HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION RESEARCH 1

GENERATIVE ADVERSARIAL NETWORKS

NGÔ QUANG MINH
minh.nq205163@sis.hust.edu.vn

Major: Information Technology
Specialization: Global ICT

Supervisor:	Associate Professor Thân Quang Khoát	______________
		Signature
Department:	Data Science Laboratory	
School:	School of Information and Communications Technology	

HANOI, 03/2023

ABSTRACT

This research explores the fundamental concept of Generative Adversarial Networks (GANs) in machine learning. GANs are a class of generative models that are capable of producing new data that is similar to the training data. The research covers the basic architecture of GANs, including the generator which takes as input a random noise vector and outputs data sample, discriminator networks that takes as input a data sample and outputs a scalar value that represents the likelihood that the data sample is real, and the training process involving a min-max game between the two networks. This research aims to provide a foundation for future research on GANs and their applications in various fields, such as computer vision or natural language processing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

Generative Adversarial Networks (GANs) are a class of deep learning models that have gained popularity in recent years due to their ability to generate highquality synthetic data that is similar to real-world data. GANs consist of two neural networks: a generator network and a discriminator network. The generator network generates synthetic data, while the discriminator network evaluates the quality of the synthetic data. The two networks are trained together in an adversarial manner, where the generator tries to mislead the discriminator into accepting its synthetic data as real, while the discriminator tries to correctly distinguish between actual and synthetic data

1.1 Background and Problems of Research

Generative Adversarial Networks (GANs) have become increasingly popular in recent years due to their ability to generate high-quality synthetic data. GANs have been used in various applications such as image synthesis, data augmentation, image-to-image translation, video synthesis, and more.

1.2 Research Objectives

This research summarized the basic concept of Generative Adversarial Networks (GANs). Using the result of GANs applying on MNIST dataset, the research aims to provide clear view on the architecture of GANs and how to implement the GANs on MNIST.

CHAPTER 2. ARCHITECTURE

The architecture of a GAN typically consists of two main components: a generator and a discriminator. Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

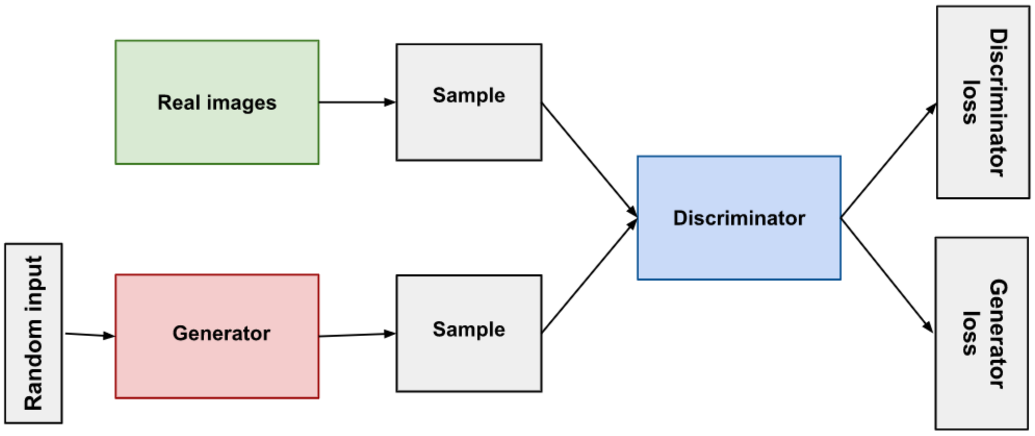


Figure 2.1: Diagram of GANs architecture

There are some techniques in training GANs include:

1. The all convolutional net
2. Batch Normalization

The first emphasizes strided convolutions (instead of pooling layers) for both: increasing and decreasing feature's spatial dimensions. And the second normalizes the feature vectors to have zero mean and unit variance in all layers. This helps to stabilize learning and to deal with poor weight initialization problems.

2.1 Discriminator Network

The discriminator network takes as input a data sample and outputs a scalar value that represents the likelihood that the data sample is real. The discriminator in a GAN is simply a classifier. It could use any network architecture appropriate to the type of data it's classifying. The goal of the discriminator network is to correctly distinguish between real and generated data.
The discriminator's training data comes from two sources:

- Real data samples from the training dataset.
- Fake data samples created by the generator.

In in this research, the discriminator used is a CNN-based image classifier with 212,865 trainable parameters. The input to the discriminator is a grayscale image of size 28x28 pixels, and the output is a value indicating the probability that the input image is real (1 for real prediction and 0 for fake prediction).

