Homework 5

Spencer Au

Introduction

The dataset used for this homework is the MNIST dataset. The MNIST dataset is a dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images. The dataset is commonly used for training and testing in the field of machine learning. The dataset is divided into 60,000 training images and 10,000 testing images. The images are 28x28 pixels, and each pixel has a value between 0 and 255. The images are labeled with the corresponding digit they represent. The MNIST dataset is a subset of a larger dataset available from the National Institute of Standards and Technology (NIST). The digits have been size-normalized and centered in a fixed-size image.

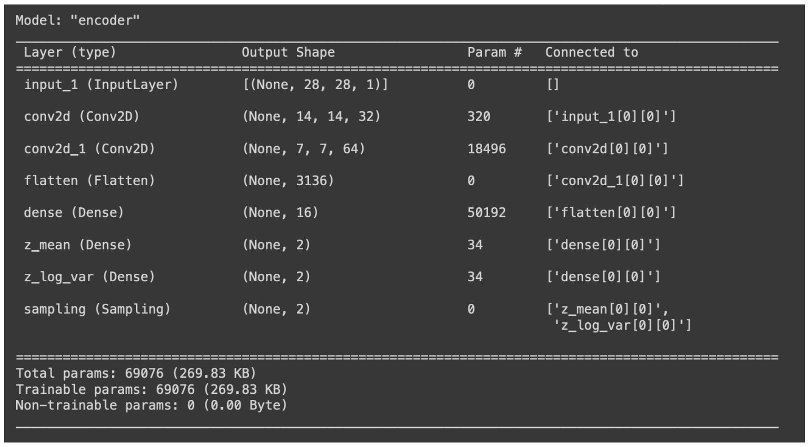
The models we are working with are a VAE, a GAN, and a CGAN. The VAE is a generative model that learns to encode and decode images. The GAN is a generative model that learns to generate images by training a generator and a discriminator. The CGAN is a generative model that learns to generate images by training a generator and a discriminator with additional class information. such as adding the condition to generate a specific digit.

Differences Between Model Architechtures, Inputs/Outputs, Loss(es) and Training

Architecture

VAE

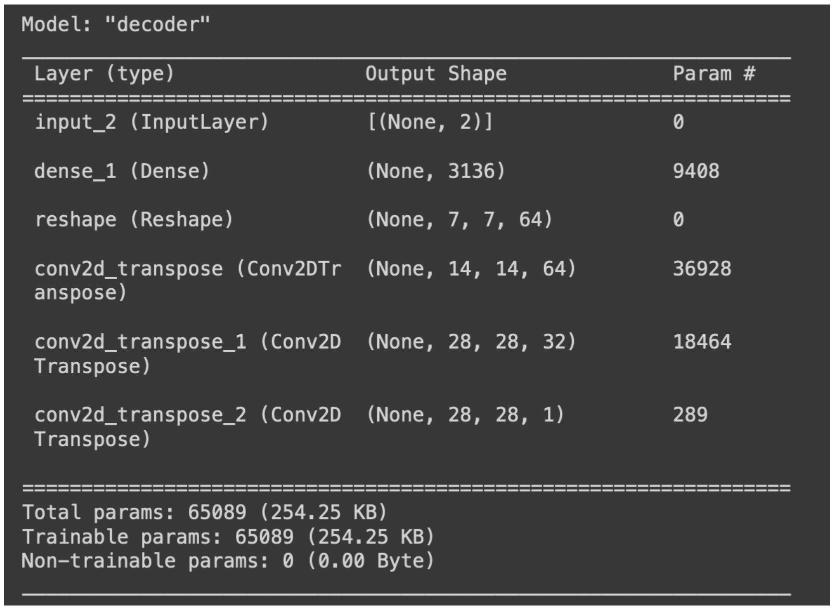
The VAE model consists of an encoder and a decoder. The encoder takes a 28x28 pixel single channel input image and encodes it into a latent space representation. The decoder takes the latent space representation and decodes it into an output image. The output is a 28x28 pixel single channel image that is similar to the input image.



```
Model: "encoder"
_________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]          0         []

 conv2d (Conv2D)             (None, 14, 14, 32)           320       ['input_1[0][0]']

 conv2d_1 (Conv2D)           (None, 7, 7, 64)             18496     ['conv2d[0][0]']

 flatten (Flatten)           (None, 3136)                 0         ['conv2d_1[0][0]']

 dense (Dense)               (None, 16)                   50192     ['flatten[0][0]']

 z_mean (Dense)              (None, 2)                    34        ['dense[0][0]']

 z_log_var (Dense)           (None, 2)                    34        ['dense[0][0]']

 sampling (Sampling)         (None, 2)                    0         ['z_mean[0][0]',
                                                                     'z_log_var[0][0]']

=================================================================================================
Total params: 69076 (269.83 KB)
Trainable params: 69076 (269.83 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________________________
```

