

**Performance Comparison between Generative Adversarial Networks (GAN) Variants  
in Generating Comic Character Images**

By

TAN JIA LER

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

## REPORT STATUS DECLARATION FORM

**Title:** Performance Comparison between Generative Adversarial  
Networks (GAN) Variants in Generating Comic Character  
Images

**Academic Session:** Jan 2024

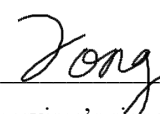
I TAN JIA LER  
**(CAPITAL LETTER)**

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

  
\_\_\_\_\_  
(Author's signature)

  
\_\_\_\_\_  
(Supervisor's signature)

**Address:**

Lot 14301, Kampung Tersusun,  
Jalan Kampung Dew, 34700  
Simpang. Perak.

Tong Dong Ling  
Supervisor's name

**Date:** 24 April 2024

**Date:** 25 Apr 2024

|  |                   |                                     |                         |
|--|-------------------|-------------------------------------|-------------------------|
| <b>Universiti Tunku Abdul Rahman</b>                                       |                   |                                     |                         |
| Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b> |                   |                                     |                         |
| Form Number: <b>FM-IAD-004</b>   | Rev No.: <b>0</b> | Effective Date: <b>21 JUNE 2011</b> | Page No.: <b>1 of 1</b> |

**FACULTY/INSTITUTE\* OF** \_ Information and Communication Technology \_\_\_\_\_

**UNIVERSITI TUNKU ABDUL RAHMAN**

Date: \_24 April 2024\_\_\_\_\_

**SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS**

It is hereby certified that \_\_\_\_\_ ***Tan Jia Ler*** \_\_\_\_\_ (ID No: 20ACB03397 ) has completed this final year project/ dissertation/ thesis\* entitled “\_Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images\_” under the supervision of \_\_Ts Dr Tong Dong Ling\_\_\_\_ (Supervisor) from the Department of \_Computer Science\_\_\_\_, Faculty/Institute\* of \_Information and Communication Technology\_\_\_\_.

I understand that University will upload softcopy of my final year project / dissertation/ thesis\* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,




\_\_\_\_\_  
(*Tan Jia Ler*)

\*Delete whichever not applicable

## DECLARATION OF ORIGINALITY

I declare that this report entitled “Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  \_\_\_\_\_

Name : \_\_\_\_\_ Tan Jia Ler \_\_\_\_\_

Date : \_\_\_\_\_ 24 April 2024 \_\_\_\_\_

## **ACKNOWLEDGEMENTS**

I would like to express my sincere thanks and appreciation to my supervisors, Ts Dr Tong Dong Ling who has given me this bright opportunity to engage in a Deep Learning project. Her guidance and support have played a pivotal role in shaping my understanding and skills in this field, marking a significant step in my journey towards a career in artificial intelligence field. Furthermore, I am truly appreciating her help and patience in answering my question. When I shared my opinion to her, she will listen thoroughly and give useful suggestion as well as provide support to me. A million thanks to you.

Besides, I also express my appreciation to my project moderator, Ts Dr Mogana a/p Vadiveloo for giving insightful suggestion from professional perspective that significantly contributed to the improvement of my project. She also points out my mistakes, so I won't repeat them again in the next project. Thank you very much your guidance and support.

## **ABSTRACT**

Generative Adversarial Networks (GANs) have emerged as a powerful framework for generating realistic and diverse data, including images. This project aims to provide a comprehensive understanding of GANs and their applications in anime face generation. Through theoretical investigation, practical implementation, and empirical analysis, the project explores the working principles of GANs, including their architecture, training dynamics, and variants. The focus is on prominent GAN architectures such as Deep Convolutional GANs (DCGAN), CycleGAN, and Spectral Normalization GAN (SNGAN).

The project conducts a thorough performance analysis of these GAN architectures in anime face generation tasks. This involves collecting and preprocessing anime face datasets, training GAN models, and evaluating their performance using quantitative metrics. The quality and diversity of generated anime face images are analyzed using FID and IS score. Furthermore, a comparative analysis of DCGAN, CycleGAN, and SNGAN is conducted to identify their strengths and weaknesses. This comparative study provides insights into the suitability of different GAN architectures for anime face generation applications. The project aims to contribute to the advancement of knowledge in the field of GANs.

# TABLE OF CONTENTS

|  |             |
|--|-------------|
| <b>TITLE PAGE</b>                      | <b>i</b>    |
| <b>REPORT STATUS DECLARATION FORM</b>  | <b>ii</b>   |
| <b>FYP THESIS SUBMISSION FORM</b>      | <b>iii</b>  |
| <b>DECLARATION OF ORIGINALITY</b>      | <b>iv</b>   |
| <b>ACKNOWLEDGEMENTS</b>                | <b>v</b>    |
| <b>ABSTRACT</b>                        | <b>vi</b>   |
| <b>TABLE OF CONTENTS</b>               | <b>vii</b>  |
| <b>LIST OF FIGURES</b>                 | <b>ix</b>   |
| <b>LIST OF TABLES</b>                  | <b>xii</b>  |
| <b>LIST OF SYMBOLS</b>                 | <b>xiii</b> |
| <b>LIST OF ABBREVIATIONS</b>           | <b>xiv</b>  |
| <br>                                   |             |
| <b>CHAPTER 1 INTRODUCTION</b>          | <b>1</b>    |
| 1.1 Problem Statement and Motivation   | 1           |
| 1.2 Objectives                         | 2           |
| 1.3 Project Scope and Direction        | 3           |
| 1.4 Contributions                      | 4           |
| 1.5 Report Organization                | 5           |
| <br>                                   |             |
| <b>CHAPTER 2 LITERATURE REVIEW</b>     | <b>6</b>    |
| 2.1 Previous works on DCGAN            | 6           |
| 2.2 Previous work on SNGAN             | 9           |
| 2.3 Previous work on CycleGAN          | 11          |
| <br>                                   |             |
| <b>CHAPTER 3 SYSTEM MODEL</b>          | <b>17</b>   |
| 3.1 System Design Diagram/Equation     | 17          |
| 3.1.1 System Architecture Diagram      | 19          |
| 3.1.2 Use Case Diagram and Description | 21          |
| 3.1.3 Activity Diagram                 | 23          |

|   |           |
|---|-----------|
| <b>CHAPTER 4 SYSTEM DESIGN</b>                    | <b>25</b> |
| 4.1 System Block Diagram                          | 25        |
| 4.2 System Components Specifications              | 26        |
| 4.3 Circuits and Components Design                | 29        |
| 4.4 System Components Interaction Operations      | 31        |
| <br>  |           |
| <b>CHAPTER 5 EXPERIMENT/SIMULATION</b>            | <b>41</b> |
| 5.1 Hardware Setup                                | 41        |
| 5.2 Software Setup                                | 41        |
| 5.3 Setting and Configuration                     | 42        |
| 5.4 System Operation (with Screenshot)            | 46        |
| 5.5 Implementation Issues and Challenges          | 54        |
| 5.6 Concluding Remark                             | 55        |
| <br>  |           |
| <b>CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION</b> | <b>56</b> |
| 6.1 System Testing and Performance Metrics        | 56        |
| 6.2 Testing Setup and Result                      | 60        |
| 6.3 Project Challenges                            | 71        |
| 6.4 Objectives Evaluation                         | 72        |
| 6.5 Concluding Remark                             | 73        |
| <br>  |           |
| <b>CHAPTER 7 CONCLUSION AND RECOMMENDATION</b>    | <b>75</b> |
| 7.1 Conclusion                                    | 75        |
| 7.2 Recommendation                                | 77        |
| <br>  |           |
| <b>REFERENCES</b>                                 | <b>78</b> |
| <b>WEEKLY LOG</b>                                 | <b>82</b> |
| <b>POSTER</b>                                     | <b>87</b> |
| <b>PLAGIARISM CHECK RESULT</b>                    | <b>88</b> |
| <b>FYP2 CHECKLIST</b>                             | <b>94</b> |



## LIST OF FIGURES

| Figure Number  | Title  | Page |
|----------------|--|------|
| Figure 2.1.1   | DCGAN model's framework  | 6    |
| Figure 2.1.2   | IS equation  | 8    |
| Figure 2.2.1   | WGAN and SNGAN   | 9    |
| Figure 2.3.1   | Adversarial loss functions for G                                 | 11   |
| Figure 2.3.2   | Adversarial loss functions for F                                 | 11   |
| Figure 2.3.3   | Cycle-consistency loss function                                  | 11   |
| Figure 2.3.4   | Overall loss function  | 11   |
| Figure 2.3.5   | Example of result of UNIT  | 13   |
| Figure 2.3.6   | Examples of CycleGAN outputs when model crashed                  | 13   |
| Figure 2.3.7   | Result of using SGD optimizer with momentum = 0.9 in<br>CycleGAN | 14   |
| Figure 2.3.8   | The blue color background is misinterpreted as hair              | 14   |
| Figure 2.3.9   | Color of the wall is mapped to the hair of character             | 14   |
| Figure 2.3.10  | CycleGAN unable to generate facial contours                      | 15   |
| Figure 2.3.11  | Comparisons between the two CycleGANs                            | 16   |
| Figure 3.1.1   | System Design Diagram  | 17   |
| Figure 3.1.2   | DCGAN and SNGAN Equation   | 17   |
| Figure 3.1.3   | Overall loss function  | 18   |
| Figure 3.1.4   | Adversarial loss functions for G                                 | 18   |
| Figure 3.1.5   | Adversarial loss functions for F                                 | 18   |
| Figure 3.1.6   | Cycle-consistency loss function                                  | 18   |
| Figure 3.1.1.1 | System Architecture Diagram of DCGAN                             | 19   |
| Figure 3.1.1.2 | System Architecture Diagram of SNGAN                             | 20   |
| Figure 3.1.1.3 | System Architecture Diagram of CycleGAN                          | 20   |
| Figure 3.1.2.1 | Use Case Diagram   | 21   |
| Figure 3.1.3.1 | Activity Diagram of DCGAN  | 23   |
| Figure 3.1.3.2 | Activity Diagram of SNGAN  | 23   |
| Figure 3.1.3.3 | Activity Diagram of CycleGAN                                     | 24   |

|               |   |    |
|---------------|---|----|
| Figure 4.1.1  | System Block Diagram                      | 25 |
| Figure 4.3.1  | Component Design of DCGAN                 | 29 |
| Figure 4.3.2  | Component Design of SNGAN                 | 29 |
| Figure 4.3.3  | Component Design of CycleGAN              | 30 |
| Figure 4.4.1  | Generator of DCGAN                        | 31 |
| Figure 4.4.2  | Discriminator of DCGAN                    | 31 |
| Figure 4.4.3  | ReLU activation                           | 32 |
| Figure 4.4.4  | Tanh activation                           | 32 |
| Figure 4.4.5  | Leaky ReLU activation                     | 33 |
| Figure 4.4.6  | Sigmoid activation                        | 34 |
| Figure 4.4.7  | Spectral Norm in SNGAN                    | 34 |
| Figure 4.4.8  | Generator of CycleGAN                     | 37 |
| Figure 4.4.9  | Discriminator of CycleGAN                 | 38 |
| Figure 5.4.1  | Runtime type and Hardware Accelerator     | 46 |
| Figure 5.4.2  | Import Libraries                          | 46 |
| Figure 5.4.3  | Integrated with Google Drive              | 47 |
| Figure 5.4.4  | Dataset loading for DCGAN and SNGAN       | 47 |
| Figure 5.4.5  | Dataset loading for CycleGAN              | 47 |
| Figure 5.4.6  | Transform for data preprocessing          | 47 |
| Figure 5.4.7  | Train and Test Dataloader                 | 48 |
| Figure 5.4.8  | Input hyperparameter                      | 48 |
| Figure 5.4.9  | Weight Initialization                     | 49 |
| Figure 5.4.10 | Define Loss function and Optimizer        | 49 |
| Figure 5.4.11 | Define Discriminator                      | 49 |
| Figure 5.4.12 | Define Generator                          | 50 |
| Figure 5.4.13 | Generators and discriminators in CycleGAN | 50 |
| Figure 5.4.14 | SNLinear layer in SNGAN                   | 51 |
| Figure 5.4.15 | Progress Bar                              | 51 |
| Figure 5.4.16 | FID and IS calculation                    | 52 |
| Figure 5.4.17 | Loss plotting                             | 52 |
| Figure 5.4.18 | FID and IS plotting                       | 52 |
| Figure 5.4.19 | Real vs Fake images plotting              | 53 |
| Figure 6.1.1  | Inception Score working                   | 56 |

|               |  |    |
|---------------|--|----|
| Figure 6.1.2  | Inception Score formulae                         | 57 |
| Figure 6.1.3  | FID score working                                | 58 |
| Figure 6.1.4  | FID score formulae                               | 58 |
| Figure 6.2.1  | Real vs Fake images in DCGAN                     | 61 |
| Figure 6.2.2  | Real vs Fake images in SNGAN                     | 62 |
| Figure 6.2.3  | Real vs Fake images (X to Y, Y to X) in CycleGAN | 62 |
| Figure 6.2.4  | Loss in DCGAN                                    | 63 |
| Figure 6.2.5  | Loss in SNGAN                                    | 64 |
| Figure 6.2.6  | Loss in CycleGAN                                 | 66 |
| Figure 6.2.7  | FID and IS in DCGAN                              | 67 |
| Figure 6.2.8  | FID and IS in SNGAN                              | 68 |
| Figure 6.2.9  | FID and IS in CycleGAN (Y to X)                  | 69 |
| Figure 6.2.10 | FID and IS in CycleGAN (X to Y)                  | 70 |

## LIST OF TABLES

| <b>Table Number</b> | <b>Title</b>             | <b>Page</b> |
|---------------------|--------------------------|-------------|
| Table 5.1.1         | Specifications of laptop | 41          |

## LIST OF SYMBOLS

$\lambda$  Weight parameter

## LIST OF ABBREVIATIONS

|       |   |
|-------|---|
| G     | Generator   |
| D     | Discriminator                                     |
| GANs  | Generative Adversarial Networks                   |
| DCGAN | Deep Convolutional Generative Adversarial Network |
| CNNs  | Convolutional Neural Networks                     |
| IS    | Inception Score                                   |
| KL    | Kullback-Leibler                                  |
| Ladv  | Adversarial Loss                                  |
| Lcls  | Domain Classification Loss                        |
| Lrec  | Reconstruction Loss                               |
| ReLU  | Rectified Linear Unit                             |
| tanh  | Hyperbolic Tangent                                |
| UNIT  | UNsupervised Image-to-Image Translation           |
| SGD   | Stochastic Gradient Descent                       |
| SNGAN | Spectral Normalised GAN                           |
| FID   | Frechet Inception Distance                        |
| IS    | Inception Score                                   |

# Chapter 1

## Introduction

The GANs, having both a generator (G) and a discriminator (D), operate in a feedback loop to make and evaluate data. The main purpose of a generator is to create fake data that is as realistic as possible. It does this by trying to generate realistic images from a noise input. In contrast, the discriminator's job is to distinguish between real and generated data. As the generator creates increasingly convincing data, the discriminator continually refines its ability to differentiate between real and fake data. This ongoing process ensures that both networks enhance their performance over time.

### 1.1 Problem Statement and Motivation

#### Problem Statement

GAN is a new technology. The complexity of GANs and their training dynamics pose a significant challenge to researchers seeking to comprehend their underlying principles. Without a comprehensive understanding of how GANs operate, it becomes difficult to optimize their performance or troubleshoot issues that may arise during training. Consequently, there is a pressing need to delve deeper into the theoretical foundations of GANs, exploring their core components, training mechanisms, and architectural variations to facilitate more effective utilization in anime face generation tasks.

Despite the widespread adoption of GANs for anime face generation, there exists a lack of systematic performance analysis of different GAN architectures in this domain. Evaluating the quality and diversity of generated anime face images requires the establishment of robust performance metrics. By conducting a comprehensive performance analysis, researchers can gain insights into the strengths and limitations of various GAN models, enabling informed decision-making regarding their applicability and suitability for specific anime face generation tasks.

With the proliferation of GAN architectures tailored for anime face generation, researchers face the challenge of selecting the most suitable model for their specific requirements. However, conducting a meaningful comparison between different GAN architectures requires

rigorous evaluation and comparative analysis. By comparing the performance of GANs, researchers can identify the relative advantages and disadvantages of each model, thereby guiding future research efforts and practical applications in anime face generation.

## **Motivation**

In the rapidly evolving world of digital design and animation, there exists a pressing issue that both amateurs and professionals face with - creating unique, high-resolution characters that captivate the audience. As industries, especially the gaming and entertainment sectors, constantly evolve, so do their demands for complex, novel characters. Researchers like [1] and [2] have shown how GANs have become torchbearers in this field, providing automated solutions that are both unique and thematically resonant. Unique character design is not just a matter of artistic achievement. It's an economic necessity. Successful character design can lead to brand recognition, merchandise opportunities, and sequels or spin-offs. Conversely, poorly designed characters can make content unrelatable or forgettable, leading to losses and missed opportunities.

## **1.2 Objectives**

This study aims to:

**Understanding the Working Principles of Generative Adversarial Networks (GANs):**

This objective focuses on gaining a comprehensive understanding of the underlying principles and mechanisms of Generative Adversarial Networks (GANs). Through literature review, experimentation, and hands-on implementation, the project aims to elucidate the core components, training dynamics, and architectural variations of GANs. By delving into the theoretical foundations and practical applications, the objective seeks to establish a solid foundation for analyzing and evaluating the performance of GANs in anime face generation tasks.

**Performance Analysis of GANs in Anime Face Generation:**

This objective entails a detailed analysis of the performance of Generative Adversarial Networks (GANs) specifically applied to the task of anime face generation. Through qualitative and quantitative evaluation, the project aims to assess the quality and diversity of generated anime face images produced by different GAN architectures, including DCGAN, CycleGAN,



and SNGAN. The analysis will involve examining key performance metrics, such as FID score, IS score and loss to gain insights into the strengths and limitations of each GAN model.

Comparative Analysis of Generative Models:

A comprehensive objective includes conducting a comparative analysis of DCGAN, CycleGAN, and StarGAN to understand their strengths and weaknesses. This comparative study aims to provide insights into the unique features of each model and identify scenarios where one model might outperform the others. The goal is to find out the most suitable model for a specific image generation task.

### **1.3 Project Scope and Direction**

This project aims to delve deep into the working principles of Generative Adversarial Networks (GANs) by conducting a thorough investigation of their theoretical foundations. The focus will be on understanding the architecture, training dynamics, and loss functions inherent to GANs. Through implementation and experimentation using Python and deep learning frameworks like PyTorch, the project seeks to gain practical insights into the workings of GAN models. Furthermore, a comprehensive literature review will be conducted to synthesize existing research on GANs, providing valuable insights into their underlying principles.

In the realm of anime face generation, this project will focus on analyzing the performance of various GAN architectures. This involves collecting and preprocessing anime face datasets to ensure diversity and quality in the training data. Multiple GAN architectures will then be trained on these datasets, and their performance will be evaluated using quantitative metrics. The goal is to analyze the quality and diversity of generated anime face images produced by different GAN architectures.

A key aspect of this project is the comparative analysis of DCGAN, CycleGAN, and SNGAN. By comparing the performance of these GAN models, insights into their strengths and weaknesses will be identified. This comparative study will provide valuable insights into the suitability of different GAN architectures for specific anime face generation applications. Ultimately, the project aims to contribute to the advancement of knowledge in the field of

GANs and their applications in anime face generation through rigorous analysis and experimentation.

#### **1.4 Contributions**

This project contributes to the theoretical understanding of GANs by exploring their foundational principles, architectural variations, and training dynamics. Through an in-depth literature review and practical experimentation, the project provides insights into the inner workings of GANs, shedding light on their mechanisms for generating realistic images. The project also offers a comprehensive analysis of GAN performance specifically in the context of anime face generation tasks. By empirically evaluating multiple GAN architectures, including DCGAN, CycleGAN, and SNGAN, the project assesses the quality and diversity of generated anime face images. This analysis provides valuable insights into the strengths and weaknesses of different GAN models for anime face synthesis.

A significant contribution of this project lies in its comparative analysis of prominent GAN architectures, highlighting their distinct characteristics and performance profiles. By systematically comparing DCGAN, CycleGAN, and SNGAN, the project identifies the relative advantages and limitations of each model in generating anime face images. This comparative study aids researchers in selecting the most suitable GAN architecture for specific anime face generation applications.

The insights generated from this project have implications for both research and practical applications in the field of generative modeling and anime face generation. Researchers can leverage the findings to deepen their understanding of GANs and explore novel approaches for improving anime face synthesis. Practitioners in industries such as character design, animation, and virtual content creation can benefit from the recommendations provided for selecting GAN models to suit their specific needs.

## **1.5 Report Organization**

The report organization for the project is structured to provide a comprehensive understanding of the research conducted on Generative Adversarial Networks (GANs) in the context of anime face generation. Chapter 1, the Introduction, sets the stage by outlining the problem statement and motivation behind the research. It also presents the objectives, project scope, contributions, and an overview of the report organization to provide a roadmap for the reader. Chapter 2, the Literature Review, delves into existing research and literature on GANs, focusing on their principles, architectures, and applications in image generation tasks, with a specific emphasis on anime face generation. Chapter 3, the System Model, introduces the design and structure of the GAN-based anime face generation system. It includes system design diagrams, equations, and descriptions of system architecture, use cases, and activities. Chapter 4, the System Design, elaborates on the technical aspects of the system, including block diagrams, component specifications, component designs, and interactions between system components, providing a detailed insight into the system's construction. Chapter 5, Experiment/Simulation, details the experimental setup and execution of the GAN models for anime face generation. It covers hardware and software setup, configuration, system operation, implementation challenges, and concluding remarks on the experimental process. Chapter 6, System Evaluation and Discussion, evaluates the performance of the GAN models through testing, performance metrics, testing setups, results, challenges encountered during the project, objective evaluation, and concluding remarks. Chapter 7, Conclusion and Recommendation, concludes the report by summarizing the findings, drawing conclusions based on the research outcomes, and providing recommendations for future research directions or improvements to the GAN-based anime face generation system.

# Chapter 2

## Literature Review

### 2.1 Previous works on DCGAN

Based on [4], DCGAN, or Deep Convolutional Generative Adversarial Network, represents a significant advancement in the realm of Generative Adversarial Networks (GANs) by incorporating Convolutional Neural Networks (CNNs). In contrast to traditional GANs, both the discriminator and generator in DCGAN employ CNNs instead of the typical multilayer perceptrons. As the name suggests, DCGAN introduces a deep convolutional architecture into the GAN framework. The key distinguishing feature is the utilization of convolutional layers without the inclusion of maximum pooling or fully connected layers. This architecture relies on the synergy between convolutional strides and transpose operations for down-sampling and up-sampling, respectively.

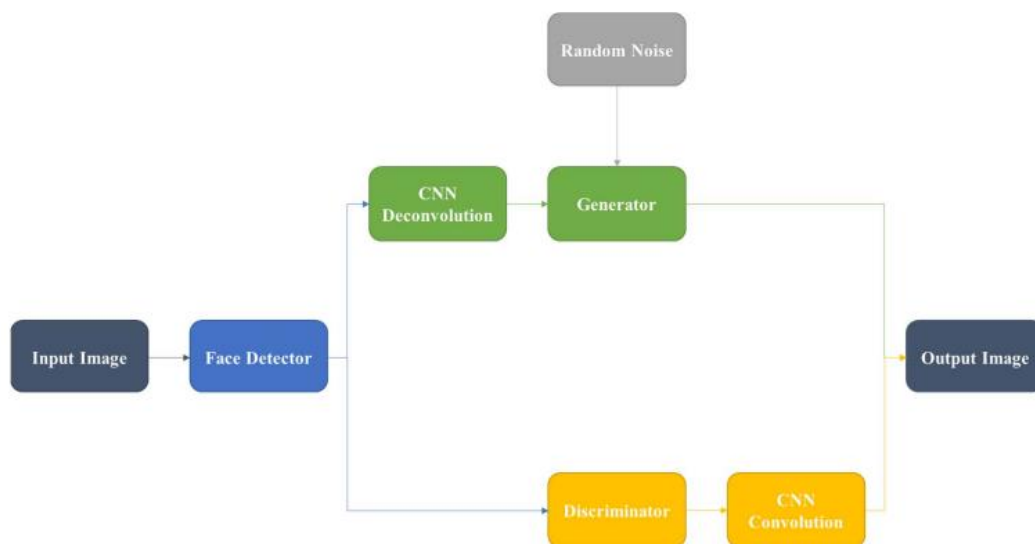


Figure 2.1.1: DCGAN model's framework

By integrating CNNs, DCGAN enhances the ability of the generator and discriminator to understand spatial hierarchies and capture complex patterns in the data. This not only contributes to the generation of more realistic and detailed images but also improves feature extraction in the discriminator. The absence of dense layers and the emphasis on convolutional operations make DCGAN particularly suitable for image-related tasks, providing a powerful framework for generating high-quality images through adversarial training.