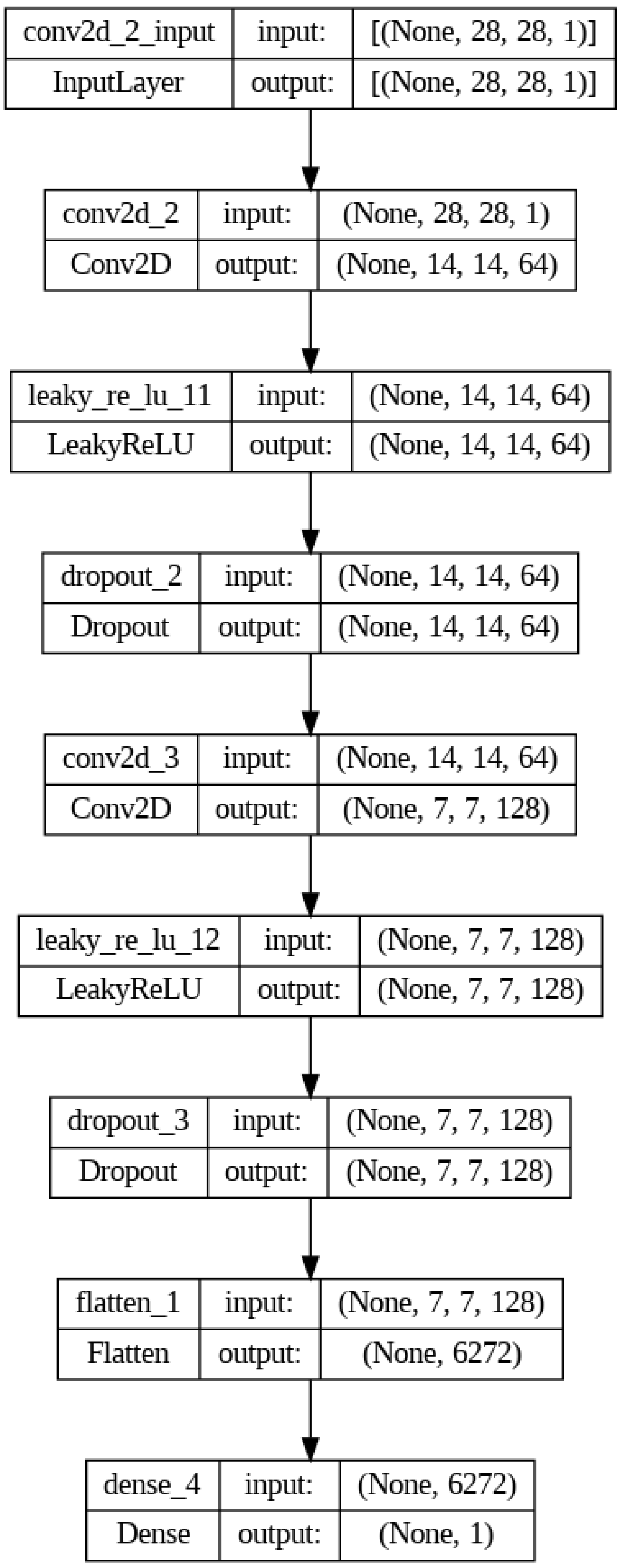

conv2d_2_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d_2 | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 14, 14, 64)
leaky_re_lu_11 | input: | (None, 14, 14, 64)
LeakyReLU | output: | (None, 14, 14, 64)
dropout_2 | input: | (None, 14, 14, 64)
Dropout | output: | (None, 14, 14, 64)
conv2d_3 | input: | (None, 14, 14, 64)
Conv2D | output: | (None, 7, 7, 128)
leaky_re_lu_12 | input: | (None, 7, 7, 128)
LeakyReLU | output: | (None, 7, 7, 128)
dropout_3 | input: | (None, 7, 7, 128)
Dropout | output: | (None, 7, 7, 128)
flatten_1 | input: | (None, 7, 7, 128)
Flatten | output: | (None, 6272)
dense_4 | input: | (None, 6272)
Dense | output: | (None, 1)

Figure 2.2: Discriminator layers

The architecture of the discriminator network in GANs can vary, but typically it consists of several convolutional layers followed by one or more fully connected layers.

- **Convolution layer(Conv2D):** 64 convolutions filters for the convolution first layer and 128 convolutions filters for the second convolution layer. The kernel size is 5 x 5 for both and the layers perform a series of strided 2 convolutions.
- **Activation fucntion:** Many activation functions will work fine with this basic GAN architecture. However, leaky ReLUs are very popular because they help the gradients flow easier through the architecture. On the other hand, a regular ReLU function works by truncating negative values to 0, blocking the gradients to flow through the network.
- **Dropout layer:** The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. Dropout layers are important in training CNNs because they prevent overfitting on the training data.
- **Flattening layer:** Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer. In other words, we put all the pixel data in one line and make connections with the final layer.
- **Dense layer:** In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

2.1.1 Generator Network

The generator network takes as input a random noise vector and outputs a synthetic data sample. It is usually a deconvolutional neural network that up samples the noise vector to the desired output size. The goal of the generator network is to generate synthetic data that is similar to the real data.

