Fingerprint Synthesis Using Deep Generative Models

Diego Andre Figueroa Llamosas,
Weizhong Tang

Master's thesis
2023:E24

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Fingerprint Synthesis Using Deep Generative Models

Diego Figueroa
di4642fi-s@student.lu.se

Weizhong Tang
we1724ta-s@student.lu.se

May 27, 2023

Master's thesis work carried out at
the Centre for Mathematical Sciences, Lund University.

Supervisors: Alexandros Sopasakis, alexandros.sopasakis@math.lth.se
Donglin Liu, donglin.liu@math.lth.se
Kerstin Johnsson, kerstin.johnsson@precisebiometrics.com

Examiner: Kalle Åström, karl.astrom@math.lth.se

Abstract

The advancements in biometric technology have amplified the need for more robust fingerprint synthesis techniques. In this thesis, We first explored the application of synthesizing live fingerprint images in high fidelity using deep generative models (e.g., generative adversarial networks and diffusion models) and created synthetic fingerprints that retain the uniqueness and complexity of the original samples. Thereafter, by employing style transfer techniques (e.g., cycleGAN and cycleWGAN-GP), we effectively blended the global structure of one fingerprint with the local features of another fingerprint to generate a new fingerprint that is both visually realistic and distinctive, such as generating a one-to-one spoof fingerprint that is highly consistent with the corresponding live fingerprint from the training set.

We employed, in the field of image generation, the state-of-the-art metrics assessing the quality of synthetic fingerprints, mainly from the perspectives of statistical analysis and subjective evaluation, and conducted a comprehensive evaluation of synthetic live and spoof fingerprints.

Our best diffusion model achieves a promising Fréchet Inception Distance (FID) score of 15.78 and is capable of generating live fingerprints that obtain even smaller False Acceptance Rate (FAR) than the real live fingerprints. Our best style transfer model - cycleWGAN-GP is capable of generating spoof fingerprints in high quality from real live fingerprints, and the distribution of these synthetic spoof fingerprints closely resembles that of real spoof fingerprints.

Our results demonstrate the potential of these methods in generating high-quality fingerprints and can be used for various applications such as security enhancement, template protection, and biometric system evaluation.

Keywords: Fingerprint Synthesis, Deep Generative Models, Style Transfer, Metrics

Acknowledgements

This thesis project was conducted in Spring 2023 and was co-directed by Lund University and Precise Biometrics AB. The aim of the thesis was to investigate a project proposed by the development department of Precise Biometrics to enable fingerprint synthesis using machine learning techniques.

We would like to thank our academic supervisors at Lund University,

Alexandros Sopasakis

Donglin Liu

We would also like to thank our supervisor at Precise Biometrics,

Kerstin Johnsson

We are truly grateful for the invaluable assistance provided by our supervisors, which encompassed challenging and instructive suggestions. The time spent during our meetings with them not only helped us enhance our project and better analyze our results but also inspired us to strive for excellence in our work.

We would like to extend a special thanks to **Anders Olsson** for all his feedback and support, as well as **Johan Windmark** for his insightful explanations of matching algorithms and spoof detection. Ultimately, we extend our appreciation to everyone who contributed their opinions and expertise, which significantly enhanced the quality of our thesis.

Diego I am deeply grateful to Weizhong for being the most exceptional thesis partner I could have hoped for. His unwavering support and dedication played a pivotal role in our continuous and successful progress throughout this project. To my beloved mother, father, and sister, I extend my heartfelt appreciation for the immeasurable love and unwavering support they have showered upon me. My mother, in particular, deserves a special mention for always believing in me and inspiring me to dream big for my future. Furthermore, I extend my sincere thanks to my colleagues at Precise for their invaluable feedback and the remarkable positive working culture they foster. I am truly grateful to everyone who has played a part in this endeavor, as their presence and encouragement have made this thesis possible. Thank you all from the bottom of my heart for your immeasurable contributions, love, and support.

Weizhong First and foremost, I would like to extend my sincere gratitude to my thesis partner, Diego. Without his strong support and excellent research skills, this thesis would not have come to fruition. I would also like to express my deepest appreciation to my family, who have always been by my side and supported me, particularly for providing me with the opportunity to study in Sweden. I am grateful for all the help received from our supervisors and colleagues to make this thesis possible. Lastly, I deeply appreciate professors Alexandros Sopasakis, Bo Bernhardsson and Pierre Nugues for providing me with invaluable opportunities and assistance. Their support has not only significantly enhanced my knowledge and capabilities, but also facilitated my progress on the academic path.

Abbreviations

CNNs - Convolutional Neural Networks

ConvNet - Convolutional Neural Networks

DCGAN - Deep Convolutional Generative Adversarial Network

DDIMs - Denoising Diffusion Implicit Models

DDPMs - Denoising Diffusion Probabilistic Models

FAR - False Acceptance Rate

FID - Fréchet Inception Distance

GANs - Generative Adversarial Networks

IS - Inception Score

KID - Kernel Inception Distance

LS - Likeness Score

PRDC - Precision, Recall, Density and Coverage

ReLU - Rectified Linear Unit

ResNet - Residual Neural Networks

SiLU - Sigmoid Linear Unit

vDDPM - Vanilla DDPM

WGAN-GP - Wasserstein Generative Adversarial Network + Gradient Penalty

Contents

Chapter 1
Introduction

1.1 Background

Biometric technologies have been widely used in various fields for identity authentication. Nowadays, technologies like fingerprint, facial and iris recognition based on biometric features are considered to be more secure, confidential, and convenient than traditional identification methods, benefiting from the fact that inherent biometric features are in general measurable, verifiable and unique. While on the other hand, collecting fingerprint data is costly and time-consuming as a large number of high quality fingerprint images are demanded in order to develop and optimize fingerprint matching algorithms. Moreover, it is also challenging and controversial when it comes to how those biometric information should be collected, utilized and stored. Under the supervision of general policies in Europe e.g., GDPR, people's privacy is well protected while on the other hand it becomes more intractable to access and collect human characteristics for the purpose of biometric authentication.