Figure 1: VAE Encoder

The encoder consists of an input layer, followed by 2 convolutional layers using reLU activation with a stride of 2 and a kernel filter of 3x3, a flatten layer, and 2 dense layers in parallel using reLU activation. One of the dense layers is used to get the normalized mean, and the other is used to get the normalized standard deviation. This means that the encoder outputs the mean and standard deviation of the latent space representation.



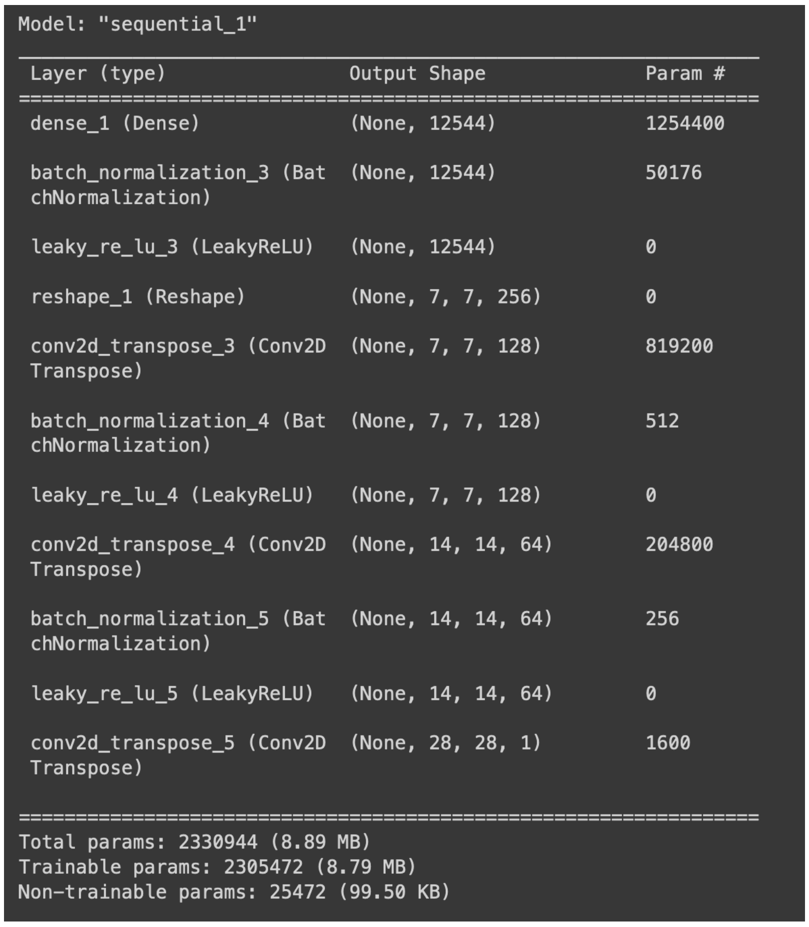
```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 2)]               0

 dense_1 (Dense)             (None, 3136)              9408

 reshape (Reshape)           (None, 7, 7, 64)          0

 conv2d_transpose (Conv2DTr  (None, 14, 14, 64)        36928
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 28, 28, 32)        18464
 Transpose)

 conv2d_transpose_2 (Conv2D  (None, 28, 28, 1)         289
 Transpose)

=================================================================
Total params: 65089 (254.25 KB)
Trainable params: 65089 (254.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2: VAE Decoder

The decoder consists of an input layer, a dense layer, a reshape layer, and 3 transposed convolutional layers with the first two each having a stride of 2, filter of 3x3, and reLU activation, and the last convolutional layer using a 3x3 filter and sigmoid activation. The dense layer is used to get the latent space representation, and the transposed convolutional layers are used to decode the latent space representation into an output image that is similar to the input image.

GAN

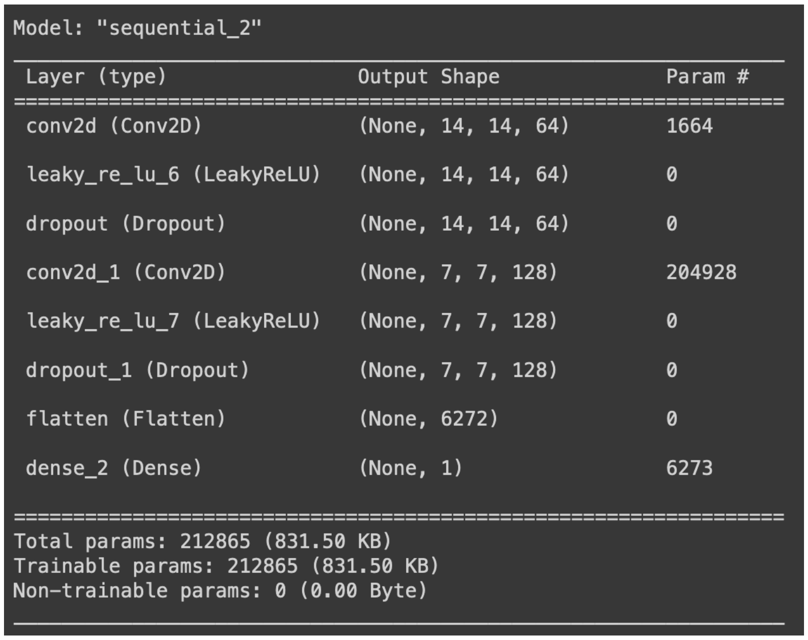
