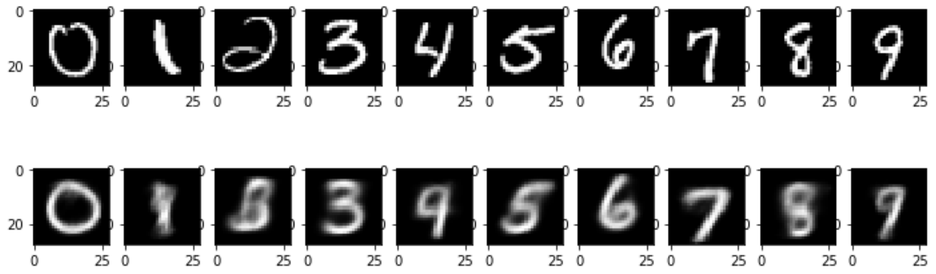
Given below are the generated images for VAE with latent dimension at 2 with varying intermediate dimensions 128, 256 and 512 for MNIST and CEFAR-10. It can be seen that larger intermediate dimension did not yield in a better quality images. In both datasets, the generated images are remain blurry despite any changes to the intermediate dimensionality.



(a) Original (top) vs generated images for VAE for MNIST with intermediate dimension at 128 with latent dimension 2 (bottom)

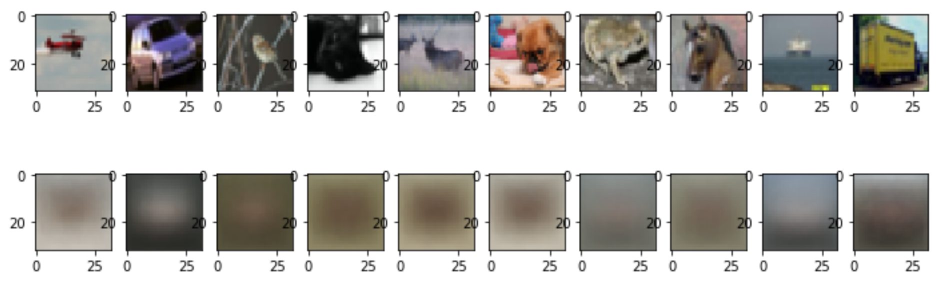


(b) Original (top) vs generated images for VAE for MNIST with intermediate dimension at 256 with latent dimension 2 (bottom)

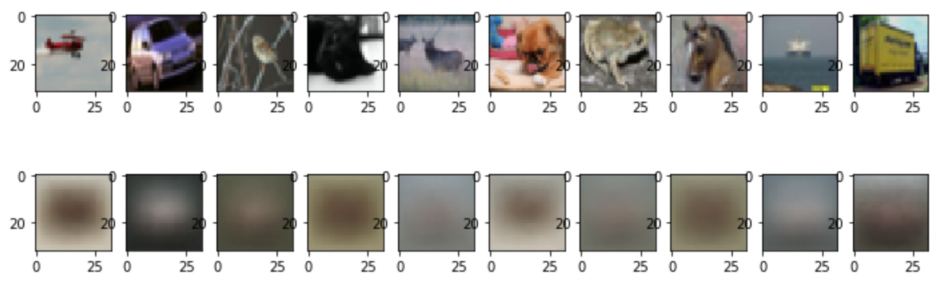


(c) Original (top) vs generated images for VAE for MNIST with intermediate dimension at 512 with latent dimension 2 (bottom)

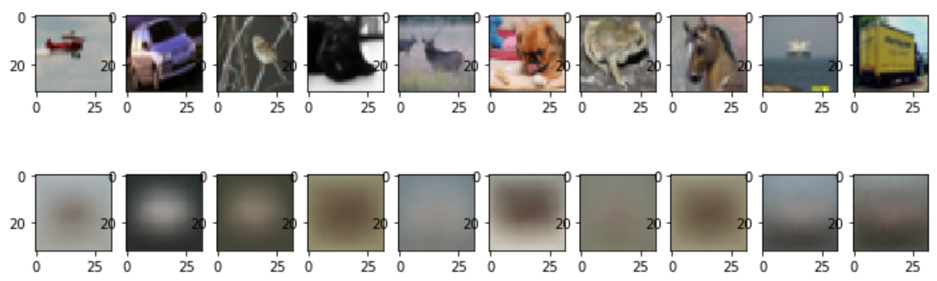
Figure 7: Plot of the generated images for VAE for MNIST



(a) Original (top) vs generated images for VAE for CEFAR-10 with intermediate dimension at 512 with latent dimension 2 (bottom)



(b) Original (top) vs generated images for VAE for CEFAR-10 with intermediate dimension at 512 with latent dimension 5 (bottom)



(c) Original (top) vs generated images for VAE for CEFAR-10 with intermediate dimension at 512 with latent dimension 10 (bottom)

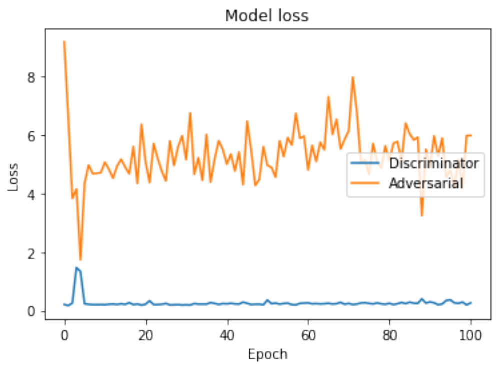
Figure 8: Plot of the generated images for VAE for CEFAR-10

Generative Adversarial Network:

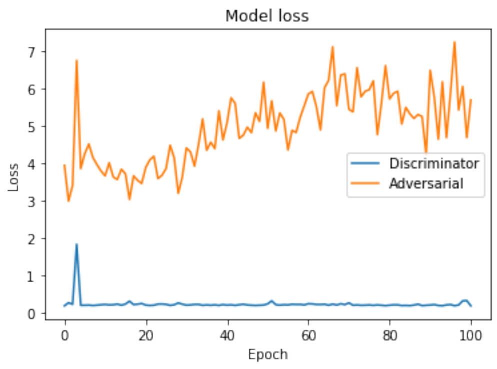
This section discusses the results of the implementation of the Generative Adversarial Network (GAN) model trained on MNIST and CIFAR10 datasets using reference.The two key hyper-parameters, latent space dimension as well as model complexity were investigated. The model complexity is defined as the intermediate dimension representing the dimension of the dense layers receiving the input. As vanilla GANs are rather unstable, an alternate is the Deep Convolution GAN (DCGAN) model which contain features like convolutional layers and batch normalization which can help with the stability of the convergence.

Varying latent space dimension:

As shown below in the model loss plot, there was no considerable change in the JSD loss when the latent dimension was modified from 100 to 10. Both discriminator and generator losses eventually oscillate over some optimum found by the GAN model, where it can't improve more, which also should mean that it has learned well enough.