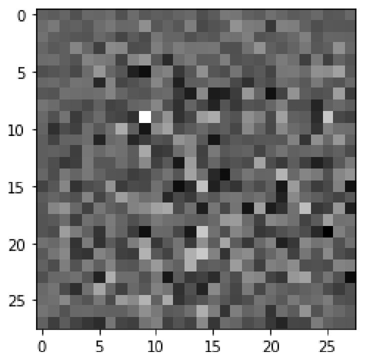
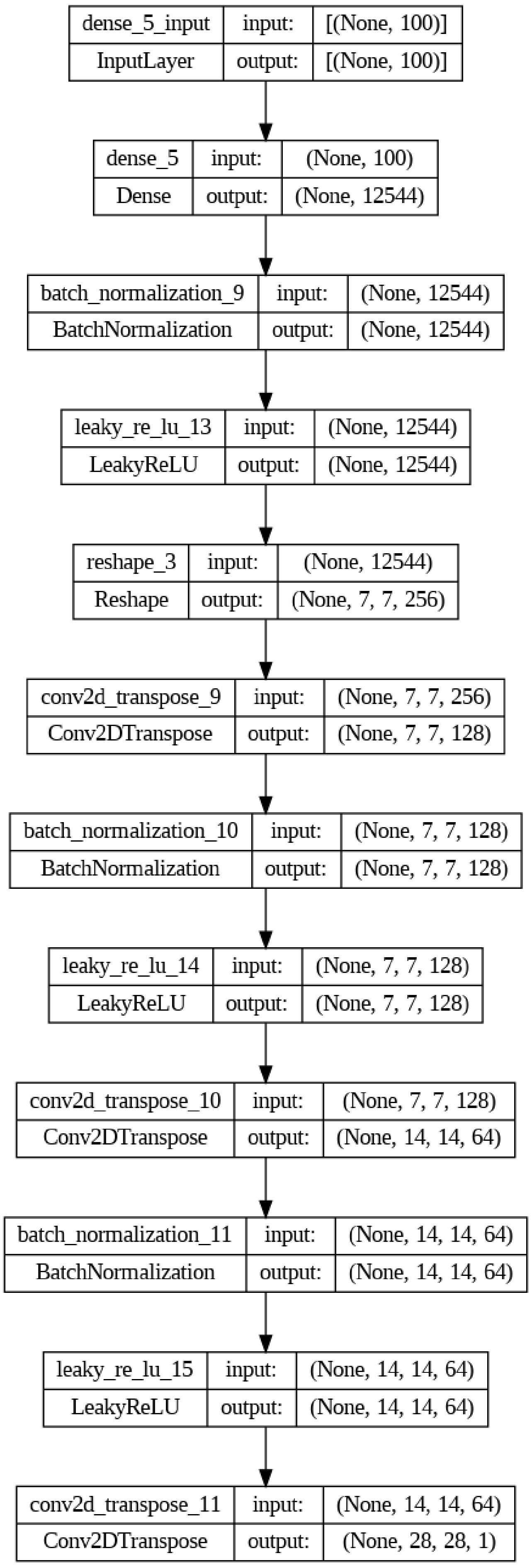


Figure 2.3: Generator output (no training)

In this research, the generator model will assign meaning to the latent points and, in turn, the latent space, until, at the end of training, the latent vector space represents a compressed representation of the output space, MNIST images, that only the generator knows how to turn into plausible MNIST images.

- **Inputs:** Point in latent space, e.g. a 100 element vector of Gaussian random numbers.
- **Outputs:** Two-dimensional square grayscale image of 28×28 pixels with pixel values in [0,1].



dense_5_input | InputLayer | input: [(None, 100)] | output: [(None, 100)]
dense_5 | Dense | input: (None, 100) | output: (None, 12544)
batch_normalization_9 | BatchNormalization | input: (None, 12544) | output: (None, 12544)
leaky_re_lu_13 | LeakyReLU | input: (None, 12544) | output: (None, 12544)
reshape_3 | Reshape | input: (None, 12544) | output: (None, 7, 7, 256)
conv2d_transpose_9 | Conv2DTranspose | input: (None, 7, 7, 256) | output: (None, 7, 7, 128)
batch_normalization_10 | BatchNormalization | input: (None, 7, 7, 128) | output: (None, 7, 7, 128)
leaky_re_lu_14 | LeakyReLU | input: (None, 7, 7, 128) | output: (None, 7, 7, 128)
conv2d_transpose_10 | Conv2DTranspose | input: (None, 7, 7, 128) | output: (None, 14, 14, 64)
batch_normalization_11 | BatchNormalization | input: (None, 14, 14, 64) | output: (None, 14, 14, 64)
leaky_re_lu_15 | LeakyReLU | input: (None, 14, 14, 64) | output: (None, 14, 14, 64)
conv2d_transpose_11 | Conv2DTranspose | input: (None, 14, 14, 64) | output: (None, 28, 28, 1)

Figure 2.4: Generator layers

When training a GAN to generate handwritten digits from the MNIST dataset, the generator typically consists of several layers of transposed convolutional layers (also known as deconvolutional layers), followed by a final output layer with a tanh activation function.

- **Transposed convolutional layer (Conv2DTranspose):** A transposed convolutional layer takes an input tensor and transforms it into a larger output tensor by performing a set of learnable filters (also called kernels or weights) on the input tensor. This will allow the generator to gradually increase the spatial dimensions of the input tensor until it matches the desired output shape.
- **Activation function:** Tanh for the final layer and LeakyRelu for others. The tanh function is symmetric around zero and maps input values to a range between -1 and 1, which matches the desired output range.
- **Batch Normalization:** Batch normalization is a layer that allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers. The activations scale the input layer in normalization. Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model.