The GAN model consists of a generator and a discriminator. The generator takes a random noise vector as input and generates a 28x28 single channel output image. The discriminator takes an input image that is 28x28x1 and outputs a probability that the image is real.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 12544)             1254400

 batch_normalization_3 (Bat  (None, 12544)             50176
 chNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 12544)             0

 reshape_1 (Reshape)         (None, 7, 7, 256)         0

 conv2d_transpose_3 (Conv2D  (None, 7, 7, 128)         819200
 Transpose)

 batch_normalization_4 (Bat  (None, 7, 7, 128)         512
 chNormalization)

 leaky_re_lu_4 (LeakyReLU)   (None, 7, 7, 128)         0

 conv2d_transpose_4 (Conv2D  (None, 14, 14, 64)        204800
 Transpose)

 batch_normalization_5 (Bat  (None, 14, 14, 64)        256
 chNormalization)

 leaky_re_lu_5 (LeakyReLU)   (None, 14, 14, 64)        0

 conv2d_transpose_5 (Conv2D  (None, 28, 28, 1)         1600
 Transpose)

=================================================================
Total params: 2330944 (8.89 MB)
Trainable params: 2305472 (8.79 MB)
Non-trainable params: 25472 (99.50 KB)
```

Figure 3: GAN Generator

The generator consists of a dense layer that uses batch normalization and leaky reLU activation, followed by 2 transposed convolutional layers, each with batch normalization, a 5x5 filter, and leaky reLU activation. The first transposed convolution uses a stride of 1, while the second uses a stride of 2. The final layer is another transposed convolutional layer using a stride of 2 and a 5x5 filter. The output of the generator is a 28x28 single channel image that is similar to the input image.