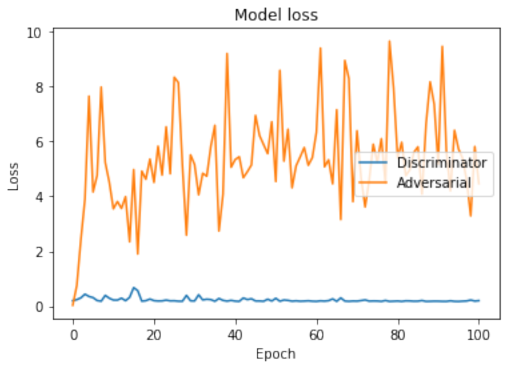


(a) Plot of model losses with latent dimension 100 for MNIST

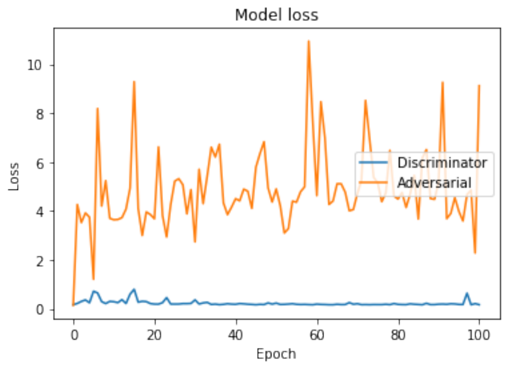


(b) Plot of model losses with latent dimension 10 for MNIST

Figure 9: Plot of the model loss for DCGAN for MNIST



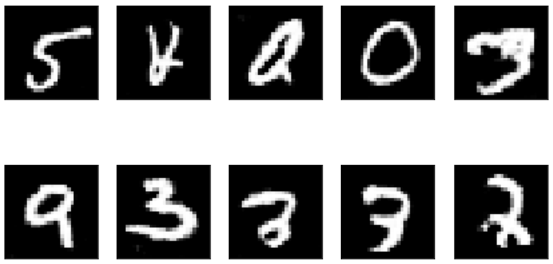
(a) Plot of model losses with latent dimension 100 for CIFAR-10



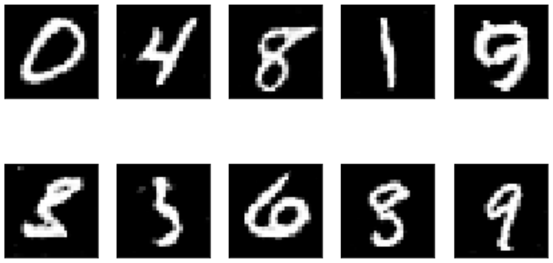
(b) Plot of model losses with latent dimension 10 for CIFAR-10

Figure 10: Plot of the model loss for DCGAN for CIFAR-10

Given below are the generated images for DCGAN with latent dimensions 100 and 10 for MNIST and CEFAR-10.As expected, it can be seen that the images are not visually differentiable in terms of their resolutions. It is easier to tell for the MNIST dataset but harder for CIFAR-10 since the images are more complex.

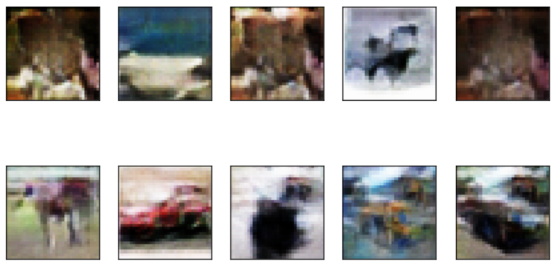


(a) Generated images with latent dimension 100 for MNIST

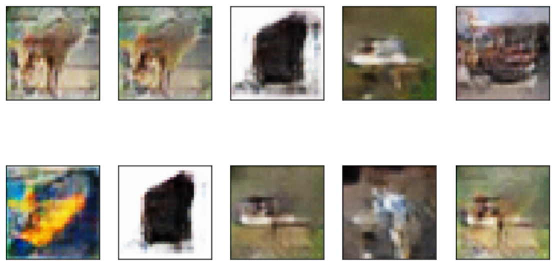


(b) Generated images with latent dimension 10 for MNIST

Figure 11: Generated images for DCGAN for MNIST



(a) Generated images with latent dimension 100 for CIFAR-10

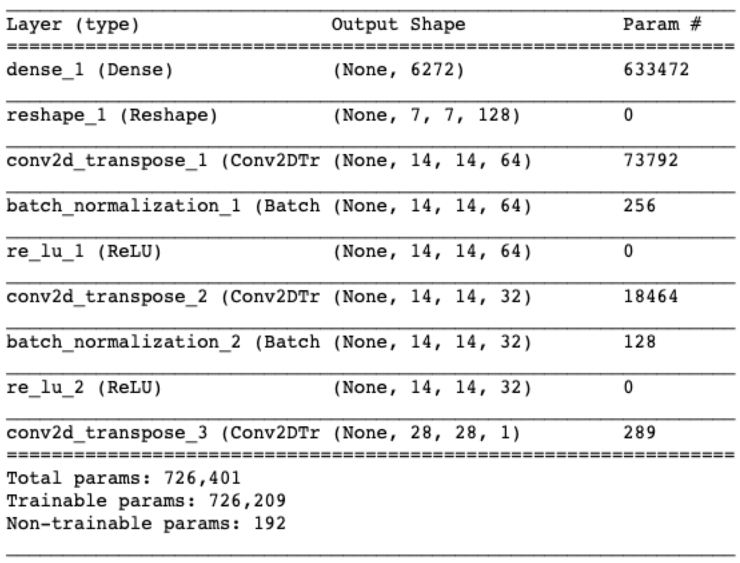


(b) Generated images with latent dimension 10 for CIFAR-10

Figure 12: Generated images for DCGAN for CIFAR-10

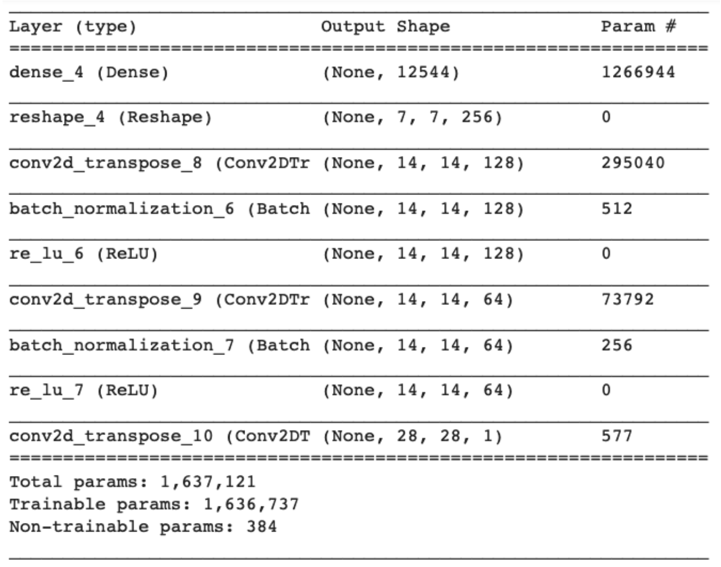
Varying model complexity:

In the case of DCGAN, modifying the model complexity can impact the performance of the model. Having rather larger dense layer accepting the generator input can lead to the generator creating the same output always, no matter what the input noise (mode collapse). In such a scenario, the generator will never be directly incentivised to cover all modes. Thus, the generator will exhibit very poor diversity amongst generated samples, which limits the usefulness of the learnt GAN. On the other hand, not using a dense layer at all can lead to the generator not learning anything meaningful after many iterations. We can reach certain bottleneck that increasing the complexity of the generator does not necessarily improve the image quality. Figure 26 and 14 shows two architecture implemented where one implements deeper layers than the other. Although the initial iteration generated more realistic images with the deeper architecture (Figure 15 and 16), the final generated images resemble each other as shown in Figure 17 and 18.