In the PyTorch implementation, ReLU activation is used in the generator, employing Tanh for output, while the discriminator utilizes LeakyReLU activation for all layers. The DCGAN framework utilizes TensorFlow's convolution function to process data into a low-dimensional matrix for the discriminator's authenticity comparison. The deconv function expands this matrix to a high-dimensional form, helping the generator in image generation. Defined functions like conv2D and deconv2d encapsulate these convolutional neural network processes.

During DCGAN training, the generative model aims to fool the discriminant model, while the discriminative model attempts to accurately distinguish between generated and real images. This adversarial training promotes a dynamic interplay between the two models, with the ideal result being that the generated images achieve a mixture of realism and falseness ( $D(G(z)) = 0.5$ ).

The discriminator is constructed as a forward convolutional neural network, while the generator maps the latent space vector ( $z$ ) to the data space through a series of 2D convolutional transposed layers. The DCGAN paper emphasizes the use of stride convolution instead of pooling for down-sampling to improve the network's ability to learn its pooling function. Batch normalization and the LeakyReLU functions promote healthy gradient flow, which is crucial for effective learning in both the generator and discriminator.

The generator's structure is similar to the discriminator's but uses a deconvolution function to reverse the convolution results, generating images from randomly generated noise ( $Z$ ). The symmetry in the number of convolution and deconvolution layers ensures the similarity of the resulting images.

In the process of collecting animation images from the Internet, approximately 50,000 images were collected using web crawlers and zip downloads. However, due to the complexity of the data, some images contain multiple faces, some images contain landscapes without characters, and some displaying characters at unusual angles, so preprocessing was required. The images were first combined into a consistent JPG format to ensure uniformity of input data. A facial recognition program, sourced from GitHub, was then used to identify and isolate faces in each image. This process resulted in more than 20,000 full-face animation images.

To normalize the data for convolutional neural network (CNN) operations, all images were resized to 96x96 pixels using a facial recognition program. The goal was to improve the

consistency of image vectorization for the same number of layers in the CNN. All experiments were conducted on a computer with a 3.6GHz Intel Core i9-9900K processor and 16GB RAM, running the Windows 10 operating system. Data preprocessing was implemented using Python 3.6.2.

The model's performance was evaluated using the Inception Score (IS), a metric that evaluates the clarity and diversity of generated images. In this case, Inception Net-V3 (the third version of the Inception Net) was used for calculating the IS. Inception Net is a picture classification network trained on the ImageNet database, containing 1.2 million RGB images across 1000 categories. The IS formula involves calculating the Kullback-Leibler (KL) divergence of conditional probability distributions. Specifically, it measures the distance between the predicted distribution and the true distribution of the main objects in the generated image. The IS equation is given by:

$$IS(G) = \exp (IE_{x \sim P_g} D_{KL}(P(y|x) || p(y)))$$

Figure 2.1.2: IS equation.

where (x) represents a generated picture, (y) represents the main object in the picture, (P<sub>g</sub>) is the generator's distribution, and (D<sub>KL</sub>) is the Kullback-Leibler divergence. A higher IS indicates greater diversity and clarity in the generated images.

## 2.2 Previous work on SNGAN

Generative Adversarial Networks (GANs) have emerged as a powerful framework for generating realistic data across various domains. However, stabilizing the training of GANs remains a significant challenge, prompting researchers to explore various techniques to enhance their performance and reliability.

Several approaches have been proposed to stabilize GAN training. Wasserstein GAN (WGAN) [22] introduced the Wasserstein distance to mitigate mode collapse and instability issues by enforcing a Lipschitz constraint on the discriminator. However, WGAN relied on weight clipping, leading to suboptimal results due to overly constrained discriminator weights.

To address this limitation, WGAN-GP introduced gradient penalty, which penalizes the norm of the discriminator's gradients with respect to interpolated samples. While effective, WGAN-GP's performance can be sensitive to hyperparameters and suffer from gradient penalty instability during training.

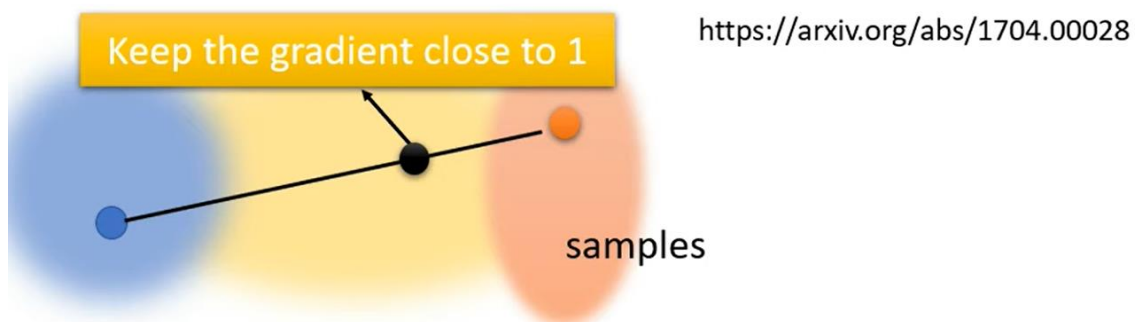
$$\max_{D \in 1\text{-Lipschitz}} \{E_{x \sim P_{data}} [D(x)] - E_{x \sim P_G} [D(x)]\}$$

- Original WGAN → Weight

Force the parameters  $w$  between  $c$  and  $-c$

After parameter update, if  $w > c$ ,  $w = c$ ; if  $w < -c$ ,  $w = -c$

- Improved WGAN → Gradient Penalty



- Spectral Normalization → Keep gradient norm smaller than 1 everywhere

<https://arxiv.org/abs/1802.05957>

Figure 2.2.1: WGAN and SNGAN

Spectral Normalization (SN) emerged as a promising technique to stabilize GAN training by enforcing Lipschitz constraints on the discriminator. [3] SN achieves this by bounding the spectral norm of weight matrices in the discriminator, leading to more stable and robust training dynamics. SN has been shown to improve the quality and diversity of generated samples across various datasets.

Spectral Normalization GAN (SNGAN) builds upon SN to further enhance stability and performance in GAN training. By applying spectral normalization to the discriminator, SNGAN ensures Lipschitz continuity, leading to improved convergence properties and generation quality. SNGAN has demonstrated superior performance compared to previous GAN variants, especially in scenarios where mode collapse and training instability are prevalent.

While SN and SNGAN have shown promising results, challenges remain in optimizing GANs for diverse datasets and tasks. The impact of architectural choices, loss functions, and hyperparameters on training stability and sample quality requires further investigation. Additionally, the scalability of SN and SNGAN to large-scale datasets and complex modalities warrants attention.

In light of the existing challenges, the proposed method introduces L2 norm regularization in the generator to complement spectral normalization in the discriminator. By directly minimizing the Euclidean distance between generated and real data, the proposed approach aims to improve stability and quality in GAN training. Initial experiments on CIFAR-10 and STL-10 datasets demonstrate promising results, highlighting the potential of the proposed method in addressing stability issues and enhancing sample quality in GANs.

Future research directions may include theoretical analyses of Lipschitz constraints and regularization techniques in GANs, as well as empirical evaluations on larger datasets and diverse modalities. Furthermore, exploring the interaction between spectral normalization and additional regularization methods could provide insights into further enhancing GAN training stability and sample quality.



### 2.3 Previous work on CycleGAN

The Cycle-Consistent Adversarial Network or CycleGAN is an unsupervised learning method that incorporates a cycle consistency loss function within the framework of GANs. The main objective of CycleGAN is image-to-image translation between two sample spaces, X and Y, with an underlying relationship. [6] The generator (denoted as G) transforms samples from X to Y, while a discriminator,  $D_Y$ , distinguishes between generated and real images in domain Y. Similarly, there is a mapping function F that converts samples in Y back to X, and a discriminator  $D_X$  for domain X.

The adversarial loss functions for G and F are defined as:

$$\begin{aligned} \mathcal{L}_{GAN}(G, D_Y, X, Y) = & \mathbb{E}_{y \sim p_{data}(y)} [\log D_Y(y)] \\ & + \mathbb{E}_{x \sim p_{data}(x)} [\log (1 - D_Y(G(x)))] \end{aligned}$$

Figure 2.3.1: Adversarial loss functions for G

$$\begin{aligned} \mathcal{L}_{GAN}(G, D_X, Y, X) = & \mathbb{E}_{x \sim p_{data}(x)} [\log D_X(x)] \\ & + \mathbb{E}_{y \sim p_{data}(y)} [\log (1 - D_X(G(y)))] \end{aligned}$$

Figure 2.3.2: Adversarial loss functions for F

Additionally, the cycle consistency loss is introduced to ensure that the mappings are cycle-consistent, meaning that every image in domain X can be reversed to the original image after the translation cycle. The cycle-consistency loss function is defined as:

$$\begin{aligned} \mathcal{L}_{cyc}(G, F) = & \mathbb{E}_{x \sim p_{data}(x)} [\| F(G(x)) - x \|_1] \\ & + \mathbb{E}_{y \sim p_{data}(y)} [\| G(F(y)) - y \|_1] \end{aligned}$$

Figure 2.3.3: Cycle-consistency loss function

Here, the L1 norm is employed to calculate the cycle-consistency loss. The overall loss function, combining adversarial and cycle-consistency components, is given by:

$$\begin{aligned} \mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{GAN}(G, D_Y, X, Y) \\ & + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{cyc}(G, F) \end{aligned}$$

Figure 2.3.4: Overall loss function

Where  $\lambda$  is a weight parameter. This comprehensive loss function guides the training process of CycleGAN to achieve meaningful and reversible mappings between the two domains. The incorporation of cycle consistency helps to the robustness and quality of the generated images. In this paper, the architecture of the generator network in CycleGAN consists of several components:

- **Convolution with ReLU Activation:** The network starts with a convolutional layer using Rectified Linear Unit (ReLU) activation. ReLU is a popular activation function known for introducing non-linearity in neural networks.
- **Two Down-sampling Blocks:** These blocks are responsible for reducing the spatial dimensions of the input images from 128 x 128 to 256 x 256. Down-sampling usually involves operations such as convolution and pooling to decrease the resolution of the image.
- **Six Residual Blocks:** Residual blocks are a key component in deep neural networks, contribute to the learning of identity mappings and aiding in the convergence of the training process. In this case, six residual blocks are employed, each contributing to the transformation of 256 x 256 images. The decision to use six residual blocks instead of nine is influenced by the input image size of 128 x 128 in this particular project.
- **Two Up-sampling Blocks:** These blocks are designed to increase the spatial dimensions of the images, taking them from 128 x 128 back to 64 x 64. Up-sampling typically involves operations like transposed convolution to increase the resolution of the image.
- **Convolution with Hyperbolic Tangent (tanh) Activation:** The last layer of the generator applies a convolution operation with a hyperbolic tangent (tanh) activation function. Tanh is commonly used for the final layer of a generator in GANs to produce pixel values in the range [-1, 1].

For the discriminator in CycleGAN, a 70 x 70 PatchGAN architecture is used. PatchGAN is a variant of the discriminator that classifies images by dividing them into smaller patches. The output of a regular GAN discriminator is a single scalar, while PatchGAN outputs an N x N array of values, where each value corresponds to a patch of the input image. This approach allows the discriminator to assess the realism of local patches, providing more detailed feedback to the generator during training.

A. Baseline Model Selection:

Baseline models serve as a reference to evaluate the performance of the trained model. In this context, the UNsupervised Image-to-Image Translation (UNIT) is introduced as a baseline model for comparison with CycleGAN. The goal is to evaluate whether CycleGAN performs better than UNIT. An experiment is carried out, as shown in Figure 2.3.5, provides visual results comparing the original input to UNIT with the corresponding transformed anime face images.



Figure 2.3.5: Example of result of UNIT

#### B. Batch Size:

The batch size is a critical parameter in machine learning, influencing the generalization performance of the model. Larger batch sizes contribute to stability and faster convergence but might impact performance. This experiment shows that increasing the batch size enhances model stability and accelerates training speed. Notably, situations such as model crashes (all outputs are the same) become less frequent. Figure 2.3.6 illustrates the impact of varying batch sizes.



Figure 2.3.6: Examples of CycleGAN outputs when model crashed.

#### C. Optimizer Selection:

Deep learning optimizers can be broadly categorized as adaptive (e.g., Adam) and acceleration-based (e.g., Stochastic Gradient Descent with momentum). The default choice for complex models like GANs is often an adaptive optimizer due to their stability. An experiment is conducted with the use of Stochastic Gradient Descent (SGD) optimizer with momentum (0.9) instead of the Adam optimizer in CycleGAN. Results, presented in Figure 2.3.7, showcases the result of this optimizer selection experiment.



Figure 2.3.7: Result of using SGD optimizer with momentum = 0.9 in CycleGAN

#### D. Analysis of Experiments:

Many measures have been proposed to objectively evaluate GANs, but there is still no consensus on a measure that comprehensively captures the strengths and weaknesses of GANs. As visual examination by humans remains one of the most intuitive ways to evaluate GANs, this research utilizes qualitative analysis to assess the results of each experiment.

#### Effect of Background and Items:

Through comparing the 3 models, it is evident that both UNIT and CycleGAN can be influenced by background or other items, such as crowns or hats. For example, in Figure 2.3.8, UNIT mistakenly interprets the blue background as the hair color, generating an anime character with blue hair. Similarly, in Figure 2.3.9, CycleGAN maps the color of the wall to the hair of the corresponding anime character.



Figure 2.3.8: The blue color background is misinterpreted as hair

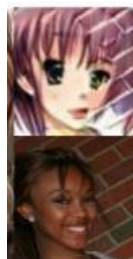


Figure 2.3.9: Color of the wall is mapped to the hair of character

### Performance on Side Faces:

Due to a lack of sufficient side images in both datasets, both UNIT and CycleGAN perform poorly in generating side faces. In some images, UNIT fails to generate approximate facial features like the mouth. In the Figure 2.3.10, CycleGAN with a batch size of 16 has difficulty generating facial contours. However, CycleGAN using the SGD optimizer demonstrates relatively good performance in accurately displaying facial features, contours, and hair.



Figure 2.3.10: CycleGAN unable to generate facial contours

### Comparative Analysis:

In conclusion, UNIT is good at learning facial features in frontal anime images through encoding and decoding in a shared latent space. CycleGAN, utilizing two discriminators and cycle consistency, learns facial features well and produces more natural facial combinations compared to UNIT. Thus, CycleGAN consistently outperforms the baseline model.

### Comparisons between CycleGAN Variants:

Figure 2.3.11 provides a comparison between two CycleGAN variants with different parameters. Randomly selecting four pictures as input for CycleGAN with a batch size of 16 and CycleGAN using the SGD optimizer with momentum, and their results are visually compared. The observation shows that CycleGAN using SGD optimizer with momentum generates corresponding anime faces more effectively. In contrast, CycleGAN with batch size 16 only captures the facial contour, resulting in blurred facial features. Furthermore, pictures generated by CycleGAN using SGD optimizer with momentum appear more natural and have less distortion compared to CycleGAN with batch size 16.













| Inputs  | Models  |  |
|---|---|--|
|   | <i>CycleGAN (batch size 16)</i>   | <i>CycleGAN (SGD)</i>  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Figure 2.3.11: Comparisons between the two CycleGANs

# Chapter 3

## System Model

### 3.1 System Design Diagram/Equation

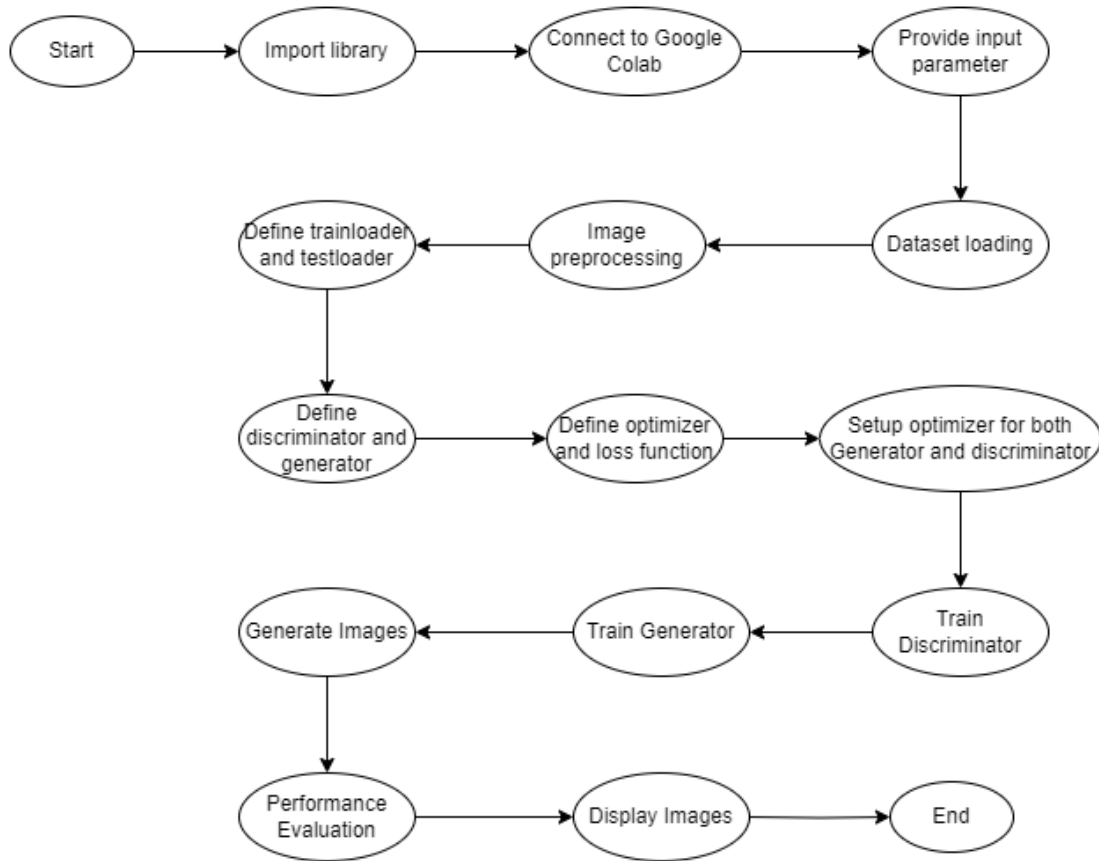


Figure 3.1.1: System Design Diagram

DCGAN[5] and SNGAN[7] have the same equation:

$$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$$
$$L_G^{CGAN} = E[\log(D(G(z), c))]$$

Figure 3.1.2: DCGAN and SNGAN Equation

CycleGAN [6]:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) &= \mathcal{L}_{GAN}(G, D_Y, X, Y) \\ &\quad + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ &\quad + \lambda \mathcal{L}_{cyc}(G, F)\end{aligned}$$

Figure 3.1.3: Overall loss function

Where:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y, X, Y) &= \mathbb{E}_{y \sim p_{data}(y)} [\log D_Y(y)] \\ &\quad + \mathbb{E}_{x \sim p_{data}(x)} [\log (1 - D_Y(G(x)))]\end{aligned}$$

Figure 3.1.4: Adversarial loss functions for G

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_X, Y, X) &= \mathbb{E}_{x \sim p_{data}(x)} [\log D_X(x)] \\ &\quad + \mathbb{E}_{y \sim p_{data}(y)} [\log (1 - D_X(G(y)))]\end{aligned}$$

Figure 3.1.5: Adversarial loss functions for F

$$\begin{aligned}\mathcal{L}_{cyc}(G, F) &= \mathbb{E}_{x \sim p_{data}(x)} [\|F(G(x)) - x\|_1] \\ &\quad + \mathbb{E}_{y \sim p_{data}(y)} [\|G(F(y)) - y\|_1]\end{aligned}$$

Figure 3.1.6: Cycle-consistency loss function

$\lambda$  is a weight parameter.



### 3.1.1 System Architecture Diagram

Deconvolution in Generator[8]: Deconvolution, also known as transpose convolution or upsampling, is used in the Generator to increase the spatial resolution of feature maps. It involves reversing the process of convolution by applying learnable filters to upsample the feature maps. Deconvolution layers expand the spatial dimensions of feature maps, allowing the Generator to generate higher-resolution images from low-dimensional noise vectors.

Convolution in Discriminator[8]: In the Discriminator network, convolutional layers are employed to process both real and generated images. These layers extract features from input images and downsample them into lower-dimensional representations. Convolution operations help the Discriminator distinguish between real and fake images by capturing discriminative features such as edges, textures, and shapes.

DCGAN: There are only one Generator and one discriminator in DCGAN.

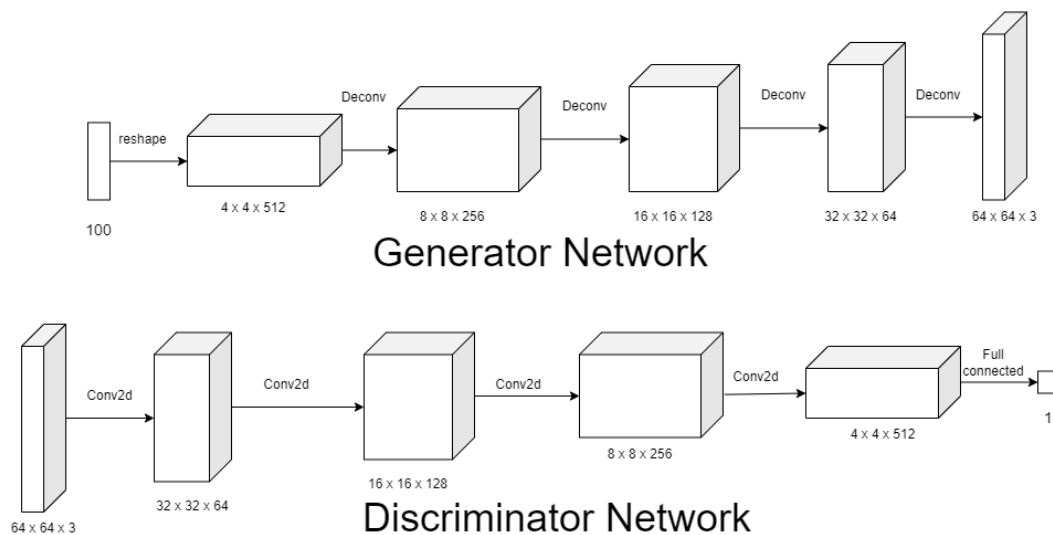


Figure 3.1.1.1: System Architecture Diagram of DCGAN

SNGAN: There are only one Generator and one discriminator in SNGAN.

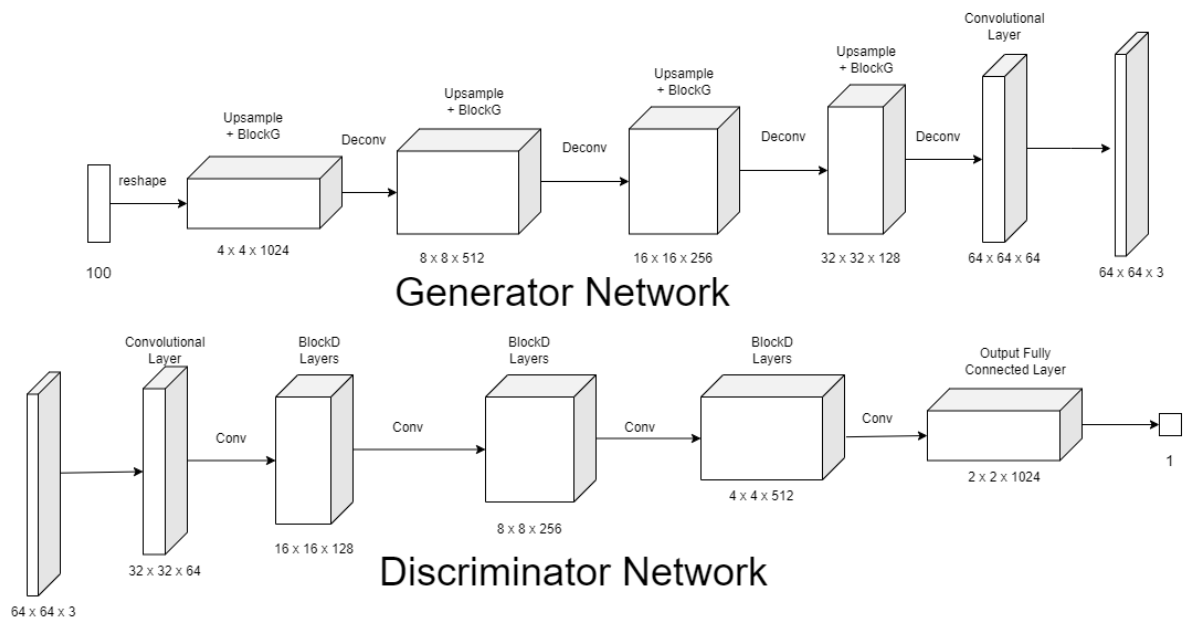


Figure 3.1.1.2: System Architecture Diagram of SNGAN

CycleGAN: There are two generators ( $G_{XY}$ ,  $G_{YX}$ ) and two discriminators ( $D_X$ ,  $D_Y$ ) in CycleGAN.