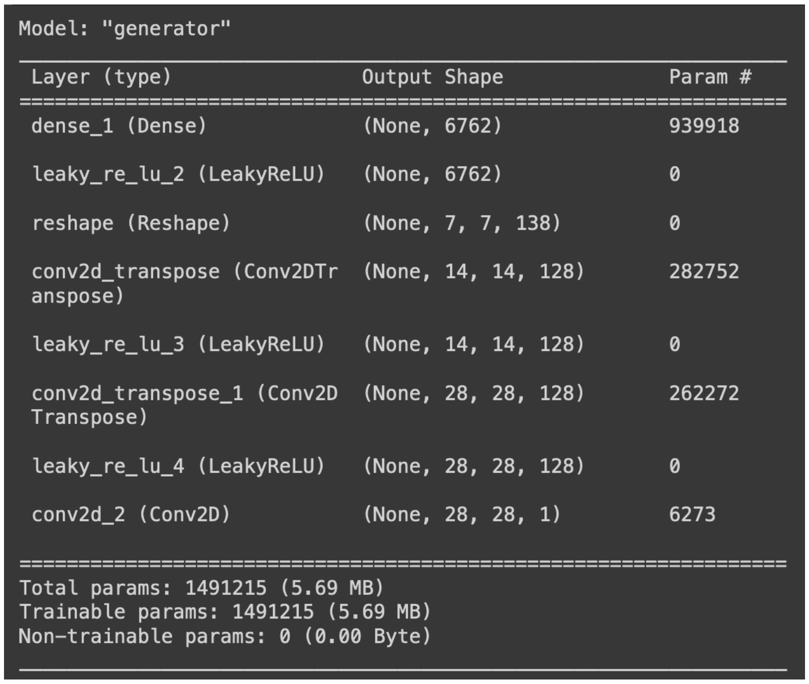
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        1664

 leaky_re_lu_6 (LeakyReLU)   (None, 14, 14, 64)        0

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         204928

 leaky_re_lu_7 (LeakyReLU)   (None, 7, 7, 128)         0

 dropout_1 (Dropout)         (None, 7, 7, 128)         0

 flatten (Flatten)           (None, 6272)              0

 dense_2 (Dense)             (None, 1)                 6273

=================================================================
Total params: 212865 (831.50 KB)
Trainable params: 212865 (831.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 4: GAN Discriminator

The discriminator consists of 2 convolutional layers, each with a stride of 2, kernel filter of 5x5, using leaky reLU activation, and a dropout of 0.3. This is followed by a flatten layer and a dense layer. The output of the discriminator is a probability that the input image is real.