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6272)              633472
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 64)        73792
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
re_lu_1 (ReLU)               (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 14, 14, 32)        18464
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
re_lu_2 (ReLU)               (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 28, 28, 1)         289
=================================================================
Total params: 726,401
Trainable params: 726,209
Non-trainable params: 192
_________________________________________________________________
```

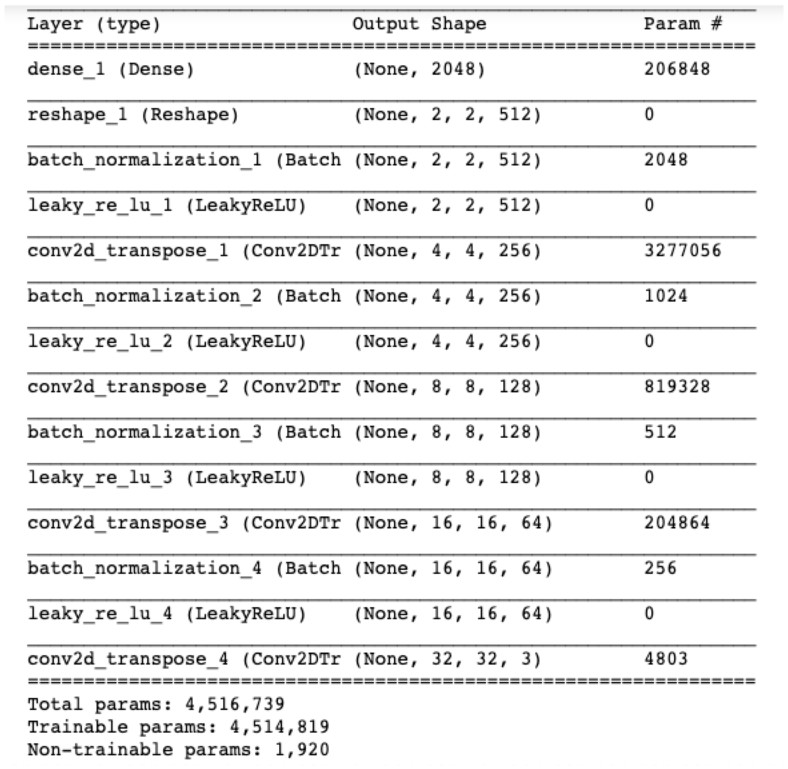
(a) Architecture 1: An effective DCGAN generator architecture for MNIST



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 12544)             1266944
_________________________________________________________________
reshape_4 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose_8 (Conv2DTr (None, 14, 14, 128)       295040
_________________________________________________________________
batch_normalization_6 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
re_lu_6 (ReLU)               (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_9 (Conv2DTr (None, 14, 14, 64)        73792
_________________________________________________________________
batch_normalization_7 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
re_lu_7 (ReLU)               (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_10 (Conv2DT (None, 28, 28, 1)         577
=================================================================
Total params: 1,637,121
Trainable params: 1,636,737
Non-trainable params: 384
_________________________________________________________________
```

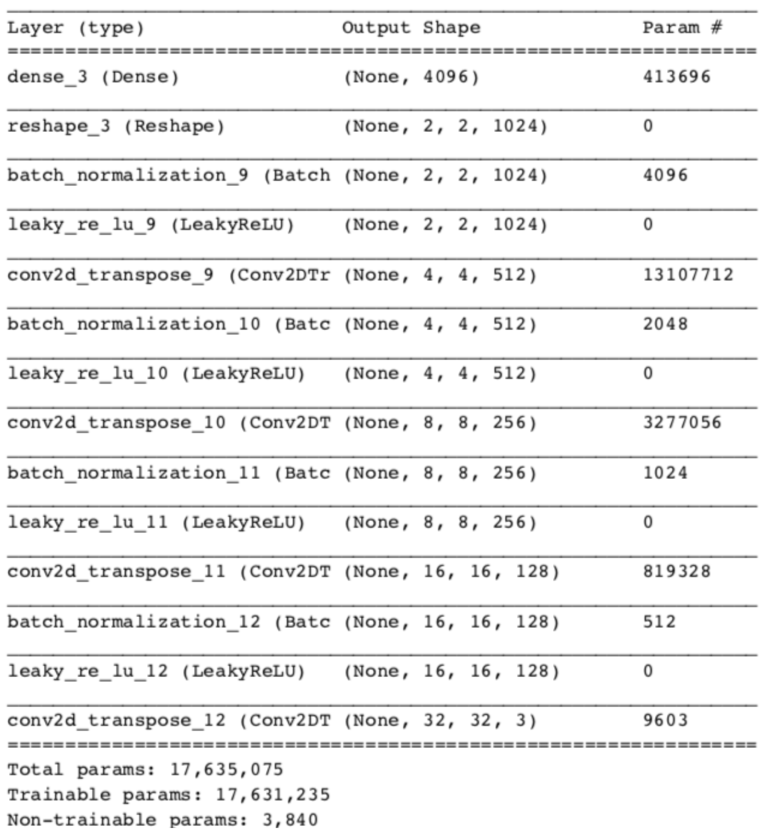
(b) Architecture 2: Deeper DCGAN generator architecture for MNIST