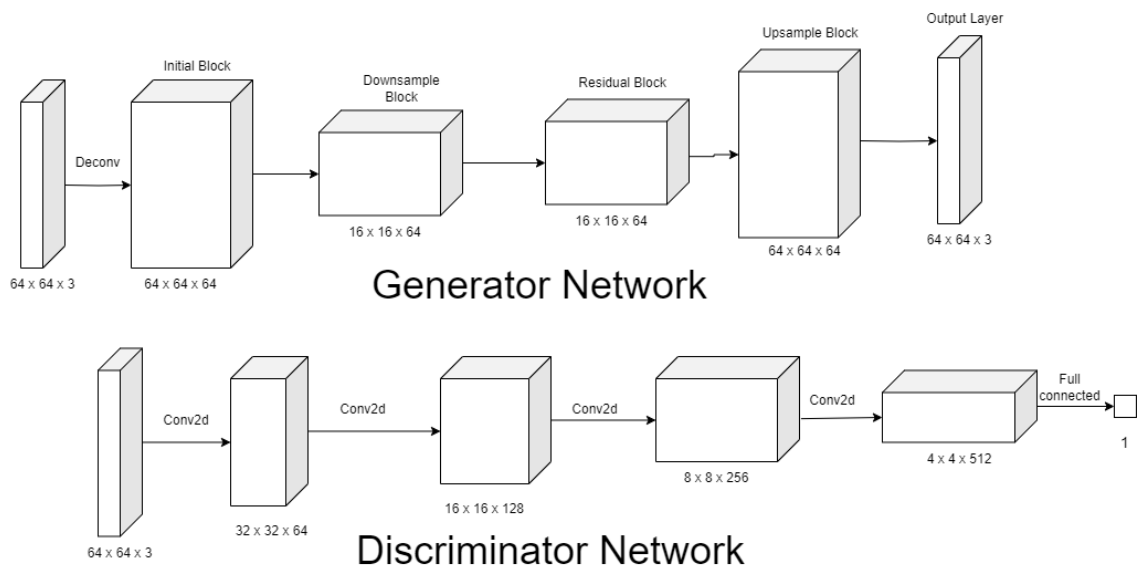


Figure 3.1.1.3: System Architecture Diagram of CycleGAN

### 3.1.2 Use Case Diagram and Description

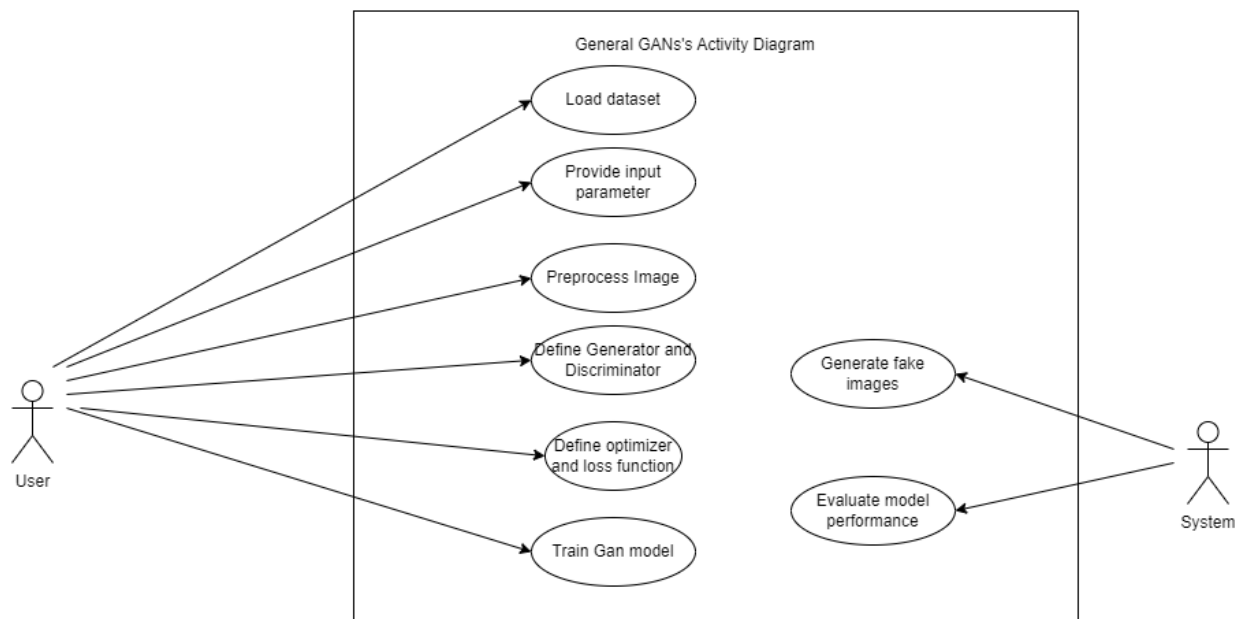


Figure 3.1.2.1: Use Case Diagram

1. Load dataset: Import the dataset containing images that will be used to train the GAN. This dataset should be representative of the type of images the GAN will generate. For cycleGAN, two datasets will be loaded.

2. Provide input parameters: Define parameters such as batch size, image size, number of channels, latent space dimension, number of epochs, learning rate, and other hyperparameters necessary for training the GAN.

3. Preprocess Image: Preprocess the images in the dataset to ensure uniformity and compatibility with the GAN architecture. Preprocessing steps include resizing, normalization, and transformation to tensor format.

4. Define Generator and Discriminator: Create the Generator and Discriminator neural network architectures. The Generator generates fake images from random noise, while the Discriminator evaluates the authenticity of generated images compared to real ones.

5. Define optimizer and loss function: Choose appropriate optimizer (In this case, Adam optimizer is used) for both the Generator and Discriminator networks. Define the loss function

(In this case, Binary Cross-Entropy Loss) used to train the GAN, where the Generator aims to minimize this loss while fooling the Discriminator, and the Discriminator aims to correctly classify real and fake images.

6. Train GAN model: Train the GAN model by iterating over the dataset multiple epochs. During training, the Generator and Discriminator are updated iteratively to improve their performance. The training process involves forward and backward passes, updating network weights based on the computed gradients.

7. Generate fake image: Once the GAN is trained, use the Generator network to generate fake images from random noise vectors sampled from the latent space. These generated images should ideally resemble the images from the training dataset.

8. Evaluate model performance: Assess the performance of the trained GAN using evaluation metrics, which are Inception Score and Frechet Inception Distance.

### 3.1.3 Activity Diagram

DCGAN[9]:

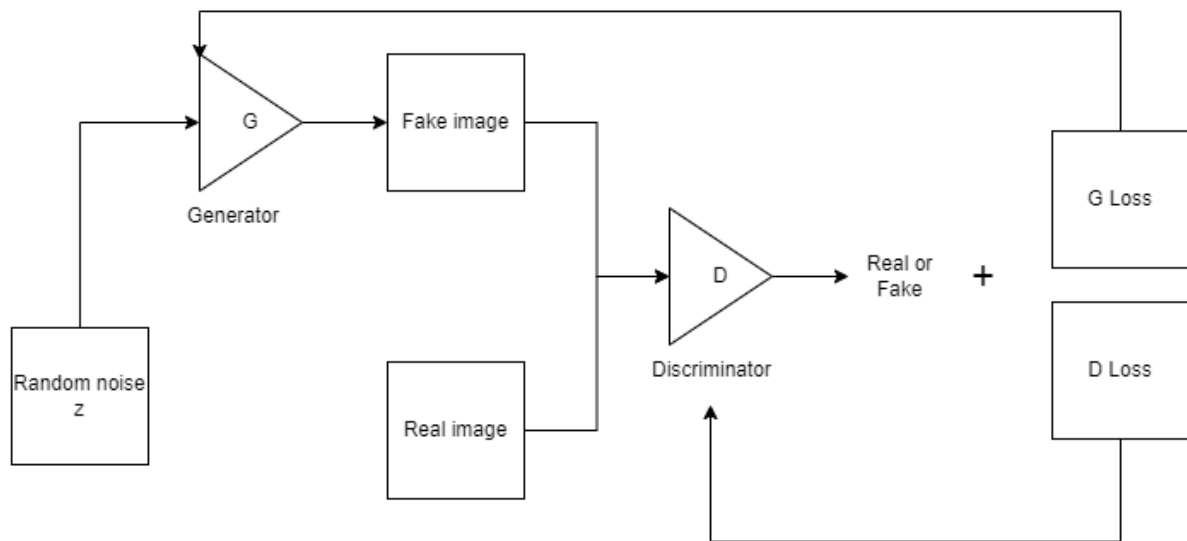


Figure 3.1.3.1: Activity Diagram of DCGAN

SNGAN[10]:

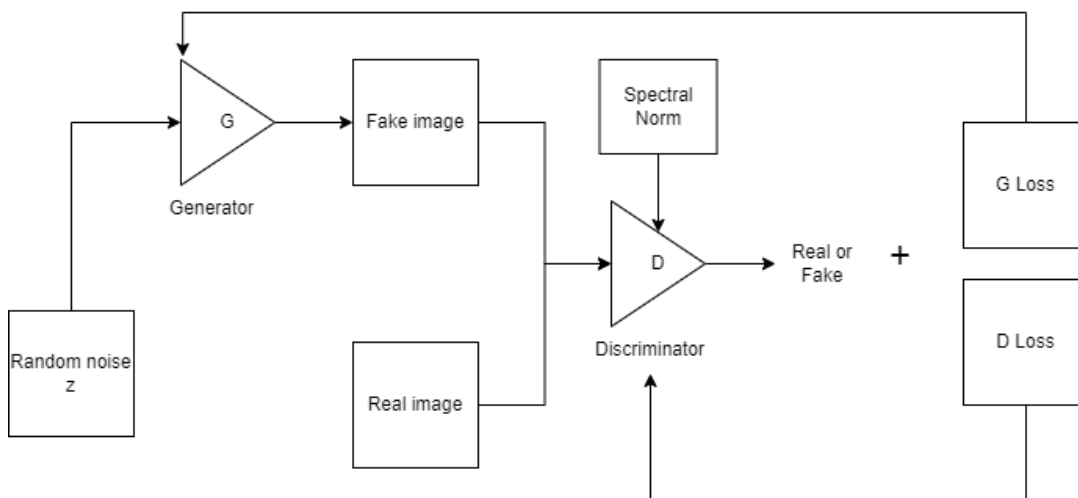


Figure 3.1.3.2: Activity Diagram of SNGAN

CycleGAN[11]:

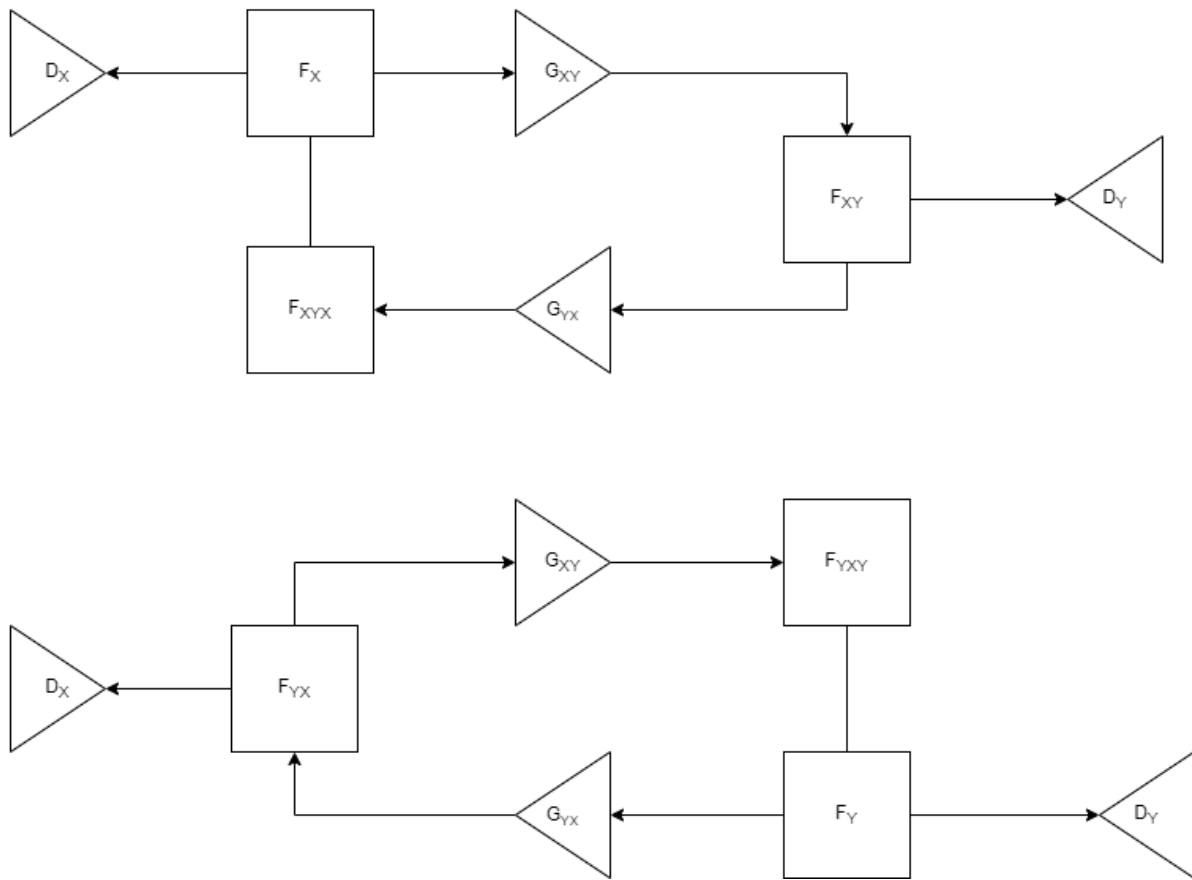


Figure 3.1.3.3: Activity Diagram of CycleGAN

# Chapter 4

## System Design

### 4.1 System Block Diagram

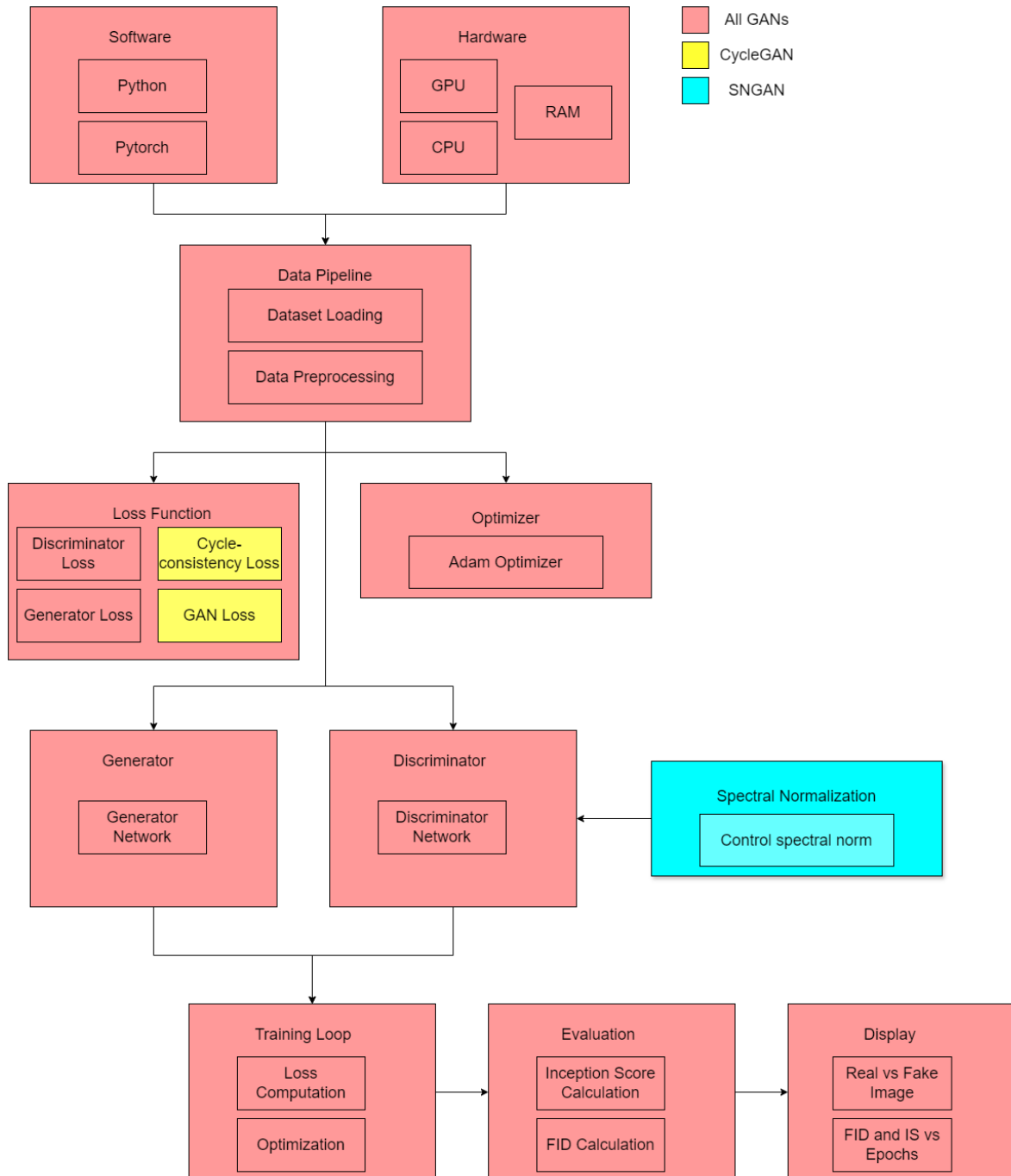


Figure 4.1.1: System Block Diagram

## 4.2 System Component Specification

### Hardware Component:

Deep learning models, including GANs, often require significant computational power for training, especially when dealing with large datasets and complex architectures. GPUs, with their parallel processing capabilities, are commonly used to accelerate the training process by performing matrix operations and backpropagation calculations much faster than CPUs. While GPUs handle most of the heavy lifting during training, CPUs are still crucial for managing overall system operations, coordinating data movement, and handling tasks that are not highly parallelizable. CPUs also play a role in preprocessing data, managing memory, and orchestrating interactions between different software components. Furthermore, sufficient RAM is necessary to store model parameters, intermediate activations, and mini-batches of data during training. The amount of memory required depends on the size of the dataset, the complexity of the model, and the batch size used during training.

### Software Component:

PyTorch is one of the popular frameworks used for implementing GANs. It offers pre-built modules for defining network architectures, loss functions, optimizers, and utilities for data loading and preprocessing. Besides, in this project, the ignite library from Pytorch is used to calculate IS score by using the pre-built InceptionScore function.

### Data Pipeline Component:

The data pipeline component manages the flow of training data into the GAN system. It involves tasks such as data loading, preprocessing and batching to prepare the data for training. It utilizes data loading utilities provided by libraries such as PyTorch, which offer functionalities for loading image datasets, applying transformations, and creating data loaders for efficient training.

### Loss Functions Component:

**Generator Loss:** Typically the negative of the discriminator's output when fed with generated samples. It encourages the generator to produce samples that resemble real data.

**Discriminator Loss:** A combination of errors made on real and fake samples. It encourages the discriminator to correctly classify real samples as real and fake samples as fake.



Cycle-consistent Loss (For CycleGAN): Measures the difference between the original input image and the image reconstructed after being translated back and forth between the two domains.

GAN Loss (For CycleGAN): The summation of all previous losses.

Optimizer Component:

The optimizer updates the parameters of both the generators and discriminators during training, typically using the Adam optimizer with a specific learning rate. It minimizes the combined loss function of the generators and discriminators.

Generator Component:

The generator component generates synthetic images by transforming random noise into realistic-looking images. It takes random noise vectors as input and produces synthetic images that resemble the training data distribution. A generator typically consists of multiple layers of transposed convolutional and batch normalization layers, followed by activation functions such as ReLU or Tanh. Parameters include the batch sizes, kernel sizes, activation functions, and other architectural hyperparameters.

Discriminator Component:

The discriminator component distinguishes between real and fake images by classifying them as genuine or synthetic. It takes input images (either real or synthetic) and outputs a probability score indicating the likelihood of the input being real. A discriminator usually comprises multiple convolutional layers followed by batch normalization and leaky ReLU activation functions. Similar to the generator, the parameter includes architectural hyperparameters such as the batch size, kernel sizes, and activation functions.

Spectral Normalization Component (For SNGAN):

It is used to stabilize training in SNGAN by normalizing the spectral norm of weight matrices in the discriminator's convolutional layers. It is applied to the weights of convolutional layers in the discriminator, typically using the power iteration method to estimate the largest singular value. This can help prevent the discriminator from becoming too powerful relative to the generator, thereby facilitating more stable training and improving sample quality.

#### Training Loop Component:

The training loop component orchestrates the training process of the GAN by optimizing the generator and discriminator components iteratively. It involves loading training data, feeding it through the generator to produce synthetic images, evaluating the discriminator's performance, calculating loss functions, and updating the network weights through backpropagation. It typically employs the adversarial training algorithm, where the generator and discriminator are trained in alternating steps to optimize their respective objectives. Parameters include training hyperparameters such as learning rate, batch size, number of epochs, and optimizer settings.

#### Evaluation Component:

The evaluation component assesses the performance of the trained GAN using quantitative metrics and qualitative analysis. It involved calculating metrics such as Inception Score and FID to measure the quality and diversity of generated images. It also included visualizations of generated images, evaluation metrics over epochs, and comparisons between real and synthetic image distributions.

#### Display Component:

**Real vs Fake Image Comparison:** side-by-side comparisons of real and generated images. This allows for qualitative assessment of the model's ability to produce realistic outputs that closely resemble the ground truth images.

**FID and IS vs Epochs:** provides insights into the image generation quality and diversity throughout the training process.

