CSI5340: Homework 4

Author: David Talson

Student Number: 8419286

1.0 Introduction

The goal of this assignment is to explore various image generation techniques. Neural networks in Artificial Intelligence can be used to train models to generate images which will be like original images it was trained on. Generative models namely, Variational Autoencoder (VAE) and Generative Adversarial Networks (GAN) will be explored. Also, Wasserstein Generative Networks (WGAN) which is a variant of the GAN will be implemented.

2.0 Methodology

In this section, the dataset used, and the model architecture will be discussed.

2.1 MNIST Dataset

The Modified National Institute of Standards and Technology (MNIST) database of handwritten digits contains 60,000 train and 10,000 test of handwritten images. The images are 28x28 in size and are commonly used for image processing tasks. This dataset is available in python and can be downloaded. The figure below shows an example of the dataset:

Figure 1: MNIST Dataset

2.2 Canadian Institute for Advanced Research, 10 Classes Dataset

The Canadian Institute for Advanced Research, 10 Classes dataset (CIFAR10) consists of 60,000 32x32 images separated into 10 classes. Each class consists of 5000 train and 1000 test images. This dataset is usually used for image processing tasks and can be downloaded from python. The figure below shows the CIFAR10 dataset:

Figure 2: CIFAR10 Dataset

2.3 Variational Autoencoder

A Variational Autoencoder (VAE) can be used to generate images due to the nature of its encoder and decoder modules. The encoder learns a representation of the input and creates a mapping into the specified latent space.

2.3.1 Encoder Network

The encoder network used to generate learn a representation of the input is a three-layer network shown below:

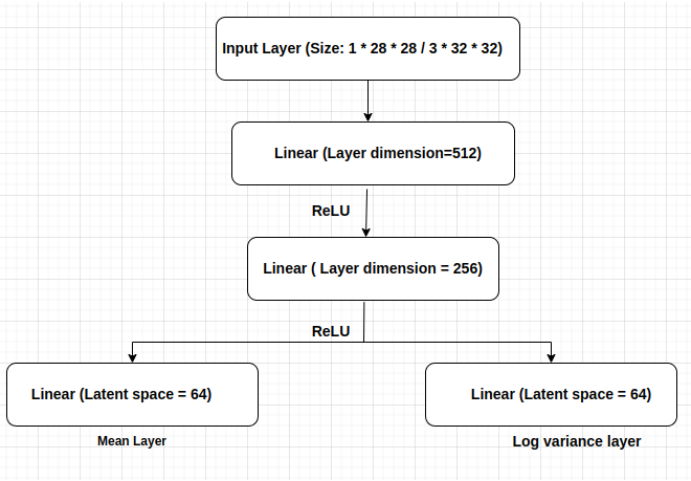

Figure 3: Encoder network layers

As can be seen from figure 3 above, the input images of size 1*28*28 for MNIST or 3*32*32 for CIFAR is taken at the input layer. Where it is passed through hidden layers which learn a representation of the data using Rectified Linear Units (ReLU) as activation functions. Finally, two outputs called the mean layer and log variance layer are produced.

2.3.2 Decoder Network

The decoder network is used to reconstruct the image from the sampled point in the latent space. This network is composed of fully connected layers. The figure below shows the architecture of the decoder network.

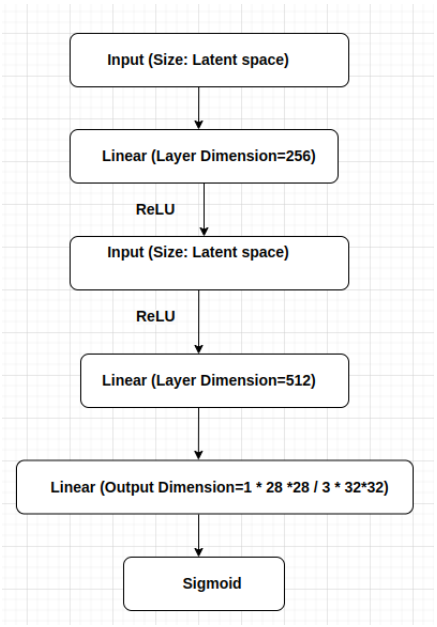

Figure 4: Decoder Architecture

As can be seen from figure 4 above, the decoder network takes an input of the specified latent space size and passes the data through linear layers of with ReLU activation functions and produces a reconstructed image. Finally, the output is passed through a sigmoid function which ensures that output values are between [0,1].