Machine Learning is a rapidly growing field that has seen significant advancements and has developed its applications in a wide range of fields, including computer vision [1], medical imaging [2], speech recognition [3], natural language processing [4] and so on. As the growth of computational power, a branch of Machine Learning, Deep Learning has shown its potential and has substantially improved the performances of Machine Learning applications in all aspects especially in computer vision and natural language processing. Undoubtedly, we have seen many examples of deep learning generative models in image generation, such as autoencoders [5], variational autoencoders [6], generative adversarial networks (GANs) [7] and the recent state-of-the-art diffusion models [8]. These models learn to generate new data by modeling the underlying probability distribution of the training data, which enables them to create new data samples that are similar but diverse to the training data.

In order to address the problem of fingerprint data shortage, we therefore propose the idea that, with a large number of real fingerprints available for training, we will focus on generating similar but unique live fingerprints, and further generating spoof fingerprints

through fingerprint-to-fingerprint transformation by implementing a variety of deep learning generative models which will be trained on the company's given dataset.

1.2 Related Work

The history of fingerprint synthesis can be traced back to the beginning of 21st century and even earlier. The paper [9] in 2000 introduced a way to generate fingerprints using Gabor-like space-variant filters which can be tuned according to the underlying ridge orientation. After two years, their work had been complemented by being able to derive a series of impressions from the same master-fingerprint, which made a large fingerprint database construction practical and possible [10]. On top of that, a modified Gabor filter was used for ridge pattern generation in another five years [11]. Thereafter, a few of researches focused on synthesizing iris by using e.g., principal component analysis (PCA) [12] or Markov Random Fields [13], and a large number of articles [14], [15], [16], [17], [18], [19] researched into fingerprint reconstruction based on minutiae templates. Later in 2013, this paper [20] demonstrated the effectiveness of its method of modeling and generating synthetic fingerprint textures.

However, since the Generative Adversarial Networks (GANs) was firstly proposed in 2014 [7], there has been an increasing trend to use deep learning in fingerprint synthesis. Finger-GAN [21] was an early trial where GANs were actually implemented to generate realistic fingerprints. After that, a variety of GANs models were used to generate fingerprints. For instance, the two-stage GANs [22] and the lightweight version of GANs [23] were based on the original standard GANs [7]. In this paper [24], based on deep convolutional generative adversarial networks (DCGAN) [25], two networks (Alex-GAN and VGG-GAN) were proposed to improve the performance of DCGAN by enhancing the performance of the discriminator, the structure of which was modified by two typical convolutional neural networks, AlexNet [26] and VGG11 [27]. Striuk, O. and Kondratenko, Y. in [28] researched into Adaptive Deep Convolutional GAN (ADCGAN) in fingerprint synthesis. This paper [29] utilized progressive growth-based GANs to develop the Clarkson Fingerprint Generator (CFG) that is capable of generating realistic fingerprints up to 512×512 pixels. Recently, two papers [30], [31] implemented cycleGAN [32] to generate fingerprints in high fidelity. Another Finger-GAN [33] used for fingerprint synthesis was proposed by embedding skip-connected DAE (SC-DAE) in the GANs as a generator, and a SpoofGAN [34] was proposed this year for generating spoof fingerprints.

Most past models have been trained on publicly available fingerprint datasets. Since the techniques of the company Precise Biometrics are highly dependent on its confidential data, which is very different from the publicly available fingerprint datasets, these past studies can hardly meet the company's requirements. Although there have been many techniques applied in the field of fingerprint generation, the most popular diffusion models do not seem to have been tried so far. We have tried to explore the performance of deep generative models in the field of fingerprint generation step by step, starting from the popular GAN models. In addition to this, we also conducted fingerprint-to-fingerprint transformation specifically for the company's non-public dataset to address the problem of learning using a small number of samples.

1.3 Goals

The current issue is that, for the company Precise Biometrics, new fingerprints sensors [1] have been developed continuously and each new sensor requires new matching and spoof detection algorithms. Due to the distinction of each sensor and the high-quality images required by the matching algorithms, each dataset can only be used for specific sensors and new fingerprints therefore need to be collected for new sensors. In order to decrease the cost of data collection, we hope to develop fingerprint generators that can generate unique fingerprint images for specific sensors. In the case that the dataset required for the new sensor is similar to the dataset used for training the old algorithms, we hope to leverage the existing datasets and achieve high-quality fingerprint synthesis by using a small and limited amount of new data.

We have seen large amount of previous publications that focused on generating synthetic fingerprints using machine learning generative models. At this stage, the master's thesis project aims at experimenting and performing mainly two machine learning generative models (WGAN-GP and DDPMs) to generate fingerprints. The training data will be given by the company in order to enrich the quantity and diversity of the company's real fingerprint data itself.

We are therefore aiming at resolving the research questions below:

- Which deep generative models perform best in live fingerprints synthesis?
- Which deep generative models can achieve fingerprint-to-fingerprint transformation in sufficient quality?
- How to measure the quality of generated fingerprints in practice and in statistics?

Ultimately, we are aiming at achieving the research objectives below:

- Generate similar but unique synthetic fingerprints in high fidelity based on real fingerprints using e.g. DCGAN[25], WGAN-GP[35] and diffusion models[8].
- Generate a variety of spoof fingerprints correspondingly based on real live fingerprints using e.g. styleGAN[36], cycleGAN[32] and diffusion models[8].

1.4 Division of Work

We started from building up our datasets for the first step, in which we roughly shared equal responsibilities. Thereafter, Diego made a significant contribution to the implementation of the first WGAN-GP model, and had done much work on tuning the model with the datasets. While training the model to synthesize live fingerprint images, Weizhong carried out his work on evaluation metrics separately.

After achieving satisfactory results in live fingerprint synthesis with WGAN-GP, we intended to implement another state-of-the-art deep generative technique - denoising diffusion probabilistic models (DDPM) to generate live fingerprint images. In this task, Weizhong

[1] The sensors are the physical devices capturing the fingerprint as an image. The sensors differ from each other depending on the matching algorithms, hardware devices, customer's demands and so on.

contributed considerably to its implementation and optimization. Later on, Diego started experimenting the next task of fingerprint-to-fingerprint transformation.