### 4.3 Component Design

DCGAN[13]:

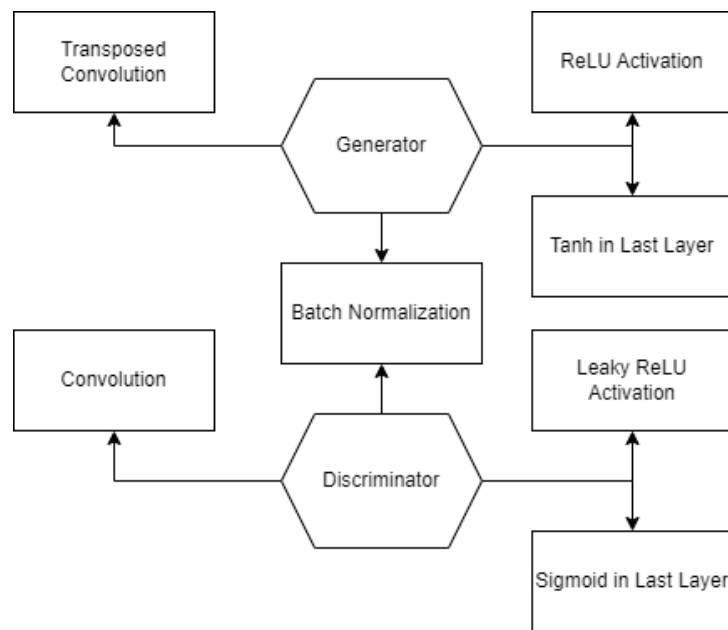


Figure 4.3.1: Component Design of DCGAN

SNGAN:

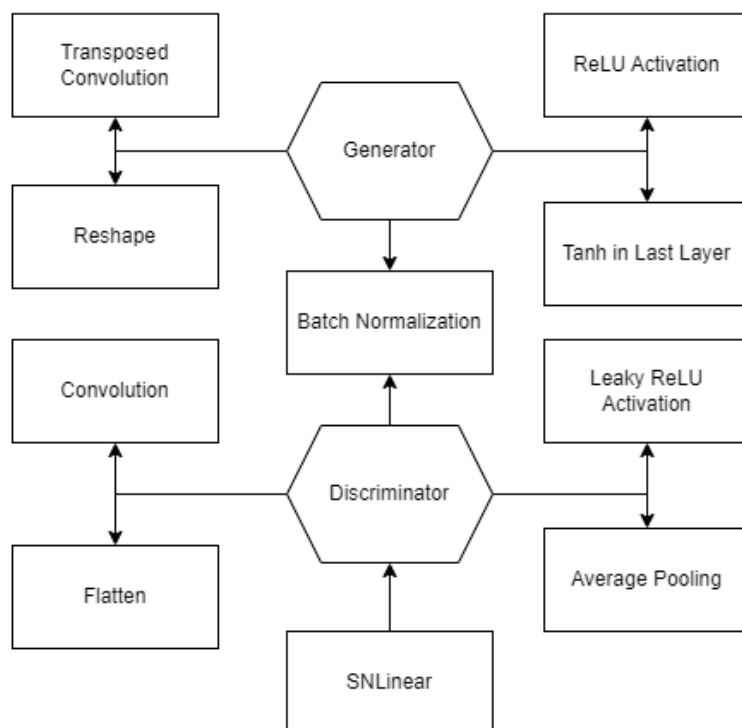


Figure 4.3.2: Component Design of SNGAN

CycleGAN:

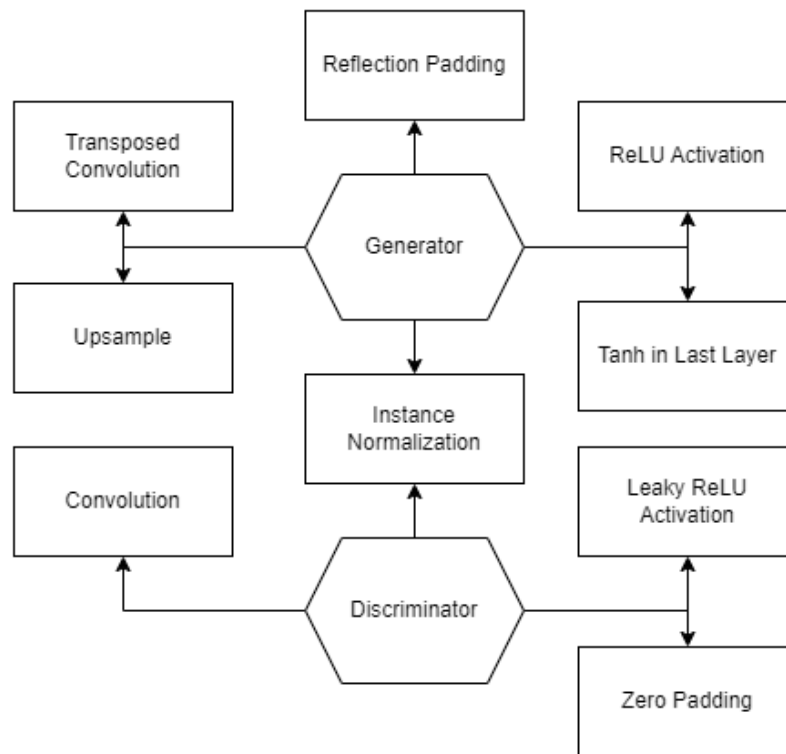


Figure 4.3.3: Component Design of CycleGAN

## 4.4 System Component Interaction Operation

DCGAN[15]:

```
Generator(  
  (main): Sequential(  
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)  
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace=True)  
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): ReLU(inplace=True)  
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): ReLU(inplace=True)  
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): ReLU(inplace=True)  
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (13): Tanh()  
  )  
)
```

Figure 4.4.1: Generator of DCGAN

```
Discriminator(  
  (main): Sequential(  
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): LeakyReLU(negative_slope=0.2, inplace=True)  
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (4): LeakyReLU(negative_slope=0.2, inplace=True)  
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (7): LeakyReLU(negative_slope=0.2, inplace=True)  
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (10): LeakyReLU(negative_slope=0.2, inplace=True)  
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)  
    (12): Sigmoid()  
  )  
)
```

Figure 4.4.2: Discriminator of DCGAN

ConvTranspose2d: These layers are crucial in the Generator as they perform the inverse operation of convolution. ConvTranspose2d layers upsample input noise vectors into higher-dimensional feature maps, gradually transforming them into images. They are responsible for spatial expansion, allowing the Generator to generate images with higher resolutions.

BatchNorm2d: Batch normalization is applied to stabilize and accelerate the training of deep neural networks. In the Generator, BatchNorm2d layers are inserted after convolutional transpose layers to normalize the activations within each mini-batch. This helps mitigate issues like internal covariate shift and enables smoother convergence during training.

ReLU Activation Functions[16]: Rectified Linear Units (ReLU) are used to introduce non-linearity into the Generator network. ReLU activation functions help the Generator learn complex mappings between input noise vectors and image outputs. They are applied after each convolutional transpose layer to introduce non-linearities and capture more complex patterns in the data.

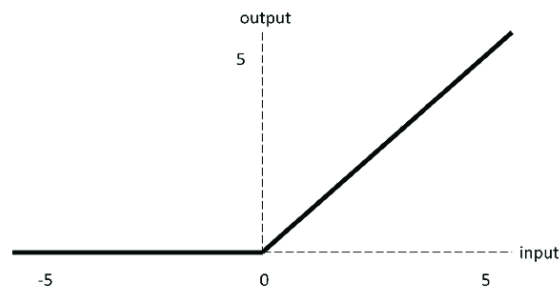


Figure 4.4.3: ReLU activation

Tanh Activation Function [17]: The Tanh activation function is commonly used in the output layer of the Generator. It scales the generated pixel values to the range [-1, 1], aligning with the input range of real images. This ensures that the generated images have pixel values similar to real images, making them visually realistic.

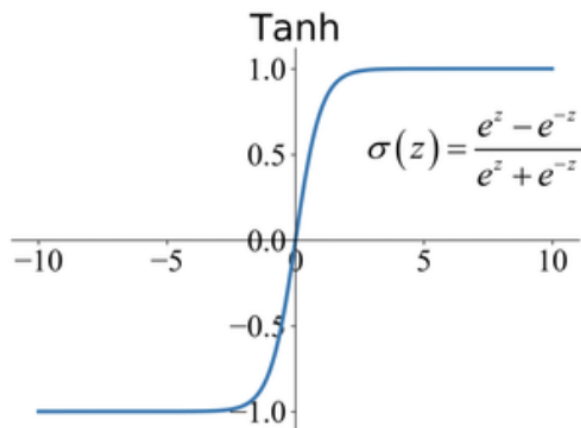


Figure 4.4.4: Tanh activation

Components in the Discriminator:

Conv2d Convolutional layers in the Discriminator network are used to extract features from input images (both real and generated). These layers apply a set of learnable filters to the input images, detecting relevant patterns and features that help differentiate between real and fake images.

BatchNorm2d: Similar to the Generator, BatchNorm2d layers are used in the Discriminator to normalize the activations within each mini-batch. This helps stabilize training and improve convergence by reducing internal covariate shift.

Leaky ReLU Activation Functions[20]: Leaky ReLU activation functions are employed in the Discriminator to introduce non-linearity. Unlike traditional ReLU, Leaky ReLU allows a small, non-zero gradient when the input is negative, preventing the "dying ReLU" problem and enabling the Discriminator to learn more robust features.

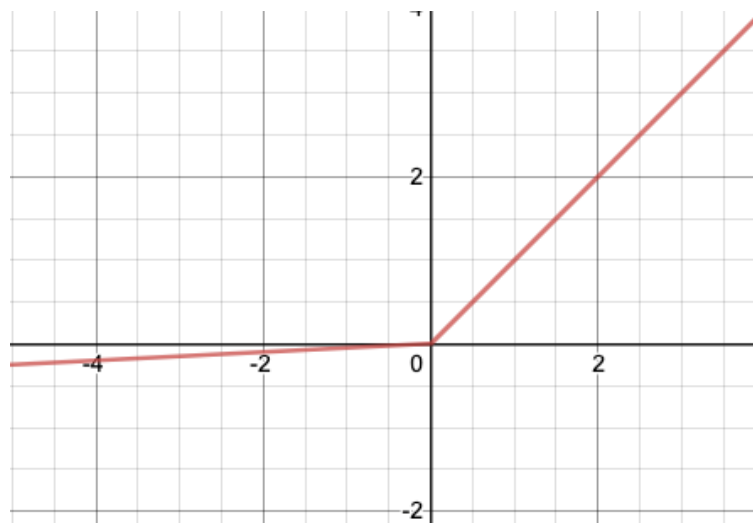


Figure 4.4.5: Leaky ReLU activation

Sigmoid Activation Function[21]: The Sigmoid activation function is typically used in the output layer of the Discriminator. It squashes the discriminator's output logits into the range [0, 1], representing the probability that an input image is real. This facilitates binary classification, with values closer to 1 indicating real images and values closer to 0 indicating fake images.

# Sigmoid Function

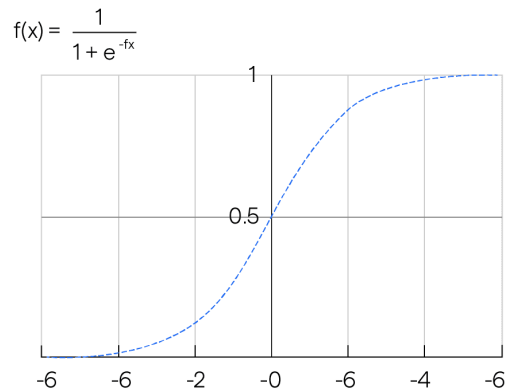


Figure 4.4.6: Sigmoid activation

SNGAN:

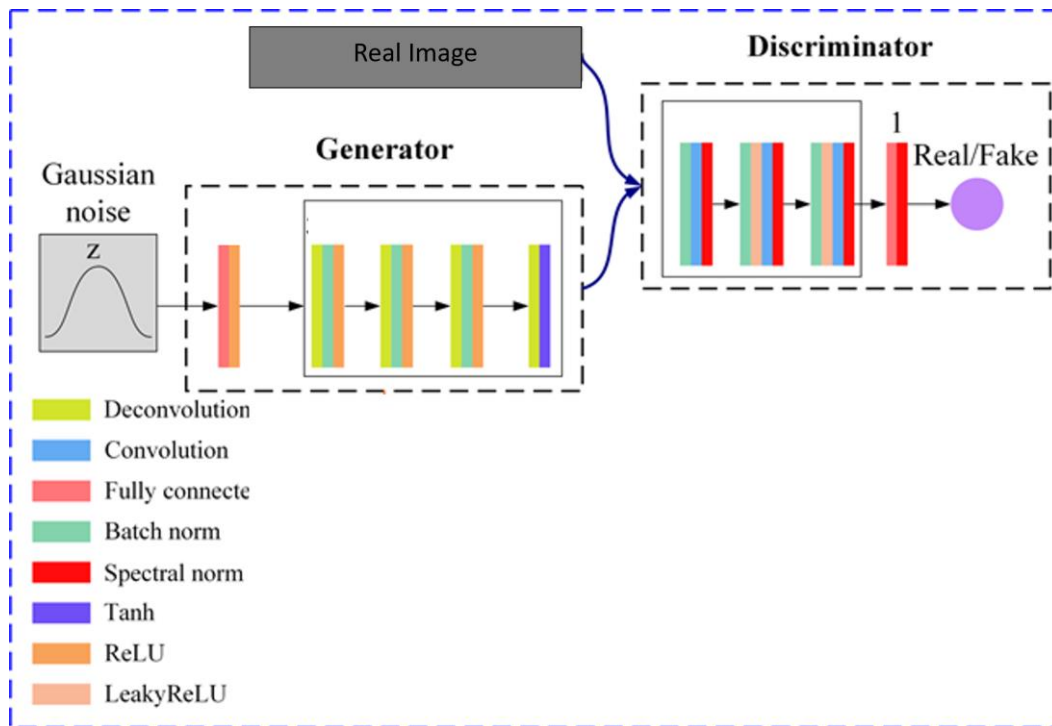


Figure 4.4.7: Spectral Norm in SNGAN

**SNLinear[10]:** This component is similar to PyTorch's Linear layer but includes Spectral Normalization to stabilize the training of the neural network. It performs a linear transformation of the input data, followed by spectral normalization. The spectral normalization ensures that



the spectral norm of weight matrices remains constant during training, which helps in stabilizing the learning process.

Components in the Generator:

**ConvTranspose2d:** This layer performs transposed convolution or deconvolution, which is used for upsampling. In the SNGAN generator, it helps in increasing the spatial resolution of the feature maps as the input noise vectors are transformed into higher-dimensional feature maps, resembling the structure of real images.

**BatchNorm2d:** Batch normalization is applied after each convolutional layer in the generator to stabilize and accelerate the training process. It normalizes the activations of each layer, making the optimization process more robust and reducing the likelihood of vanishing or exploding gradients.

**ReLU:** Rectified Linear Unit activation function introduces non-linearity by outputting the input directly if it is positive, otherwise, it outputs zero. ReLU is used after each batch normalization layer in the generator to introduce non-linearities, allowing the network to learn complex mappings from the input noise space to the output image space.

**Tanh:** The hyperbolic tangent activation function is used in the output layer of the generator to squash the pixel values to the range  $[-1, 1]$ . Since the real images are typically normalized to this range, Tanh ensures that the generated images have similar pixel value distributions, making them visually more realistic.

**Linear:** This layer performs a linear transformation of the input noise vectors to a higher-dimensional space. In the SNGAN generator, it typically maps low-dimensional noise vectors to higher-dimensional feature representations that can be reshaped into 2D feature maps.

**Reshape:** Reshape layer is used to reshape the output of the linear layer to a 2D shape that can be further processed by convolutional layers. It is commonly used in the generator to convert the flattened output of the linear layer into 2D feature maps.

Components in the Discriminator:

**Conv2d:** Convolutional layers are used in the discriminator to extract features from input images. These layers convolve input feature maps with learnable filters to detect spatial patterns and structures in the images.

**BatchNorm2d:** Similar to the generator, batch normalization is applied after each convolutional layer in the discriminator to stabilize and accelerate training. It helps in normalizing the activations and reducing internal covariate shift, making the training process more efficient.

**ReLU:** Rectified Linear Unit activation function introduces non-linearity after each convolutional layer in the discriminator. It helps in capturing complex patterns and features in the input images, making the discriminator more discriminative in distinguishing between real and fake images.

**AvgPool2d:** Average pooling layers are used in the discriminator for downsampling the spatial dimensions of the feature maps. They compute the average value of each feature map region, reducing the spatial resolution while retaining important information about the image.

**Identity:** Identity function is used in residual blocks to create bypass connections that allow gradients to flow directly through the network without any transformation. This helps in mitigating the vanishing gradient problem and enables the discriminator to learn more efficiently.

**AdaptiveAvgPool2d:** Adaptive average pooling layers are used to convert variable-sized feature maps into fixed-sized representations. They perform spatial averaging to generate output feature maps with a predefined size, ensuring that the discriminator can process images of different resolutions effectively.

**Flatten:** Flatten layer is used to convert the multi-dimensional feature maps into a 1D tensor. It is typically applied before the fully connected layers in the discriminator to flatten the spatial dimensions of the feature maps and feed them into the linear layers for classification.

LeakyReLU: Leaky Rectified Linear Unit activation function is similar to ReLU but allows a small negative slope for negative input values. It helps in preventing the saturation of neurons and encourages the flow of gradients, improving the learning process in the discriminator.

CycleGAN:

```
(8): ResidualBlock(
  (block): Sequential(
    (0): ReflectionPad2d((1, 1, 1, 1))
    (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
    (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False,
track_running_stats=False)
    (3): ReLU(inplace=True)
    (4): ReflectionPad2d((1, 1, 1, 1))
    (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
    (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False,
track_running_stats=False)
  )
)
)
)
(upsample_blocks): Sequential(
  (0): Upsample(scale_factor=2.0, mode='nearest')
  (1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (2): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False,
track_running_stats=False)
  (3): ReLU(inplace=True)
  (4): Upsample(scale_factor=2.0, mode='nearest')
  (5): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
  (7): ReLU(inplace=True)
)
(output): Sequential(
  (0): ReflectionPad2d((3, 3, 3, 3))
  (1): Conv2d(64, 3, kernel_size=(7, 7), stride=(1, 1))
  (2): Tanh()
)
)
```

Figure 4.4.8: Generator of CycleGAN

```

Discriminator D_Y:
Discriminator(
  (model): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (9): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): ZeroPad2d((1, 0, 1, 0))
    (12): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
  )
)

```

Figure 4.4.9: Discriminator of CycleGAN

#### Generator Components:

**ReflectionPad2d:** This component is used to perform zero-padding on the input tensor. In CycleGAN, reflection padding is often used to maintain the spatial dimensions of the feature maps during convolution operations. It ensures that the edges of the input image are properly handled and prevents artifacts such as checkerboard patterns in the generated images.

**Conv2d:** Convolutional layers are fundamental components in the generator for feature extraction and transformation. They apply filters to the input image to learn hierarchical representations, capturing essential features at different spatial scales. These features are crucial for transforming the input image from one domain to another, enabling the generation of realistic output images.

**InstanceNorm2d:** Instance normalization is applied to normalize the activations of each layer independently. It helps stabilize the training process by reducing internal covariate shift, leading to faster convergence and better generalization. In CycleGAN, instance normalization ensures that the network can effectively learn domain-specific features without being biased by the distribution of input images.

**ReLU:** Rectified Linear Unit (ReLU) is an activation function used to introduce non-linearity into the network. It replaces negative values in the feature maps with zero, effectively capturing the positive aspects of the learned features. ReLU activation is applied after convolutional and

normalization layers to enable the generator to learn complex mappings between input and output images.

**Upsample:** The upsampling operation increases the spatial dimensions of the feature maps. In CycleGAN, upsampling is typically used to enlarge the feature maps before applying convolutional layers. It helps recover spatial details lost during downsampling and enables the generator to generate high-resolution output images with fine-grained textures.

**Tanh:** The hyperbolic tangent (Tanh) activation function is applied in the output layer of the generator. It squashes the pixel values of the generated images to the range  $[-1, 1]$ , ensuring that the output images have the same range as the input images. Tanh activation helps stabilize the training process and produces visually appealing results by preventing pixel intensity saturation.

**Discriminator Components:**

**Conv2d:** Similar to the generator, convolutional layers in the discriminator are used for feature extraction and transformation. They analyze the input images and learn discriminative features that distinguish between real and generated images. Convolutional filters in the discriminator capture texture, shape, and color information, enabling effective image classification.

**LeakyReLU:** Leaky Rectified Linear Unit (LeakyReLU) is an activation function used in the discriminator to introduce non-linearity. Unlike the standard ReLU function, LeakyReLU allows a small, non-zero gradient for negative input values, preventing the issue of "dying" neurons. This ensures that the discriminator can effectively learn from both real and generated images, improving its robustness and performance.

**InstanceNorm2d:** Instance normalization is applied in the discriminator to normalize the activations of each layer independently, similar to its use in the generator. It helps stabilize the training process and improves the discriminator's ability to generalize across different input images.

ZeroPad2d: Zero-padding is used in the discriminator to maintain the spatial dimensions of the feature maps. It ensures that the convolutional operations preserve the spatial information of the input images, preventing information loss at the edges. Zero-padding is particularly important in the discriminator to maintain consistency in feature extraction across different regions of the image.

# Chapter 5

## Experiment/ Stimulation

### 5.1 Hardware Setup

The hardware utilized in this project is a computer, specifically employed to run the Google Chrome, facilitating the utilization of Google Colab for the construction, training, and execution of the GAN models.

Table 5.1.1 Specifications of laptop

| Description      | Specifications                                    |
|------------------|---|
| Model            | ROG Strix G531GT_G531GT                           |
| Processor        | Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz |
| Operating System | Windows 11  |
| Graphic          | Intel(R) UHD Graphics 630                         |
| Memory           | 4.00 GB RAM                                       |
| Storage          | 475GB   |

### 5.2 Software Setup

Google Colab is configured for project execution, utilizing its advanced hardware accelerators and high-RAM capabilities. Runtime type selected is Python 3. Compute units were purchased so more powerful hardware accelerator can be ran. Among the five available types of hardware accelerators (CPU, A100 GPU, V100 GPU, T4 GPU and TPU), T4 GPU was selected based on its speed and consumption of compute units. The integration of advanced hardware accelerators will enhance the speed and performance of the generative models, allowing for efficient experimentation and development. Three different generative models: DCGAN, CycleGAN and SNGAN will be implemented. The project setup involves adapting template Python notebooks for DCGAN from [12]; CycleGAN from [19]; SNGAN from [23]. To maximize computational efficiency and accelerate model training, the chosen GPU will be employed during the execution of the notebooks in the Google Colab environment. Additionally, the platform's integration with Google Drive ensured efficient data handling, allowing convenient access to extensive anime image dataset.

### 5.3 Setting and Configuration

#### Data Collection

In this project, anime face dataset from [18] is used. This dataset contains 63,632 high-quality anime faces, ensuring diverse and robust training for the GAN model. For DCGAN and SCGAN, only the one dataset from [18] is required, while for CycleGAN. The dataset from [18] is set as dataset for domain Y (various hair and eye colour) and dataset from [24] is set as dataset for domain X (blue hair and blue eye) for training.

#### Data Loading

In data loading, the first step involves the mounting of Google Drive, providing access to the dataset stored within it. Subsequently, the anime faces dataset is extracted by unzipping the relevant files, and then the images in zip files will be extracted to the dataroot. To efficiently handle the dataset, the project leverages the PyTorch framework, employing the ImageFolder class for loading and preprocessing. This step ensures that the dataset is prepared in a format compatible with the subsequent model training and evaluation processes.

#### Data Preprocessing

To ensure consistency in the input data, all images undergo a resizing process. The images are adjusted to a consistent size of 64 x 64 pixels. The processed and resized images contribute to the formation of the training dataset. This dataset becomes the primary input for training the GAN models, containing various facial expressions and features present in anime faces. To facilitate the flow of data during model training and testing, essential components known as dataloaders are generated. Two specific dataloaders are created, which are train dataloader and test dataloader. These dataloaders efficiently handle the batching and loading of data, ensuring a smooth interaction between the dataset and the GAN models.

#### Setting Hyperparameter

All the three GANs are set with same hyperparameter for comparisons purpose.

**Learning Rate:** This parameter controls the step size during optimization. A typical value for GANs is 0.0002.

**Number of Epochs:** This is the number of times the entire dataset is passed forward and backward through the neural network. Set with 50.



**Batch Size:** It determines the number of samples propagated through the network before the parameters are updated. Set with 128 for DCGAN and SNGAN. However, for the CycleGAN, I can just follow the sample to set with 5. This is because with higher batch sizes, the memory requirements also increase significantly. Each image in the batch consumes memory for both forward and backward passes through the network. This can quickly exhaust the available GPU memory, leading to out-of-memory errors. Indeed, my CycleGAN faced the out-of memory error when I tried to increase the batch size.

**Latent Space Dimension:** The dimensionality of the latent space or the size of the input noise vector. Set as 100. Through experimentation, researchers [27] found that a latent dimension of 100 often provides a good balance between representational capacity and model complexity. It's large enough to capture meaningful variations in the data distribution while not being excessively high, which could lead to overfitting or computational inefficiency.

**Optimizer Parameters:** Hyperparameters related to the optimizer, such as beta1 for Adam optimizer's exponential decay rates of moment estimates. Set as 0.5.

**Number of Workers for DataLoader (workers):** The number of subprocesses to use for data loading. Set as 2.

### Model Architecture Design

In this section, the architecture of the generator and discriminator neural networks were defined. The G is responsible for transforming random noise into synthetic anime face images. The architecture typically contains several layers, including convolutional layers, batch normalization, and activation functions. These components work together to learn and generate complex features present in anime faces. Details of G's architecture, such as the number of layers, filter sizes, and activation functions, are key determinants of its ability to produce realistic and diverse images.

On the other side of the adversarial spectrum, the D evaluates the authenticity of generated images by distinguishing between real and synthetic samples. The architecture of D involves convolutional layers followed by batch normalization and activation functions. The

discriminator's design is focused on capturing and classifying key features that differentiate between real and generated anime face images.

For DCGAN and SNGAN, only one generator and one discriminator is defined. While for CycleGAN, for the purpose of domain-to-domain translation, two generators and two discriminators are defined, which are  $G_{XY}$  and  $G_{YX}$ ,  $D_X$  and  $D_Y$ .

$G_{XY}$  and  $G_{YX}$ :

$G_{XY}$  and  $G_{YX}$  are the generators responsible for translating images from domain X to domain Y and vice versa, respectively. These generators play a crucial role in the cycle-consistent adversarial learning framework by enabling the translation of images between two domains without requiring paired training data.

$D_X$  and  $D_Y$ :

$D_X$  and  $D_Y$  are used to distinguish between real and translated images in domains X and Y, respectively. They are trained adversarially against their corresponding generators to improve the quality of the generated images and ensure that they are indistinguishable from real images in their respective domains.

In the SNGAN, there is an additional layer called SNLinear that required define for performing spectral normalization.

Besides, network weights were initialized using a custom weight initialization function. Proper weight initialization is essential for stabilizing GAN training and preventing issues like vanishing or exploding gradients. A custom weight initialization function (`weights_init`) is employed to initialize the weights of both G and D. This function systematically initializes the network weights, contributing to a more effective and stable learning process during training.

The result of this architectural definition and weight initialization process is the generation of two key models: the Generator G and the Discriminator D. These models encapsulate the learned parameters and architectures, ready to undergo the adversarial training process. The generator aims to produce anime face images that can deceive the discriminator, while the discriminator aims to accurately classify the authenticity of the generated images.

### Model Training

Training is implemented to customarily update the generator and discriminator models iteratively. In DCGAN, the adversarial loss is calculated using the BCE Loss function; In SNGAN, hinge loss is calculated; In CycleGAN, adversarial loss, cycle-consistency loss and GAN Loss are calculated, providing the necessary feedback for model updates. The GAN undergoes Iterative Training over multiple epochs. Throughout each epoch, the models are updated based on the training data and the adversarial loss computed. This iterative process allows the GAN to progressively enhance its ability to generate anime face images that closely resemble real samples.