Note: There are two common ways to do this upsampling process, sometimes called deconvolution. One way is to use an UpSampling2D layer (like a reverse pooling layer) followed by a normal Conv2D layer. The other and perhaps more modern way is to combine these two operations into a single layer, called a Conv2D Transpose.

CHAPTER 3. LOSS FUNCTION

The loss function in Generative Adversarial Networks (GANs) is a key component of the training process, as it determines how well the generator and discriminator are performing their respective tasks.

3.1 Binary Cross Entropy

Binary Classification is a problem where we must separate our observations in any of the two labels based on the features. Binary Cross-Entropy (BCE) is a loss function that is commonly used in binary classification problems. It is used to measure the difference between the predicted probabilities and the actual target values, which are binary in nature (0 or 1). The BCE loss function is defined as:

$$BCE(y, \hat{y}) = -y \log(\hat{y}) - (1-y) \log(1-\hat{y}) \tag{3.1}$$

where y is the ground truth label (either 0 or 1) and $\hat{y}$ is the predicted probability of the positive class. The logarithm is taken with base 10, but it could also be taken with base e (natural logarithm).
The BCE loss penalizes the model more when it makes a large mistake, such as predicting a high probability for the positive class when the target value is 0. On the other hand, it only penalizes the model slightly when it makes a small mistake, such as predicting a low probability for the positive class when the target value is 1.

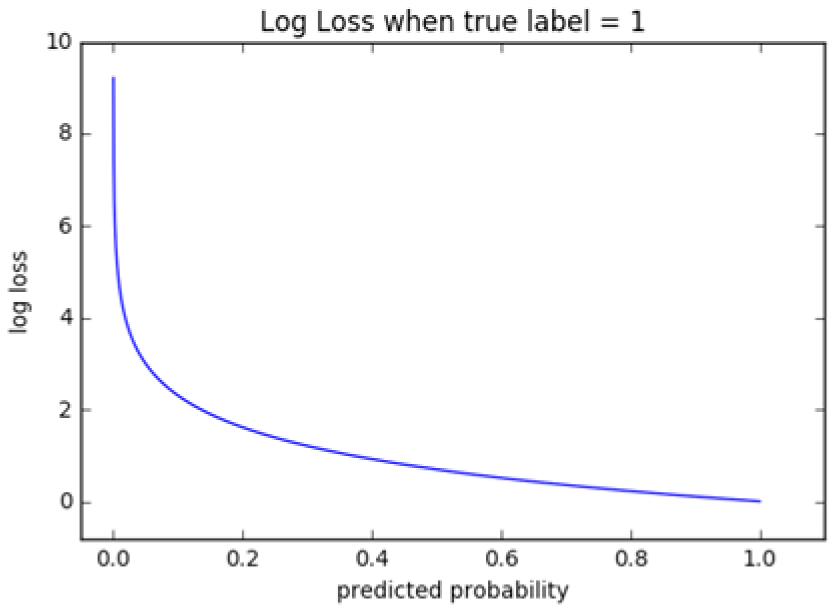


Figure 3.1: Binary Cross Entropy

The graph above shows the range of possible loss values given a true observation. As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly.

3.2 Discriminator Loss

Binary Cross-Entropy (BCE) is commonly used as a loss function in the discriminator of a Generative Adversarial Network because it is well-suited for binary classification problems, which is the task of the discriminator in a GAN.
For a real input, y = 1. Substituting this in the binary cross entropy loss function gives the below loss value:

$$L(1, \widehat{y}) = -(1 \cdot \log(D(x)) + (1-1) \cdot \log(1 - D(x)) = -\log(D(x)) \tag{3.2}$$

Similarly for a fake input, y = 0, which gives the below loss value:

$$L(0, \widehat{y}) = -(0 \cdot \log(D(G(z))) + (1-0) \cdot \log(1 - D(G(z))) = -\log(1 - D(G(z))) \tag{3.3}$$

Combining both the above losses for the discriminator, one for a real input and the other for a fake input gives the below discriminator loss.

$$L(Discriminator) = -[\log(D(x)) + \log(1 - D(G(z)))] \tag{3.4}$$

The goal of the discriminator is to minimise this loss,

$$L(Discriminator) = \min[-[\log(D(x)) + \log(1 - D(G(z)))]] \tag{3.5}$$

Removing the negative sign will change min to max, hence the final discriminator loss for a single datapoint can be written as:

$$L(Discriminator) = \max[\log(D(x)) + \log(1 - D(G(z)))] \tag{3.6}$$

3.3 Generator Loss

For the generator, loss is calculated from the discriminator loss. During the training of the generator, the discriminator is frozen. Hence only one input is possible to the discriminator, which is the fake input.
The generator is trying to fool the discriminator into classifying the fake data as real data. This implies that the generator tries to minimise the second term in the discriminator loss equation. The generator loss function for single generated

datapoint can be written as:

$$L(Generator) = \min[\log(1 - D(G(z)))] \tag{3.7}$$

Note: to avoid bad early stages of GAN training when the discriminator's job is very easy, it suggested modifying the generator loss so that the generator tries to maximize $\log D(G(z))$.

