CSI 5340 – Deep Learning and Reinforcement Learning

Assignment 4

Student Name: Nathaniel Bowness

Student Number: 7869283

Due Date: December 7th, 2023

This assignment asks students to train three VAE, GAN, and WGAN models on both the MNIST and CIFAR 10 datasets. As part of the training evaluation, the "loss" for the different models will be analyzed, and the generated images from each model type will be compared to the original images. The first main section will show the results for all three trained models on the MNIST dataset. The second main section will then show the results of three trained models on the CIFAR 10 dataset.

MNIST Dataset:

Real Images:

Some sample images from the MNIST dataset are shown in Figure 1 to compare with the results shown in later sections.

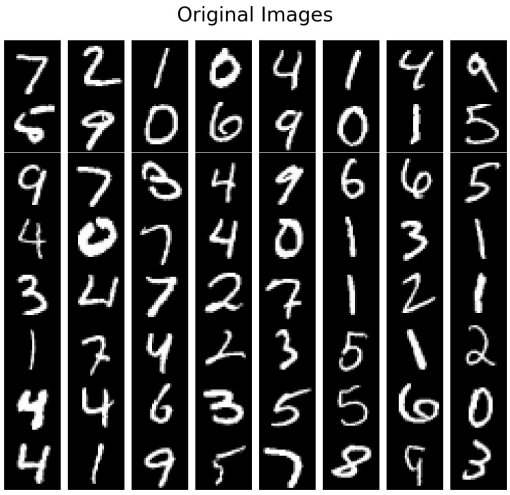


Figure 1: *Sample images from the original test MNIST dataset.*

Variational Autoencoder (VAE) Model

The VAE model was the simplest, in terms of complexity, used in this assignment. However, it still did a good job of reproducing and generating numbers based on the inputted test data tensors. For the VAE model, I compared models with latent space dimensions of 2, 32, and 128, respectively. The model was configured as shown in Table 1 below. Between each main network layer was a ReLU and Dropout layer as well. Different values of the hidden dimension and the latent space dimension size were tested.

Table 1*: VAE Model parameters used for training on the MNIST dataset.*

Parameter	Value	Details
Encoding Layers	3 Layers - ReLU - Dropout=0.2	784 → HiddenDim HiddenDim → HiddenDim HiddenDim → LatentSize
Decoding Layers	3 Layers - ReLU - Dropout=0.2	LatentSize → HiddenDim HiddenDim → HiddenDim HiddenDim → 784
Hidden Dimension	[64, 512]	-
Latent Space Dimension	[2, 32, 128]	-
Epochs	15	-
Optimizer	Adam	-
Total Loss	Binary Cross-Entropy + KL Divergence	-
Learning Rate	0.001	-

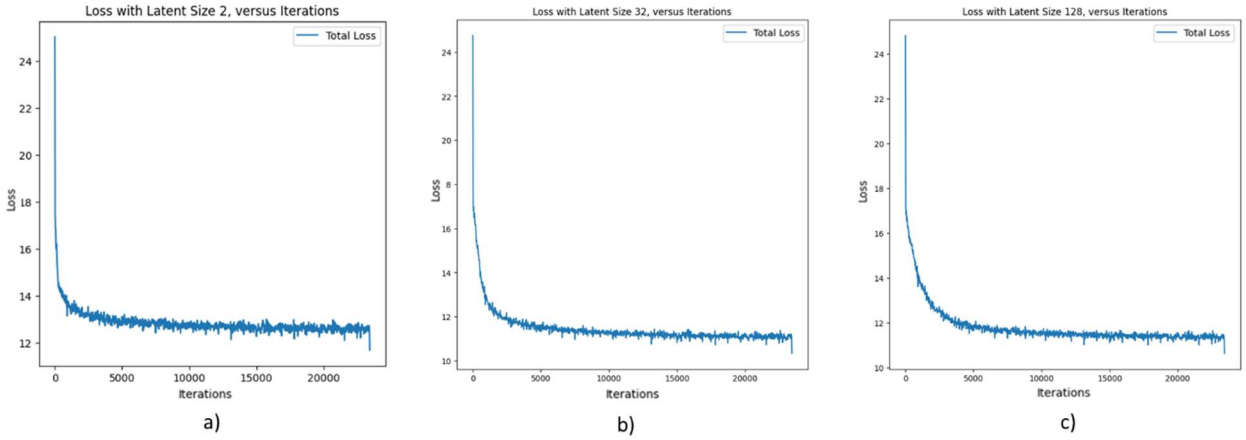


Figure 2: *Total Loss (Binary Cross-Entropy + KL Divergence) for the VAE model, with a hidden dimension of 512 and plotted against the number of iterations through 15 epochs; a), b), c) shows the results with a latent space dimension of 2, 32 and 128 respectively.*