Figure 13: Architectures explored for DCGAN for MNIST



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 2048)              206848    
_________________________________________________________________
reshape_1 (Reshape)          (None, 2, 2, 512)         0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 2, 2, 512)         2048      
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 2, 2, 512)         0         
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 4, 4, 256)         3277056   
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 256)         1024      
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 4, 4, 256)         0         
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 8, 8, 128)         819328    
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 128)         512       
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 128)         0         
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 16, 16, 64)        204864    
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 32, 32, 3)         4803      
=================================================================
Total params: 4,516,739
Trainable params: 4,514,819
Non-trainable params: 1,920
```

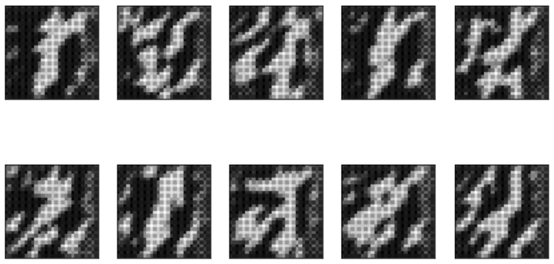
(a) Architecture 1: An effective DCGAN generator architecture for CIFAR-10



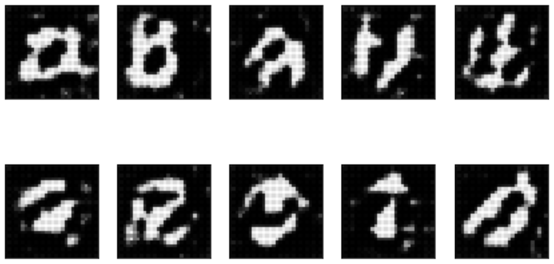
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 4096)              413696    
_________________________________________________________________
reshape_3 (Reshape)          (None, 2, 2, 1024)        0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 2, 2, 1024)        4096      
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 2, 2, 1024)        0         
_________________________________________________________________
conv2d_transpose_9 (Conv2DTr (None, 4, 4, 512)         13107712  
_________________________________________________________________
batch_normalization_10 (Batc (None, 4, 4, 512)         2048      
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 4, 4, 512)         0         
_________________________________________________________________
conv2d_transpose_10 (Conv2DT (None, 8, 8, 256)         3277056   
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 256)         1024      
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 8, 8, 256)         0         
_________________________________________________________________
conv2d_transpose_11 (Conv2DT (None, 16, 16, 128)       819328    
_________________________________________________________________
batch_normalization_12 (Batc (None, 16, 16, 128)       512       
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 16, 16, 128)       0         
_________________________________________________________________
conv2d_transpose_12 (Conv2DT (None, 32, 32, 3)         9603      
=================================================================
Total params: 17,635,075
Trainable params: 17,631,235
Non-trainable params: 3,840
```

(b) Architecture 2: Deeper DCGAN generator architecture for CIFAR-10

Figure 14: Architectures explored for DCGAN for CIFAR-10

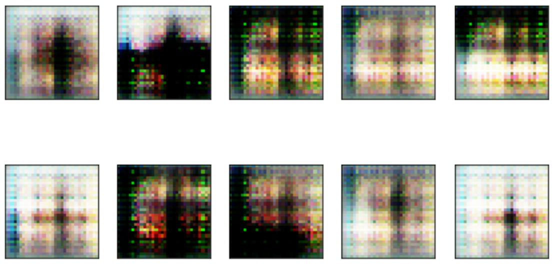


(a) Generated images after first iteration for Architecture 1 for MNIST

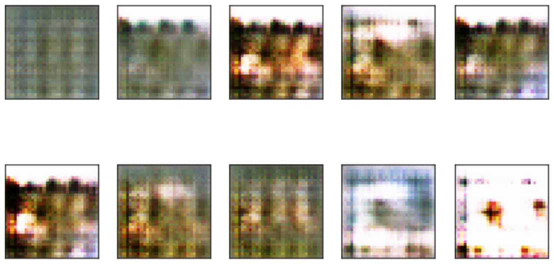


(b) Generated images after first iteration for Architecture 2 for MNIST

Figure 15: Generated images of architectures explored for DCGAN for MNIST

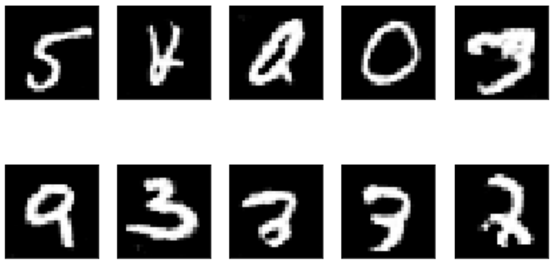


(a) Generated images after first iteration for Architecture 1 for CIFAR-10

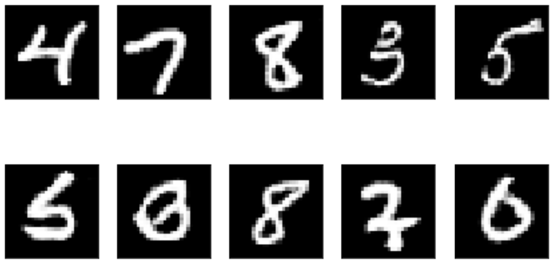


(b) Generated images after first iteration for Architecture 2 for CIFAR-10

Figure 16: Generated images of architectures explored for DCGAN for CIFAR-10

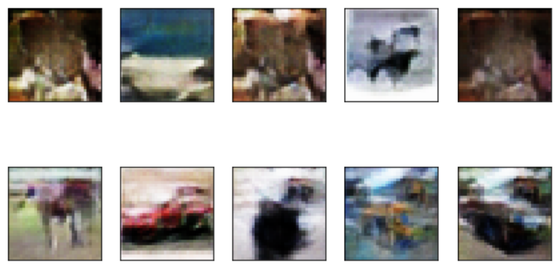


(a) Generated images after final iteration for Architecture 1 for MNIST

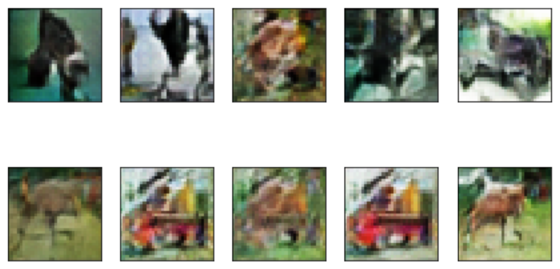


(b) Generated images after final iteration for Architecture 2 for MNIST

Figure 17: Generated images of architectures explored for DCGAN for MNIST



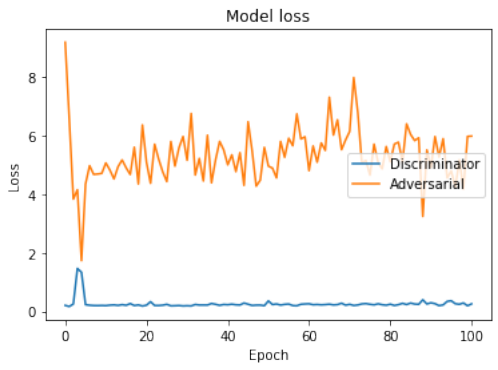
(a) Generated images after final iteration for Architecture 1 for CIFAR-10



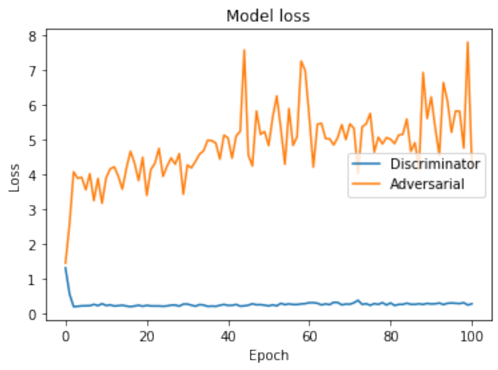
(b) Generated images after final iteration for Architecture 2 for CIFAR-10

Figure 18: Generated images of architectures explored for DCGAN for CIFAR-10

The corresponding loss curves for both architectures are given below.