3.4 Minmax Loss

A GAN can be conceptually thought of as a minimax game played between the generator model and the discriminator model. Both models are trained simultaneously where one model tries to minimise the loss while the other tries to maximise the loss.

$$\min_{G} \max_{D} \mathrm{V}(G, D) = \mathbb{E}\mathbf{x} \sim p_{\text{data}}(\mathbf{x})[\log D(\mathbf{x})] + \mathbb{E}\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})[\log(1 - D(G(\mathbf{z})))] \tag{3.8}$$

where G is the generator, D is the discriminator, $\mathbf{x}$ is a real data point, $\mathbf{z}$ is a random noise vector sampled from the prior distribution $p_{\mathbf{z}}$, and $\log$ is the natural logarithm. The first term in the loss encourages the discriminator to correctly classify real data as real, while the second term encourages it to correctly classify fake data generated by the generator as fake. The generator tries to minimize this loss by generating fake data that fools the discriminator.

In the perfect equilibrium, the generator would capture the general training data distribution. As a result, the discriminator is always unsure of whether its inputs are real or not. In practice, achieving convergence can be challenging, as the generator and discriminator are trained in opposition to each other through adversarial training, which can lead to instability in the training process.

CHAPTER 4. TRAINING PRCOESS

Training a GAN involves updating both the generator and the discriminator networks alternately. The training process can be broken down into the following steps:

4.1 Sample real data

The training dataset is MNIST dataset include 28 x 28 pixel handwritten iamges which have been normalized into -1 to 1 sacle.

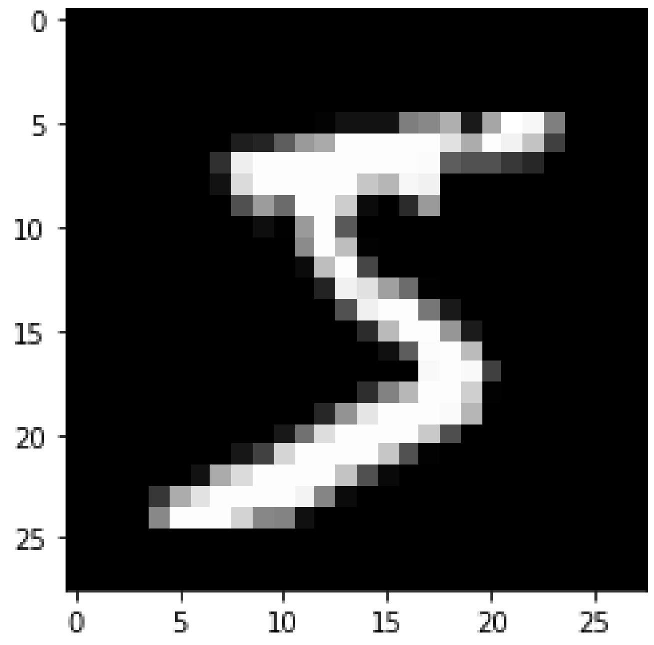


Figure 4.1: Training Data

A batch of real data is sampled from the data distribution that you want to model. The purpose of this is to provide the discriminator with real data to learn from and improve its ability to differentiate between real and generated data.
The size of the batch can vary depending on the specific problem and architecture of the GAN. It is important to select a batch size that is large enough to provide sufficient data for the discriminator to learn from, but not too large as to cause memory issues during training. In this research, the train dataset size is 60,000, batch size equals to 256 so there are 234 batches per epoch.

4.2 Generate new data

The generator network takes as input a random noise vector and generates a batch of synthetic data. Experiments suggest that the distribution of the noise not having significant effect on the training process. The noise use in this research is

100-dimensions noise with uniform distribution.