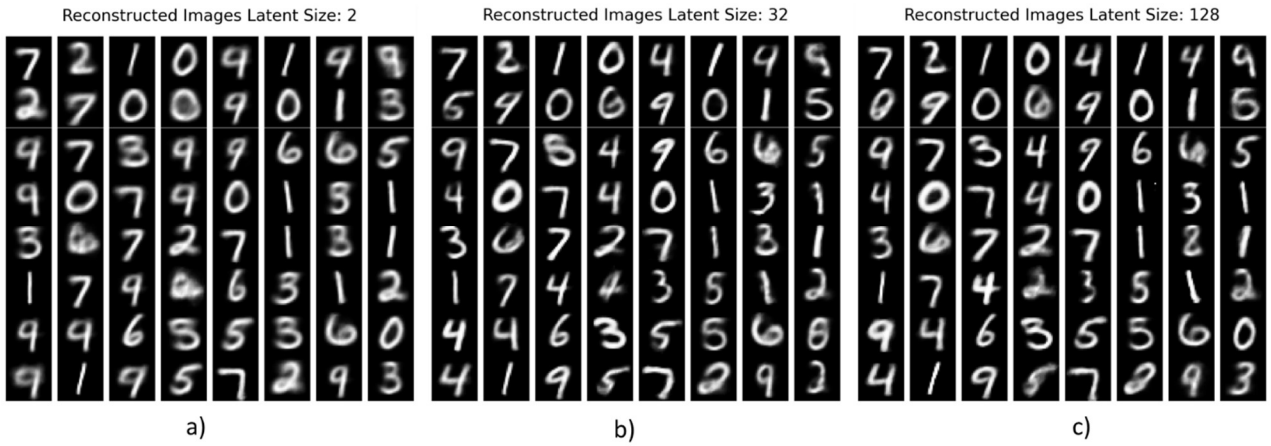


Figure 3: *Generated MNIST images, with a hidden dimension of 512, after 15 epochs using the same random seed to enable easy comparison; a), b), c) shows the results with a latent space dimension of 2, 32 and 128, respectively.*

As can be seen in Figure 2, the loss was the least with the highest latent space dimension. However, with the higher latent space dimension, it took more iterations for the model to converge. This shows the more complex model will take additional time to train as expected.

The results and comparison show that as the latent space size increases, so does the performance and accuracy of the images. The output results, seen in Figure 3 c) for a latent space size of 128 look better than a space size of 32. This can partially be attributed to the larger NN size used for training with a vector size of 128 and can store more weights.

During some experimentation not shown here, a much simpler model with a hidden dimension of 64 was tested. I found the results were much worse. The straightforward network was not able to generate images like the original ones. The hidden size of 64 led to images created that were primarily black with some white “globs.”

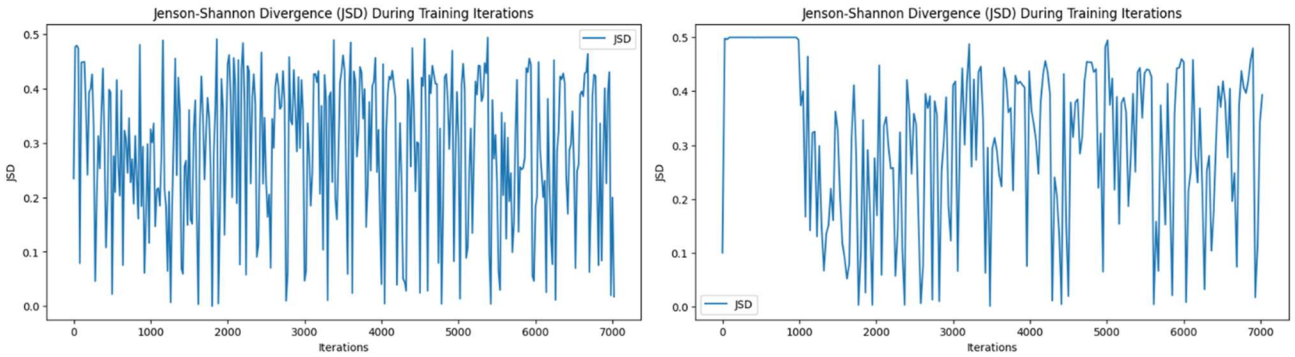
Generative Adversarial Network (GAN) Model

During research and through some experimentation, all the GAN and WGAN networks will be implemented using convolutional networks as they perform much better than traditional linear networks. So, unless otherwise stated, all the experiments shown are on DC-GAN and DC-WGAN style models, which perform much better than GAN and WGAN. I initially tried to create a linear GAN network model for this question, and the images output were relatively poor. They were hard to distinguish as any handwritten number.

The model used for the GAN is shown below in Table 2. It uses 4 convolution layers, with 1 convolutional output layer. Each of the first 4 layers has a ReLU and BatchNorm layer between them. The output layer for the encoder has a Tanh function applied at the output. The Decoding layer used LeakyReLU between the layers, with a Sigmoid function after the output layer. The latent space size was changed between 64 and 256, and the two variations will have separate plots below.

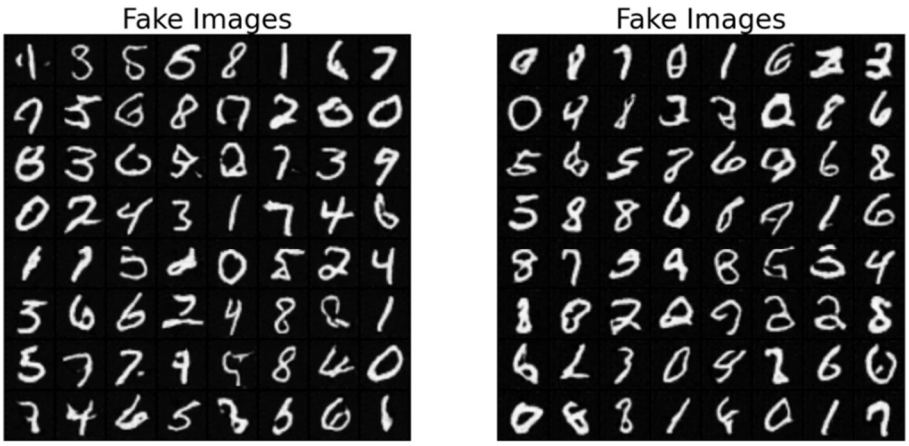
***Table 2**: Convolutional GAN model parameters used for training on the MNIST dataset. For reference: K=Kernel Size, S=Stride, P=Padding.*