CGAN

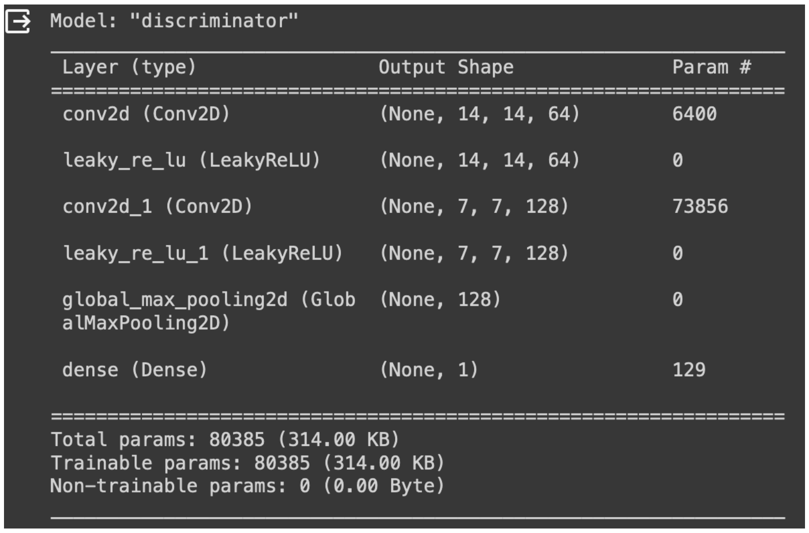
The CGAN model is similar to the GAN model, but it also takes a class label as input to the generator and discriminator. The class label is a one-hot encoded vector that represents the digit that the generator should generate.



```
Model: "generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 6762)              939918

 leaky_re_lu_2 (LeakyReLU)   (None, 6762)              0

 reshape (Reshape)           (None, 7, 7, 138)         0

 conv2d_transpose (Conv2DTr  (None, 14, 14, 128)       282752
 anspose)

 leaky_re_lu_3 (LeakyReLU)   (None, 14, 14, 128)       0

 conv2d_transpose_1 (Conv2D  (None, 28, 28, 128)       262272
 Transpose)

 leaky_re_lu_4 (LeakyReLU)   (None, 28, 28, 128)       0

 conv2d_2 (Conv2D)           (None, 28, 28, 1)         6273

=================================================================
Total params: 1491215 (5.69 MB)
Trainable params: 1491215 (5.69 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 5: CGAN Generator

The generator takes a random noise vector and a class label as input and generates a 28x28 single channel output image. The generator consists of a dense layer with leaky reLU activation, followed by a reshape layer. This is followed by 2 transposed convolutional layers each with leaky reLU activation, a stride of 2, and a 4x4 kernel filter. The final layer is a convolutional layer with a 7x7 kernel filter and sigmoid activation. The output of the generator is a 28x28 single channel image that is similar to the input image.



```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        6400

 leaky_re_lu (LeakyReLU)     (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         73856

 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0

 global_max_pooling2d (Glob  (None, 128)               0
 alMaxPooling2D)

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 80385 (314.00 KB)
Trainable params: 80385 (314.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

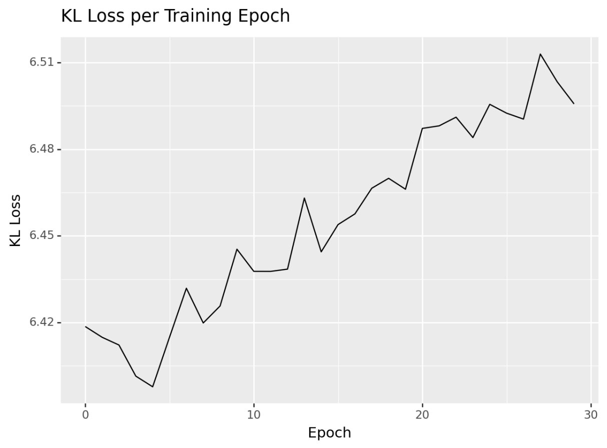
Figure 6: CGAN Discriminator

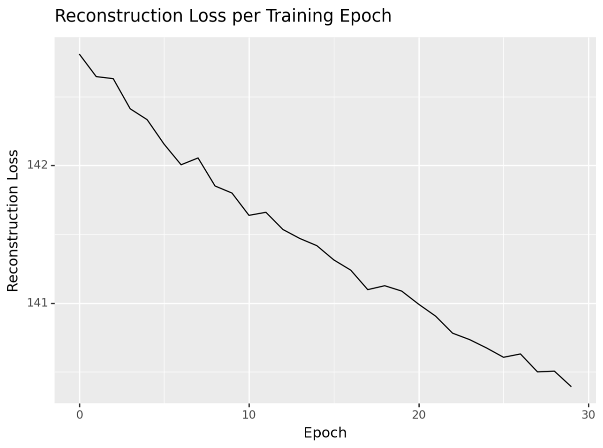
The discriminator takes an input image that is 28x28x1 and a class label and outputs a probability that the image is real. The discriminator consists of a dense layer, followed by 2 convolutional layers with a 3x3 kernel filter, a stride of 2, and leaky reLU activation. This is followed by a global max pooling layer and a dense layer. The output of the discriminator is a probability that the input image is real.