In terms of thesis writing, each of us has mostly carried out the relevant sections corresponding to the programming work, i.e., sections related to GANs were assigned to Diego and sections related to diffusion models and metrics were assigned to Weizhong. Apart from that, Weizhong completed most written work in chapter 1, Introduction and sections 2.3, Machine Learning Techniques. The rest parts of the written work has been conducted together throughout the period. All the unmentioned content is considered to be equally contributed.

Chapter 2
Methodology

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [7] are a powerful class of generative models that involves setting up a competition between two networks. The first network, called the generator G, transforms random noise into data samples. The second network, known as the discriminator D, receives input samples from either the generator or real data and must identify which is which. The generator's objective is to deceive the discriminator into classifying its generated samples as real. Mathematically, this process is represented by the minimax objective equation (2.1) between the generator G and the discriminator D where $\mathbb{P}_data$ is the data distribution and $\mathbb{P}_model$ is the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z}), \boldsymbol{z} \sim p(\boldsymbol{z})$ (the input $\boldsymbol{z}$ to the generator is sampled from some simple noise distribution p, such as the uniform distribution or a spherical Gaussian distribution).

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_data} [\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_model} [\log(1 - D(\tilde{\boldsymbol{x}}))] \tag{2.1}$$

2.1.1 Deep Convolutional GAN

Deep Convolutional Generative Adversarial Networks (DCGAN) [25] is a type of GAN [7] that uses convolutional neural networks (CNNs) as the generator and discriminator networks was introduced in a 2015 paper [25]. DCGAN is designed to generate high-quality images that closely resemble real-world images.

The main innovation of DCGAN is the use of convolutional layers in both the generator and discriminator networks. This allows the networks to learn hierarchical representations of the input data, capturing both low-level features such as edges and textures and high-level features such as object shapes and compositions.

DCGAN also includes several architectural guidelines for building the generator and discriminator networks. The first is using the all convolutional net [37] which replaces any

pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator). Second is removing fully connected hidden layers for deeper architectures, using, for example, global average pooling which has been utilized in image classification models [38]. See Fig. (2.1) for a visualization of an example model architecture. Third is using Batch Normalization [39] in both the generator and the discriminator which helps to stabilize the learning process by normalizing the input to each unit to have zero mean and unit variance. These guidelines help improve the stability of the training process, helps gradient flow, prevent mode collapse, and produce more visually pleasing results.

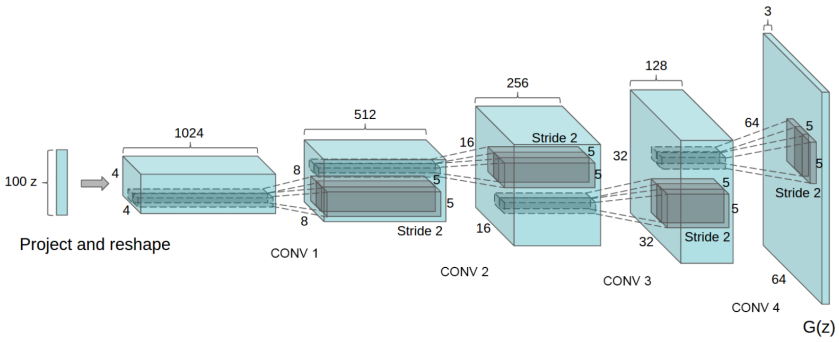


Figure 2.1: A uniform distribution Z with 100 dimensions is transformed into a small spatial extent convolutional representation that has many feature maps. Then, a sequence of four fractionally-strided convolutions is applied to convert the high level representation into an image with a resolution of 64 × 64 pixels. It's important to note that neither fully connected layers nor pooling layers are utilized in this process. However, it should be noted that some recent papers inaccurately refer to these fractionally-strided convolutions as deconvolutions. Cited from [25]

Apart for these guidelines, the authors made use of ReLU activation [40] in the generator for all layers except for the output, which uses Tanh. For all the layers on the discriminator, it is used LeakyReLU activation [41].

During training, the generator network takes random noise as input and generates images that are then fed to the discriminator network, which tries to distinguish between the generated images and real images from a training dataset. The generator and discriminator networks are trained together, with the generator trying to fool the discriminator into thinking its generated images are real, and the discriminator trying to correctly classify the images as real or fake.

The authors demonstrated the effectiveness of DCGAN on several image generation tasks, including generating images of faces, bedrooms, and street scenes. Since then, DCGAN has become a widely used architecture for various generative modeling tasks in computer vision and beyond, including image super-resolution, text-to-image synthesis, and video prediction.

2.1.2 Improved Wasserstein GAN

Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [35] was introduced in a 2017 paper [35]. The authors proposed a modification to the loss function

used in the original Wasserstein GAN (WGAN) [42] Eq. (2.2)[1] which uses the Wasserstein distance [43] (also known as earth mover's distance) as the objective function for training the generator and discriminator networks. This distance metric measures the difference between the probability distributions of real and fake data, making it more stable and easier to optimize than traditional GAN loss Eq. (2.1).

$$\mathcal{L}_{\text{WGAN}} = \min_{G} \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] \tag{2.2}$$

In addition to using the Wasserstein distance, WGAN-GP introduces a gradient penalty term to the loss function as an alternative method for enforcing the Lipschitz constraint. By considering the fact that a differentiable function is 1-Lipschitz if and only if its gradients have a norm of 1 everywhere, the authors suggest directly limiting the norm of the critic's output gradients with respect to its input. To address issues of computational feasibility, a penalty is applied to the gradient norm for random samples obtained from $\mathbb{P}_{\hat{\boldsymbol{x}}}$, effectively implementing a softened version of the constraint. The revised objective function is thus formulated accordingly,

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original WGAN loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Gradient penalty}}. \tag{2.3}$$

The penalty term helps prevent the discriminator from becoming too powerful and dominating the training process, which encourages smoothness in the decision boundary between real and fake data and can contribute to mode collapse.