Parameter	Value	Details
Encoding Layers	4 layers - BatchNorm - ReLU + Output layer - Tanh	LatentSize → 512 (K=4, S=1, P=0) 512 → 256 (K=4, S=2, P=1) 256 → 128 (K=4, S=2, P=1) 128 → 64 (K=4, S=2, P=1) 64 → 1
Decoding Layers	4 Layers - BatchNorm - LeakyReLU + Output layer - Sigmoid	1 → 64 (K=4, S=2, P=1) 64 → 128 (K=4, S=2, P=1) 128 → 256 (K=4, S=2, P=1) 256 → 512 (K=4, S=2, P=1) 512 → LatentSize (K=4, S=1, P=0)
Latent Space Size	[64, 256]	-
Epochs	15	-
Optimizer	Adam	-
Loss Function	Binary Cross Entropy	-
Learning Rate	0.0002	-
Batch Size	128	-



a) Latent Size=64 b) Latent Size=256

Figure 4: *Jenson-Shannon Divergence (JSD) during the training iterations of 15 epochs; a) and b) show the results with a latent space dimension of 64 and 128, respectively.*



a) Latent Size=64 b) Latent Size=256

Figure 5: *Generated MNIST images of the GAN model after 15 epochs; a) and b) show the results with a latent space dimension of 64 and 128, respectively.*

From Figure 4, the JSD curve for both models with different latent space dimensions appears similar. The ideal goal of the JSD curve is to have it fluctuating above and below the 0.5 value, which means the generator is "fooling" the discriminator about 50% of the time. In both graphs, the JSD value appears to hover around 30-40%, which shows that while starting to get close, the model hasn't hit that ideal value yet. The slightly lower value could also be attributed to the images' simplicity, where the generator consistently outperforms the discriminator.

In Figure 5b), the output of the latent space dimension model seems to be more noisy and not quite as clear as the 5a). I think that can be attributed to more training time being required for the larger latent space size used to give as clear outputs. This was also noted in the JSD curve, where the larger the latent space, the more training time is needed for convergence of the model. I will mention that the images in

Figure 5b) do appear to be thinner and more concise shapes, so if given enough time to train, there may have been much better results.

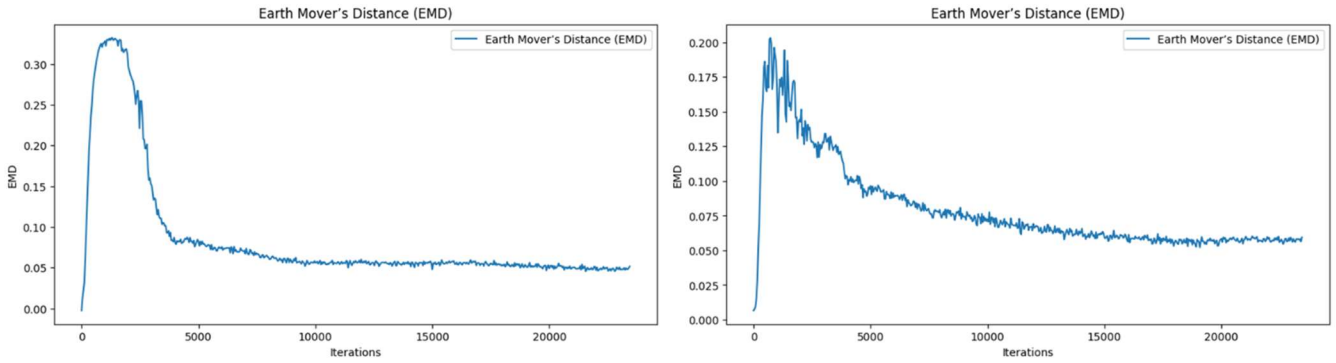
Wasserstein Generative Adversarial Network (WGAN) Model

Similar to GAN above, the WGAN was implemented with convolutional layers as they are much more performant on images, meaning the implementation is technically DC-WGAN. The model parameters and configuration can be seen in Table 3 below. The generator and discriminator each contained 4 layers + 1 additional output layer. The generator like in GAN, uses BatchNorm and ReLU between each layer and in the output, the layer applies the Tanh function. The discriminator layer uses LeakyReLU instead of the regular layer, and the output layer does not perform any transformations after it.

During experimentation, I tried both Adam and RMSprop, and I found that the performance of RMSprop was much better and produced much better images. The WGAN model also used weight clipping, which was implemented with a value of 0.01. I noticed the impact of weight clipping did improve the model performance, but too high a value led to poor results.