### Evaluation

The evaluation process involves the computation of two metrics: the FID and IS. For the FID calculation, all the three GANs are using the FID function and InceptionV3 from [25]. While the IS score calculation is done with the pre-built InceptionScore function from the Ignite library.

## 5.4 System Operation

Select runtime type and hardware accelerator

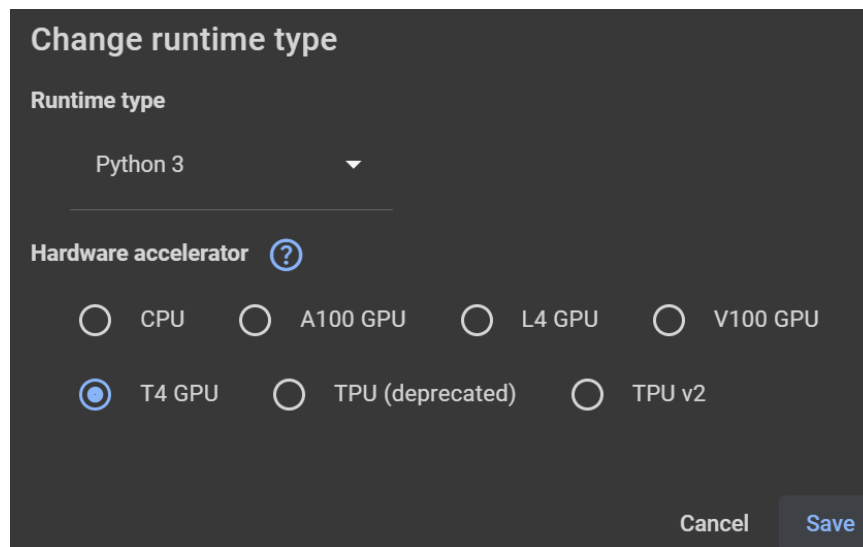


Figure 5.4.1: Runtime type and Hardware Accelerator

Import necessary libraries.

```
import os
import cv2
import torch
import torch.nn as nn
import numpy as np
from PIL import Image
import shutil
import itertools
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
import torchvision.transforms as transforms
from torchvision.utils import make_grid
```

Figure 5.4.2: Import Libraries

Data Loading:

Integrated with Google Drive

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

Mounted at /content/drive
```

Figure 5.4.3: Integrated with Google Drive

For DCGAN and SNGAN, only one dataset is loaded

```
class AnimeFace:
    base_dir = "/content/drive/My Drive/MyData_DCGAN/dataset"
    img_folder = "images"
```

Figure 5.4.4: Dataset loading for DCGAN and SNGAN

For CycleGAN, two datasets are loaded

```
MONET_IMAGES_PATH = "/content/drive/My Drive/rem_preprocessed_512"
TEST_IMAGES_PATH = "/content/drive/My Drive/MyData_DCGAN/dataset/images"
```

Figure 5.4.5: Dataset loading for CycleGAN

Data Preprocessing

```
# train generative models on whole dataset
transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.RandomHorizontalFlip(0.5), # random horizontal flipping
    transforms.ToTensor(), # 0..255 RGB to [0, 1] (C, H, W) Tensor
    transforms.Normalize((0.5, ) * 3, (0.5, ) * 3) # rescale to [-1, 1]
])
traindata = AnimeFace(split="all", transform=transform)
```

Figure 5.4.6: Transform for data preprocessing

## Create the train and test dataloader

```
train_loader = DataLoader(  
    ImageDataset(directory_x=MONET_IMAGES_PATH, directory_y=TEST_IMAGES_PATH, test=False, transforms=transforms_dataset),  
    batch_size = BATCH_SIZE,  
    shuffle = True,  
    num_workers = 2  
)  
  
test_loader = DataLoader(  
    ImageDataset(directory_x=MONET_IMAGES_PATH, directory_y=TEST_IMAGES_PATH, test=True, transforms=transforms_dataset),  
    batch_size = BATCH_SIZE,  
    shuffle = False,  
    num_workers = 2  
)
```

Figure 5.4.7: Train and Test Dataloader

## Model Initialization:

### Set the input hyperparameter

```
# Number of workers for dataloader  
workers = 2  
  
# Batch size during training  
batch_size = 128  
  
# Spatial size of training images. All images will be resized to this  
# size using a transformer.  
image_size = 64  
  
# Number of channels in the training images. For color images this is  
nc = 3  
  
# Size of z latent vector (i.e. size of generator input)  
nz = 100  
  
# Size of feature maps in generator  
ngf = 64  
  
# Size of feature maps in discriminator  
ndf = 64  
  
# Number of training epochs  
num_epochs = 50  
  
# Learning rate for optimizers  
lr=0.0002  
  
# Beta1 hyperparam for Adam optimizers  
beta1 = 0.5
```

Figure 5.4.8: Input hyperparameter

## Weight Initialization

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

Figure 5.4.9: Weight Initialization

## Define the loss function and optimizer

```
# Loss Functions & Optimizers
# Initialize BCELoss function
criterion = nn.BCELoss()

# Create batch of latent vectors that we will use to visualize
# the progression of the generator
fixed_noise = torch.randn(64, nz, 1, 1, device=device)

# Establish convention for real and fake labels during training
""" adding label smoothing """
real_label=0.9
fake_label=0.1

# Setup Adam optimizers for both G and D
optimizerD = optim.Adam(netD.parameters(), lr=0.0002, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=0.0002, betas=(beta1, 0.999))
```

Figure 5.4.10: Define Loss function and Optimizer

## Define the discriminator

```
class Discriminator(nn.Module):
    def __init__(self, in_channels):
        super(Discriminator, self).__init__()
        self.scale_factor = 16

        self.model = nn.Sequential(
            nn.Conv2d(in_channels, 64, 4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, stride=2, padding=1),
            nn.InstanceNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, 4, stride=2, padding=1),
            nn.InstanceNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 512, 4, stride=2, padding=1),
            nn.InstanceNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.ZeroPad2d((1, 0, 1, 0)),
            nn.Conv2d(512, 1, 4, padding=1),
        )

    def forward(self, x):
        return self.model(x)
```

Figure 5.4.11: Define Discriminator

## Define the Generator

```
class GeneratorResNet(nn.Module):
    def __init__(self, in_channels, num_residual_blocks=9):
        super(GeneratorResNet, self).__init__()

        self.initial = nn.Sequential(
            nn.ReflectionPad2d(in_channels),
            nn.Conv2d(in_channels, 64, 2 * in_channels + 1),
            nn.InstanceNorm2d(64),
            nn.ReLU(inplace=True)
        )

        self.downsample_blocks = nn.Sequential(
            nn.Conv2d(64, 128, 3, stride=2, padding=1),
            nn.InstanceNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 256, 3, stride=2, padding=1),
            nn.InstanceNorm2d(256),
            nn.ReLU(inplace=True)
        )

        self.residual_blocks = nn.Sequential(*[ResidualBlock(256) for _ in range(num_residual_blocks)])

        self.upsample_blocks = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='nearest'),
            nn.Conv2d(256, 128, 3, stride=1, padding=1),
            nn.InstanceNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Upsample(scale_factor=2, mode='nearest'),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.InstanceNorm2d(64),
            nn.ReLU(inplace=True)
        )

        self.output = nn.Sequential(
            nn.ReflectionPad2d(in_channels),
            nn.Conv2d(64, in_channels, 2 * in_channels + 1),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.initial(x)
        x = self.downsample_blocks(x)
        x = self.residual_blocks(x)
        x = self.upsample_blocks(x)
        return self.output(x)
```

Figure 5.4.12: Define Generator

For CycleGAN, create the two generators and the two discriminators

```
G_XY = GeneratorResNet(3, num_residual_blocks=9)
D_Y = Discriminator(3)

G_YX = GeneratorResNet(3, num_residual_blocks=9)
D_X = Discriminator(3)
```

Figure 5.4.13: Generators and discriminators in CycleGAN



## SNLinear layer in SNGAN

```
class SNLinear(nn.Module):
    dtype = torch.float32

    def __init__(self, in_ft, out_ft, bias=True, use_gamma=True, pow_iter=1, lip_const=1):
        super(SNLinear, self).__init__()
        self.in_ft = in_ft
        self.out_ft = out_ft
        self.use_gamma = use_gamma
        self.pow_iter = pow_iter
        self.lip_const = lip_const
        self.weight = nn.Parameter(torch.empty((out_ft, in_ft), dtype=self.dtype))
        if bias:
            self.register_parameter(
                "bias", nn.Parameter(torch.zeros((out_ft, ), dtype=self.dtype)))
        else:
            self.register_buffer("bias", None)
        self.lip_const = lip_const
        self.register_buffer("u", torch.randn((out_ft, ), dtype=self.dtype))
        if use_gamma:
            self.register_parameter(
                "gamma", nn.Parameter(torch.ones((1, ), dtype=self.dtype)))
        else:
            self.register_parameter("gamma", None)

        # initialize the parameters
        nn.init.kaiming_normal_(self.weight, a=0., mode="fan_in")
```

Figure 5.4.14: SNLinear layer in SNGAN

## Training Loop:

Print the progress bar

```
1/50 epochs: 100% ██████████ 284/284 [05:48<00:00, 1.23s/it, dis_loss=0.774, gen_loss=0.945]
2/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.618, gen_loss=0.954]
3/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.513, gen_loss=0.965]
4/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.468, gen_loss=0.976]
5/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.392, gen_loss=0.995]
6/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.37, gen_loss=0.995]
7/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.353, gen_loss=0.999]
8/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.356, gen_loss=0.998]
9/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.326, gen_loss=0.995]
10/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.35, gen_loss=1]
11/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.333, gen_loss=1.01]
12/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.353, gen_loss=0.999]
13/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.339, gen_loss=0.992]
14/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.352, gen_loss=0.988]
15/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.324, gen_loss=0.991]
16/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.322, gen_loss=0.99]
17/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.311, gen_loss=0.988]
18/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.308, gen_loss=0.998]
19/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.297, gen_loss=1]
20/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.301, gen_loss=0.993]
21/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.304, gen_loss=1]
22/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.278, gen_loss=1.01]
23/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.283, gen_loss=1.01]
24/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.277, gen_loss=1.01]
25/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.257, gen_loss=1.01]
26/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.24, gen_loss=1.02]
27/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.238, gen_loss=1.03]
28/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.237, gen_loss=1.03]
29/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.223, gen_loss=1.04]
30/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.217, gen_loss=1.03]
31/50 epochs: 100% ██████████ 284/284 [05:46<00:00, 1.22s/it, dis_loss=0.209, gen_loss=1.03]
32/50 epochs: 100% ██████████ 284/284 [05:45<00:00, 1.22s/it, dis_loss=0.212, gen_loss=1.04]
33/50 epochs: 100% ██████████ 284/284 [05:48<00:00, 1.23s/it, dis_loss=0.214, gen_loss=1.04]
34/50 epochs: 100% ██████████ 284/284 [05:47<00:00, 1.22s/it, dis_loss=0.205, gen_loss=1.05]
35/50 epochs: 100% ██████████ 284/284 [05:48<00:00, 1.23s/it, dis_loss=0.207, gen_loss=1.05]
```

Figure 5.4.15: Progress Bar

Evaluation:

Perform FID and IS calculation

```
#fid_metric.reset()
is_metric.reset()

#fid_metric.update((fake_Y.cpu(), real_Y.cpu())) # Update FID metric with
is_metric.update(fake_X.cpu()) # Update IS metric with

# Compute scores
fid_score=calculate_frechet(real_Y, fake_X, v3model)
is_score = is_metric.compute()

fid_values.append(fid_score)
is_values.append(is_score)
```

Figure 5.4.16: FID and IS calculation

Plot the graph of loss

```
# Plot the losses
plt.plot(dis_losses, label='Discriminator Loss')
plt.plot(gen_losses, label='Generator Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('GAN Training Losses')
plt.legend()
plt.show()
```

Figure 5.4.17: Loss plotting

Plot the graph of FID and IS vs epochs

```
fig, ax1 = plt.subplots()

plt.title("Evaluation Metric During Training")

color = 'tab:red'
ax1.set_xlabel('epochs')
ax1.set_ylabel('IS', color=color)
ax1.plot(is_values, color=color)

ax2 = ax1.twinx()

color = 'tab:blue'
ax2.set_ylabel('FID', color=color)
ax2.plot(fid_values, color=color)

fig.tight_layout()
```

Figure 5.4.18: FID and IS plotting

## Plot the graph of Real vs Fake Image

```
%matplotlib inline

# Grab a batch of real images from the dataloader
real_batch = next(iter(train_dataloader))

# Plot the real images
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.axis("off")
plt.title("Real Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0][:64], padding=5, normalize=True).cpu(), (1,2,0)))