2.3.3 Reparametization

Reparametization is an important step in the VAEs as it is used to introduce randomness during training. A sampling from a normal distribution is done and it is scaled using the standard deviation of the log variance. The mean and sampled point are added and used for back propagation. The image below shows the code for reparametization:

```
def reparameterize(self, mean, log_variance):
    std = torch.exp(0.5*log_variance)
    eps = torch.randn_like(std)
    sample = mean + eps * std
    return sample
```

Figure 5: Reparametization

2.3.4 Loss Function

The loss function used in VAE is used during training to improve accuracy. The loss functions used in this process are the reconstruction loss, Kullback-Leiber (KL) divergence and total loss. The reconstruction loss was derived using the Mean Squared Error (MSE) between the reconstructed image and original image. The equation below shows the MSE equation:

[Equation]

Where X_i = Original image

X_i^r = Reconstructed image

The KL divergence loss is the implemented for the latent space. It is the difference between the learned latent distribution and the desired distribution. The equation below shows the KL divergence loss function.

[Equation]

Where μ_i = Original image

μ_i = Reconstructed image

The total loss is the sum of the reconstruction loss and KL divergence shown below:

Total Loss = MSE Loss + KL Divergence Loss

The figure below shows the code implementation for loss functions of the VAE:

```
reconstructed_input, mean, log_variance = model(images)
reconstructed_input = reconstructed_input.view(-1, self.num_channels * self.img_size * self.img_size)

mse_loss = F.mse_loss(reconstructed_input, images, reduction='sum')
kl_divergence = -0.5 * torch.sum(1+ log_variance - mean.pow(2) - log_variance.exp())
```

Figure 6: VAE Loss Calculation

As can be seen from figure 6 above, the loss for each iteration is calculated and backpropagated.

2.4 Generative Adversarial Network

The Generative Adversarial Network (GAN) can also be used for image generation as it employs the use of a Generator and Discriminator neural network modules. The Generator network produces fake images for the Discriminator which tries to identify the differences between the fake and

original images. The Generator continues to suffer a penalty until the Discriminator is unable to distinguish between the fake generated images and real images.

2.4.1 Generator Architecture

The Generator module is designed to created noise images initially but through learning will learn to create images that resemble the real images. The Generator was designed using convolution layers, batch normalization, and transposed convolution layers. The architecture of the Generator is shown below:

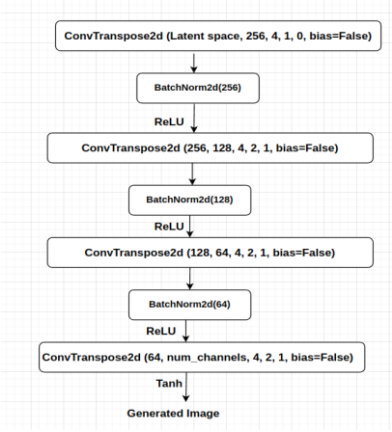

Figure 7: Generator Architecture

As can be seen from figure 7 above, a latent vector is passed through a transposed convolution layer for upsampling, then batch normalization is applied for stabilization. Then a ReLU activation function is applied to introduce non-linearity. Another transposed convolution layer is used to further upsample the feature map. Batch normalization and ReLU activation are applied again. Another layer of transposed convolution layer, batch normalization, and ReLU activation is applied. Finally, a transposed convolution layer to transform the feature map to image size then a tanh activation function is applied to normalize pixel values between [-1,1].

2.4.2 Discriminator Architecture

The discriminator module is used to classify between real and fake images. The goal of the discriminator module is to continually learn to predict between real and fake images. The architecture of the discriminator module is shown below:

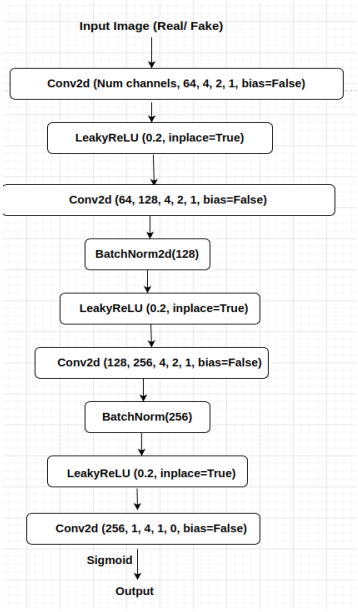

Figure 8: Discriminator Architecture

As can be seen from figure 8 above, the discriminator takes a real or fake image then passes it through a convolutional layer for downsampling. Leaky ReLU is then applied to prevent weights from going to zero. Another convolutional layer, batch normalization, and Leaky ReLU activation is applied. The final convolutional layer is used to reduce the dimensions to a single value. A sigmoid activation function is used to output a value of 0 or 1.

2.4.3 Loss Function

The loss function used to evaluate training is the Jensen Shannon Divergence (JSD) which calculates the difference between the probability distributions of the real and fake images. The equation below shows the JSD loss function:

[Equation]

Where D_{KL} = KL Divergence described above

M = Average Distribution

2.5 Wasserstein Generative Adversarial Network

The Wasserstein Generative Adversarial Network (WGAN) is the like the GAN but uses something called Earth Mover's Distance (EMD). The WGAN still uses the Generator and Critic/Discriminator components just like the GAN but is more stable.

2.5.1 Generator Architecture

The architecture of the WGAN is like the Generator architecture shown in figure 7.

2.5.2 Critic Architecture

The architecture of the critic module is like the architecture in figure 8 above but the sigmoid layer is not used which means the output values are not limited to 0 or 1.

2.5.3 Loss Function

The loss function used is the Earth Mover Distance which can be thought of to minimize the difference in scores of the critic module assigned to the real and generated samples. The loss function is shown below:

[Equation]

Where inf is the greatest lower bound

2.5.4 Gradient Penalty

The gradient penalty method is used to stabilize training in WGAN, the Lipschitz constant is used. Interpolated samples between the real and generated data are created which are fed tot the discriminator and gradients are computed. A gradient penalty is then calculated and returned.

3.0 Discussion

In this section, the performance and results of each image generation technique is discussed. After each model was developed, some hyperparameters were tuned to improve the performance of the model. The hyperparameters chosen for tuning in this task are the latent space and number of epochs. The latent space was chosen because the latent space is where the model encodes representations of the input data. The is important because a small latent space may lead to underfitting, and a large space may lead to overfitting. The number of epochs was chosen because an optimal number of epochs which will let the model converge is needed a small value will lead to underfitting and a large value will lead to overfitting.

3.1 Variational Autoencoder

The initial hyperparameters used to train the VAE is shown in the table below:

Table 1: Initial Hyperparameters

Hyperparameter	Value
Latent space	2 for MNIST 64 for CIFAR10
Epochs	20
Learning rate	0.0001
Batch size	128

3.1.1 MNIST Dataset

Using the values from table 1 the VAE was trained on the MNIST. After 20 epochs, the generated and reconstructed images are shown below:

Figure 9: Generated Image

Figure 10: Original Image

As can be seen from figures 9 and 10 above, the generated images look like the original image but appear blurry. To improve this model the latent space was increased to 64. The figures below show the generated and original images for a latent space of 64.

Figure 11: Generated image with latent space of 64

Figure 12: Original Image

As can be seen from figures 11 and 12 above, there is an improvement in the generated image. However, training for more epochs will improve the quality of the generated image. For 100 epochs, the generated and original images are shown below:

Figure 13: Generated Image for 200 epochs

Figure 14: Original Image

As can be seen from figure 13 above the quality of the generated image is improved. The figures below show the KL divergence loss MSE Loss and total loss for the train and test data in 100 epochs:

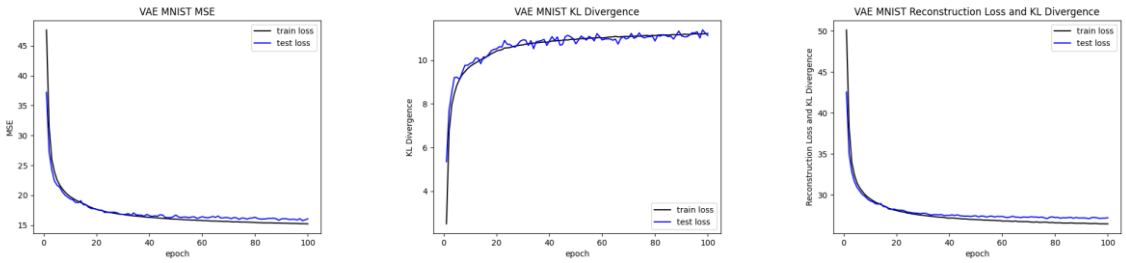

Figure 15: Losses

As can be seen from figure 15 above, the KL divergence and MSE stabilize at around 60 epochs.