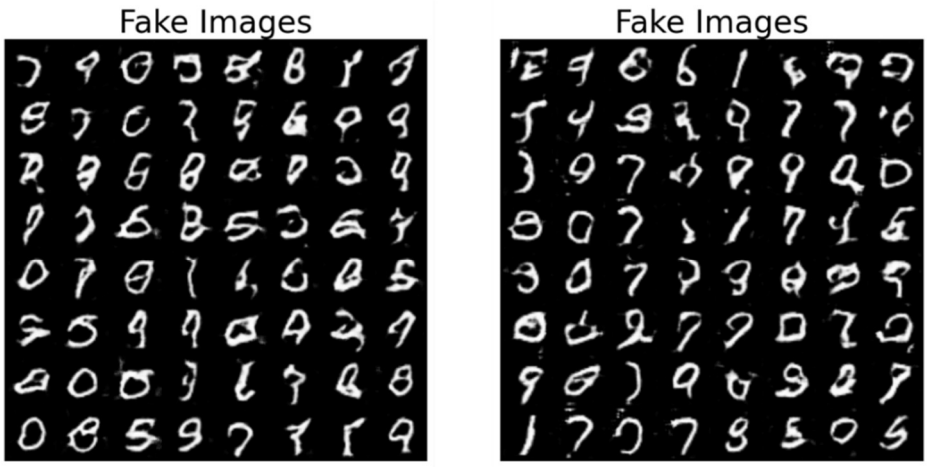
***Table 3**: Convolutional WGAN model parameters used for training on the MNIST dataset. For reference: K=Kernel Size, S=Stride, P=Padding.*

Parameter	Value	Details
Generator Layers	4 layers - BatchNorm - ReLU + Output layer - Tanh	LatentSize → 128 (K=4, S=1, P=0) 128 → 64 (K=4, S=2, P=1) 64 → 32 (K=4, S=2, P=1) 32 → 16 (K=4, S=2, P=1) 16 → 1
Discriminator Layers	4 Layers - BatchNorm - LeakyReLU + Output layer	1 → 16 (K=4, S=2, P=1) 16 → 32 (K=4, S=2, P=1) 32 → 64 (K=4, S=2, P=1) 64 → 128 (K=4, S=2, P=1) 128 → 1 (K=4, S=1, P=0)
Latent Space Size	[64, 256]	-
Epochs	25	-
Optimizer	RMSprop	-
Loss Function	Binary Cross Entropy	-
Learning Rate	0.0002	-
Batch Size	128	-
Weight Clipping	0.01	-



a) Latent Size=64 b) Latent Size=256

Figure 6: *Earth Mover's Distance (EMD) during the training iterations of 15 epochs; a) and b) show the results with a latent space dimension of 64 and 256, respectively.*



a) Latent Size=64 b) Latent Size=256

Figure 7: *Generated MNIST images of the GAN model after 25 epochs; a), and b) show the results with a latent space dimension of 64 and 128, respectively.*

The Earth Mover's Distance (EMD) can be seen above in Figure 6 for each of the training iterations with latent space dimensions of 64 and 256. A bit counter-intuitively, I found the results from the lower latent space dimension model to be better than the higher latent dimension of 256. The EMD curve (Figure 6a) is smoother and converged much quicker to a lower value than in Figure 6b). It's possible that a more complex latent space size would have reached the same value with more iterations. My intuition also thinks the convolutional network dimensions may have slightly impacted the result. With the higher latent dimension size, the larger value would have been the input layer, which may have negatively impacted the results.

The results of the generated images can be seen in Figure 7. Objectively, I think the results in Figure 7a) with the smaller latent space dimensions look better, as 7b) has more noise and the images are less concise. I think some additional testing to let the larger model run through more iterations would show if it had not completed training or if that large latent space dimension was too large for the other small network dimensions and resulted in poor results.

Typically, I would expect the larger latent space dimension to be better with enough training time.

Comparison

Overall, the MNIST-generated data from the simple models was quite impressive. The VAE images look the most like the training data, but they were developed by taking a test vector and outputting an image from that test vector, so that is to be expected. I think the WGAN model did the best for generating images from random vectors. However, it also took longer to train effectively. The GAN model with a latent space dimension of 64 was quite impressive as it was quick to train and had good results compared to the others.

CIFAR Dataset

Real Images:

For comparison, some sample images from the CIFAR dataset are shown in Figure 8 to compare with the image results that will be shown later.

Figure 8: *Sample images from the original test CIFAR dataset.*

Variational Autoencoder (VAE) Model

The VAE model was the simplest in terms of complexity used to try and regenerate the CIFAR images. While it performed decently on the MNIST dataset, it struggled on the CIFAR dataset to reproduce reasonable results. To try to improve the results, a convolutional model of VAE was also implemented, and the results are shown below.

For the VAE model, I compared models with the latent space vector dimensions of 2 and 128. The results for the latent vector dimension of 32 are not shown for the CIFAR dataset, as there was little discernible difference between 32 and 128. The construction parameters for the linear model can be seen below in Table 4.