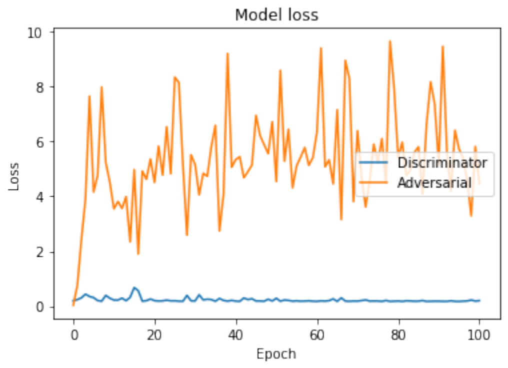


(a) Plot of model losses with Architecture 1 for MNIST

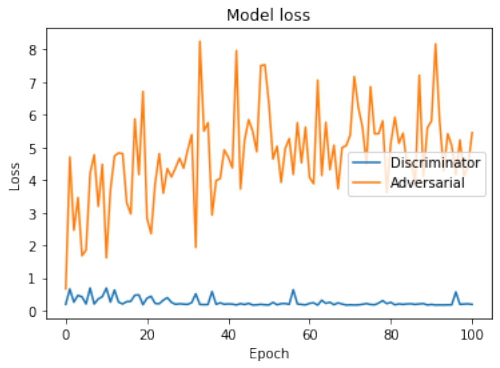


(b) Plot of model losses with Architecture 2 for MNIST

Figure 19: Plot of the model loss for DCGAN for MNIST



(a) Plot of model losses with Architecture 1 for CIFAR-10



(b) Plot of model losses with Architecture 2 for CIFAR-10

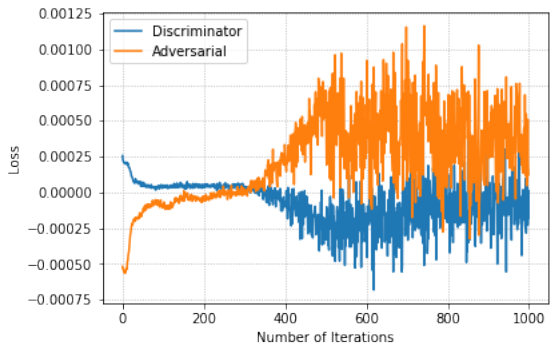
Figure 20: Plot of the model loss for DCGAN for CIFAR-10

Wasserstein Generative Adversarial Network:

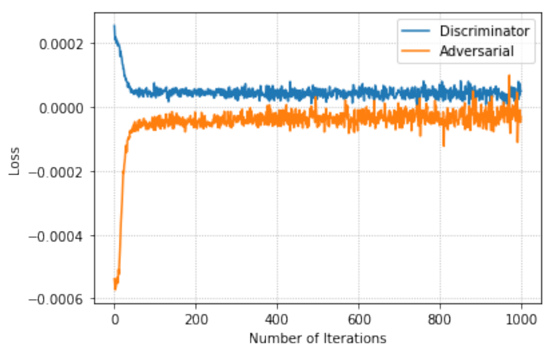
The goal of WGAN is to better stabilize GAN training and diminish some disadvantages of other GAN models such as mode collapse and uninformative loss. Thus, WGAN was implemented with properties such as a meaningful loss metric to correlate generator's convergence and quality of samples. Also, it can improve the overall stability of the optimization process.

Varying latent space dimension:

Given below is the estimated EMD loss of WGAN with latent dimensions 100 and 10. We find the loss starts converging after 600 epoch for latent space of 100. In the case of the smaller dimension, the loss converges fasters. More training seems to result in better quality generated images especially after reaching a state of convergence the quality remains about the same for the model.