# Plot the fake images from the last epoch
plt.subplot(1,2,2)
plt.axis("off")
plt.title("Fake Images")
plt.imshow(np.transpose(vutils.make_grid(img_list[-1], padding=2, normalize=True).cpu(), (1,2,0)))
```

Figure 5.4.19: Real vs Fake images plotting

## 5.5 Implementation Issues and Challenges

The implementation of GAN training required large amount of time and high Internet speed. It is normal that the system stopped when I ran the GAN training until halfway due to the network issue, then I had to rerun the training again although it already ran 2 hours. Furthermore, if using the free CPU provided in the system, the system can run but the speed will be much slower than using the GPU, that is why I purchased the computing units to use the GPU. Every day I restart the system, the system needed to run from beginning. The first-time training of GAN is much time consuming compared to you run it again after the training had completed. That is why if I want to make a big change on the system, I will empty one-day time to just train the GAN, because it is time wasting if I chose to run the first-time training every day.

The next challenge is the GPU memory constraints. Training GANs demands significant computational resources, particularly GPU memory. Large datasets or complex network architectures exacerbate this challenge. My dataset from [18] is quite large, it comprised over 60000 images. This made the system ran slow every first time it wanted to load my dataset. For my CycleGAN, the system even encountered “run out of memory” error, due to the large batch size that I set. Large batch size means the GAN will take more images in one iterations for training and the CycleGAN involved more complex architecture than the two other GANs. As a result, I had to limit the batch size to 5, mirroring the sample provided.

The selection of evaluation metrics takes a lot of time. Initially, the three GANs are using three different kinds of calculation function. For example, DCGAN used the pre-built FID function in ignite libraries, CycleGAN used the FID function from [26], and SNGAN used the pre-built FID function in Pytorch (Not in ignite). The integration of FID and IS calculation into a GAN consumed a lot of time because I faced many problems, such as the matrix problem (the input is not as expected), the imaginery component problem (the FID value is too large) and the shape problem (input has different shape with inceptionV3 model). I spent long time to understand the reasons of these problems to integrate them into my system. Once I had successfully integrated them, my supervisor told me that the three GANs should employ the same FID and IS function so they can compare with each other. I spent another time to find a FID function that can fit all the GANs, and finally I use the FID function from [25] and it successfully fit the three GANs.

## 5.6 Conclusion Remark

In conclusion, Chapter 5 delves into the experimental setup, implementation, and challenges encountered during the execution of the project. The hardware setup involved the utilization of a laptop equipped with Google Chrome for accessing Google Colab, which provided advanced hardware accelerators for model training. Various GAN architectures, including DCGAN, CycleGAN, and SNGAN, were implemented and trained using Google Colab, leveraging its computational resources and integration with Google Drive for efficient data handling.

The software setup included the configuration of Google Colab with T4 GPU hardware accelerators to expedite model training. Data collection involved the use of an anime face dataset, while data loading and preprocessing ensured the dataset's compatibility with the PyTorch framework. Hyperparameters were carefully selected and set for all GAN architectures to ensure consistency and facilitate performance comparison.

Model architecture design and weight initialization were pivotal steps in defining the generator and discriminator networks for each GAN architecture. The training process involved iterative updates to the models based on adversarial loss calculations, aiming to enhance the GANs' ability to generate realistic anime face images.

Evaluation metrics, including FID and IS were computed to assess the performance of the trained models. Challenges such as network issues, GPU memory constraints, and selection of evaluation metrics were encountered during implementation. Despite these challenges, the experimentation and implementation process provided valuable insights into the practical aspects of training GANs for anime face generation.

# Chapter 6

## System Evaluation and Discussion

### 6.1 System Testing and Performance Matrix

#### Image Quality Evaluation

can be divided into two types:

- 1) Qualitative evaluation involves visualization of generated images to determine realism, diversity, and overall quality.
- 2) Quantitative evaluation employs metrics such as Inception Score, Frechet Inception Distance (FID) to measure image fidelity, diversity, and similarity to real images.

#### Training Stability

Evaluate the stability and convergence of the training process. Done through observing loss curves for both the generator and discriminator networks to ensure they converge to reasonable values without oscillations or divergence.

#### **Inception Score Calculation**

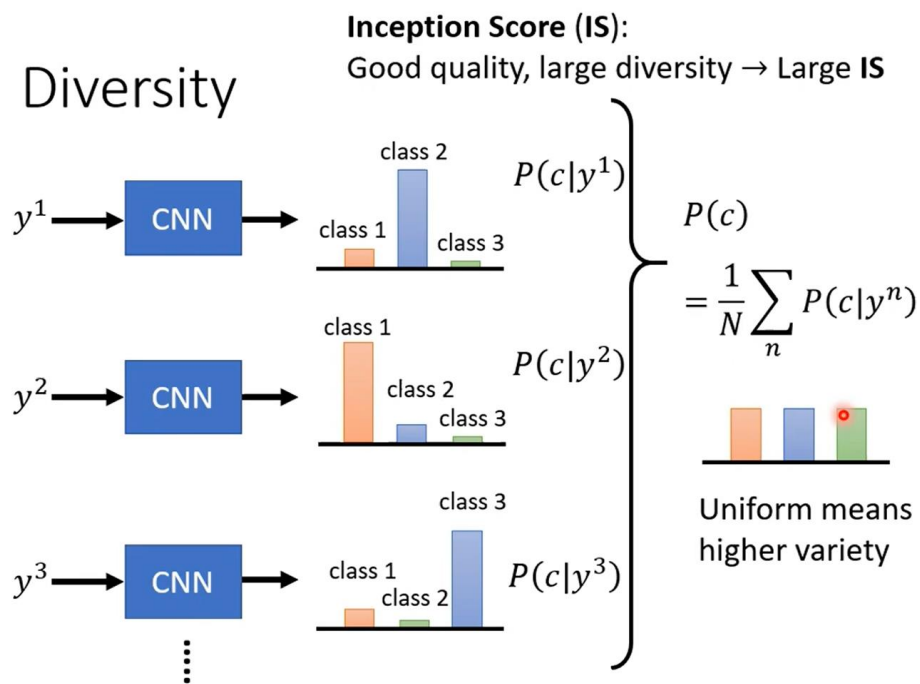


Figure 6.1.1: Inception Score working

**Generate Images:** Firstly, a set of generated images is produced by the generative model being evaluated. These images can be samples from the model's output, typically generated from random noise vectors.

**Image Classification:** Each generated image is fed into a pre-trained Inception model, which is typically InceptionV3 or a similar architecture. This model has been trained on a large dataset for image classification tasks and has learned to recognize various objects and patterns in images.

**Calculate Class Probabilities:** For each generated image, the Inception model produces a probability distribution over the classes it was trained to recognize. This distribution reflects the model's confidence in assigning the image to different categories.

**Compute Inception Score:** The Inception Score is calculated based on these class probabilities. It consists of two components:

- **Entropy:** The entropy of the class distribution for each generated image measures the diversity of predictions made by the Inception model. High entropy indicates that the model is uncertain about the class of the image, suggesting diversity.
- **Kullback-Leibler Divergence:** The Kullback-Leibler (KL) divergence between the marginal class distribution of generated images and the conditional class distribution given the entire set of generated images measures how closely the distribution of classes in the generated images matches that of the overall dataset used to train the Inception model. Low KL divergence indicates that the generated images cover a wide range of classes similar to those in the training dataset.

**Aggregate Scores:** Finally, the average entropy and KL divergence scores across all generated images are computed to obtain the overall Inception Score. Higher Inception Scores indicate that the generated images are both diverse and resemble the classes found in the training dataset.

$$IS(G) = \exp \left( E_{x \sim P_G} D_{KL}(P(y|x) || p(y)) \right)$$

Figure 6.1.2: Inception Score formulae

## Fréchet Inception Distance Calculation

# Fréchet Inception Distance (FID)

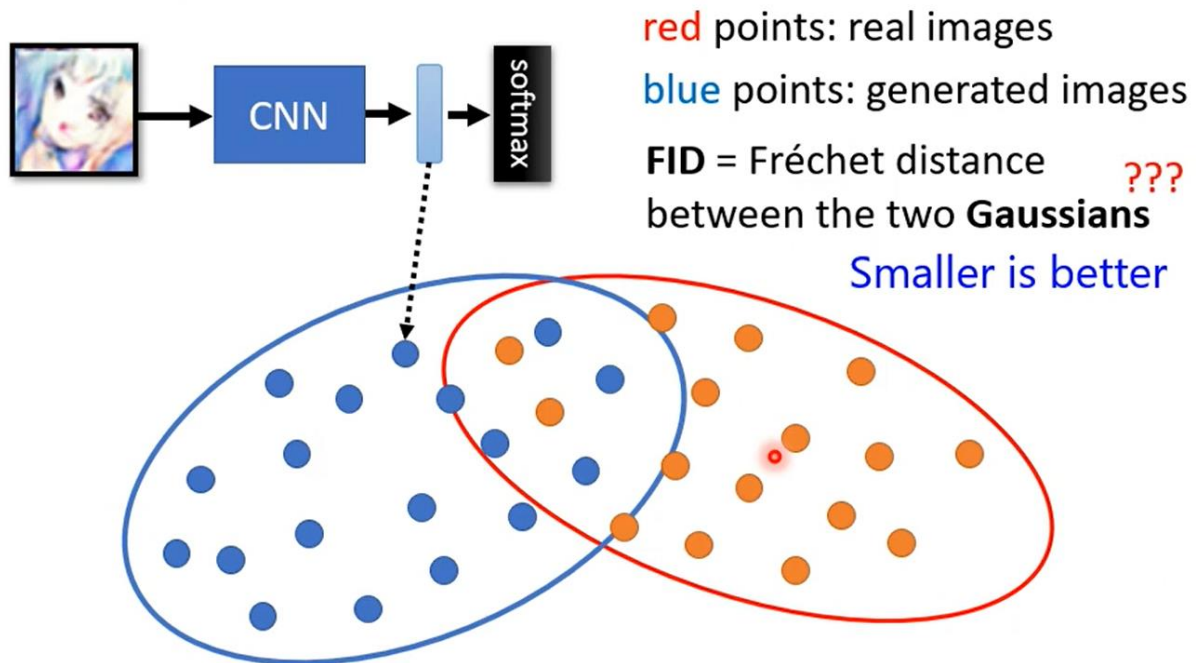


Figure 6.1.3: FID score working

**Feature Extraction:** Firstly, a pre-trained InceptionV3 neural network is used to extract features from both real and generated images. This network has been trained on a large dataset for image classification tasks and has learned to represent images in a feature space.

**Calculate Mean and Covariance:** Next, the mean ( $\mu$ ) and covariance ( $\Sigma$ ) of these feature representations are calculated for both the real and generated images. This step essentially summarizes the distribution of features in each set of images.

**Calculate Fréchet Distance:** The Fréchet distance between these two multivariate Gaussian distributions (one for real images, one for generated images) is then computed. The Fréchet distance is a measure of similarity between two probability distributions in a metric space.

$$FID = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

Figure 6.1.4: FID score formulae



Compute FID Score: Finally, the FID score is obtained by combining the mean squared difference between the means ( $\mu$ ) and the trace of the covariance matrices ( $\Sigma$ ) along with a constant factor. The lower the FID score, the better the quality and diversity of the generated images, as it indicates that the distribution of generated images is closer to that of real images. [14]

## 6.2 Testing Setup and Result

### Testing Setup

Generate Fake Images:

During the training process of the GAN, the generator network creates fake images intended to resemble the real images in the dataset.

Append Fake Images into Image Array:

As the fake images are generated, they are appended into an image array.

Plot Comparison Graph:

Using visualization tools like matplotlib in Python, plot a graph to visually compare the real and fake images. Each image is represented in the graph, allowing for a side-by-side comparison of their visual characteristics.

Quantitative analysis:

Downloading the Ignite Library:

The Ignite library is a high-level library for training neural networks in PyTorch. By downloading and importing this library, we gain access to various utilities and functions that streamline the training and evaluation process.

Importing Functions:

Import the InceptionScore function from Ignite.

Defining FID Function and InceptionV3 Model:

To calculate the FID, define a function that computes the distance between the feature representations of real and generated images using the InceptionV3 model, a pre-trained deep learning model.

Calculating FID and Inception Score: Use the defined FID function and the InceptionScore function imported from Ignite to calculate the FID and Inception Score, respectively.

### Calculating Generator and Discriminator Loss:

To further evaluate the performance of the GAN, we calculate the generator loss and discriminator loss. For DCGAN and SNGAN, these are the primary losses used to optimize the generator and discriminator networks. For CycleGAN, two additional losses are considered: cycle consistency loss and GAN loss. These losses provide insights into how well the GAN is able to learn and generate images that match the desired distribution.

Appending Values and Losses: The calculated FID and Inception Score values, as well as the generator and discriminator losses, are appended into respective arrays (fid\_value, is\_value, and loss). This allows for easy tracking and visualization of these metrics over the course of training.

### Result Analysis

#### Qualitative Analysis

#### DCGAN:



Figure 6.2.1: Real vs Fake images in DCGAN

SNGAN:

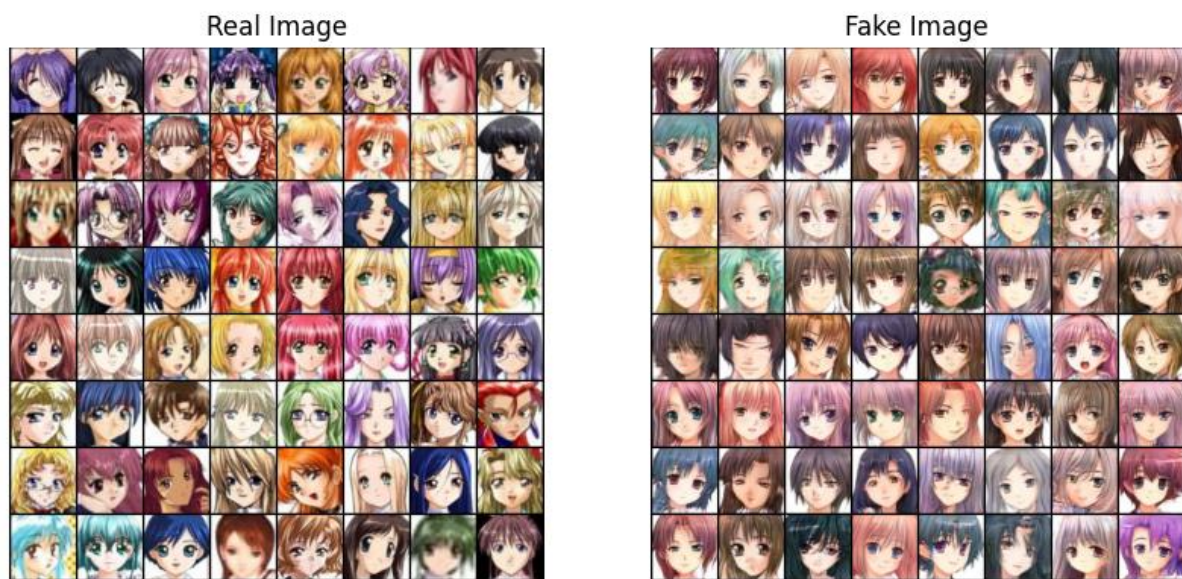


Figure 6.2.2: Real vs Fake images in SNGAN

Comparison:

When comparing DCGAN and SNGAN, it becomes evident that SNGAN consistently generates higher-quality images compared to DCGAN. In SNGAN, most images depict complete facial structures, with well-positioned and proportionate eyes. Although occasional instances of characters with closed eyes are observed in SNGAN, the eyes maintain uniform size and placement. Conversely, DCGAN-generated images exhibit inconsistencies, such as characters without eyes or eyes of varying sizes and positions, resulting in a somewhat unsettling appearance. Both GANs showcase a variety of hair colors, but DCGAN tends to produce a wider range of eye sizes, including large, medium, and sometimes no eyes, while SNGAN predominantly generates images with medium-sized and closed eyes.

CycleGAN:



Figure 6.2.3: Real vs Fake images (X to Y, Y to X) in CycleGAN

For cycleGAN, domain X is for various hair color and eye color, whereas domain Y is for blue hair and blue eye. In the generated Images X, it can see most characters' eyes had translated into blue color. For colors that are brighter (yellow) or closed to blue (purple), they are easier to translate to blue. The brown color hair character after train for 50 epochs still retains much of their original hair color. For translate domain Y to X, the hair color mostly translates to those colors closed to blue, but most eye color translate successfully to brown or yellow.

However, due to memory limitations resulting in a small batch size of 5, CycleGAN struggles to produce images with high diversity. Consequently, in translations from domain Y to X, most generated images feature blue hair rather than other colors like yellow or brown. Increasing the number of images used for training could potentially address this limitation and enable the generation of a wider range of hair colors.

### Quantitative Analysis

DCGAN:

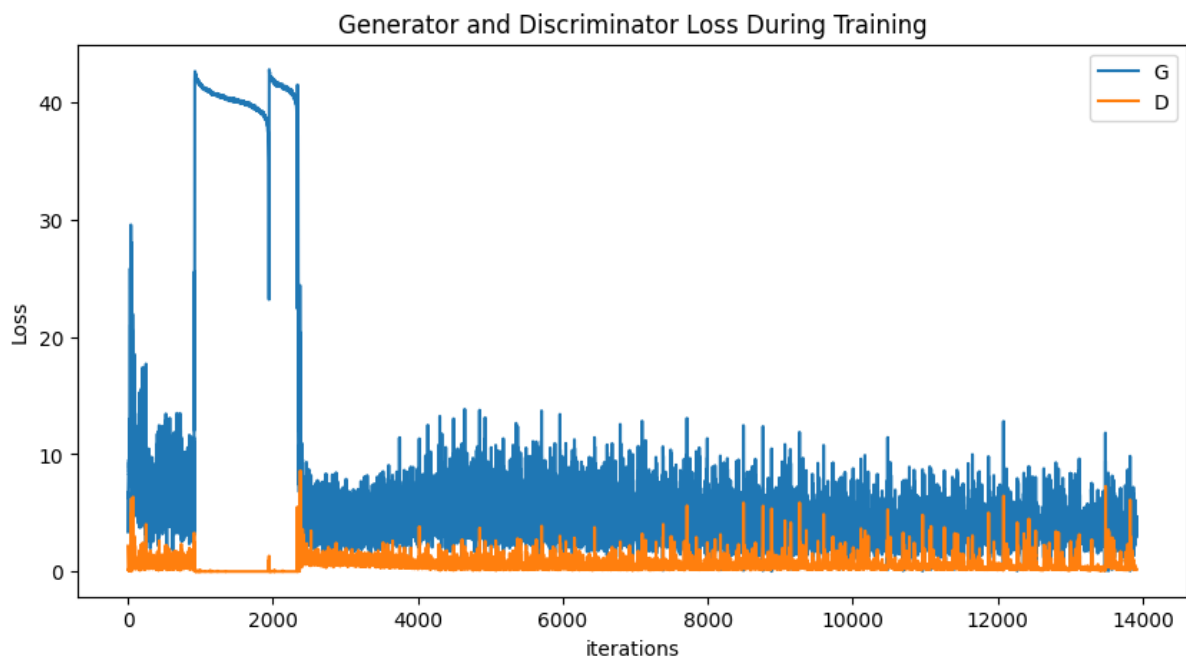


Figure 6.2.4: Loss in DCGAN

Generator Loss:

Initially, the generator loss is high as the generator struggles to produce meaningful images that can fool the discriminator.

As training progresses, the generator loss decreases. This indicates that the generator is improving its ability to generate realistic images that are more difficult for the discriminator to distinguish from real images. Towards convergence, the generator loss may stabilize at a certain level. This suggests that the generator has reached a relatively optimal state where it produces realistic images consistently.

#### Discriminator Loss:

Initially, the discriminator loss is high as it learns to distinguish between real and fake images. As training progresses, the discriminator loss decreases. This indicates that the discriminator becomes better at distinguishing between real and fake images. Towards convergence, the discriminator loss may stabilize at a certain level. This suggests that the discriminator has reached a relatively optimal state where it can effectively differentiate between real and fake images.

#### SNGAN:

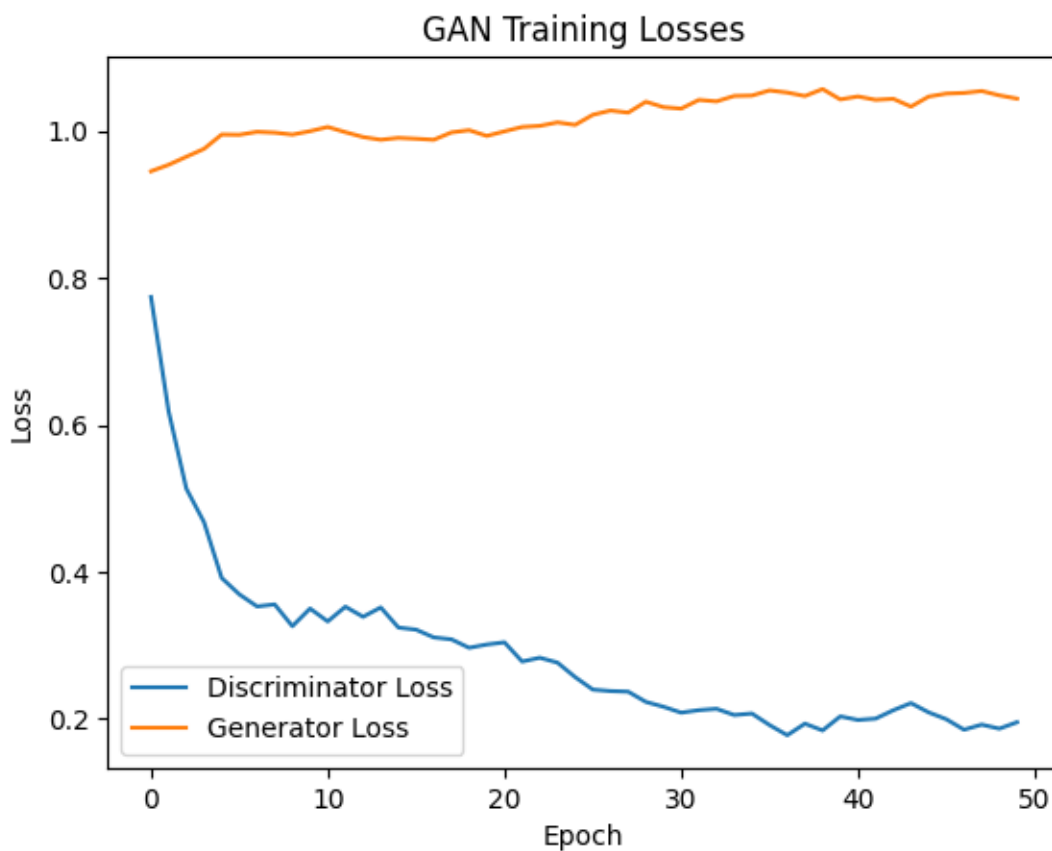


Figure 6.2.5: Loss in SNGAN

### Generator Loss:

Towards the end of training, the generator loss kept increasing. This increase in the generator loss could be due to several reasons:

- 1) Mode Collapse: The generator start to collapse, meaning it produces limited varieties of images, or even repeats the same image. This results in the discriminator becoming more capable of distinguishing generated images from real ones, causing an increase in the loss.
- 2) Difficulty in Learning: As the training progresses, it becomes increasingly challenging for the generator to improve its performance further. It struggle to generate more diverse and realistic images, leading to an increase in the loss.
- 3) Competition with Discriminator: The generator-loss increase could also be a result of the discriminator becoming more effective at distinguishing real from fake images, putting pressure on the generator to produce better outputs.

Combine with the IS score result, it can be concluded that the SNGAN faced the problem of mode collapse.

### Discriminator Loss:

Initially, the discriminator loss is high as it learns to distinguish between real and fake images. As training progresses, the discriminator loss decreases. This indicates that the discriminator becomes better at distinguishing between real and fake images. Towards convergence, the discriminator loss stabilizes at a certain level. This suggests that the discriminator has reached a relatively optimal state where it can effectively differentiate between real and fake images.

## CycleGAN:

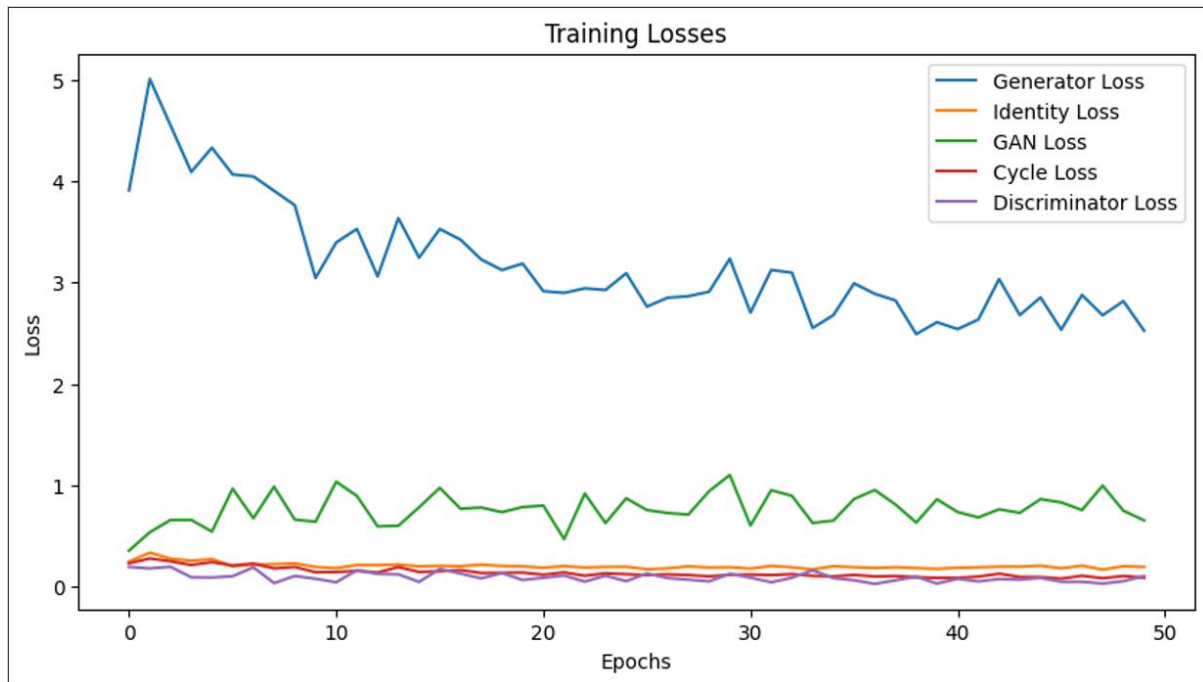


Figure 6.2.6: Loss in CycleGAN

### Generator Loss:

Initially high and then gradually decreases over epochs: At the beginning of training, the generator loss is high as the model learns to translate images between domains effectively. As training progresses, the generator loss tends to decrease as the model improves its translation capability and converges towards generating realistic images. Throughout training, fluctuations occur in the generator loss due to changes in the discriminator's performance.

### Discriminator Loss:

In the early stages, the discriminator quickly learns to distinguish between real and fake images, resulting in low loss values. However, as the generators improve, the discriminator faces a more challenging task, leading to fluctuations in its loss. Over time, the discriminator loss stabilizes as both the generator and discriminator reach a balanced state, where the generator produces realistic images that are difficult for the discriminator to distinguish from real ones.



### Cycle Loss:

The cycle loss measures how effectively the generators can reconstruct the original input image after translation. In the graph, the cycle loss remain at a low level, indicates good reconstruction performance.

### GAN Loss:

GAN loss measures the how effectively the CycleGAN model successfully learns to translate images between domains while producing realistic-looking outputs. Fluctuation occurs due to complexity of the image translation task.

### Comparison:

Comparing these GANs, DCGAN and SNGAN share similarities in their generator and discriminator loss behaviors, with DCGAN exhibiting more stable training dynamics. SNGAN faces challenges like mode collapse, leading to an increase in the generator loss towards the end of training. CycleGAN, on the other hand, focuses on image translation between domains, with fluctuations in both generator and discriminator losses, along with specific losses like cycle loss and GAN loss, indicating the translation quality and realism of generated images. The training of CycleGAN is stable due to the small batch size.

### DCGAN:

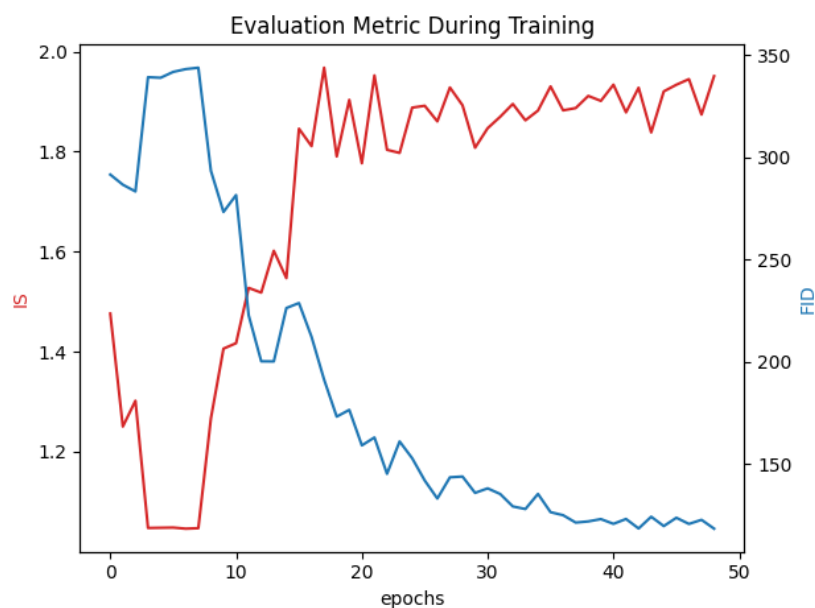


Figure 6.2.7: FID and IS in DCGAN

The FID values fluctuate within a certain range, suggesting that the model encounter variations in performance during training. Towards the later epochs, the FID values tend to stabilize and decrease at a slower rate, indicating that the model is converging to a more stable state.

Similar to FID, the IS values fluctuate within a certain range, indicating variations in image quality and diversity during training. Towards the later epochs, the IS values stabilize and increase at a slower rate, indicating that the model is producing more consistent and diverse images.

SNGAN:

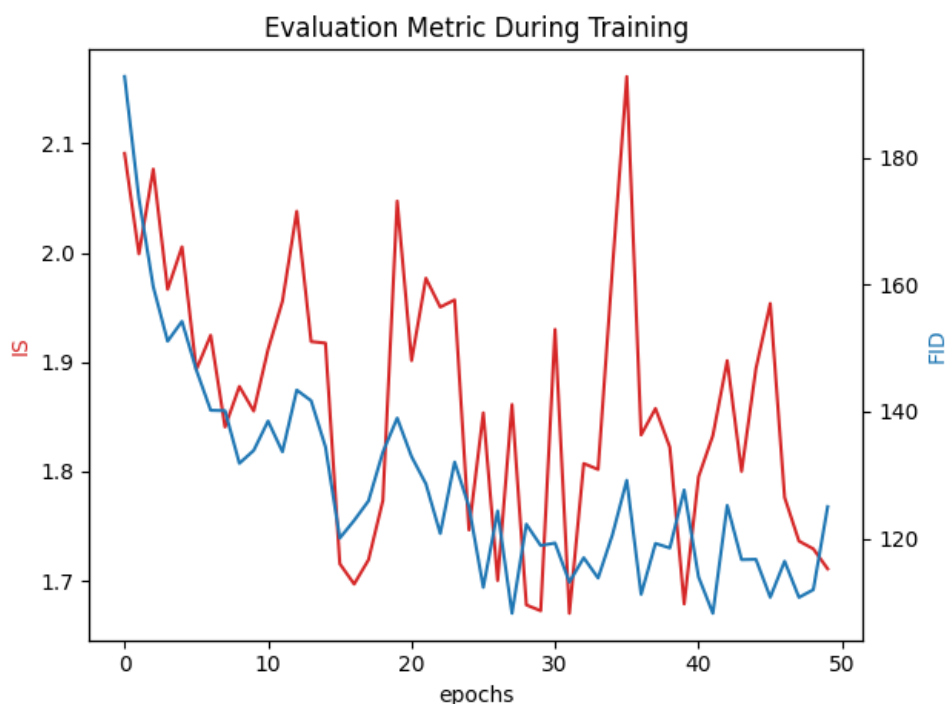


Figure 6.2.8: FID and IS in SNGAN

The FID score decreases initially and then fluctuates around a certain range. In the provided data, the FID score starts at a relatively high value and gradually decreases over the first few epochs. Afterward, it seems to oscillate around a certain value without a clear decreasing trend. This oscillation indicate that the model's performance stabilizes or converges to a certain level, and further training iterations do not lead to significant improvements in the FID score.

The IS score decreases initially and then fluctuates around a certain range. This look weird because the IS pattern should be from low to high, but the graph shown a different result. This

unexpected changes in the IS score pattern occur due to factors such as mode collapse, where the generator fails to produce diverse images, or changes in training dynamics. Combine with the previous result, it can be concluded that the SNGAN faced the problem of mode collapse. From the generated fake images, it is obvious that more pink color hair characters are produced and this reduced the diversity and thus reduced the IS score.

Comparing between DCGAN and SNGAN:

Through comparison, we can conclude that both GANs are producing high quality images than beginning, because the FID score both decreased. DCGAN FID range from 343.735949 to 118.587097 while SNGAN changed from 192.845002 to 108.192419. However, through comparing the FID range, the SNGAN produce image with relatively low range of FID compared to DCGAN, indicating it is more stable to produce high quality fake image.

Through comparing the IS score, although the DCGAN produce high diverse image compared to beginning, SNGAN encounter mode collapse so the IS score at the end is lower than beginning, but we see that the IS score of DCGAN is ranged from 1.046320 to 1.967778 while the SNGAN range from 1.670244 to 2.16116. This indicated the SNGAN is more consistent and stable than DCGAN to produce image with high IS score although it encounters mode collapse.

CycleGAN:

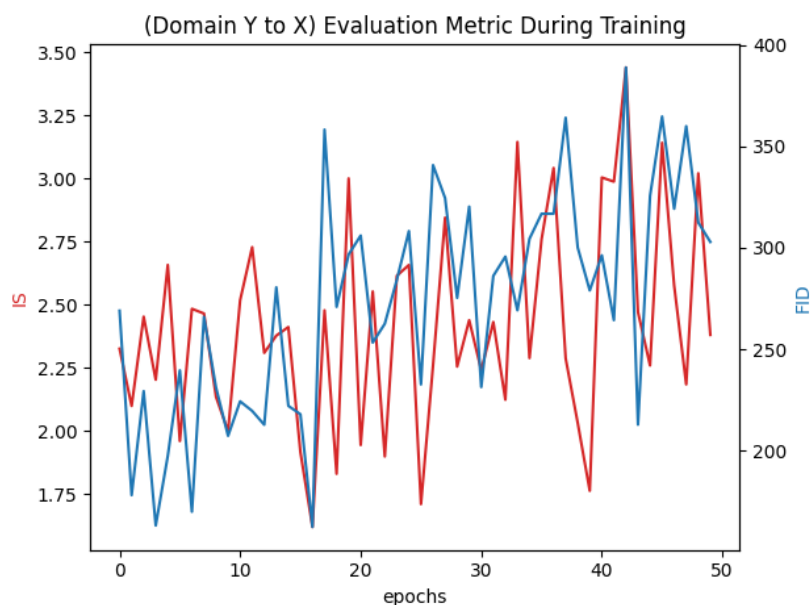


Figure 6.2.9: FID and IS in CycleGAN (Y to X)

For the FID score, the FID score increased from approximately 269.0384 to 302.9035, meaning that the quality of images decreased.

For IS score, although it encountered fluctuation in the middle, but the final IS score is almost same with that in the beginning but increase a bit (from 2.3254 to 2.3800), meaning that the diversity has no big changes. From the display fake images, the hair color of characters mostly remains the original color, but there are still some changes in the eye color (from blue to yellow), causing the a bit increasing in IS score.

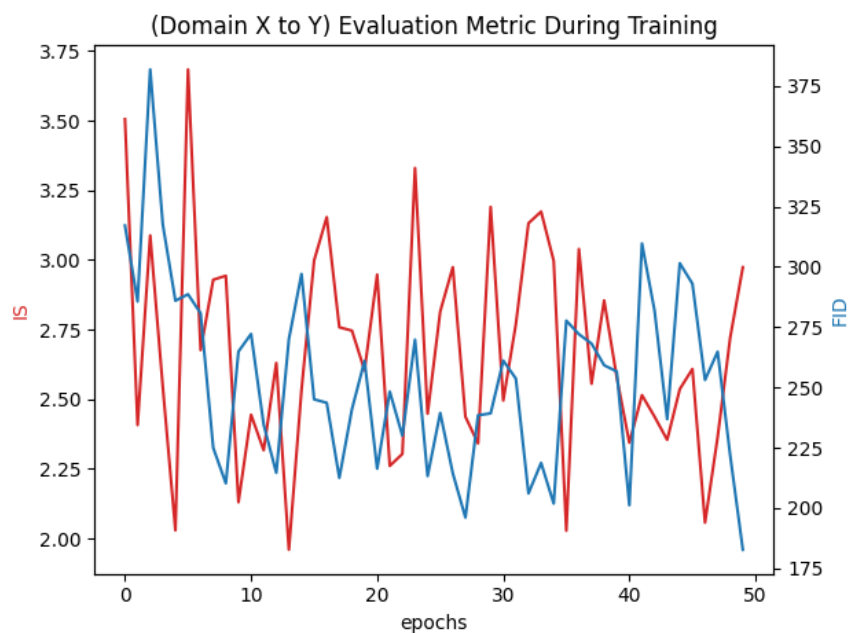


Figure 6.2.10: FID and IS in CycleGAN (X to Y)

The FID value initially high when the generator's performance is poor and then stabilize finally decrease, indicating better image quality.

For IS score, it is initially 3.50 and finally 3.00. This reduction is due to the characters are translated to all blue hair and blue eyes, thus decreased the diversity.

### 6.3 Project Challenges

GAN is a relatively new technology that was first introduced in 2017. When I first began studying GANs, I found myself lacking in understanding due to the complexity of the vocabulary used in research papers. Despite my efforts to comprehend the material, I struggled to grasp certain concepts. However, I was fortunate to come across a 10-hour lecture on YouTube that covered the basics of GANs. This resource proved to be immensely helpful because the teaching materials included clear explanations accompanied by visual aids. Unlike reading research papers, which often presented challenges in visualization, this lecture provided a more accessible way for beginners like myself to comprehend the workings of GANs. The combination of verbal explanations and visual illustrations made it easier for me to grasp complex concepts compared to relying solely on written text. Overall, I found this method to be highly effective in bridging the gap between my limited understanding and the advanced concepts of GAN technology.