***Table 4**: Linear VAE model parameters used for training on the CIFAR dataset.*

Parameter	Value	Details
Encoding Layers	3 Linear Layers	784 → HiddenDim HiddenDim → HiddenDim HiddenDim → LatentSize
Decoding Layers	3 Linear Layers	LatentSize → HiddenDim HiddenDim → HiddenDim HiddenDim → 784
Hidden Dimension	[64, 512]	-
Latent Vector Size	[2, 128]	-
Epochs	20	-
Optimizer	Adam	-
Total Loss	Binary Cross-Entropy + KL Divergence	-
Learning Rate	0.001	-

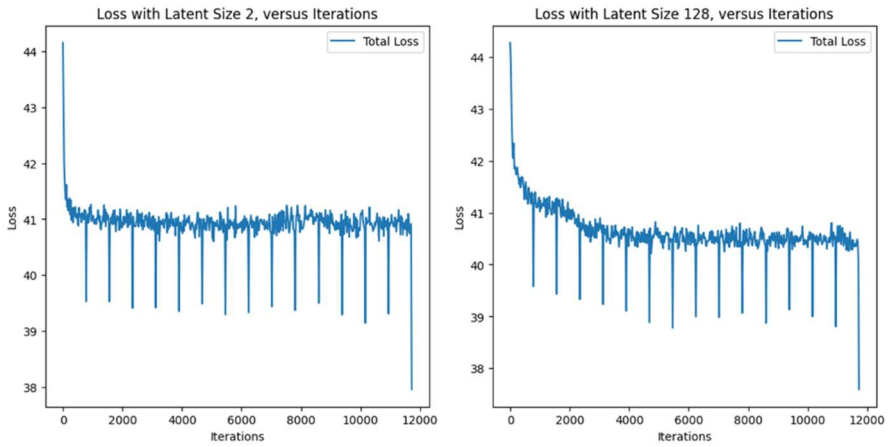


***Figure 9:** Total Loss (Binary Cross-Entropy + KL Divergence) for the linear VAE model, with a hidden dimension of 512 and plotted against the number of iterations through 20 epochs; a), b) show the results with a latent space dimension of 2 and 128 respectively.*

The total loss of the linear VAE model can be seen in Figure 9. The total loss of around 41 was not good and showed the model could not converge to a lower loss value. This was very apparent from the created images in Figure 10 below. All the photos are grey blurs that do not show the colour or overall shape of the image well. The model performs about the same with a latent space size of 2 and 128, with a slight improvement at a latent space size of 128. Overall, though, the results are poor.

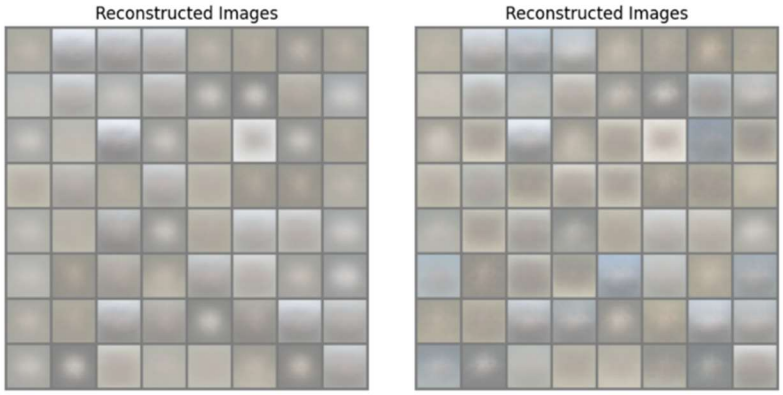


Figure 10: *Generated CIFAR images by the linear VAE model, with a hidden dimension of 512, after 20 epochs using the same random seed, to enable easy comparison; a), b) shows the results with a latent space dimension of 2 and 128 respectively.*

To improve the results that were seen with the linear VAE model, I did some research to find that convolution VAE models can perform much better on image data. So, I created a second VAE model with the layout in Table 5 below. For the encoder, the 3 convolutional layers have a ReLU layer between them with a final linear output layer. For the decoder, there is a linear input layer with 3 convolution layers after it, that end with a sigmoid function before the output.

Table 5*: Convolutional VAE model parameters used for training on the CIFAR dataset. For reference: K=Kernel Size, S=Stride, P=Padding.*

Parameter	Value	Details
Encoding Layers	3 Conv2d Layers - ReLU 1 Linear Layer	3 → 32 (K=4, S=2, P=1) 32 → 64 (K=4, S=2, P=1) 64 → 128 (K=4, S=2, P=1) 2048 → LatentSize*2
Decoding Layers	1 Linear Layer 3 Conv2d Layers - ReLU	LatentSize → 2048 128 → 64 (K=4, S=2, P=1) 64 → 32 (K=4, S=2, P=1) 32 → 3 (K=4, S=2, P=1)
Latent Vector Size	[2, 128]	-
Epochs	15	-
Optimizer	Adam	-
Total Loss	Binary Cross-Entropy + KL Divergence	-
Learning Rate	0.001	-

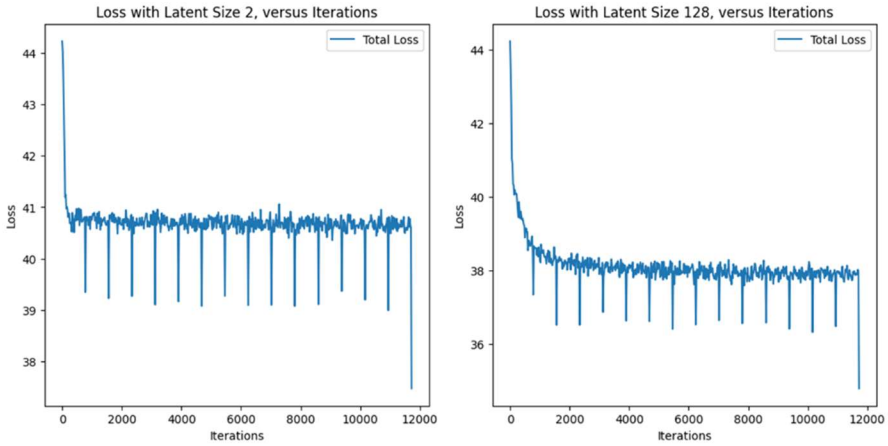


Figure 11: *Total Loss (Binary Cross-Entropy + KL Divergence) for the convolutional VAE model, plotted against the number of iterations through 20 epochs; a), b) show the results with a latent space dimension of 2 and 128 respectively.*