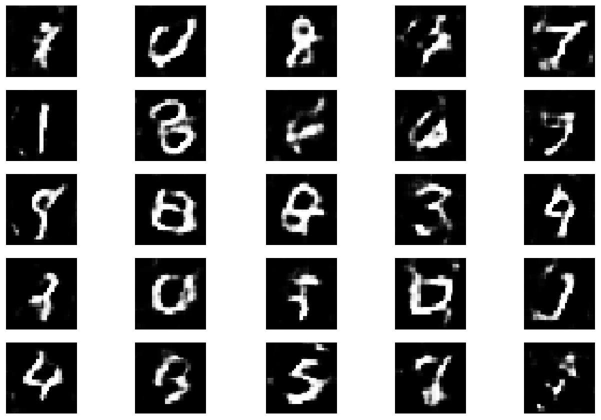


(a) Plot of model losses with latent dimension 100 for MNIST

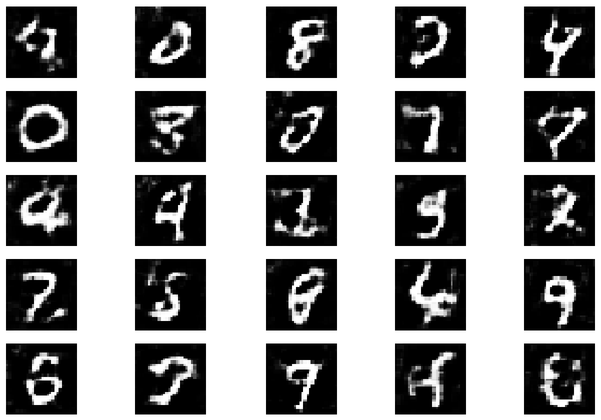


(b) Plot of model losses with latent dimension 10 for MNIST

Figure 21: Plot of the model loss for WGAN for MNIST

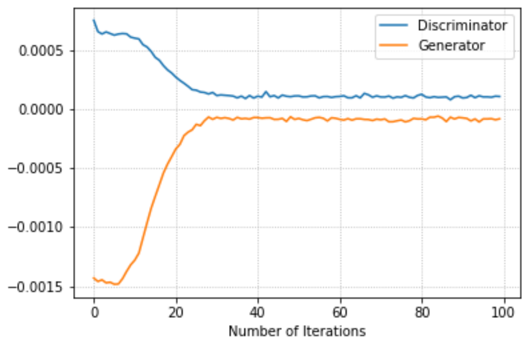


(a) Generated images with latent dimension 100 for MNIST

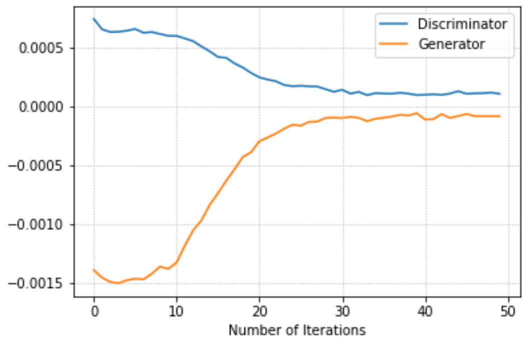


(b) Generated images with latent dimension 10 for MNIST

Figure 22: Generated images for WGAN for MNIST

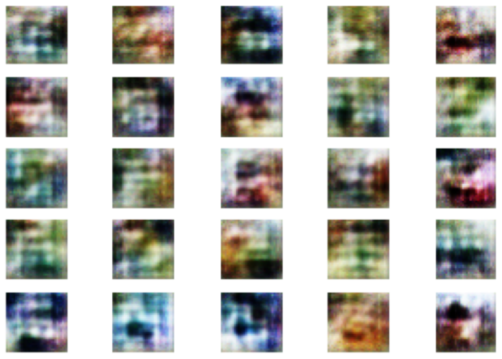


(a) Plot of model losses with latent dimension 100 for CIFAR-10

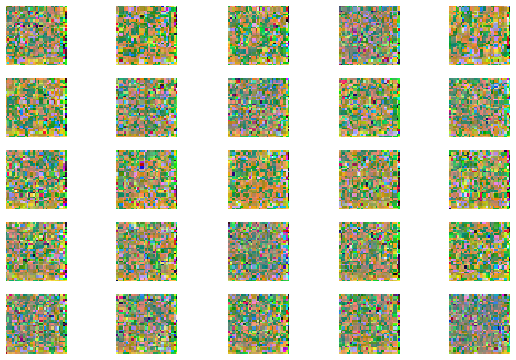


(b) Plot of model losses with latent dimension 10 for CIFAR-10

Figure 23: Plot of the model loss for WGAN for CIFAR-10



(a) Generated images with latent dimension 100 for CIFAR-10

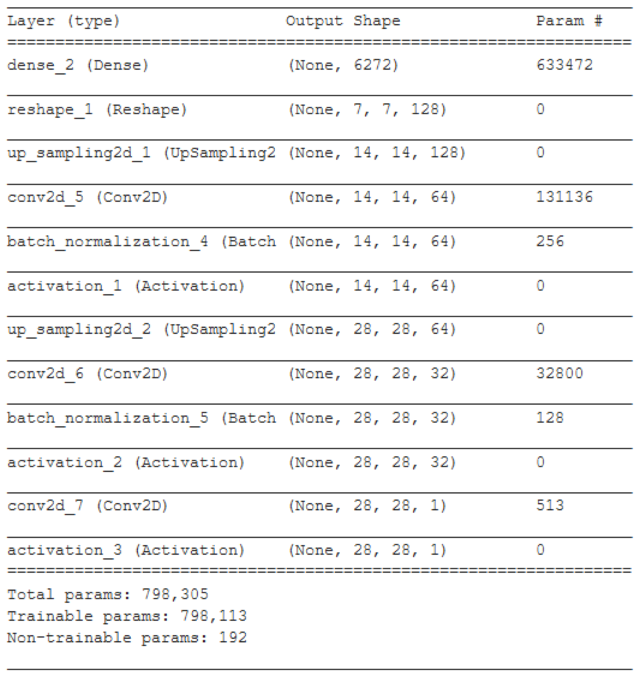


(b) Generated images with latent dimension 10 for CIFAR-10

Figure 24: Generated images for WGAN for CIFAR-10

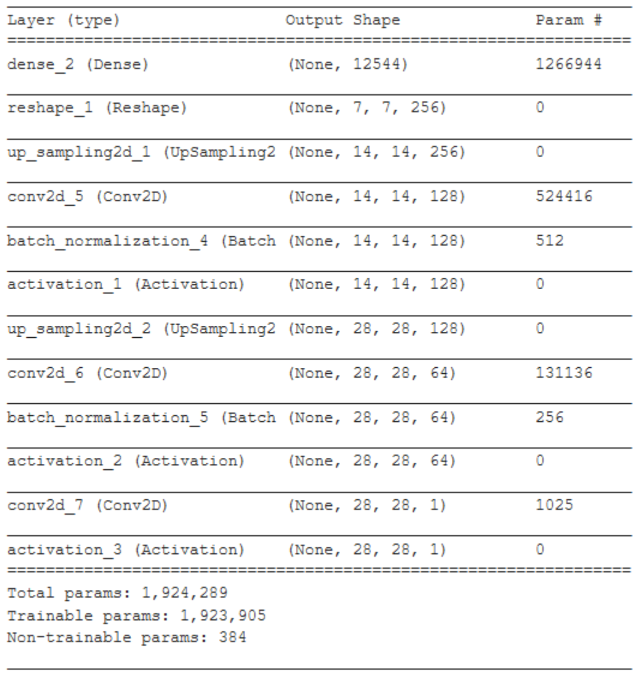
Varying model complexity:

After testing numerous model architecture, the two that are given below are chosen to demonstrate for comparison purposes. The deeper architecture did improve the generated images to create more plausible digits. If the architecture is too simple, the fabricated images remained more like noise. Although the difference is small, the images generated after first 50 epoch with the deeper architecture are preliminary stages of forming edges. With continued tuning of the complexity, it maybe possible to improve results further. CIFAR-10 is a more complex dataset thus the model requires larger number of iterations to generate more realistic looking images than what was explored in the assignment.



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 6272)              633472
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 128)         0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        131136
_________________________________________________________________
batch_normalization_4 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
activation_1 (Activation)    (None, 14, 14, 64)        0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 32)        32800
_________________________________________________________________
batch_normalization_5 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 1)         513
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 798,305
Trainable params: 798,113
Non-trainable params: 192
_________________________________________________________________
```

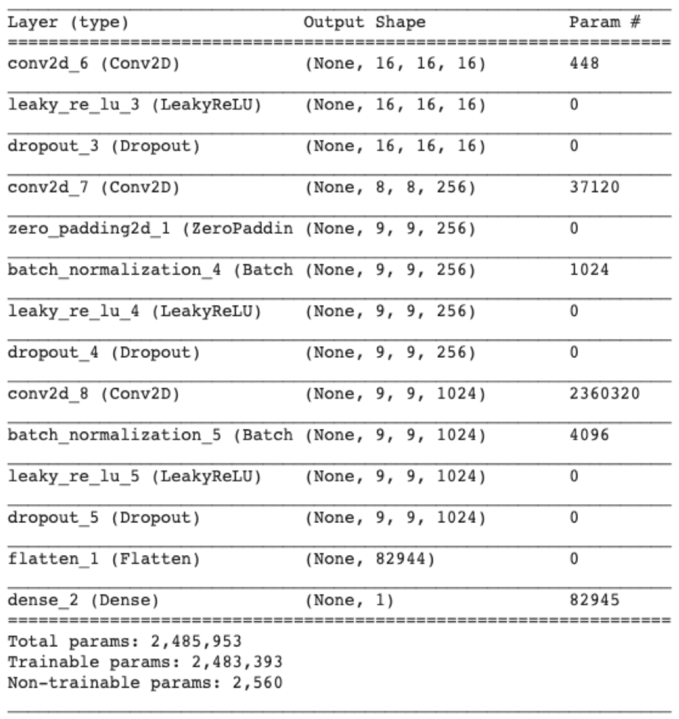
(a) Architecture 1: An effective WGAN generator architecture for MNIST



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 12544)             1266944
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 128)       524416
_________________________________________________________________
batch_normalization_4 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 14, 14, 128)       0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 64)        131136
_________________________________________________________________
batch_normalization_5 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 1)         1025
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 1,924,289
Trainable params: 1,923,905
Non-trainable params: 384
_________________________________________________________________
```