There are made two further modifications to the training procedure. First, it is noted that using batch normalization in both the generator and discriminator can affect the discriminator's problem formulation and make the penalized training objective invalid. As a solution, it is suggested omitting batch normalization in the critic and using normalization schemes that do not introduce correlations between examples, such as layer normalization. Second, it is proposed using a two-sided penalty instead of the traditional one-sided penalty. Empirically, it is found that this change did not constrain the critic too much and led to slightly better performance.

2.1.3 CycleGAN

Cycle-Consistent Adversarial Network (CycleGAN) [32] is a type of GAN [7] that is used for unsupervised image-to-image translation. Introduced in a 2017 paper [32] by Jun-Yan Zhu and his colleagues, CycleGAN is designed to learn mappings between two different domains of images without the need for paired training data.

The key innovation of CycleGAN is the use of cycle consistency [32], which allows the model to learn the mappings between the two domains $\boldsymbol{X}$ and $\boldsymbol{Y}$ given training samples $\{x_i\}_{i=1}^N$ where $x_i \in \boldsymbol{X}$ and $\{y_j\}_{j=1}^N$ where $y_j \in \boldsymbol{Y}$. Specifically, the model consists of two GANs, each with a generator $\boldsymbol{G}$ and $\boldsymbol{F}$ and a discriminator network $\boldsymbol{D_Y}$ and $\boldsymbol{D_X}$. One GAN is trained to map images from domain $\boldsymbol{X}$ to domain $\boldsymbol{Y}$, while the other GAN is trained to map images from domain $\boldsymbol{Y}$ to domain $\boldsymbol{X}$.

[1] Equations (2.2) and (2.3) are referred from [35]

The objective contains two types of terms: adversarial losses [7] for matching the distribution of generated images to the data distribution in the target domain; and cycle consistency losses [32] to prevent the learned mappings G and F from contradicting each other.

The adversarial loss [7] for the mapping function $G : XY$ and its discriminator D_Y is express as,

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \tag{2.4}$$

in a similar way, the adversarial loss for the mapping function $F : YX$ and its discriminator D_X is express as,

$$\mathcal{L}_{\text{GAN}}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log(1 - D_X(F(y))]. \tag{2.5}$$

The cycle consistency loss is express as,

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]. \tag{2.6}$$

The full objective function [32] is then express as,

$$\mathcal{L}(F, D_X, Y, X) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F), \tag{2.7}$$

where λ controls the relative importance of the two objectives.

2.1.4 CycleWGAN-GP

Yanming Zhu et al. [44] proposed a new comprehensive objective function to enhance the training stability of CycleGAN by extending the concept of WGAN [42] and Improved WGAN [35] to it.

Using Eq. (2.2), the proposed adversarial losses are expressed as,

$$\begin{aligned} \mathcal{L}_{\text{WGAN}}(G, D_Y, X, Y) &= \mathbb{E}_{y \sim p_Y(y)}[D_Y(y)] - \mathbb{E}_{x \sim p_X(x)}[D_Y(G(x)], \\ \mathcal{L}_{\text{WGAN}}(F, D_X, Y, X) &= \mathbb{E}_{x \sim pX(x)}[D_X(x)] - \mathbb{E}_{y \sim p_Y(y)}[D_X(F(x)]. \end{aligned} \tag{2.8}$$

By introducing gradient penalty [35] Eq. (2.3), the full objective function is expressed as,

$$\begin{aligned} \mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{WGAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{WGAN}}(F, D_X, Y, X) + \lambda_0 \mathcal{L}_{\text{cyc}}(G, F) \\ + \lambda_1 \mathop{\mathbb{E}}_{\hat{x}_1 \sim \mathbb{P}_{x_1}} \left[\left(\left\| \nabla_{\hat{x}_1} D_Y(\hat{\boldsymbol{x}}_1) \right\|_2 - 1 \right)^2 \right] + \lambda_2 \mathop{\mathbb{E}}_{\hat{x}_2 \sim \mathbb{P}_{x_2}} \left[\left(\left\| \nabla_{\hat{x}_2} D_X(\hat{\boldsymbol{x}}_2) \right\|_2 - 1 \right)^2 \right]. \end{aligned} \tag{2.9}$$

2.2 Diffusion Models

The original idea of diffusion models inspired by non-equilibrium statistical physics was initially presented in the paper [45] in 2015. As its name indicates, any substance that exists in a diffusible form can be spread uniformly throughout space by random motion, which turns out that its own localized distribution will eventually transform into uniform distribution. This theory was then referred again by Jonathan H., Ajay J. and Pieter A. who proposed a state-of-the-art deep learning generative model called Denoising Diffusion Probabilistic Models (DDPMs) [8].

2.2.1 Denoising Diffusion Probabilistic Models

Given latent variables $\mathbf{x}_1$,..., $\mathbf{x}_T$ with the same dimensionality as $\mathbf{x}_0$, a DDPM starts from adding Gaussian noise into the input sample i.e. $\mathbf{x}_0$ step by step until it is converted to a purely isotropic Gaussian sample i.e. $\mathbf{x}_T$. The diffusion steps are carried out on the assumption of Markov chain which indicates that the future state of the process only depends on the current state of the process. In Eq. (2.10)[2], $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is controlled by the previous state $\mathbf{x}_{t-1}$, $\mathbf{I}$ is the identity matrix, and the trainable noise variable β_t controlling the noise level in each step. We then call it the forward process as illustrated in Fig. (2.2),

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \quad q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \tag{2.10}$$

In the forward process, a notable property is that $\mathbf{x}_t$ can be sampled at any arbitrary time step t in a closed form using the reparameterization trick,

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\right), \tag{2.11}$$

where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$.