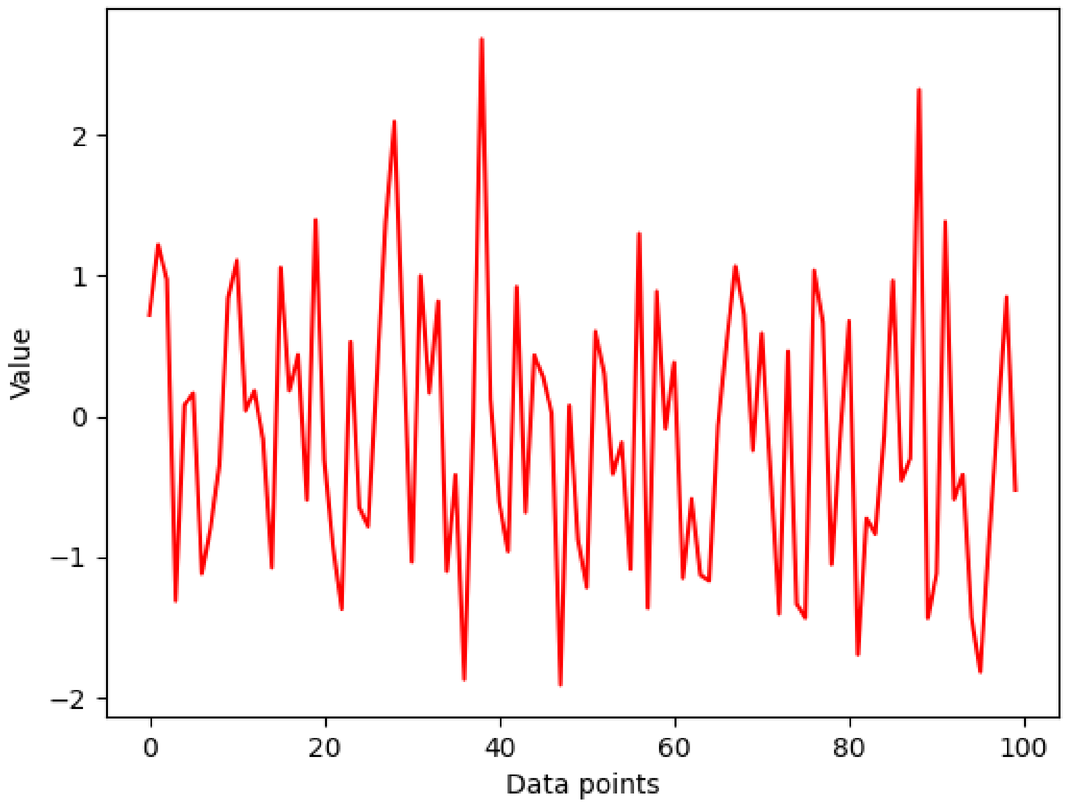


Figure 4.2: Input noise

4.3 Update the discriminator network

The discriminator network is trained on both the real data and the synthetic data. The discriminator network's parameters are updated based on the cross-entropy loss between its predictions and the true labels.

4.4 Update the generator network

The generator network is trained to generate synthetic data that is indistinguishable from real data. The generator network's parameters are updated based on the loss function that compares the discriminator network's predictions on the synthetic data with a target label of 1 (indicating that the synthetic data is real).

4.5 Repeat

The above steps are repeated until the generator network can generate synthetic data that the discriminator network cannot distinguish from real data. For this experiment, the discriminator and generator is trained simultaneously and equally.

4.6 Notice

As the generator improves with training, the discriminator performance gets worse because the discriminator cannot easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. This progression creates a problem: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on invaluable feedback, and its own quality may decrease.

In the scope of this research, the results cover the output of the generator and the losses of discriminator and generator.

5.1 Discriminator and generator losses

Due to the huge amount of steps, for devices that do not support GPU or GPU is not configured enough, the processing time of an epoch is very large. With the support of Google Colab GPU processing, the average runtime dropped sharply to 40 - 60 seconds per an epoch.

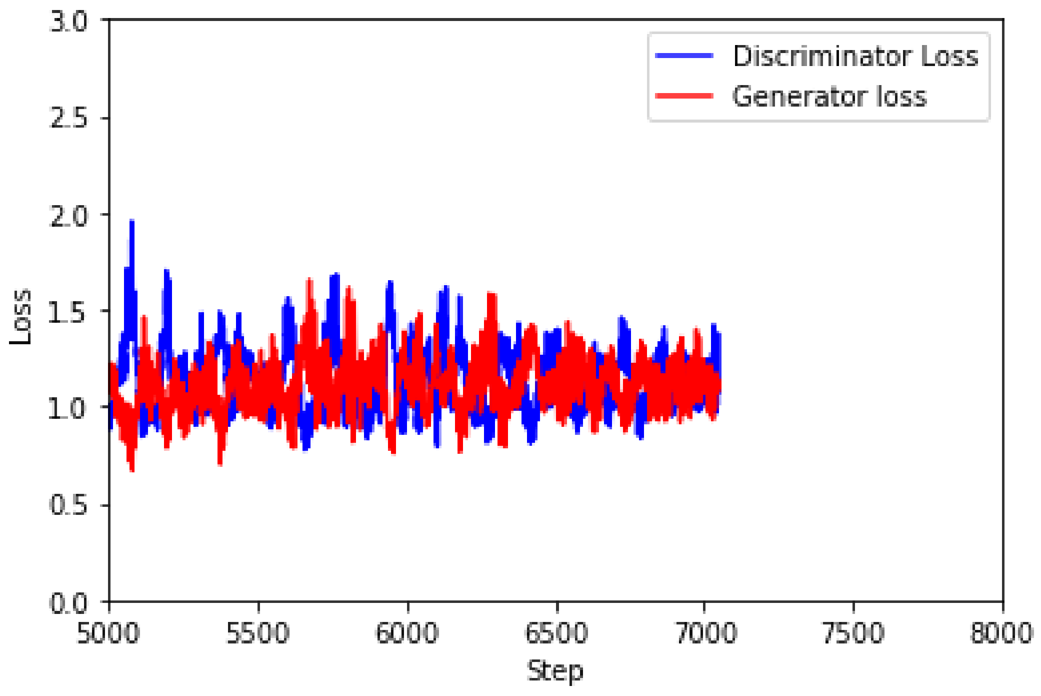


Figure 5.1: Losses

The Discriminator loss and the generator loss vary through time, starting with large distance and end up closer.