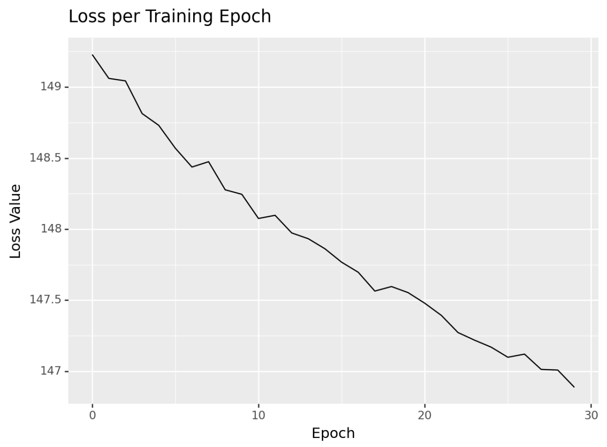
Training and Generation

VAE

The VAE is trained using KL divergence loss which ensures that the latent space representation is close to a normal distribution, reconstruction loss which measures the difference between the original input data and its reconstruction produced by the decoder, and total loss, which is the sum of the reconstruction loss and the KL divergence loss. It represents the overall objective that the VAE tries to minimize during training. The total loss is used to compute the gradients and update the model parameters via backpropagation. The VAE is trained using the Adam optimizer with a default initial learning rate, and the model is trained for 30 epochs with a batch size of 128. The VAE generates samples by sampling from the latent space representation and decoding the samples using the decoder.





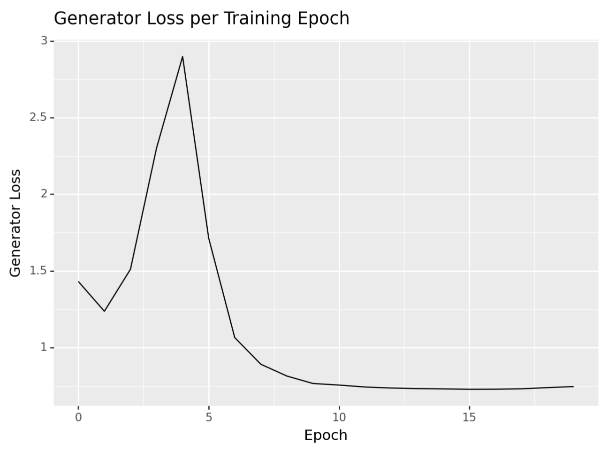


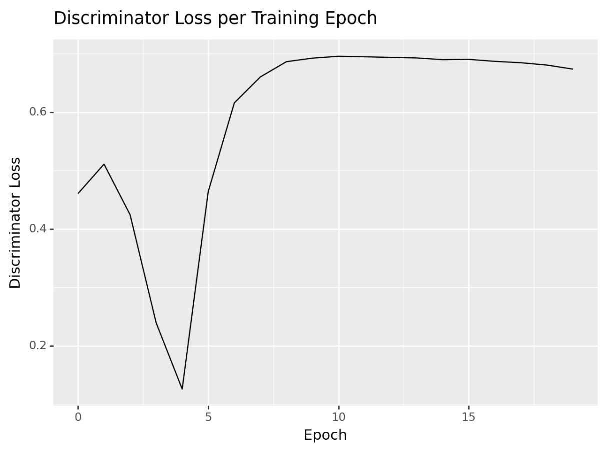
GAN

The GAN is trained using the binary cross-entropy loss, which measures the difference between the predicted probability that an image is real and the actual label. The generator is trained to generate images that the discriminator cannot distinguish from real images, while the discriminator is trained to distinguish between real and fake images. The GAN is trained using the Adam optimizer with an intial learning rate of 1e-4 for both the generator and discriminator, and the model is trained for 50 epochs. The GAN generates samples by sampling from a random noise vector and decoding the samples using the generator.