Likewise, the reverse process $p_\theta(\mathbf{x}_{0:T})$ is conducted through subtracting or denoising the purely noisy sample $\mathbf{x}_T$ step by step in a reverse way,

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \quad p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{2.12}$$

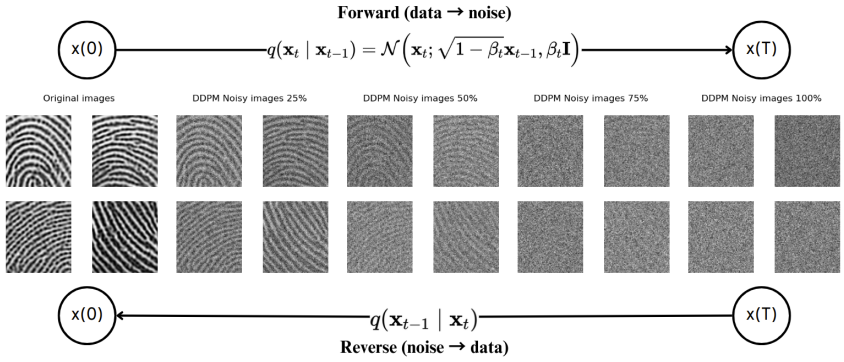


Figure 2.2: The forward and reverse processes of DDPMs. In the forward process, the input sample is randomly added with Gaussian noise constantly in each step, and becoming an isotropic Gaussian noise image in the end. In the reverse process, a random noise sample drawn from an isotropic Gaussian distribution is denoised in a level estimated by a deep learning model (typically a U-Net) in each step, and being reconstructed completely after going through all the time steps.

Given the condition that $\mathbf{x}_0$ is known,

[2]Equations (2.10), (2.11), (2.12), (2.13), (2.14), (2.15) and (2.16) are referred from DDPM [8].

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right), \tag{2.13}$$

where,

$$\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{2.14}$$

However, $\mathbf{x}_0$ remains unknown in image generation until the last step. It is tractable to estimate the next step based on Bayes Theorem and probability as shown in the forward process, but this is unfortunately not feasible for the reverse process. As the knowledge of the whole data distribution is required to resolve the backward conditional probability, the sample in the previous step needs to be instead predicted by deep learning models in the reverse process.

Without knowing the $\mathbf{x}_0$, by sampling $\mathbf{x}_t$ step by step, the reverse sampling process then goes to,

$$\mathbf{x}_{t-1} = \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2.15}$$

where,

$$\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right) = \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t\right)\right)\right) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right) \tag{2.16}$$

where $\boldsymbol{\epsilon}_\theta$ is a function approximator given by the deep generative model for predicting $\boldsymbol{\epsilon}$ from $\mathbf{x}_t$. The desired image can then be reconstructed eventually.

2.2.2 Denoising Diffusion Implicit Models

Unlike DDPMs that require the diffusion processes to be Markovian, denoising diffusion implicit models (DDIMs) [46] propose a new way that can dramatically accelerate the forward processes defined as a class of Non-Markovian processes. DDIMs have been demonstrated to be able to generate samples in high quality from 10 up to 50 times faster than the time DDPMs need.

The authors chose one of the non-Markovian inference processes in which the reverse process relies on the conditional probability $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)$ rather than $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$. Thereafter, the non-Markovian forward process according to the Bayes Theorem is derived as [3],

$$q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0\right) = \frac{q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_0\right)}{q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0\right)}, \tag{2.17}$$

where,

$$q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 \boldsymbol{I}\right) \tag{2.18}$$

[3] Note: All the α_t in DDIMs equal to $\bar{\alpha}_t$ in DDPMs

In section 2.2.1, we have shown how $\mathbf{x}_{t-1}$ is generated by subtracting the noise map $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ from the noise image $\mathbf{x}_t$. However, here $\mathbf{x}_{t-1}$ is generated by first predicting the desired image $\mathbf{x}_0$ and then adding the predicted noise term to $\mathbf{x}_0$, i.e.,

$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{“predicted } \boldsymbol{x}_0 \text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\boldsymbol{x}_t)}_{\text{“direction pointing to } \boldsymbol{x}_t \text{”}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}} . \tag{2.19}$$

2.3 Machine Learning Techniques

In this section, we will discuss a variety of machine learning techniques which have been extensively used for a number of popular model structures. Some of them constitute the core parts of machine learning models, for example, the attention blocks in a transformer, the U-Net structure in the reverse process of a diffusion model and so on.

We will briefly introduce several typical deep neural networks (DNNs), a couple of normalization methods and activation functions used in the thesis. We will then, in Chapter 3, point out what the roles of these machine learning modules are in our task.

2.3.1 U-Net

U-Net is a popular deep learning architecture specifically designed for biomedical image segmentation tasks. It was first proposed by Olaf Ronneberger, Philipp Fischer and Thomas Brox in 2015 [47]. Since then, the architecture has been adopted and applied to various other image segmentation tasks in different fields.

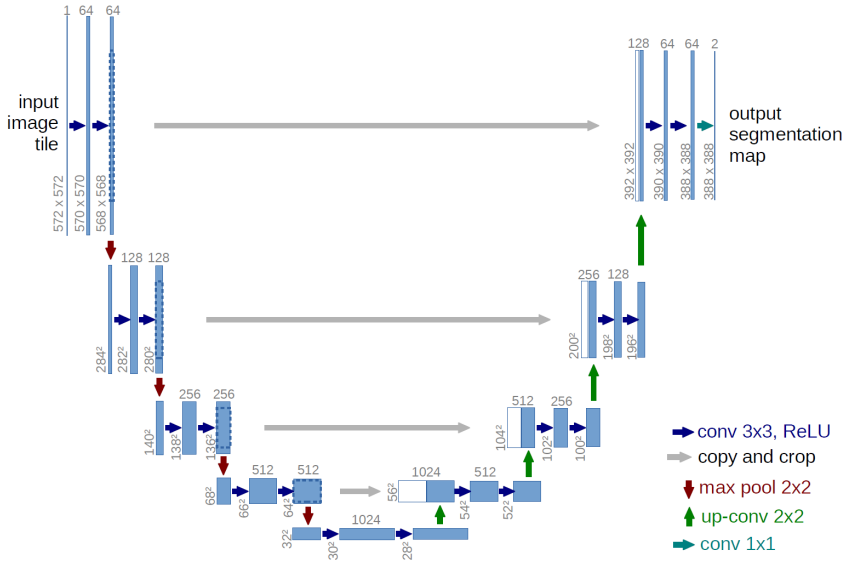


Figure 2.3: A typically U-Net structure. Cited from [47]

The U-Net architecture consists of a systolic (encoder) path and an extended (decoder) path (see Fig. (2.3)), which together form a "U" shape, hence the name. The encoder path is a

series of convolutional layers followed by a maximum set layer, which helps capture contextual information and reduces the spatial size of the input image. The decoder path consists of a series of transposed convolutional layers, which helps to reconstruct the segmented image from the feature map generated by the encoder.