5.2 Output of generator

The result for MNIST dataset of generator through out the training process change significantly in the early state. The results in the starting phase is really different from the handwritten digits since the generator have not learn much.

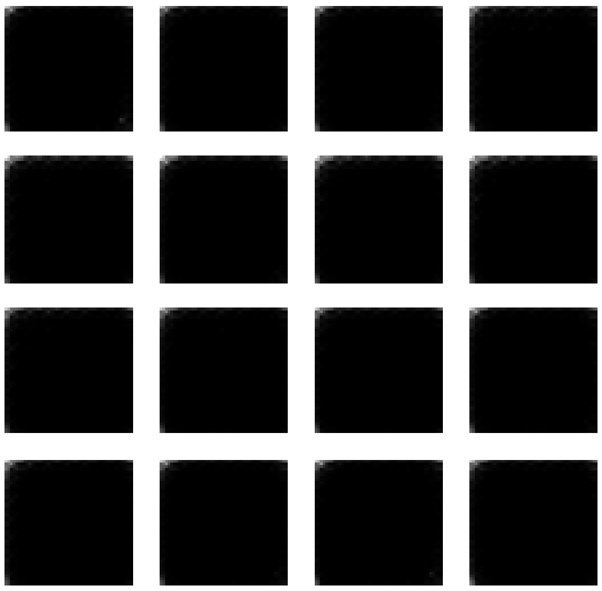


Figure 5.2: Image at epoch 1

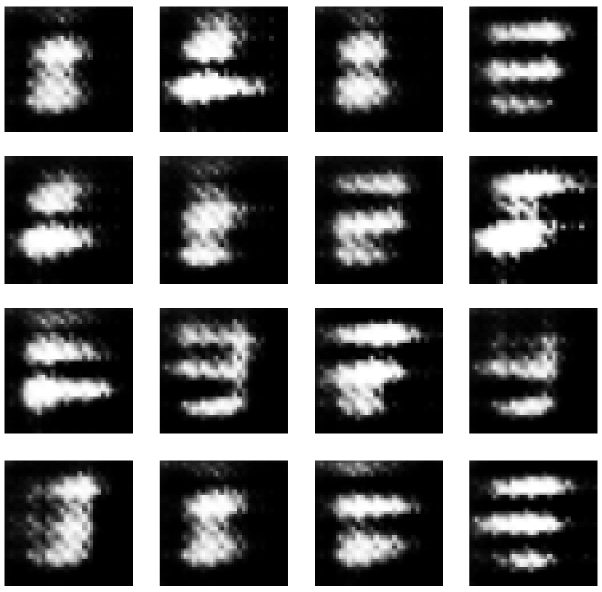


Figure 5.3: Image at epoch 2

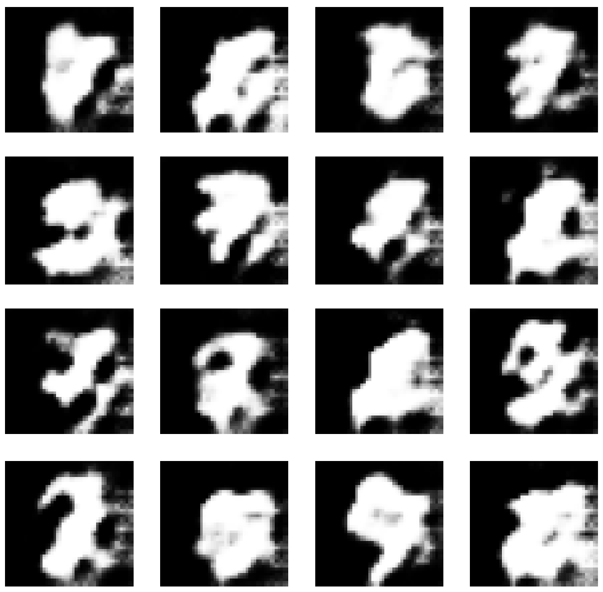


Figure 5.4: Image at epoch 10

From epoch 10 to epoch 20, there are still some changes in the output, the generator keep learning and updating the network to provide better result.