The next challenge arose when deciding between SNGAN and StarGAN for my research. Initially, I was inclined towards StarGAN due to its unique ability to perform various feature translations on a single image, unlike DCGAN and CycleGAN, which are focused on specific types of image generation and domain-to-domain translation, respectively. However, as I delved deeper into my decision-making process, I realized that SNGAN, an enhanced version of DCGAN incorporating spectral normalization for improved stability during training, might be a more suitable choice. Despite this realization, I faced a significant obstacle: the scarcity of labeled anime datasets available online. These datasets would have been crucial for training and evaluating models based on attributes such as hair color, eye color, facial expression, and etc. Despite extensive searching on platforms like Kaggle, I was unable to find a suitable labeled anime dataset. Moreover, creating such a dataset from scratch would have been extremely time-consuming, especially given the vast number of images required for a comprehensive comparison with other GAN architectures. This scarcity of labeled data ultimately influenced my decision to pivot from StarGAN to SNGAN.

## 6.4 Objective Evaluation

Understanding the Working Principles of Generative Adversarial Networks (GANs):

The project begins with an in-depth literature review. This involves studying research papers, lecture video, and tutorials on GANs to grasp the theoretical foundations, architecture, training dynamics, and loss functions of GANs. Additionally, experiment is conducted by implementing various GAN architectures, which are DCGAN, CycleGAN, and SNGAN, using Python and deep learning frameworks like PyTorch. Through this practical exploration and experimentation, the project gains insights into the core components and mechanisms of GANs, facilitating a comprehensive understanding of their working principles.

Performance Analysis of GANs in Anime Face Generation:

The project collects and preprocesses anime face datasets suitable for training GAN models. Multiple GAN architectures are then trained on these datasets, and their performance is evaluated using qualitative analysis and quantitative analysis (Inception Score and FID). By analyzing the loss, quality, and diversity of the generated anime face images, the project assesses the performance of different GAN architectures in anime face generation tasks.

Comparative Analysis of Generative Models:

The project conducts a systematic comparison of between DCGAN, CycleGAN, and SNGAN. Although many hyperparameters of the three GANs are set to be same for comparisons but due to the out of memory issue in CycleGAN, it can only take in 5 for its batch size, so most comparisons only done between DCGAN and SNGAN. Each architecture is evaluated based on metrics such as FID, IS and loss. By identifying the strengths and weaknesses of each GAN model, the project provides insights into their suitability for specific anime face generation applications.

## 6.5 Concluding Remark

In conclusion, Chapter 6 presents the system evaluation and remarks on the project's objectives. The evaluation encompasses qualitative and quantitative analyses of the generated images, training stability, and performance metrics. Additionally, challenges encountered during the project, including learning curve hurdles and model selection dilemmas, are discussed.

The testing setup and results provide a comprehensive assessment of the performance of DCGAN, CycleGAN, and SNGAN in anime face generation tasks. Qualitative analysis reveals visual differences in image quality and diversity among the three architectures, while quantitative metrics offer insights into their quality and similarity to real images.

In qualitative analysis, SNGAN outperforms DCGAN in generating high-quality images with more consistent facial features, while CycleGAN demonstrates successful translation between image domains despite limitations in batch size affecting diversity. SNGAN produces images with better facial structures and eye consistency compared to DCGAN, while CycleGAN effectively translates between hair and eye color domains, albeit with constraints on diversity due to memory limitations. These findings underscore the strengths and limitations of each GAN architecture in generating diverse and realistic images.

In quantitative analysis, DCGAN demonstrates stable training with decreasing losses over epochs, while SNGAN encounters mode collapse, leading to an increase in generator loss. CycleGAN exhibits fluctuations in losses due to its focus on image translation between domains, with stable training facilitated by a small batch size.

The FID and IS calculation concluded DCGAN and SNGAN both show improvements in image quality, but SNGAN demonstrates greater stability and consistency in generating high-quality fake images despite encountering mode collapse. For translation from domain X to Y, CycleGAN exhibits increase in image quality but with a reduction in diversity towards the end due to the blue hair and eye. For translation from domain Y to X, the image quality decreased but the diversity increased a bit although hair color almost same as real image, but eye color had changed.

Challenges such as the complexity of GAN technology and the selection of appropriate architectures underscore the learning curve and decision-making processes involved in GAN research. Overcoming these challenges required a combination of theoretical understanding, practical experimentation, and problem-solving skills.

The project's objectives are achieved through a systematic exploration of GAN principles, performance analysis in anime face generation, and comparative evaluation of different generative models. By fulfilling these objectives, the project contributes to the understanding of GANs and their applications in generating anime face images.



# Chapter 7

## 7.1 Conclusion

In conclusion, this project embarked on a comprehensive exploration of Generative Adversarial Networks (GANs) in the context of anime face generation. Through a meticulous journey encompassing theoretical study, practical implementation, and systematic evaluation, significant insights into the workings and performance of different GAN architectures were gleaned.

The project began with a thorough review of GAN literature, delving into the foundational concepts, architecture designs, and training dynamics. This theoretical groundwork provided the necessary foundation for the subsequent phases of the project. Practical implementation involved the construction and training of three distinct GAN architectures: DCGAN, CycleGAN, and SNGAN. Leveraging powerful computational resources and advanced software tools like Google Colab, the project was able to execute complex training processes efficiently.

Central to the project's objectives was the evaluation of GAN performance in anime face generation tasks. This evaluation was multifaceted, encompassing both qualitative and quantitative analyses. Qualitative assessment involved visual inspection of generated images to discern aspects of realism, diversity, and overall quality. Quantitative evaluation relied on established metrics IS and FID to provide objective measures of image quality and similarity to real images.

The qualitative analysis highlights SNGAN's superiority in generating high-quality images with consistent facial features compared to DCGAN, while CycleGAN excels in domain translation despite diversity limitations. Quantitatively, DCGAN shows stable training with decreasing losses, contrasting SNGAN's mode collapse issue causing increase generator loss and CycleGAN's fluctuating losses. Despite encountering mode collapse, SNGAN demonstrates greater stability in generating high-quality images. Although DCGAN can generate high quality and diversity fake images, its stability is lower than SNGAN. Additionally, CycleGAN achieves successful domain translation, albeit with changes in

image diversity. These insights underscore the strengths and limitations of each GAN architecture in image generation and translation tasks.

Challenges encountered along the way, including network issues, lacking knowledge, model and evaluation selection dilemmas, and computational constraints. Most of them were addressed through a combination of perseverance, problem-solving, and adaptation.

Through these endeavors, the project not only achieved its objectives but also contributed to the broader understanding of GAN technology and its applications in generating anime face images. By bridging the gap between theory and practice, this project lays a solid foundation for future research and innovation in the field of generative modeling.

## 7.2 Recommendation

### Dataset Expansion and Diversity:

Consider expanding the dataset to include a broader range of anime face images with labeled attributes. Increasing dataset diversity enhances the training process and enables more comprehensive evaluations of image quality, diversity, and realism across different GAN architectures. By incorporating a more diverse dataset, GANs can capture a wider range of facial features, expressions, and styles, leading to more realistic and varied generated images. With labeled attribute, GANs that required labeled dataset such as StarGAN can be implemented to analyze its performance and compare with other GANs.

### Memory Optimization for CycleGAN:

Explore memory optimization techniques to overcome the limitations imposed by batch size in CycleGAN training. Techniques such as gradient accumulation or data augmentation can help alleviate memory constraints and allow for larger batch sizes without compromising model performance. By optimizing memory usage, more efficient training can be facilitated and improve the overall effectiveness of CycleGAN in generating high-quality images.

### Optimization of Training Parameters:

Experiment with various training parameters such as learning rates, batch sizes, and optimization algorithms to enhance the performance of GAN models. Fine-tuning these parameters can help mitigate issues like mode collapse in SNGAN and improve the stability of CycleGAN training. By systematically adjusting these parameters, the training process can be optimized to achieve better convergence and image quality.

## REFERENCES

- [1] S. Ruan, "Anime Characters Generation with Generative Adversarial Networks," 2022, [Online]. Available: <https://ieeexplore-ieee-org.libezp2.utar.edu.my/document/9918869>
- [2] D. E. V, V. V. H. Kumar, R. D. Kumar and V. M. Sahithi, "Generation of Hilarious Animated Characters using GAN," 2023, [Online]. Available: <https://ieeexplore-ieee-org.libezp2.utar.edu.my/document/10125904>
- [3] C. Xiaopeng, C. Jiangzhong, L. Yuqin and D. Qingyun, "Improved Training of Spectral Normalization Generative Adversarial Networks," 2020, [Online]. Available: <https://ieeexplore-ieee-org.libezp2.utar.edu.my/document/9143310>
- [4] Z. Li and Q. Wan, "Generating Anime Characters and Experimental Analysis Based on DCGAN Model," [Online]. Available: <https://ieeexplore-ieee-org.libezp2.utar.edu.my/stamp/stamp.jsp?tp=&arnumber=9708652>
- [5] Madhu Sanjeevi "Ch:14.1 Types of GAN's with Math.," 2019, [Online]. Available: <https://medium.com/deep-math-machine-learning-ai/ch-14-1-types-of-gans-with-math-5b0dbc1a491d>
- [6] L. Quan and H. Zhang, "Facial Animation Using CycleGAN," [Online]. Available: <https://ieeexplore-ieee-org.libezp2.utar.edu.my/stamp/stamp.jsp?tp=&arnumber=9591087>
- [7] “【機器學習 2021】生成式對抗網路 (Generative Adversarial Network, GAN) (二) – 理論介紹與 WGAN,” [Online]. Available: [www.youtube.com](http://www.youtube.com).  
<https://youtu.be/jNY1WBb8l4U?feature=shared>
- [8] “Deconvolution vs Convolutions,” *GeeksforGeeks*, [Online]. Available: <https://www.geeksforgeeks.org/deconvolution-vs-convolutions/> (accessed Apr. 24, 2024).

[9] M. Nayak, "Deep Convolutional Generative Adversarial Networks(DCGANs)," *Medium*, 2018. [Online]. Available: <https://medium.datadriveninvestor.com/deep-convolutional-generative-adversarial-networks-dcgans-3176238b5a3d>

[10] Kaixin. "Data-Augmented Manifold Learning Thermography for Defect Detection and Evaluation of Polymer Composites", 2022, [Online]. Available: [https://www.researchgate.net/figure/SNGAN-data-augmentation-architecture-for-thermal-image-generation\\_fig1\\_366687713](https://www.researchgate.net/figure/SNGAN-data-augmentation-architecture-for-thermal-image-generation_fig1_366687713)

[11] Parmar. "Effectiveness of Cross-Domain Architectures for Whisper-to-Normal Speech Conversion." 2019, [Online]. Available: [https://www.researchgate.net/publication/335840935\\_Effectiveness\\_of\\_Cross-Domain\\_Architectures\\_for\\_Whisper-to-Normal\\_Speech\\_Conversion](https://www.researchgate.net/publication/335840935_Effectiveness_of_Cross-Domain_Architectures_for_Whisper-to-Normal_Speech_Conversion)

[12] "Google Colaboratory,"[Online]. Available: <https://colab.research.google.com/github/pytorch-ignite/pytorch-ignite.ai/blob/gh-pages/blog/2021-08-11-GAN-evaluation-using-FID-and-IS.ipynb>

[13] Cheng, "An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset," 2020, [Online]. Available: [https://www.researchgate.net/figure/Significant-components-of-DCGAN-model\\_fig3\\_338979046](https://www.researchgate.net/figure/Significant-components-of-DCGAN-model_fig3_338979046)

[14] R. Gupta and V. Gupta, "Performance Analysis of Different GAN Models: DC-GAN and LS-GAN,"[Online]. Available: <https://ieeexplore.ieee.org/document/9673478>

[15] “【Tensorflow2.0 深度学习】从 GAN 到 DCGAN、cGAN、ACGAN、InfoGAN：原理全解析与 Tensorflow 代码实践 {14} 附：1 8 版本的 Tensorflow 学习基础,” [Online]. Available: [www.youtube.com](http://www.youtube.com). <https://youtu.be/EXK5djTw8J8?feature=shared>

[16] H. Sultan, “Multi-Classification of Brain Tumor Images Using Deep Neural Network.” 2019, [Online]. Available: [https://www.researchgate.net/figure/ReLU-activation-function\\_fig7\\_333411007](https://www.researchgate.net/figure/ReLU-activation-function_fig7_333411007)

[17] “Papers with Code - Tanh Activation Explained,” [Online]. Available: <https://paperswithcode.com/method/tanh-activation>

[18] S. Churchill, “Anime face dataset,” Kaggle, 2019, [Online]. Available: <https://www.kaggle.com/datasets/splcher/animefacedataset>

[19] “Google Colaboratory,” [Online]. Available: <https://colab.research.google.com/github/nivedwho/Colab/blob/main/CycleGAN.ipynb>

[20] S. Singh, “Leaky ReLU as an Activation Function in Neural Networks”, 2020, [Online]. Available: <https://deeplearninguniversity.com/leaky-relu-as-an-activation-function-in-neural-networks/>

[21] “Sigmoid Function: Types and Applications | BotPenguin,” [Online]. Available: <https://botpenguin.com/glossary/sigmoid-function>.

[22] “CoCalc -- SNGAN.ipynb,” [Online]. Available: [https://cocalc.com/github/y33-j3T/Coursera-Deep-Learning/blob/master/Build%20Basic%20Generative%20Adversarial%20Networks%20\(GANs\)/Week%203%20-%20Wasserstein%20GANs%20with%20Gradient%20Penalty/SNGAN.ipynb](https://cocalc.com/github/y33-j3T/Coursera-Deep-Learning/blob/master/Build%20Basic%20Generative%20Adversarial%20Networks%20(GANs)/Week%203%20-%20Wasserstein%20GANs%20with%20Gradient%20Penalty/SNGAN.ipynb)

[23] “Claim Your Anime Waifu with SNGAN,” [Online]. Available: <https://www.kaggle.com/code/tqch2020/claim-your-anime-waifu-with-sngan>

[24] “Re:Zero Rem Anime Faces For GAN Training,” [Online]. Available: <https://www.kaggle.com/datasets/andy8744/rezero-rem-anime-faces-for-gan-training>

[25] “GAN in Pytorch with FID,” [Online]. Available: <https://www.kaggle.com/code/ibtesama/gan-in-pytorch-with-fid>

[26] J. Brownlee, “How to Implement the Frechet Inception Distance (FID) for Evaluating GANs,” 2019, [Online]. Available: <https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/>

[27] Marin, “The Effect of Latent Space Dimension on the Quality of Synthesized Human Face Images.” 2021, [Online]. Available: [https://www.researchgate.net/publication/351832396\\_The\\_Effect\\_of\\_Latent\\_Space\\_Dimension\\_on\\_the\\_Quality\\_of\\_Synthesized\\_Human\\_Face\\_Images](https://www.researchgate.net/publication/351832396_The_Effect_of_Latent_Space_Dimension_on_the_Quality_of_Synthesized_Human_Face_Images)

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

|  |                          |
|--|--------------------------|
| <b>Trimester, Year: January, 2024</b>  | <b>Study week no.: 4</b> |
| <b>Student Name &amp; ID: Tan Jia Ler 2003397</b>  |                          |
| <b>Supervisor: Ts Dr Tong Dong Ling</b>  |                          |
| <b>Project Title: Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images</b> |                          |

## 1. WORK DONE

- Met supervisor for the 1st time for this semester during w4
- briefing on task to be done (exploring StarGAN)
- Reviewed on the project objectives and results of FYP1.

## 2. WORK TO BE DONE

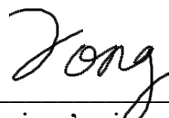
- Explore StarGAN codes.
- Adjust the hyperparameter in CycleGAN and DCGAN to be the same. Also apply to StarGAN.

## 3. PROBLEMS ENCOUNTERED

-

## 4. SELF EVALUATION OF THE PROGRESS

**Good**



Supervisor's signature



Student's signature



# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

|  |                          |
|--|--------------------------|
| <b>Trimester, Year: January, 2024</b>  | <b>Study week no.: 6</b> |
| <b>Student Name &amp; ID: Tan Jia Ler 2003397</b>  |                          |
| <b>Supervisor: Ts Dr Tong Dong Ling</b>  |                          |
| <b>Project Title: Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images</b> |                          |

## 1. WORK DONE

- Met supervisor for the 2<sup>nd</sup> time for this semester during w6
- Listened to supervisor's explanation on defragmentation.
- Failed to run StarGAN (lack of labeled dataset), find another GAN which is SNGAN to continue the project.
- Successfully ran the SNGAN.

## 2. WORK TO BE DONE

- try to integrate FID and IS calculation into SNGAN.

## 3. PROBLEMS ENCOUNTERED

- Lack of labeled dataset in StarGAN
- Dilemma in selecting suitable SNGAN sample code

## 4. SELF EVALUATION OF THE PROGRESS

**Good**



\_\_\_\_\_  
Supervisor's signature



\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

|   |                   |
|---|-------------------|
| Trimester, Year: January, 2024  | Study week no.: 8 |
| Student Name & ID: Tan Jia Ler 2003397  |                   |
| Supervisor: Ts Dr Tong Dong Ling  |                   |
| Project Title: Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images |                   |

## 1. WORK DONE

- Met supervisor for the 3<sup>rd</sup> time for this semester during w8
- Successfully integrated FID and IS calculation into SNGAN.
- Listened to supervisor's advice to improve the knowledge related to GANs, and changing all GANs to have same FID and IS calculation.

## 2. WORK TO BE DONE


- Read more paper related to GANs.
- Apply same FID and IS calculation to all the three GANs.


## 3. PROBLEMS ENCOUNTERED

- Problems in integrating evaluation metrics in SNGAN.

## 4. SELF EVALUATION OF THE PROGRESS

Not Good

  
\_\_\_\_\_  
Supervisor's signature

  
\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

|   |                    |
|---|--------------------|
| Trimester, Year: January, 2024  | Study week no.: 10 |
| Student Name & ID: Tan Jia Ler 2003397  |                    |
| Supervisor: Ts Dr Tong Dong Ling  |                    |
| Project Title: Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images |                    |

## 1. WORK DONE

- Met supervisor for the 4<sup>th</sup> time for this semester during w10
- Successfully changed FID and IS function in all GANs into the same one.
- Discussed the results with supervisors.
- Listened to a lecture on YouTube, learnt a lot of knowledges related to GANs.

## 2. WORK TO BE DONE

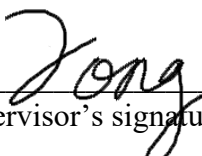
- further improve the GANs according to supervisor's advice.
- start to write the FYP report
- think explanation to explain the results


## 3. PROBLEMS ENCOUNTERED

- Unable to explain some of the results
- Mistake in displaying the results.

## 4. SELF EVALUATION OF THE PROGRESS

Good

  
\_\_\_\_\_  
Supervisor's signature

  
\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

|  |                           |
|--|---------------------------|
| <b>Trimester, Year: January, 2024</b>  | <b>Study week no.: 12</b> |
| <b>Student Name &amp; ID: Tan Jia Ler 2003397</b>  |                           |
| <b>Supervisor: Ts Dr Tong Dong Ling</b>  |                           |
| <b>Project Title: Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character Images</b> |                           |

## 1. WORK DONE

- Fixed the mistake in the GANs.
- Explored some resources aiding in writing FYP report.

## 2. WORK TO BE DONE


- Write FYP report
- Prepare the presentation slides.

## 3. PROBLEMS ENCOUNTERED

-

## 4. SELF EVALUATION OF THE PROGRESS

Good



Supervisor's signature



Student's signature

# POSTER

## Performance Comparison between GAN Variants in Generating Comic Character Images



### A. Introduction

Exploring Deep Learning for Anime Character Generation using GAN

### B. Objectives

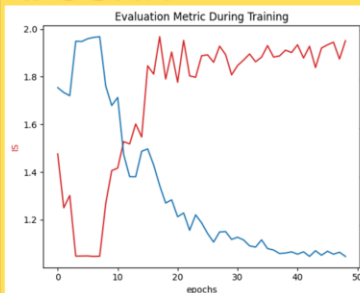
- Understanding the Working Principles of GANs
- Performance Analysis of GANs in Anime Face Generation.
- Comparison Analysis of Generative Models

### C. System Steps

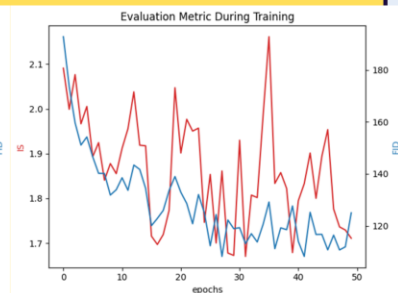
- Import Library
- Connect to Google Drive
- Provide input parameter
- Data Loading
- Image Preprocessing
- Define G and D
- Define optimizer and loss function
- Setup optimizer
- Train D and G
- Generate fake images
- Performance evaluation
- Display results

### D. Results

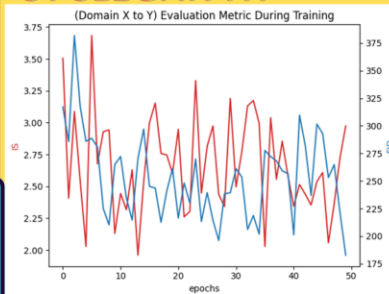
#### DCGAN



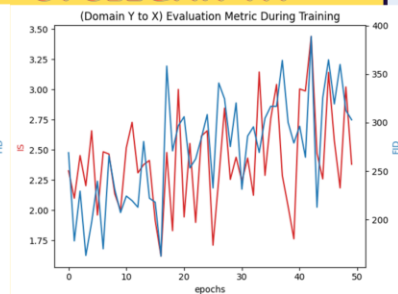
#### SNGAN



#### CYCLEGAN XY



#### CYCLEGAN YX



### E. Conclusion

| GANs        | Quality | Diversity | Stability | Issue            |
|-------------|---------|-----------|-----------|------------------|
| DCGAN       | High    | High      | Low       | None             |
| SNGAN       | High    | Low       | High      | Mode Collapse    |
| CycleGAN XY | High    | Low       | High      | Small Batch size |
| CycleGAN YX | Low     | High      | High      |                  |

### F. Recommendations

- Dataset Expansion
- Memory Optimization for CycleGAN
- Optimization for Training Parameters