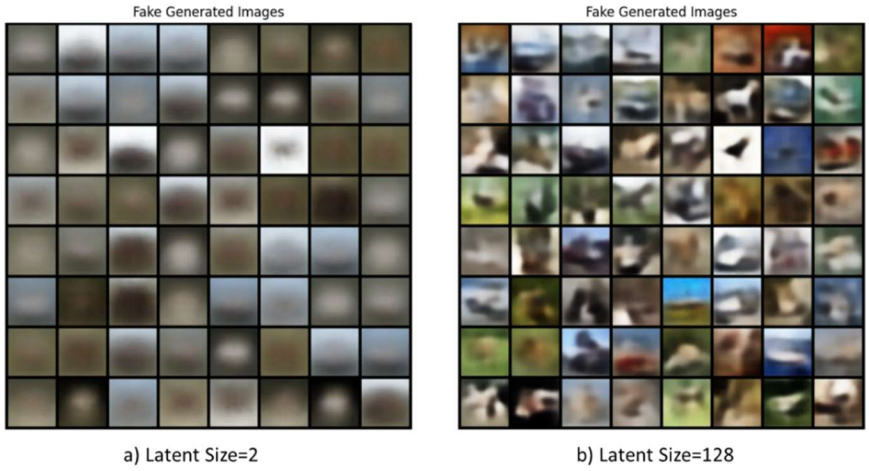


Figure 12: *Generated CIFAR images by the convolutional VAE model after 20 epochs; a) and b) show the results with a latent space dimension of 2 and 128, respectively.*

The loss of the convolutional VAE model for the smaller latent space dimension of 2 did not see much improvement, as seen in Figures 11a) and 12a). The total loss still hovered around 41, and the images contain minimal colour. However, there is a large improvement when increasing the latent size to 128. The loss decreases to around 38, as seen in Figure 11 b), and the images contain colour and start to represent some of the originals in Figure 12 b).

This example does a great job of demonstrating how the increased latent space dimension is essential to the model. A small value will distort the output of the same model using a large value.

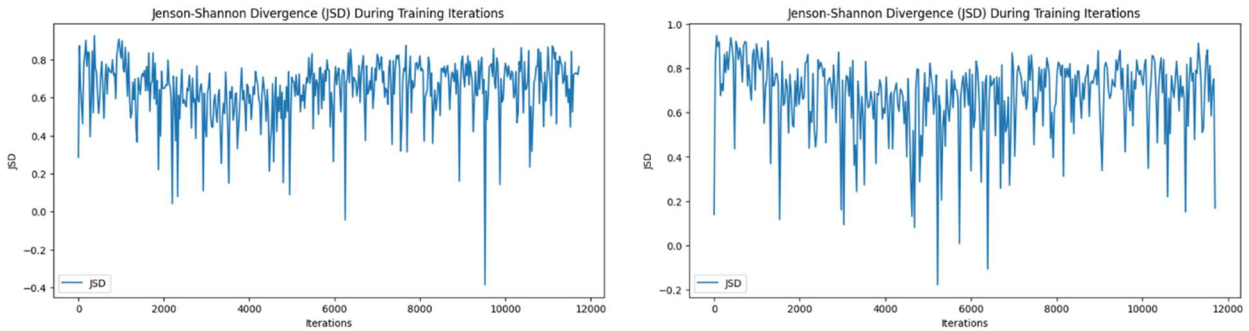
Overall, the VAE model, even when using convolution layers, has trouble representing the CIFAR images and reproducing them. I think the results would be better with a more complex model with more layers or a larger width in each layer, but overall, it does not do well with the more complex dataset.

Generative Adversarial Network (GAN) Model

To get better results, the GAN model used for generating CIFAR images was another convolutional network. The details of the model can be seen in Table 6 below. Like the model used for MNIST, the 4 encoding layers use BatchNorm and ReLU, whereas the Decoding layers use BatchNorm and LeakyReLU between each. The output layer of the encoding layer uses a Tanh function, whereas the decoding layer uses a sigmoid function. The latent size was modified during a test to see the effect on the overall generated images. The size of the individual layers was also modified during testing until I found the current model that was relatively quick to train and had reasonable results.