One of the main features of U-Net is the introduction of jump connections between the encoder and decoder paths. These connections allow the model to preserve and utilize the high-resolution spatial information from the earlier layers of the encoder when reconstructing the segmented image in the decoder path. This design choice greatly improves the performance of U-Net, especially when working with images containing fine details or complex structures.

2.3.2 Residual Neural Network

ResNet [48], which stands for Residual Network, is a deep learning architecture designed for image recognition and classification tasks. Because of its ability to effectively train very deep neural networks, ResNet has since become a popular and influential architecture in the field of computer vision.

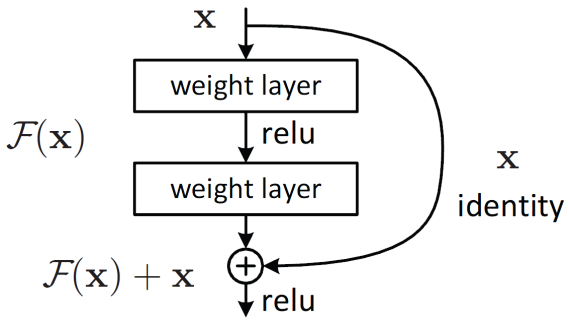


Figure 2.4: Overview of residual connections. Cited from [48]

The key innovation of ResNet is, in Fig. (2.4), the introduction of residual connections (also known as skip connections or shortcut connections) that allow the network to learn residual functions associated with layer inputs instead of learning unreferenced functions. These connections allow the model to bypass certain layers during training, effectively creating shorter paths for the flow of gradients during backpropagation. This helps alleviate the gradient disappearance problem common in deep neural networks and allows the trained network to be deeper than before.

2.3.3 ConvNeXt

ConvNeXt [49] proposed in 2022 is a deep neural network constructed entirely from standard Convolutional Neural Networks (ConvNets) modules. The authors demonstrated that ConvNeXt was capable of competing with state-of-the-art hierarchical vision Transformers [50] across various computer vision benchmarks, while maintaining the simplicity and efficiency of standard ConvNets.

ConvNeXt is considered to be an alternative to ResNet in standard DDPM [8].

2.3.4 Self-Attention Technique

Self-attention [51] is a technique used in deep learning models to capture long-range dependencies and contextual information within sequences, such as text or time series data.

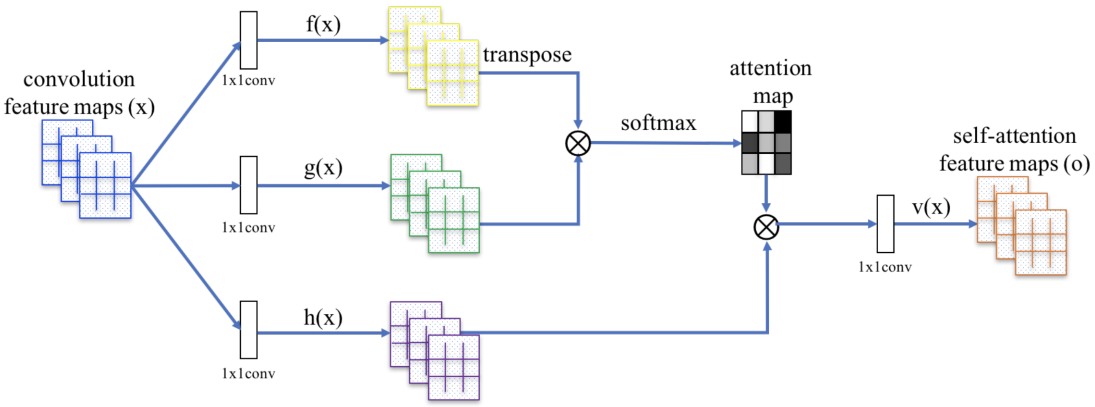


Figure 2.5: Visual expression of self-attention mechanism. Cited from [52]

As shown in Fig. (2.5), self-attention mechanism calculates the relationship between elements in a sequence by assigning a weight to each pair of elements, indicating how much one element contributes to the other. This is done by computing dot product between element representatives (usually vectors) and then normalizing the weights using a soft key operation. The resulting weighted sum of the sequence elements forms the output representation, which captures the dependencies between the elements. Self-attention has become a crucial component in various state-of-the-art natural language processing (NLP) and sequence modeling architectures.

2.3.5 Time Schedule

For diffusion models, in the forward process formulated by Eq. (2.10) and (2.11), one needs to define the time schedule of the noise level β_t for formulating the degree of noise at each time step. There are two time schedules we used in this project.

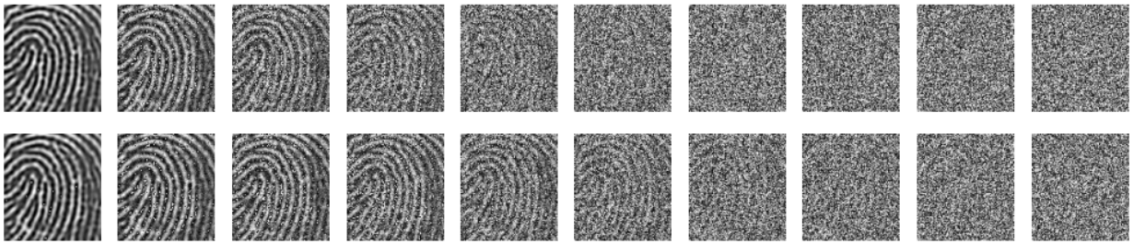


Figure 2.6: The upper one is linear. The lower one is cosine.