3.1.2 CIFAR10 Dataset

The figures below show the generated and original images for a latent space of 64 and 20 epochs:

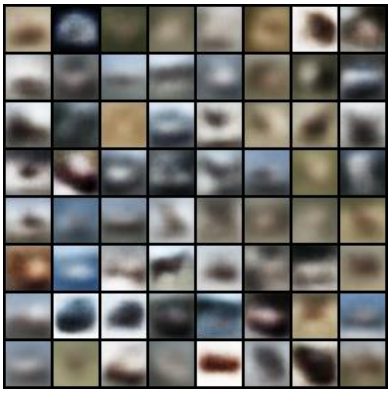

Figure 16: Generated Image

Figure 17: Original Image

As can be seen from figures 16 and 17 the generated image has a poor quality. To improve this, the latent space was increased to 512. This is because the CIFAR10 images are coloured meaning it has 3 channels so the models would need more dimensions to represent the image data. The images below show the generated and original images using a latent space of 512.

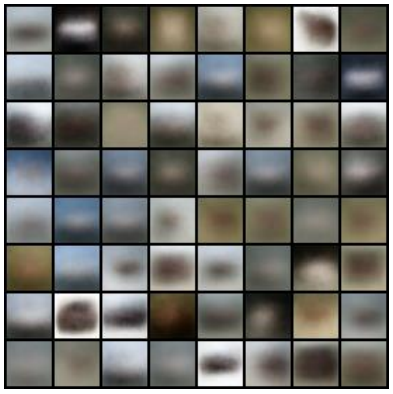

Figure 18: Generated image latent space 512, epochs 20

Figure 19: Original image

As can be seen from figure 18 and 19 above, the there is a slight improvement in the generated image, but the quality of the generated image is still poor. The next improvement is increasing the number of epochs. The figures below show the generated image and original image for 100 epochs:

Figure 20: Generated image latent space 512, epochs 100

Figure 21: Original image

As can be seen from figure 20 above, there is significant improvement in the generated image. The figures below show the losses:

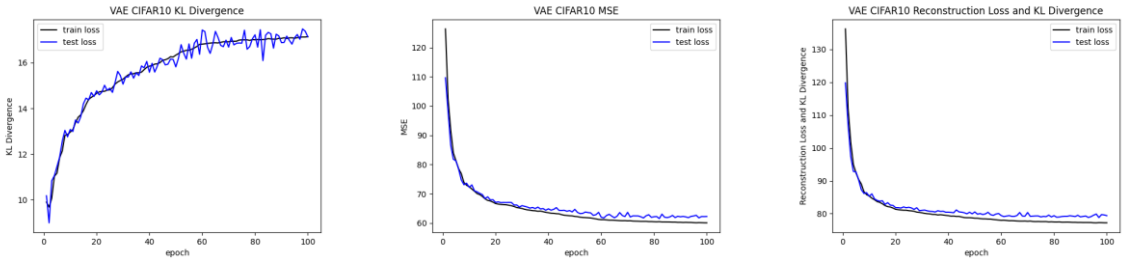

Figure 22: Losses

3.2 Generative Adversarial Network

The initial hyperparameters used to train the GAN are shown in the table below:

Table 2: Initial Hyperparameters for GAN

Hyperparameter	Value
Latent space	2 for MNIST 64 for CIFAR10
Epochs	20
Learning rate	0.0001
Batch size	64

3.2.1 MNIST Dataset

The figures below show the generated image and original image using the initial hyperparameters:

Figure 23: Generated image

Figure 24: Original image

As can be seen from figures 23 and 24 above, the generated image is not accurate, and this is because the latent space is too small to create representations for the input image. Using a latent space of 64, the generated image is shown below:

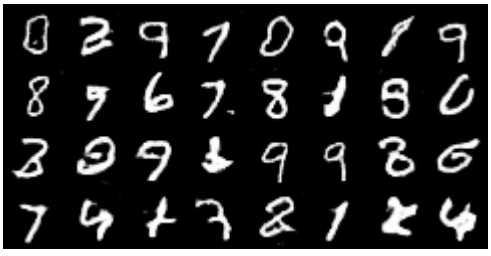

Figure 25: Generated image

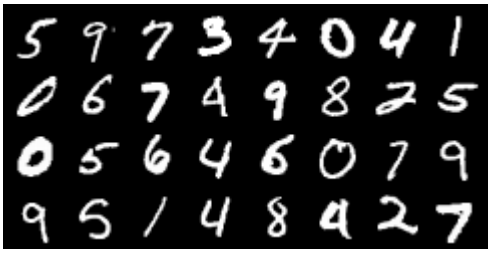

Figure 26: Original image

As can be seen from figure 25 above, the generated image has a better quality. To further improve the quality of the image, training was done for 100 epochs and the result is shown below:

Figure 27: Generated image

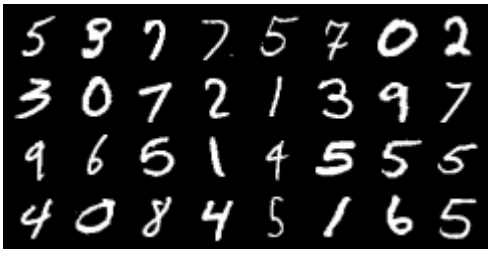

Figure 28: Original image

The JSD loss is shown below:

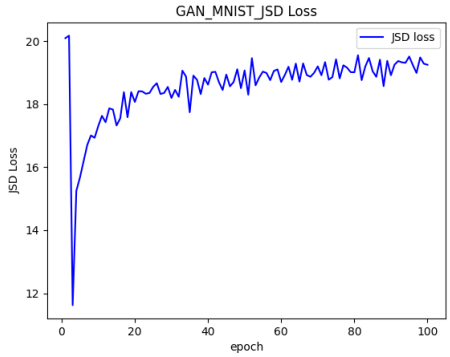

Figure 29: GAN MNIST JSD loss

3.2.2 CIFAR10 Dataset

The generated and original image for CIFAR10 using a latent space of 64 and 20 epochs are shown below:

Figure 30: Generated image

Figure 31: Original image

As can be seen from figure 30, the generated image has a good quality but can be improved. To further improve model, the latent space was increased to 512 and training was done for 20 epochs with the result shown below:

Figure 32: Generated image, latent space 512

Figure 33: Original image

As can be seen from figure 32, the quality of the image is improved but can be further improved. The number of epochs was increased to 100 and the images below show the generated and original images:

Figure 34: Generated image, latent space 512, epochs 100

Figure 35: Original image

The JSD loss plot is shown below:

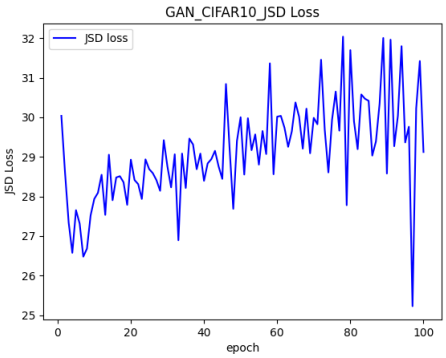

Figure 36: JSD loss plot

3.3 Wasserstein Generative Adversarial Network

The initial hyperparameters used for the WGAN are shown in the table below:

Hyperparameter	Value
Latent space	2 for MNIST 64 for CIFAR10
Epochs	20
Learning rate	0.00005
Batch size	64
Gradient penalty coefficient	0.