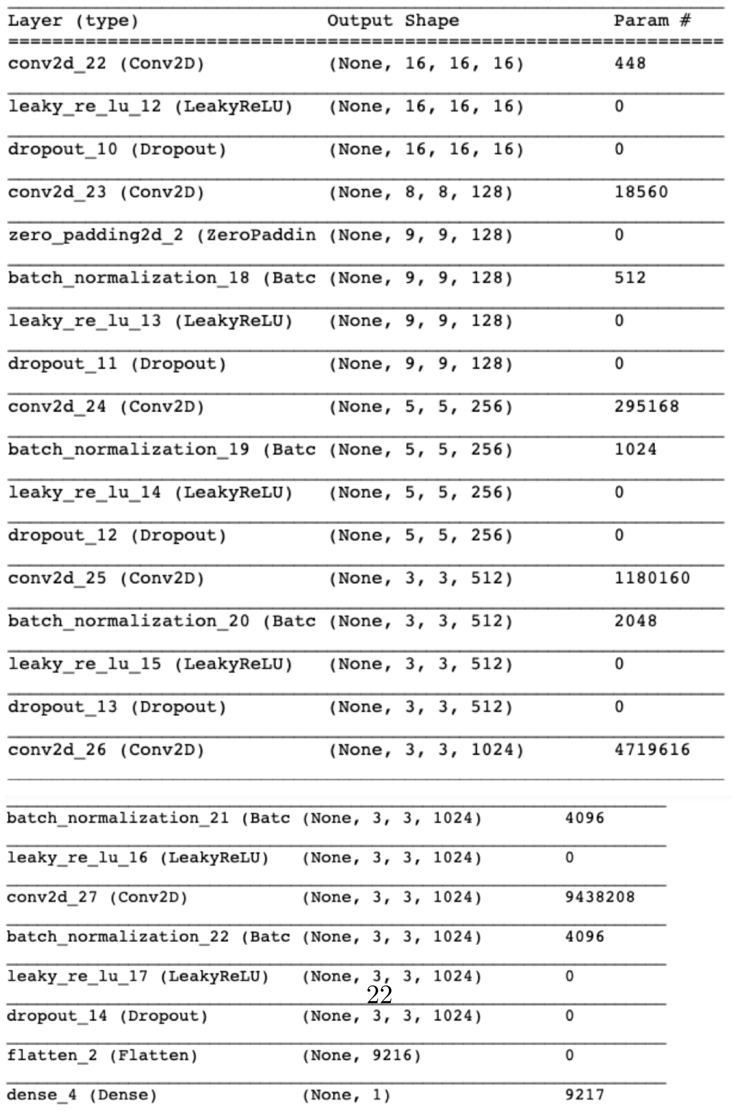
(b) Architecture 2: Deeper WGAN generator architecture for MNIST

Figure 25: Architectures explored for WGAN for MNIST

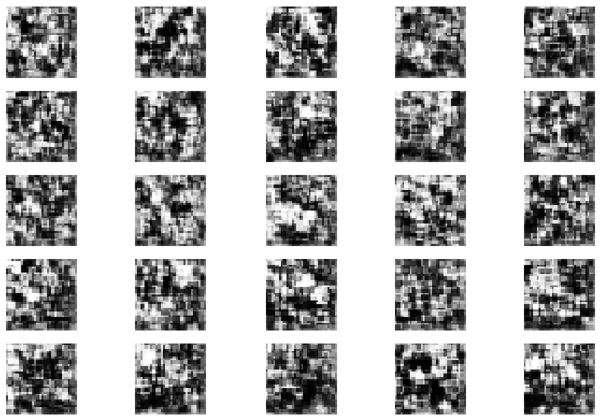


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 16, 16, 16)        448
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 16)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 256)         37120
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 9, 9, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 9, 256)         1024
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 9, 9, 256)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 9, 9, 1024)        2360320
_________________________________________________________________
batch_normalization_5 (Batch (None, 9, 9, 1024)        4096
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 9, 9, 1024)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 9, 9, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 82944)             0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 82945
=================================================================
Total params: 2,485,953
Trainable params: 2,483,393
Non-trainable params: 2,560
_________________________________________________________________
```

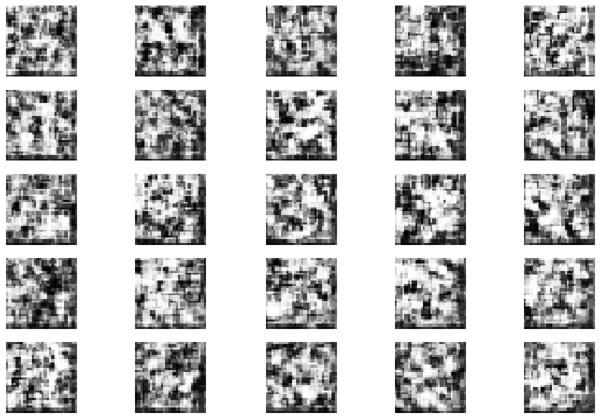
(a) Architecture 1: An effective WGAN generator architecture for CIFAR-10



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 16, 16, 16)        448
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 16, 16, 16)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 8, 8, 128)         18560
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 9, 9, 128)         0
_________________________________________________________________
batch_normalization_18 (Batc (None, 9, 9, 128)         512
_________________________________________________________________
leaky_re_lu_13 (LeakyReLU)   (None, 9, 9, 128)         0
_________________________________________________________________
dropout_11 (Dropout)         (None, 9, 9, 128)         0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 5, 5, 256)         295168
_________________________________________________________________
batch_normalization_19 (Batc (None, 5, 5, 256)         1024
_________________________________________________________________
leaky_re_lu_14 (LeakyReLU)   (None, 5, 5, 256)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 3, 3, 512)         1180160
_________________________________________________________________
batch_normalization_20 (Batc (None, 3, 3, 512)         2048
_________________________________________________________________
leaky_re_lu_15 (LeakyReLU)   (None, 3, 3, 512)         0
_________________________________________________________________
dropout_13 (Dropout)         (None, 3, 3, 512)         0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 3, 3, 1024)        4719616
_________________________________________________________________
_________________________________________________________________
batch_normalization_21 (Batc (None, 3, 3, 1024)        4096
_________________________________________________________________
leaky_re_lu_16 (LeakyReLU)   (None, 3, 3, 1024)        0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 3, 3, 1024)        9438208
_________________________________________________________________
batch_normalization_22 (Batc (None, 3, 3, 1024)        4096
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)   (None, 3, 3, 1024)        0
_________________________________________________________________
dropout_14 (Dropout)         (None, 3, 3, 1024)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 9217
=================================================================
```