Linear Beta Schedule

As shown in the original DDPMs [8], a linear beta schedule was proposed to depict the evolution of the noise level. Suppose the range of β_t is from 0.0001 to 0.02, and the total time step $T = 1000$, a linear beta schedule is given by,

$$\beta_t = \frac{t}{T}(\beta_{max} - \beta_{min}). \tag{2.20}$$

Cosine Beta Schedule

A cosine beta schedule was proposed in the improved DDPMs [53] to smooth the noise schedule and avoid abrupt changes,

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2. \tag{2.21}$$

where s stands for a parameter to prevent β_t from being too small when t converges to 0. From the Eq. (2.21) defining the term $\bar{\alpha}_t$, the definition of β_t is derived as,

$$\bar{\beta}_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}. \tag{2.22}$$

2.3.6 Time Embedding

Similar to word embeddings, time embedding aims to encode and represent the time data into vectors so that it can be interpreted by the machine learning models. In the diffusion models, this technique refers the knowledge of position embedding proposed in the introduction of transformer [51].

Sinusoidal Embedding

In order to represent the information of time steps in diffusion models, a sinusoidal time embedding is utilized to vectorize a series of time steps from e.g., 1 to 1000,

$$\overrightarrow{p}_t^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases}, \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}. \tag{2.23}$$

Eq. (2.23) provides the definition of time embedding, where $\overrightarrow{p}_t^{(i)}$ represents the output vector, and d stands for the length of the vector.

2.4 Metrics

As typical black-box models, deep generative models have been constantly worried about the lack of explainability and transparency. It is therefore necessary to either clarify the internal model structure beforehand or perform all kinds of post hoc explanation methods afterwards.

2.4.1 False Acceptance Rate

In biometric security systems, False Acceptance Rate (FAR) and False Rejection Rate (FRR) are commonly used for assessing the performance of biometric authentication. A FAR indicates the percentage of an unauthorized person being incorrectly accepted by the biometric security system.

In this case, we evaluated the quality of generated fingerprints using the company's algorithm that calculates the score of every pair of real (training) fingerprints, and the score of every pair of generated fingerprints respectively. These scores formed two distributions where we compared the FAR of them, and hence obtained a relatively comparable metric.

2.4.2 Inception Score

Inception Score (IS)[54] is a well-performing algorithm used to assess the quality of images created by a generative model such as a GAN. It tries to learn the statistics from the generated images and correlate with human judgement in some way.

First of all, a group of (typically around 30,000) generated images is fed into a pre-trained Inception v3 image classification model [55] in order to obtain the conditional probability of each generated image $p(\mathbf{y}|x_i)$. In other words, as the pre-trained model is able to classify an input image into a thousand classes, the conditional probability here indicates the probability that the image would be labelled to one of the one thousand classes. The marginal probability $p(\mathbf{y})$ is then computed by averaging all the conditional probabilities in the group.

Thereafter, the inception score is given by calculating the KL divergence of these two distributions,

$$D_{KL} = \frac{1}{N}\sum^{N} p(\mathbf{y}|x_i)\log(\frac{p(\mathbf{y}|x_i)}{p(\mathbf{y})}). \tag{2.24}$$

Intuitively, we hope that the marginal probability and have fairly equal weights on as many classes as possible. On the other hand, we hope the conditional probability can be more distinct (showing high probability in one or two classes but extremely low probability in most classes). We therefore can measure the fidelity and diversity of the generated dataset.

As can be seen in Eq. (2.24), the real dataset for training is not involved in the calculation, which implies one of the disadvantages that Inception Score is not referring the statistics of the real samples.

2.4.3 Fréchet Inception Distance

Fréchet Inception Distance (FID) [56] is a metric used to evaluate the similarity of the synthetic images to the real ones by calculating the Fréchet Distance [57] between two feature vectors. The pre-trained Inception v3 image classification model [55] without the last activation layer is used again to transform two groups of image datasets into two groups of feature vectors, in which each image will be first resized to 256x256 pixels and then be transformed to a 1x2048 feature vector. $F_r = \phi(R)$, $F_g = \phi(G)$ are two groups of feature vectors extracted from real and generated image datasets. We consider that the distributions of F_r, F_g are multivariate Gaussian,

$$F_r \sim N(\mu_r, \Sigma_r); F_g \sim N\left(\mu_g, \Sigma_g\right). \tag{2.25}$$

The difference of two Gaussians is measured by the Frechet distance [57],

$$\mathrm{FID}(R, G) = \left\|\mu_r - \mu_g\right\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r \Sigma_g\right)^{\frac{1}{2}}\right). \tag{2.26}$$

FID has been tested on a variety of deep generative models and large image datasets, and has shown more robust performance than Inception Score. A FID score reveals the similarity between two probability distributions.

2.4.4 Kernel Inception Distance

Kernel Inception Distance (KID) [58] is a metric similar to FID used to evaluate the quality of generated images. It uses the squared MMD between Inception representations (See Eq. (2.27)) in combination with a kernel function to measure the similarity between the distribution of real images and the distribution of generated images.

$$MMD_u^2(X, Y) = \frac{1}{m(m-1)} \sum_{i \neq j}^{m} k\left(x_i, x_j\right) + \frac{1}{n(n-1)} \sum_{i \neq j}^{n} k\left(y_i, y_j\right) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k\left(x_i, y_j\right). \tag{2.27}$$

Unlike FID that relies on Gaussian assumption, KID provides a more flexible measure that only assumes the kernel is suitable enough. KID has proven its remarkable performance compared to FID and IS in [59], and has become a popular tool for evaluating generative models such as generative adversarial networks (GANs) and variational autoencoders (VAEs). KID is also used as a loss function for training generative models, as it provides a measure of how well the generated images match the real images. KID is a powerful evaluation metric that has been shown to correlate well with human judgments of image quality. However, it is important to note that KID is not perfect and may not capture all aspects of image quality, such as diversity and novelty.

2.4.5 Likeness Score

A novel and distance-based measure - Likeness Score [60] was proposed to evaluate GAN-based generated images without projecting images into feature vectors. It calculates the Euclidean distances of data and also the KS distances [61] between distributions.