CGAN

The CGAN is also trained using the binary cross-entropy loss, which measures the difference between the predicted probability that an image is real and the actual label. The generator is trained to generate images that the discriminator cannot distinguish from real images, while the discriminator is trained to distinguish between real and fake images. The CGAN is trained using the Adam optimizer with an intial learning rate of 0.0003 for both the generator and discriminator, and the model is trained for 20 epochs. The CGAN generates samples by sampling from a random noise vector and class label and decoding the samples using the generator.





Comparison of Outputs (e.g. quality, ease of getting new samples, interpolation/smoothness)

Overall it seems like the VAE struggles to generate images that are more "complicated" with more curves, etc unlike zero. The images generated by the VAE are pretty blurry and its even hard to make out certain digits. That being said, barring the "blurry" part, its pretty simple to make out which digit the VAE was attempting to recreate. On the other hand, both the GAN and CGAN seem to be able to generate various digits in a pretty clear manner, without any significant blurriness. Howver, it is often difficult to make out which digit they were trying to recreate, and the "width" of the digitsdoesnt sem to be uniform. Many of the generated samples don't seem to be very realistic either. Overall the CGAN seemed to outperform both the GAN and VAE in terms of quality of the generated images.

Generated images

VAE

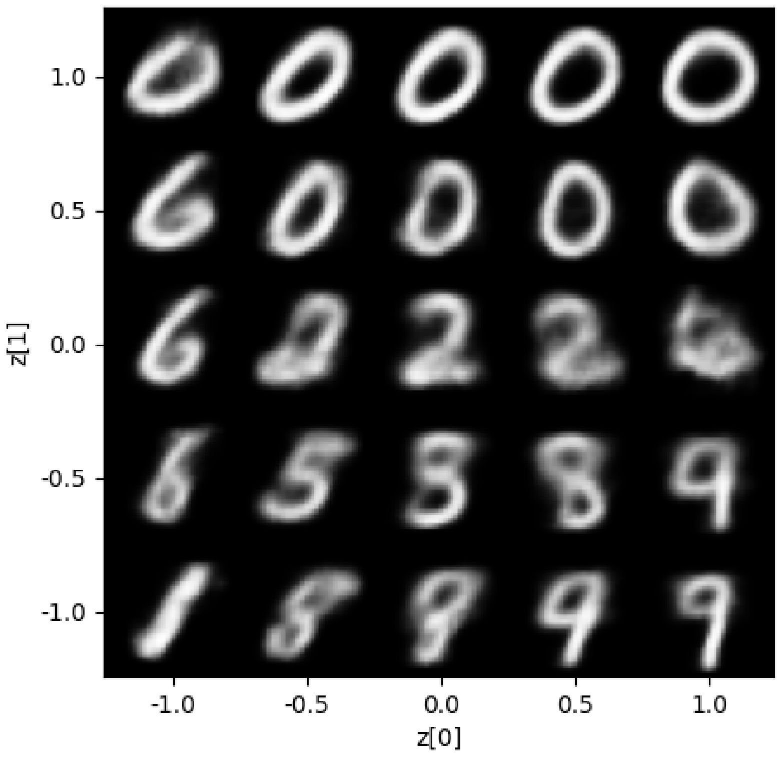


Figure 7: VAE Imagess

GAN

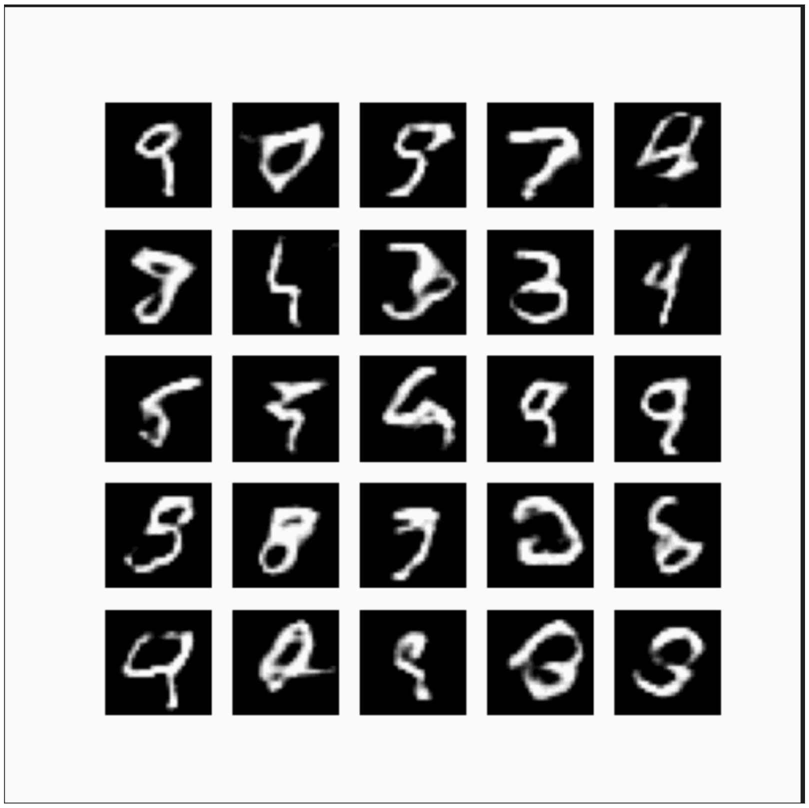


Figure 8: GAN Images

CGAN

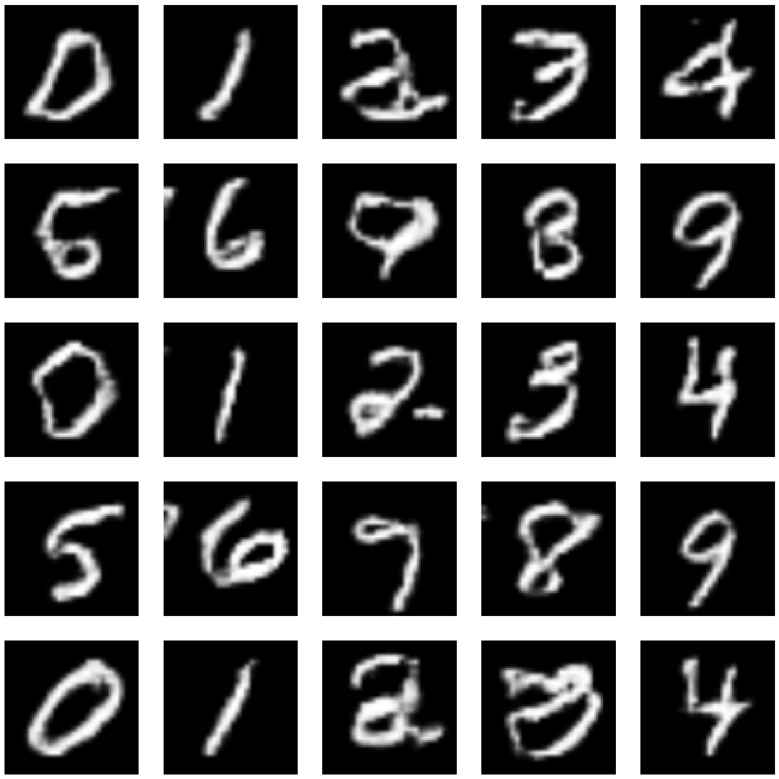


Figure 9: CGAN Images