Prepared by: Tan Jia Ler. Supervisor: Ts Dr Tong Dong Ling

# PLAGIARISM CHECK RESULT

## Turnitin Originality Report

Processed on: 25-Apr-2024 10:23 +08

ID: 2360370867

Word Count: 13131

Submitted: 4

2003397.docx By Jia Ler Tan

2% match (Li Quan, Haiyi Zhang. "Facial Animation Using CycleGAN", 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), 2021)

[Li Quan, Haiyi Zhang. "Facial Animation Using CycleGAN", 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering \(ICECCME\), 2021](#)

Similarity Index

15%

Similarity by Source

|                   |     |
|-------------------|-----|
| Internet Sources: | 6%  |
| Publications:     | 11% |
| Student Papers:   | 5%  |

1% match (Sanaa Kaddoura. "A Primer on Generative Adversarial Networks", Springer Science and Business Media LLC, 2023)

[Sanaa Kaddoura. "A Primer on Generative Adversarial Networks", Springer Science and Business Media LLC, 2023](#)

1% match (Internet from 12-Apr-2024)

[https://assets.researchsquare.com/files/rs-4198952/v1\\_covered\\_8376683c-c1f4-4316-bb45-159f078ce8ff.pdf?c=1712666610](https://assets.researchsquare.com/files/rs-4198952/v1_covered_8376683c-c1f4-4316-bb45-159f078ce8ff.pdf?c=1712666610)

1% match (Kelvin K. L. Wong. "Cybernetical Intelligence", Wiley, 2023)

[Kelvin K. L. Wong. "Cybernetical Intelligence", Wiley, 2023](#)

< 1% match (Internet from 10-Oct-2023)

[http://eprints.utar.edu.my/5563/1/fyp\\_IA\\_2023\\_ORFY.pdf](http://eprints.utar.edu.my/5563/1/fyp_IA_2023_ORFY.pdf)

< 1% match (Internet from 20-Jan-2024)

[http://eprints.utar.edu.my/5507/1/fyp\\_CT\\_2023\\_SKL.pdf](http://eprints.utar.edu.my/5507/1/fyp_CT_2023_SKL.pdf)

< 1% match (Internet from 15-Dec-2022)

[http://eprints.utar.edu.my/4674/1/fyp\\_CS\\_2022\\_TCH.pdf](http://eprints.utar.edu.my/4674/1/fyp_CS_2022_TCH.pdf)

< 1% match (Internet from 16-Jan-2024)

[http://eprints.utar.edu.my/6029/1/fyp\\_CS\\_2023\\_AIQ.pdf](http://eprints.utar.edu.my/6029/1/fyp_CS_2023_AIQ.pdf)

< 1% match (Internet from 30-Mar-2023)

[http://eprints.utar.edu.my/4738/1/fyp\\_IB\\_2022\\_LZH.pdf](http://eprints.utar.edu.my/4738/1/fyp_IB_2022_LZH.pdf)

< 1% match (Internet from 15-Dec-2022)

[http://eprints.utar.edu.my/4653/1/fyp\\_CS\\_2022\\_LCS.pdf](http://eprints.utar.edu.my/4653/1/fyp_CS_2022_LCS.pdf)

< 1% match (Internet from 16-Sep-2023)

[http://eprints.utar.edu.my/5526/1/fyp\\_IA\\_2023\\_ACD.pdf](http://eprints.utar.edu.my/5526/1/fyp_IA_2023_ACD.pdf)

< 1% match (Internet from 30-Mar-2023)

[http://eprints.utar.edu.my/4755/1/fyp\\_IB\\_2022\\_OWK.pdf](http://eprints.utar.edu.my/4755/1/fyp_IB_2022_OWK.pdf)

< 1% match (Internet from 13-Mar-2024)

[http://eprints.utar.edu.my/6034/1/fyp\\_CS\\_2023\\_HIZQ.pdf](http://eprints.utar.edu.my/6034/1/fyp_CS_2023_HIZQ.pdf)

< 1% match (Internet from 15-Dec-2022)

[http://eprints.utar.edu.my/4621/1/fyp\\_%2D\\_CN\\_%2D\\_LW1\\_%2D\\_1702593\\_.pdf](http://eprints.utar.edu.my/4621/1/fyp_%2D_CN_%2D_LW1_%2D_1702593_.pdf)

< 1% match (Internet from 20-Apr-2023)

[https://assets.researchsquare.com/files/rs-2828816/v1\\_covered.pdf?c=1682014259](https://assets.researchsquare.com/files/rs-2828816/v1_covered.pdf?c=1682014259)

< 1% match (Internet from 09-Apr-2024)

[https://assets.researchsquare.com/files/rs-4137066/v1\\_covered\\_1157b056-0410-47e5-bd35-638323cc8d70.pdf?c=1712550874](https://assets.researchsquare.com/files/rs-4137066/v1_covered_1157b056-0410-47e5-bd35-638323cc8d70.pdf?c=1712550874)

< 1% match (student papers from 13-Mar-2023)

[Submitted to Liverpool John Moores University on 2023-03-13](#)

< 1% match (student papers from 18-May-2023)

[Submitted to Liverpool John Moores University on 2023-05-18](#)

< 1% match (student papers from 05-Jun-2023)

[Submitted to Liverpool John Moores University on 2023-06-05](#)

< 1% match ("Generative Adversarial Learning: Architectures and Applications", Springer Science and Business Media LLC, 2022)  
["Generative Adversarial Learning: Architectures and Applications", Springer Science and Business Media LLC, 2022](#)

---

< 1% match (Francesco Mercaldo, Luca Brunese, Fabio Martinelli, Antonella Santone, Mario Cesarelli. "Generative Adversarial Networks in Retinal Image Classification", Applied Sciences, 2023)  
[Francesco Mercaldo, Luca Brunese, Fabio Martinelli, Antonella Santone, Mario Cesarelli. "Generative Adversarial Networks in Retinal Image Classification", Applied Sciences, 2023](#)

---

< 1% match (student papers from 20-Jun-2023)  
[Submitted to University of Western Sydney on 2023-06-20](#)

---

< 1% match (student papers from 18-Jun-2023)  
[Submitted to University of Western Sydney on 2023-06-18](#)

---

< 1% match (Behzod Mustafaev, Sungwon Kim, Eungsoo Kim. "Enhancing Metal Surface Defect Recognition through Image Patching and Synthetic Defect Generation", IEEE Access, 2023)  
[Behzod Mustafaev, Sungwon Kim, Eungsoo Kim. "Enhancing Metal Surface Defect Recognition through Image Patching and Synthetic Defect Generation", IEEE Access, 2023](#)

---

< 1% match (student papers from 25-Mar-2024)  
[Submitted to University of Strathclyde on 2024-03-25](#)

---

< 1% match (student papers from 14-Aug-2023)  
[Submitted to University of Strathclyde on 2023-08-14](#)

---

< 1% match (student papers from 02-Apr-2023)  
[Submitted to University of Strathclyde on 2023-04-02](#)

---

< 1% match (Internet from 24-Feb-2024)  
<https://d-nb.info/1321302266/34>

---

< 1% match (student papers from 21-Apr-2024)  
[Submitted to University of Cape Town on 2024-04-21](#)

---

< 1% match (Xu Li, Bowei Li, Minghao Fang, Rui Huang, Xiaoran Huang. "BaMSGAN: Self-Attention Generative Adversarial Network with Blur and Memory for Anime Face Generation", Mathematics, 2023)  
[Xu Li, Bowei Li, Minghao Fang, Rui Huang, Xiaoran Huang. "BaMSGAN: Self-Attention Generative Adversarial Network with Blur and Memory for Anime Face Generation", Mathematics, 2023](#)

---

< 1% match (student papers from 28-Feb-2021)  
[Submitted to Erasmus University of Rotterdam on 2021-02-28](#)

---

< 1% match (publications)  
 n Duc Thang University

---

< 1% match (student papers from 11-Jun-2023)  
[Submitted to Higher Education Commission Pakistan on 2023-06-11](#)

---

< 1% match (Rashed Iqbal Nekvi, Sajal Saha, Yaser Al Mtawa, Anwar Haque. "Examining Generative Adversarial Network for Smart Home DDoS Traffic Generation", 2023 International Symposium on Networks, Computers and Communications (ISNCC), 2023)  
[Rashed Iqbal Nekvi, Sajal Saha, Yaser Al Mtawa, Anwar Haque. "Examining Generative Adversarial Network for Smart Home DDoS Traffic Generation", 2023 International Symposium on Networks, Computers and Communications \(ISNCC\), 2023](#)

---

< 1% match (student papers from 09-May-2023)  
[Submitted to University of Surrey on 2023-05-09](#)

---

< 1% match (student papers from 16-Apr-2023)  
[Submitted to Australian National University on 2023-04-16](#)

---

< 1% match (student papers from 29-Apr-2020)  
[Submitted to GGS IP University Delhi on 2020-04-29](#)

---

< 1% match (E. Thirumagal, K. Saruladha. "GAN models in natural language processing and image translation", Elsevier BV, 2021)  
[E. Thirumagal, K. Saruladha. "GAN models in natural language processing and image translation", Elsevier BV, 2021](#)

---

< 1% match (Wei Shuhan, Yu Chengzhi, Liao Xiaoxiao, Wang Siyu. "Smart Infrastructure Design: Machine Learning Solutions for Securing Modern Cities", Sustainable Cities and Society, 2024)  
[Wei Shuhan, Yu Chengzhi, Liao Xiaoxiao, Wang Siyu. "Smart Infrastructure Design: Machine Learning Solutions for Securing Modern Cities", Sustainable Cities and Society, 2024](#)

---

< 1% match (Internet from 09-Jul-2022)  
<https://tel.archives-ouvertes.fr/tel-03517304/document>

---

< 1% match (student papers from 16-Apr-2024)

[Submitted to Meppco Schlenk Engineering college on 2024-04-16](#)

< 1% match (student papers from 27-Jun-2023)

[Submitted to October University for Modern Sciences and Arts \(MSA\) on 2023-06-27](#)

< 1% match (Hao Li, Liu Zhang, Heng Sun, Zhenhong Rao, Haiyan Ji. "Discrimination of unsound wheat kernels based on deep convolutional generative adversarial network and near-infrared hyperspectral imaging technology", Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy, 2021)

[Hao Li, Liu Zhang, Heng Sun, Zhenhong Rao, Haiyan Ji. "Discrimination of unsound wheat kernels based on deep convolutional generative adversarial network and near-infrared hyperspectral imaging technology". Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy. 2021](#)

< 1% match (Internet from 09-Dec-2023)

<https://iq.opengenus.org/key-terms-in-deep-learning/>

< 1% match (Bassel H. Zeno, Ilya A. Kalinovskiy, Yuri N. Matveev. "Identity preserving face synthesis using generative adversarial networks", Proceedings of the 5th International Conference on Engineering and MIS - ICEMIS '19, 2019)

[Bassel H. Zeno, Ilya A. Kalinovskiy, Yuri N. Matveev. "Identity preserving face synthesis using generative adversarial networks". Proceedings of the 5th International Conference on Engineering and MIS - ICEMIS '19, 2019](#)

< 1% match (Ying Chen, Hongping Lin, Wei Zhang, Wang Chen, Zonglai Zhou, Ali Asghar Heidari, Huiling Chen, Guohui Xu. "iCycle-GAN: Improved cycle generative adversarial networks for liver medical image generation", Biomedical Signal Processing and Control, 2024)

[Ying Chen, Hongping Lin, Wei Zhang, Wang Chen, Zonglai Zhou, Ali Asghar Heidari, Huiling Chen, Guohui Xu. "iCycle-GAN: Improved cycle generative adversarial networks for liver medical image generation". Biomedical Signal Processing and Control. 2024](#)

< 1% match (Internet from 07-Feb-2021)

<https://coek.info/pdf-medgan-medical-image-translation-using-gans-.html>

< 1% match (Internet from 19-Apr-2024)

[http://dspace.uju.ac.bd/bitstream/handle/52243/2958/MSc\\_thesis\\_report\\_of\\_Mohammad\\_Hasan.pdf?isAllowed=y&sequence=1](http://dspace.uju.ac.bd/bitstream/handle/52243/2958/MSc_thesis_report_of_Mohammad_Hasan.pdf?isAllowed=y&sequence=1)

< 1% match (student papers from 04-Mar-2024)

[Submitted to Ouachita Baptist University on 2024-03-04](#)

< 1% match (student papers from 26-Jun-2023)

[Submitted to University of Lagos on 2023-06-26](#)

< 1% match (student papers from 23-Feb-2024)

[Submitted to University of West Attica on 2024-02-23](#)

< 1% match (Internet from 12-Dec-2023)

<https://ijrset.in/index.php/ijrset/article/download/635/455/>

< 1% match (student papers from 19-Sep-2023)

[Submitted to American InterContinental University on 2023-09-19](#)

< 1% match (student papers from 19-Apr-2023)

[Submitted to Heriot-Watt University on 2023-04-19](#)

< 1% match (Muhayyuddin Ahmed, Ahsan Baidar Bakht, Taimur Hassan, Waseem Akram et al. "Vision-Based Autonomous Navigation for Unmanned Surface Vessel in Extreme Marine Conditions", 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023)

[Muhayyuddin Ahmed, Ahsan Baidar Bakht, Taimur Hassan, Waseem Akram et al. "Vision-Based Autonomous Navigation for Unmanned Surface Vessel in Extreme Marine Conditions", 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems \(IROS\), 2023](#)

< 1% match (R. Shanmuga Priya Rajendran, K. Vani K. "Vegetation Change Detection and Recovery Assessment on Post-fire Satellite Imagery using Deep Learning", Research Square Platform LLC, 2024)

[R. Shanmuga Priya Rajendran, K. Vani K. "Vegetation Change Detection and Recovery Assessment on Post-fire Satellite Imagery using Deep Learning". Research Square Platform LLC. 2024](#)

< 1% match (Tianjie Fu, Peiyu Li, Shimin Liu. "An imbalanced small sample slab defect recognition method based on image generation", Journal of Manufacturing Processes, 2024)

[Tianjie Fu, Peiyu Li, Shimin Liu. "An imbalanced small sample slab defect recognition method based on image generation". Journal of Manufacturing Processes. 2024](#)

< 1% match (student papers from 10-Mar-2023)

[Submitted to University of Lancaster on 2023-03-10](#)

< 1% match (Internet from 06-Apr-2024)

[https://keep-dev.lib.asu.edu/search?f%5B1%5D=linked\\_agents%3AReyna%2C+Janet&f%5B2%5D=linked\\_agents%3ASankar%2C+Lalitha&search\\_api\\_fulltext](https://keep-dev.lib.asu.edu/search?f%5B1%5D=linked_agents%3AReyna%2C+Janet&f%5B2%5D=linked_agents%3ASankar%2C+Lalitha&search_api_fulltext)

< 1% match ("Computer Vision – ECCV 2018", Springer Science and Business Media LLC, 2018)

["Computer Vision – ECCV 2018", Springer Science and Business Media LLC, 2018](#)



< 1% match ("Inventive Communication and Computational Technologies", Springer Science and Business Media LLC, 2020)

["Inventive Communication and Computational Technologies", Springer Science and Business Media LLC, 2020](#)

---

< 1% match (student papers from 03-Apr-2024)

[Submitted to Coventry University on 2024-04-03](#)

---

< 1% match (Jianbin Zheng, Daqing Liu, Chaoyue Wang, Minghui Hu, Zuopeng Yang, Changxing Ding, Dacheng Tao. "MMoT: Mixture-of-Modality-Tokens Transformer for Composed Multimodal Conditional Image Synthesis", International Journal of Computer Vision, 2024)

[Jianbin Zheng, Daqing Liu, Chaoyue Wang, Minghui Hu, Zuopeng Yang, Changxing Ding, Dacheng Tao. "MMoT: Mixture-of-Modality-Tokens Transformer for Composed Multimodal Conditional Image Synthesis", International Journal of Computer Vision, 2024](#)

---

< 1% match (student papers from 18-Jan-2024)

[Submitted to Manchester Metropolitan University on 2024-01-18](#)

---

< 1% match (student papers from 10-Feb-2023)

[Submitted to Universidade do Porto on 2023-02-10](#)

---

< 1% match (Internet from 27-Jan-2024)

<https://community.acer.com/en/discussion/comment/1241504>

---

< 1% match (Rong Lan, Di Guo, Pengyan Du, YuHao Sun, Zhao Feng, Haiyan Yu, Lu Zhang. "Multi-scale Cross-layer Feature Interaction GAN for Underwater Image Enhancement", Digital Signal Processing, 2023)

[Rong Lan, Di Guo, Pengyan Du, YuHao Sun, Zhao Feng, Haiyan Yu, Lu Zhang. "Multi-scale Cross-layer Feature Interaction GAN for Underwater Image Enhancement", Digital Signal Processing, 2023](#)

---

< 1% match (Wei Yuan, Shiyu Zhao, Li Wang, Lijia Cai, Yong Zhang. "Online course evaluation model based on graph auto-encoder", Intelligent Data Analysis, 2024)

[Wei Yuan, Shiyu Zhao, Li Wang, Lijia Cai, Yong Zhang. "Online course evaluation model based on graph auto-encoder", Intelligent Data Analysis, 2024](#)

---

< 1% match (Elshan Baghirov. "Evaluating the Performance of Different Machine Learning Algorithms for Android Malware Detection", 2023 5th International Conference on Problems of Cybernetics and Informatics (PCI), 2023)

[Elshan Baghirov. "Evaluating the Performance of Different Machine Learning Algorithms for Android Malware Detection", 2023 5th International Conference on Problems of Cybernetics and Informatics \(PCI\), 2023](#)

---

< 1% match (Kun Wu, Yan Qiang, Kai Song, Xueting Ren, WenKai Yang, WanJun Zhang, Akbar Hussain, Yanfen Cui. "Image synthesis in contrast MRI based on super resolution reconstruction with multi-refinement cycle-consistent generative adversarial networks", Journal of Intelligent Manufacturing, 2019)

[Kun Wu, Yan Qiang, Kai Song, Xueting Ren, WenKai Yang, WanJun Zhang, Akbar Hussain, Yanfen Cui. "Image synthesis in contrast MRI based on super resolution reconstruction with multi-refinement cycle-consistent generative adversarial networks", Journal of Intelligent Manufacturing, 2019](#)

---

< 1% match (MohammadReza Nehzati. "Integrating Convolutional Neural Networks for Improved Software Engineering: A Collaborative and Unbalanced Data Perspective", Memories - Materials, Devices, Circuits and Systems, 2024)

[MohammadReza Nehzati. "Integrating Convolutional Neural Networks for Improved Software Engineering: A Collaborative and Unbalanced Data Perspective", Memories - Materials, Devices, Circuits and Systems, 2024](#)

---

< 1% match (Internet from 17-Feb-2022)

<https://deepai.org/publication/a-survey-for-deep-rgb-t-tracking>

---

< 1% match (Internet from 02-Aug-2023)

<https://ebin.pub/computational-intelligence-for-clinical-diagnosis-3031236823-9783031236822.html>

---

< 1% match (Internet from 21-Nov-2022)

[https://www.utn.uu.se/sts/student/wp-content/uploads/2019/06/1906\\_salam\\_jadari.pdf](https://www.utn.uu.se/sts/student/wp-content/uploads/2019/06/1906_salam_jadari.pdf)

---

< 1% match (Internet from 17-May-2023)

<https://learninfun.github.io/learn-with-ai/ai-knowledge-hub/it/artificial-intelligence/natural-language-processing/text-classification/convolutional-neural-networks-cnns/>

---

< 1% match (Internet from 15-Mar-2022)

<https://lup.lub.lu.se/luur/download?fileOid=9074364&func=downloadFile&recordOid=9074363>

---

< 1% match (Internet from 24-Nov-2021)

<https://www.hindawi.com/journals/sp/2021/8340779/>

---

< 1% match (Internet from 07-Aug-2023)

[http://www.ijirset.com/upload/2023/june/128\\_Deep\\_NC.pdf](http://www.ijirset.com/upload/2023/june/128_Deep_NC.pdf)

---

< 1% match (Internet from 14-Sep-2023)

<https://www.mdpi.com/1424-8220/23/17/7543>

---

< 1% match (Danlan Huang, Xiaoming Tao, Jianhua Lu, Minh N. Do. "Geometry-Aware GAN for Face Attribute Transfer", IEEE Access, 2019)  
[Danlan Huang, Xiaoming Tao, Jianhua Lu, Minh N. Do. "Geometry-Aware GAN for Face Attribute Transfer". IEEE Access, 2019](#)

---

< 1% match (Jianmin Bian, Qian Wang, Siyu Nie, Hanli Wan, Juanjuan Wu. "Understanding nitrogen transport in the unsaturated zone with fluctuations in groundwater depth", Water Supply, 2021)  
[Jianmin Bian, Qian Wang, Siyu Nie, Hanli Wan, Juanjuan Wu. "Understanding nitrogen transport in the unsaturated zone with fluctuations in groundwater depth", Water Supply, 2021](#)

---

< 1% match (Joonyoung Song, Jae-Heon Jeong, Dae-Soon Park, Hyun-Ho Kim, Doo-Chun Seo, Jong Chul Ye. "Unsupervised Denoising for Satellite Imagery Using Wavelet Directional CycleGAN", IEEE Transactions on Geoscience and Remote Sensing, 2020)  
[Joonyoung Song, Jae-Heon Jeong, Dae-Soon Park, Hyun-Ho Kim, Doo-Chun Seo, Jong Chul Ye. "Unsupervised Denoising for Satellite Imagery Using Wavelet Directional CycleGAN", IEEE Transactions on Geoscience and Remote Sensing, 2020](#)

---

< 1% match (Qi Wang, Weidong Min, Qing Han, Qian Liu, Cheng Zha, Haoyu Zhao, Zitai Wei. "Inter-Domain Adaptation Label for Data Augmentation in Vehicle Re-identification", IEEE Transactions on Multimedia, 2021)  
[Qi Wang, Weidong Min, Qing Han, Qian Liu, Cheng Zha, Haoyu Zhao, Zitai Wei. "Inter-Domain Adaptation Label for Data Augmentation in Vehicle Re-identification", IEEE Transactions on Multimedia, 2021](#)

---

< 1% match (Yongyi Lu, Yu-Wing Tai, Chi-Keung Tang. "Chapter 18 Attribute-Guided Face Generation Using Conditional CycleGAN", Springer Science and Business Media LLC, 2018)  
[Yongyi Lu, Yu-Wing Tai, Chi-Keung Tang. "Chapter 18 Attribute-Guided Face Generation Using Conditional CycleGAN". Springer Science and Business Media LLC, 2018](#)

---

< 1% match (Internet from 04-Jul-2023)  
[https://fenix.tecnico.ulisboa.pt/downloadFile/1407770020546511/thesis\\_79092.pdf](https://fenix.tecnico.ulisboa.pt/downloadFile/1407770020546511/thesis_79092.pdf)

---

< 1% match (Internet from 15-Apr-2024)  
<https://www.journal.esrgroups.org/jes/article/download/1263/1006/2134>

|  |            |                            |                  |
|--|------------|----------------------------|------------------|
| <b>Universiti Tunku Abdul Rahman</b>   |            |                            |                  |
| <b>Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b> |            |                            |                  |
| Form Number: FM-IAD-005  | Rev No.: 0 | Effective Date: 01/10/2013 | Page No.: 1 of 1 |



**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

|                                     |   |
|-------------------------------------|---|
| <b>Full Name(s) of Candidate(s)</b> | Tan Jia Ler   |
| <b>ID Number(s)</b>                 | 20ACB03397  |
| <b>Programme / Course</b>           | Bachelor of Computer Science (Honours)  |
| <b>Title of Final Year Project</b>  | Performance Comparison between Generative Adversarial Networks (GAN) Variants in Generating Comic Character |

| <b>Similarity</b>   | <b>Supervisor's Comments<br/>(Compulsory if parameters of originality exceeds the limits approved by UTAR)</b> |
|---|--|
| <b>Overall similarity index: <u>15</u> %</b><br><br><b>Similarity by source</b><br>Internet Sources: <u>6</u> %<br>Publications: <u>11</u> %<br>Student Papers: <u>5</u> %  |  |
| <b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>   |  |
| <b>Parameters of originality required and limits approved by UTAR are as Follows:</b><br>(i) Overall similarity index is 20% and below, and<br>(ii) Matching of individual sources listed must be less than 3% each, and<br>(iii) Matching texts in continuous block must not exceed 8 words<br><i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i> |  |

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

***Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.***

*Tong*

\_\_\_\_\_  
Signature of Supervisor

Name: Tong Dong Ling

Date: 25 Apr 2024

\_\_\_\_\_  
Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_



**UNIVERSITI TUNKU ABDUL RAHMAN**

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY  
(KAMPAR CAMPUS)**

**CHECKLIST FOR FYP2 THESIS SUBMISSION**

|                 |                      |
|-----------------|----------------------|
| Student Id      | 20ACB03397           |
| Student Name    | Tan Jia Ler          |
| Supervisor Name | Ts Dr Tong Dong Ling |

| TICK (✓) | DOCUMENT ITEMS  |
|----------|---|
|          | Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.            |
| ✓        | Title Page  |
| ✓        | Signed Report Status Declaration Form   |
| ✓        | Signed FYP Thesis Submission Form   |
| ✓        | Signed form of the Declaration of Originality   |
| ✓        | Acknowledgement   |
| ✓        | Abstract  |
| ✓        | Table of Contents   |
| ✓        | List of Figures (if applicable)   |
| ✓        | List of Tables (if applicable)  |
| ✓        | List of Symbols (if applicable)   |
| ✓        | List of Abbreviations (if applicable)   |
| ✓        | Chapters / Content  |
| ✓        | Bibliography (or References)  |
| ✓        | All references in bibliography are cited in the thesis, especially in the chapter of literature review  |
|          | Appendices (if applicable)  |
| ✓        | Weekly Log  |
| ✓        | Poster  |
| ✓        | Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)  |
| ✓        | I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report. |

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

\_\_\_\_\_  
(Signature of Student)

Date: 25 April 2024