3.3.1 MNIST Dataset

The figures below show the generated and original image using the original hyperparameters:

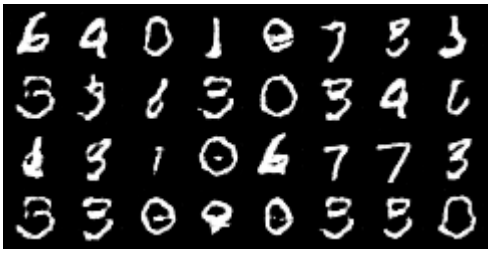

Figure 37: Generated image

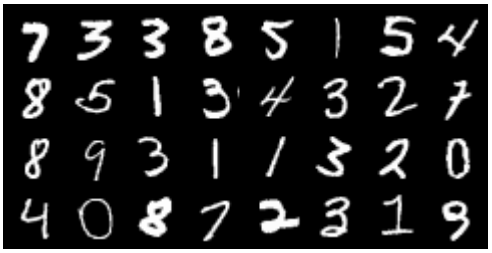

Figure 38: Original image

As can be seen from figure 37 above, the generated image has a good quality but can be improved. The latent space was increased to 64 and training was done. The generated and original images are shown below:

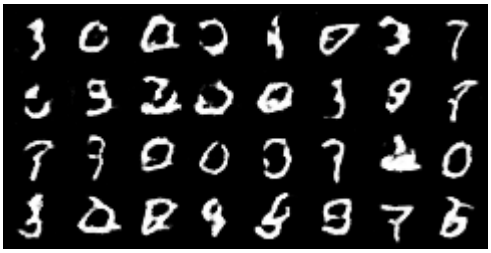

Figure 39: Generated image, latent space 64, epochs 20

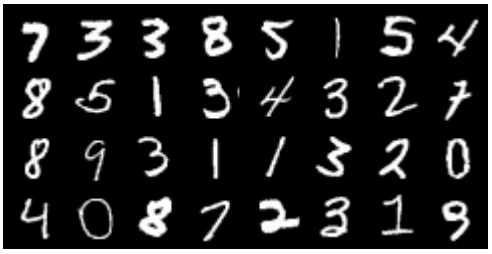

Figure 40: Original image

Finally, training was done for 100 epochs and the generated and original images are shown below:

Figure 41: Generated image, latent space 64, epochs 100

Figure 42: Original image

The EMD loss plot is shown below:

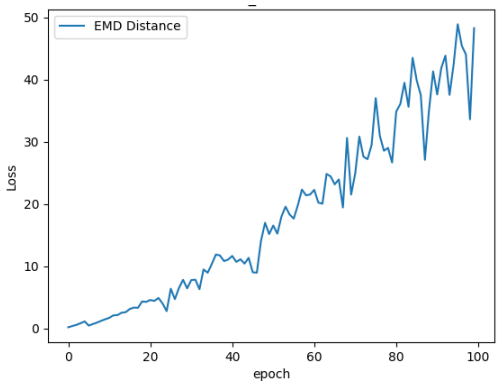

Figure 43: EMD distance

3.3.2 CIFAR10 Dataset

The generated and original images using a latent space of 64 and 20 epochs are shown below:

Figure 44: Generated image

Figure 45: Original image

As can be seen from figure 44, the generated image does not have a very good quality. This is understandable as the latent space is too small. The latent space was increased to 512 and generated and original images are shown below:

Figure 46: Generated image, latent space 512, epochs 20

Figure 47: Original image

As can be seen from figure 46 above, the image quality is improved. To further improve the image quality. Training was done for 100 epochs and the generated and original images are shown below:

Figure 49: Original image

Figure 48: Generated image, latent space 512, epochs 100

As can be seen from figure 48, the quality of the generated image has a very good quality. The figure below shows the EMD distance plot:

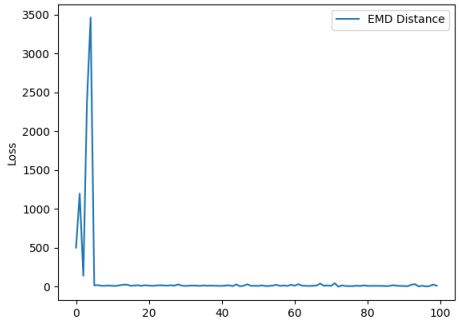

Figure 50: EMD distance

4.0 Conclusion

Through this exercise, more knowledge was gained on image generation techniques and hyperparameter tuning. Due to resource limitation, there was a limit to the model architecture complexity and number of epochs.

Through this exercise, the limitations of the VAE, GAN, and WGAN was understood. The VAE suffers from capturing complex patterns as linear layers was used for the architecture. The GAN suffers from instability which was solved by the WGAN using EMD and gradient penalty. Also, we saw the effect of the latent space and number of epochs on the performance of the model.