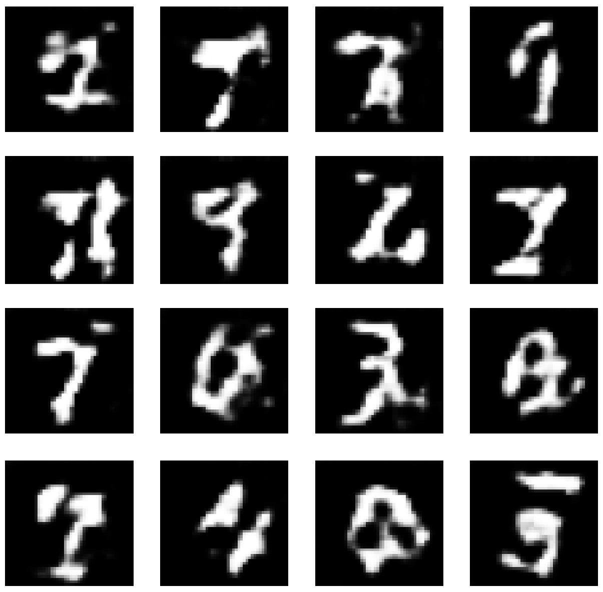


Figure 5.5: Image at epoch 20

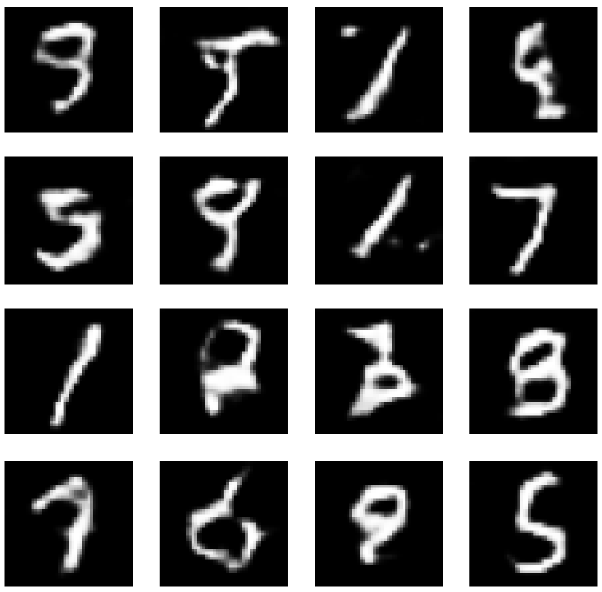


Figure 5.6: Image at epoch 40

After epoch 40, the results and the handwritten digits look more alike. The generator have updated its weight so the output distribution is nearly the same to the training data. However, there are not many considerable changes between the result from epoch 40 and epoch 50 (or further epoch).

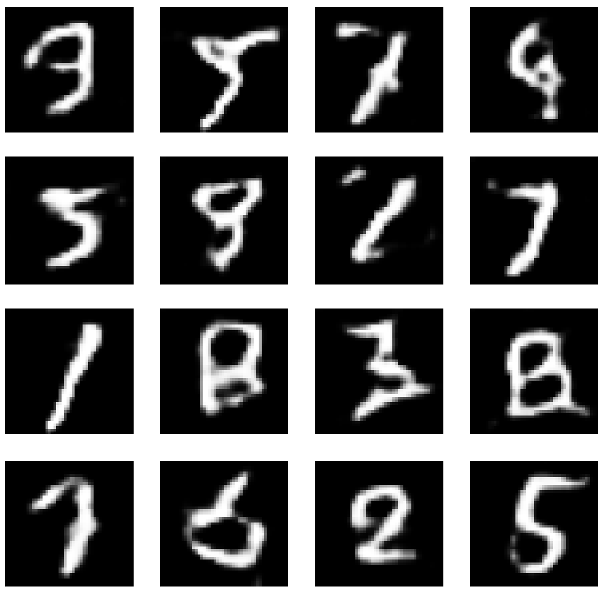


Figure 5.7: Image at epoch 50

CHAPTER 6. CONCLUSIONS

6.1 Summary

In conclusion, this research have summarized the basic of Generative Adversarial Networks. GANs are composed of two neural networks, a generator and a discriminator, that work together in a game-theoretic manner to generate realistic data. The generator creates synthetic data and the discriminator tries to distinguish between real and synthetic data. Through the training process, both networks improve their performance until the generator produces high-quality synthetic data that is indistinguishable from the real data.

However, this research have not covered the evaluation for the model which is still challenging problem. Due to the lack of resources, the results provided might not be enough for further analysis.

Overall, GANs have shown immense promise in generating realistic data and have opened up new avenues for research and development in the field of artificial intelligence. With continued research and development, GANs will likely play an increasingly important role in advancing AI and transforming various industries.

6.2 Suggestion for Future Works

Some improvement can be conducted in the future:

- GANs evaluation: using Inception Score (IS) or Fréchet Inception Distance (FID) to compare the similarity between the real data and generated data.
- Train with labels: Making use of labels in the GANs improves image quality.
- One-sided label smoothing: Using targets for real examples in the discriminator with a value of 0.9 or targets with a stochastic range delivers better results.
- Add random noise to the labels in the discriminator.

SHORT NOTICES ON REFERENCE

"Generative Adversarial Nets" Ian J. Goodfellow, Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair† , Aaron Courville, Yoshua Bengio‡, QC H3C 3J7, 10 June 2014.

"Unsupervised representation learning with deep convolutional generative adversarial networks" Alec Radford Luke Metz, Soumith Chintala, 7 Jan 2016

"NIPS 2016 Tutorial: Generative Adversarial Networks" Ian Goodfellow, 3 Apr 2017

"Effect of Input Noise Dimension in GANs" Padala Manisha , Debojit Das† , and Sujit Gujar‡, April 16, 2020

"Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions" Divya Saxena, Jiannong Cao

APPENDIX