***Table 6**: Convolutional GAN model parameters used for training on the CIFAR dataset. For reference: K=Kernel Size, S=Stride, P=Padding.*

Parameter	Value	Details
Encoding Layers	4 layers - BatchNorm - ReLU + Output layer - Tanh	LatentSize → 512 (K=4, S=1, P=0) 512 → 256 (K=4, S=2, P=1) 256 → 128 (K=4, S=2, P=1) 128 → 64 (K=4, S=2, P=1) 64 → 1
Decoding Layers	4 Layers - BatchNorm - LeakyReLU + Output layer - Sigmoid	1 → 64 (K=4, S=2, P=1) 64 → 128 (K=4, S=2, P=1) 128 → 256 (K=4, S=2, P=1) 256 → 512 (K=4, S=2, P=1) 512 → LatentSize (K=4, S=1, P=0)
Latent Space Size	[64, 256]	-
Epochs	15	-
Optimizer	Adam	-
Loss Function	Binary Cross Entropy	-
Learning Rate	0.0002	-
Batch Size	128	-



a) Latent Size=64 b) Latent Size=256

***Figure 13:** Jenson-Shannon Divergence (JSD) of the GAN model, during the training iterations of 25 epochs; a), b) show the results with a latent space dimension of 64, and 256 respectively.*

The JSD of the models plotted against the used iterations can be seen in Figure 12 above. The JSD curve shows that both models have consistently competing generators and discriminators, and neither was over-trained. With a latent of 64, the JSD value averages around 0.7 after the training iterations, whereas with a latent space size of 256, the JSD value averages around 0.6. Ideally, the JSD value would be around 0.5, but overall, the higher latent space size had better results. This can also be seen from the generated images below in Figure 14. The photos in 14 a) with a latent space size of 64 appear less concise and blurrier. The pictures in 14 b) appear to be much clearer. However, what they are showing is not exactly obvious. A couple images appear to be car-like, but the animals only represent the shapes – they are not discernable by my eye.

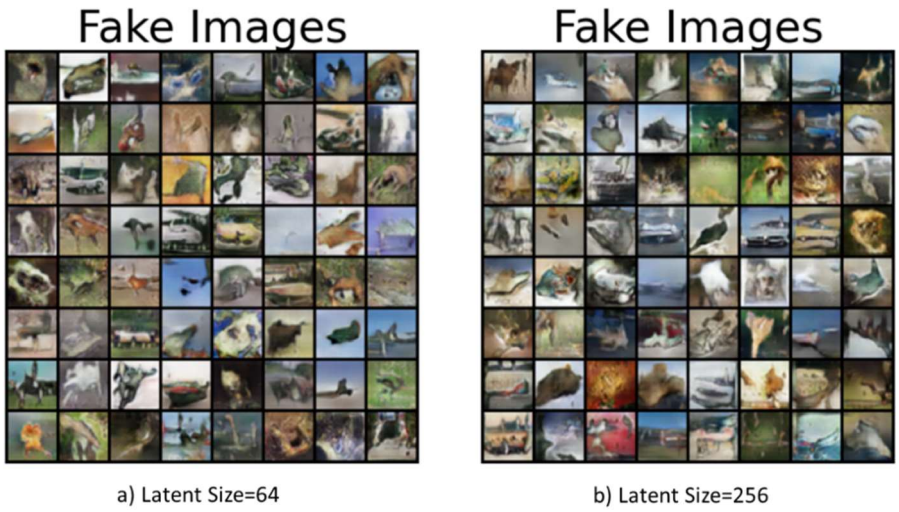


Figure 14: *Generated CIFAR images by the convolutional GAN model after 25 epochs; a) b) Show the results with latent space dimensions of 64 and 256, respectively.*

Overall, the GAN model performed much better than the VAE convolution model in generating images. The convolutional layers once again output better results for the images, and the larger latent space size also yielded better results.

Wasserstein Generative Adversarial Network (WGAN) Model

The final WGAN model was only implemented with Convolutional layers because of the better performance compared to linear layers. Like the MNIST model, the WGAL used 4 layers + 1 output layers for the generator and discriminator. The details of the model can be seen in Table 7 below. Like previous models, the generator uses ReLU and a Tanh output function, while the discriminator uses LeakyReLU, and no function is applied at the output.

For WGAN, I tested both the Adam optimizer and the RMSprop and found the results from RMSprop to be much better, so the results displayed will be using RMSprop. The weight clipping value used in the model was also set to 0.01 a popular value I found online.

***Table 7**: Convolutional WGAN model parameters used for training on the CIFAR dataset. For reference: K=Kernel Size, S=Stride, P=Padding.*

Parameter	Value	Details
Generator Layers	4 layers - BatchNorm - ReLU + Output layer - Tanh	LatentSize → 256 (K=4, S=1, P=0) 256 → 128 (K=4, S=2, P=1) 128 → 64 (K=4, S=2, P=1) 64 → 32 (K=4, S=2, P=1) 32 → 1
Discriminator Layers	4 Layers - BatchNorm - LeakyReLU + Output layer	3 → 32 (K=4, S=2, P=1) 32 → 64 (K=4, S=2, P=1) 64 → 128 (K=4, S=2, P=1) 128 → 256 (K=4, S=2, P=1) 256 → 1 (K=4, S=1, P=0)
Latent Space Size	[64, 256]	-
Epochs	25	-
Optimizer	RMSprop	-
Loss Function	Binary Cross Entropy	-
Learning Rate	0.0002	-
Batch Size	128	-
Weight Clipping	0.01	-