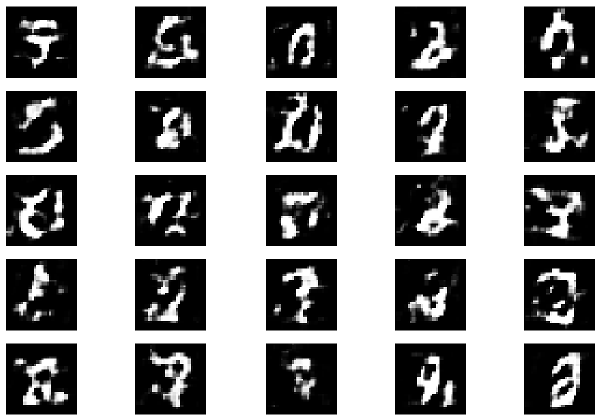


(a) Generated images with Architecture 1 for MNIST after initial 50 epoch

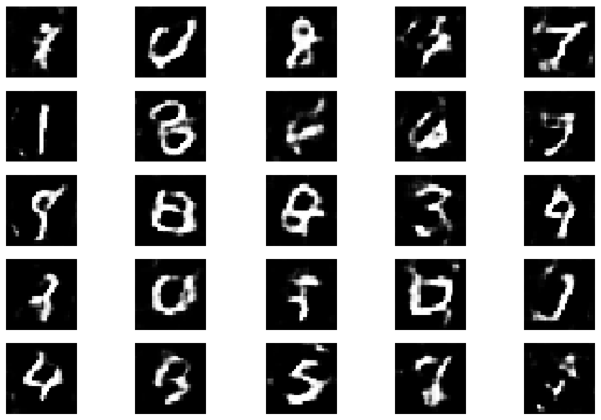


(b) Generated images with Architecture 2 for MNIST after initial 50 epoch

Figure 27: Generated images for WGAN for MNIST

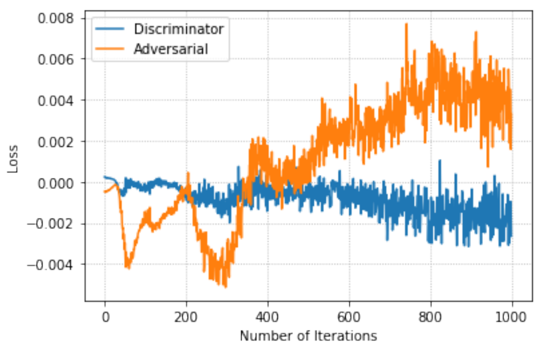


(a) Generated images with Architecture 1 for MNIST after final epoch

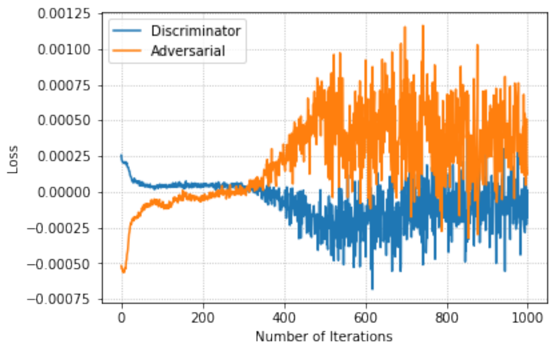


(b) Generated images with Architecture 2 for MNIST after final epoch

Figure 28: Generated images for WGAN for MNIST

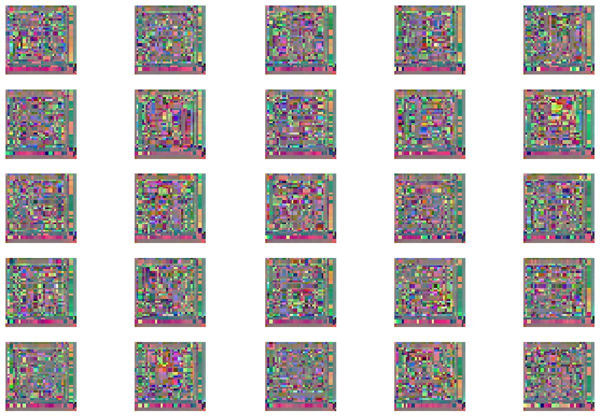


(a) Plot of model losses with Architecture 1 for MNIST

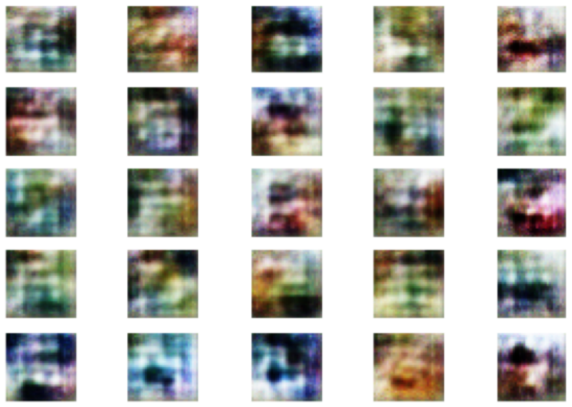


(b) Plot of model losses with Architecture 2 for MNIST

Figure 29: Plot of the model loss for WGAN for MNIST

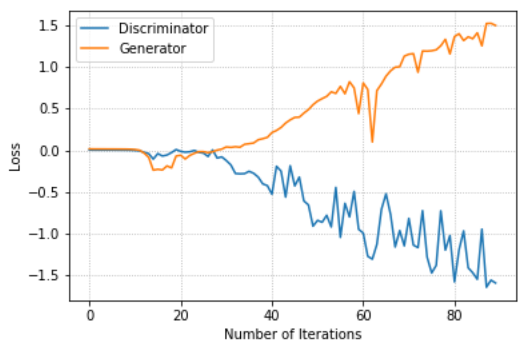


(a) Generated images with Architecture 1 for CIFAR-10 after final epoch

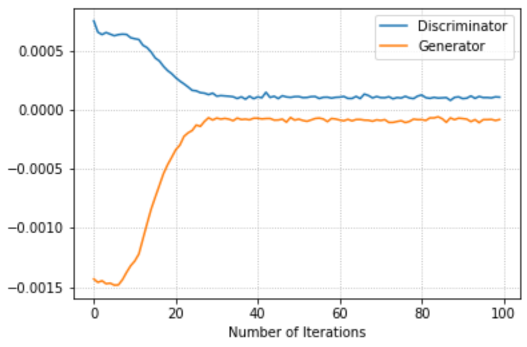


(b) Generated images with Architecture 2 for CIFAR-10 after final epoch

Figure 30: Generated images for WGAN for CIFAR-10



(a) Plot of model losses with Architecture 1 for CIFAR-10



(b) Plot of model losses with Architecture 1 for CIFAR-10

Figure 31: Plot of the model loss for WGAN for CIFAR-10