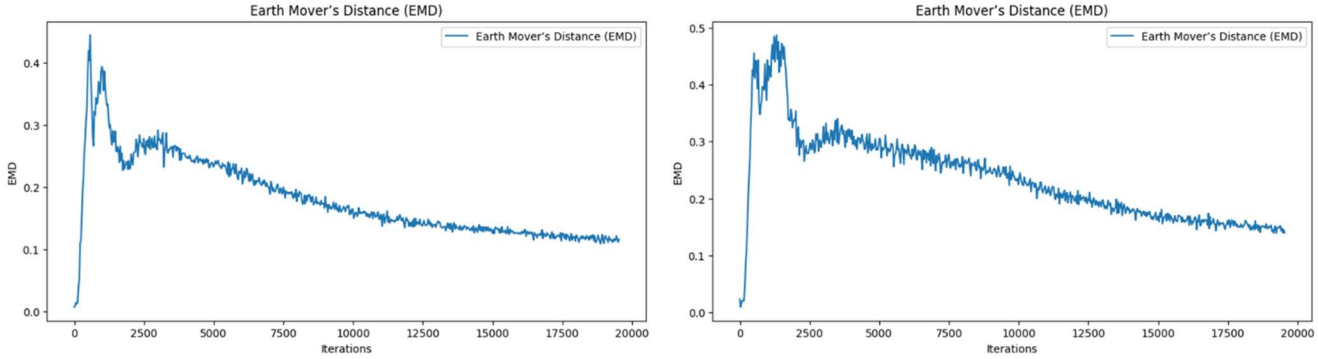


***Figure 15:** Earth Mover's Distance (EMD) of the WGAN model, during the training iterations of 25 epochs; a), b) show the results with a latent space dimension of 64 and 256, respectively.*

The Earth Mover's Distance (EMD) of the model over 25 epochs can be seen in Figure 15 above. In an ideal scenario, the EMD value would eventually converge closer to 0, showing that the delta between the discriminator and generator is almost none. In the two curves above, the EMD values converge to around 0.2. The EMD curve in Figure 15 a) for the latent space size of 64 almost converges to around 0.1, showing that the difference between the generator and discriminator was less than in Figure 15 b).

Based on the results, it seems that with the larger latent space size of 256 the model likely needed more iterations to train and become more accurate. So in a future iteration I would like to do 30-40 epochs to see if that curve would also converge closer to 0.

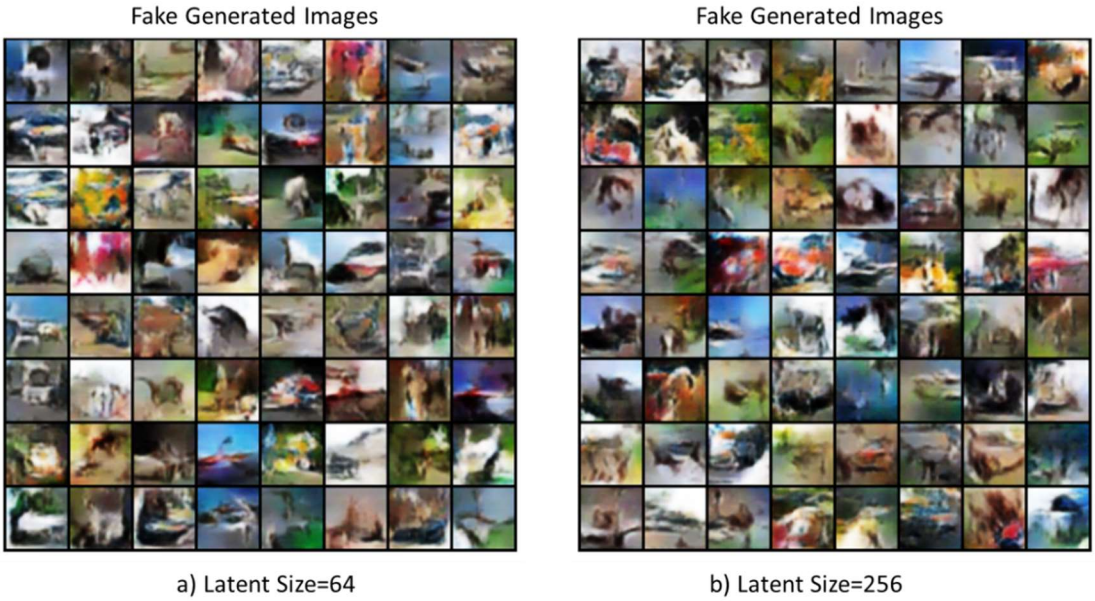


Figure 16: *Generated CIFAR images by the convolutional GAN model after 25 epochs; a), and b) show the results with a latent space dimension of 64 and 256, respectively.*

The results for the WGAN model with the two different latent space dimensions can be seen in Figure 16 above. Both sets of generated images look nice and colourful. I don't think either of them generated images that would be mistaken for the originals by the human eye, but the shapes and colours are quite similar. The images for the latent space size of 256 are slightly better to my eye, as they are less blurry and more detailed. But that is just my opinion, both look similar.

Overall Comparison

The VAE model did very well on regenerating images for the simple MNIST dataset but poorly on CIFAR, likely because of the complexity of the data. Overall, I think the GAN and WGAN generated images were the best, as expected. I believe the GAN-generated MNIST images were better than the WGAN images, but that may be because I did not have enough training iterations for the WGAN model. The CIFAR images from the WGAN model were the best, and it could have even used more training to bring down the EMD curve approach closer to 0. The most crucial performance gain I saw throughout GAN and WGAN was using convolutional layers over linear layers to process these images. The convolutional layers did much better and ended up with good results. This assignment was an excellent opportunity to test and create different models in different configurations.