The definition of LS follows the steps in Algorithm 4 (In Appendix).
Fig. (2.7) illustrates how ICD and BCD sets interpret the quality of generated samples in terms of creativity, diversity and inheritance. Only peaks of BCD showing in the beginning of the histograms indicate the generated samples are highly overlapped with the real samples. This lack of creativity may cause problems of generating identical samples. Having a peak of generated ICD ahead indicates the generated samples follow only a partial distribution of real samples. In addition to that, if the distributions of ICD and BCD sets are not sufficiently matched with each other, the lack of inheritance means the generated samples may not learn from the real samples at all.

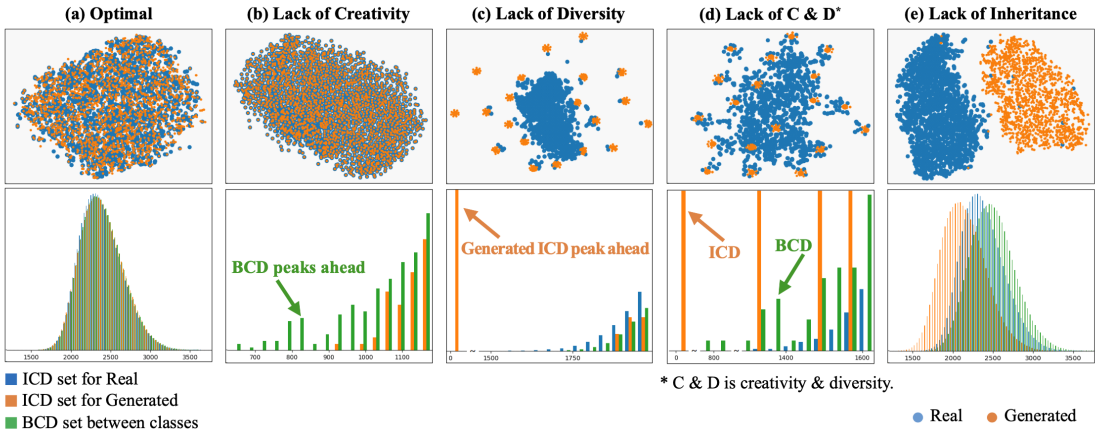


Figure 2.7: Example of how Intra-Class Distance (ICD) and Between-Class Distance (BCD) interpret the quality of generated sample in terms of creativity, diversity and inheritance. First row: the 2D tSNE plots of real (blue) and generated (orange) data points from each virtual GAN. Second row: histograms of ICDs and BCD for real and generated datasets. The histograms in (b)-(d) are zoomed to the beginning of plots. Cited from [60]

2.4.6 Precision and Recall

The FID score is the most commonly used metric to assess the similarity between real and generated images. However, it does not distinguish between fidelity and diversity of the generated images, leading recent papers to introduce variants of precision and recall to evaluate these properties separately. Precision and Recall [62] proposed in 2019 are evaluation metrics that can dependably and independently gauge factors in image generation tasks by creating explicit, non-parametric representations of real and generated data manifolds. The purpose of this metric is to evaluate the quality and coverage of the generated samples. Fig. (2.8) illustrates the definition of precision and recall in a brief way.

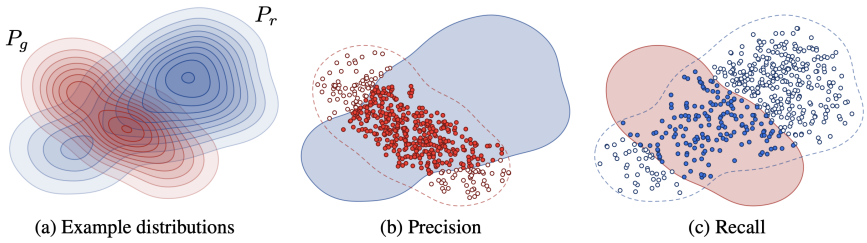


Figure 2.8: Definition of precision and recall. (a) denotes the distribution of real samples with Pr (blue) and the distribution of generated samples with Pg (red). (b) denotes the definition of precision: the percentage of generated samples falling within the distribution of real samples Pr. (c) denotes the definition of recall: the percentage of real samples falling within the distribution of generated samples Pg. Cited from [63]

$$f(\phi, \mathbf{\Phi}) = \begin{cases} 1, \text{ if } \|\phi - \phi'\|_2 \leq \|\phi' - \mathbf{NN}_k(\phi', \mathbf{\Phi})\|_2 \text{ for at least one } \phi' \in \mathbf{\Phi} \\ 0, \text{ otherwise,} \end{cases} \tag{2.28}$$

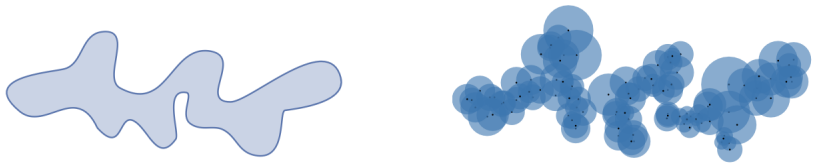


Figure 2.9: The left graph is a true manifold in a feature space. The right graph is an estimate of the real manifold using k-nearest neighbors. Cited from [62]

$$\text{precision}\left(\mathbf{\Phi}_r, \mathbf{\Phi}_g\right) = \frac{1}{\left|\mathbf{\Phi}_g\right|} \sum_{\boldsymbol{\phi}_g \in \mathbf{\Phi}_g} f\left(\boldsymbol{\phi}_g, \mathbf{\Phi}_r\right) \quad \text{recall}\left(\mathbf{\Phi}_r, \mathbf{\Phi}_g\right) = \frac{1}{\left|\mathbf{\Phi}_r\right|} \sum_{\boldsymbol{\phi}_r \in \mathbf{\Phi}_r} f\left(\boldsymbol{\phi}_r, \mathbf{\Phi}_g\right) \tag{2.29}$$

In the Eq. (2.28), ϕ denotes a given generated or real sample, and $\mathbf{\Phi}$ denotes either $\mathbf{\Phi}_r$ or $\mathbf{\Phi}_g$ depending on calculating precision or recall. Basically, this equation calculates if a given sample is included within the manifold (See in Fig. (2.9)) of a distribution estimated by sampling a group of points and surrounding each point with a hypersphere that reaches its kth nearest neighbor. By averaging the relevant binary score for each sample $f(\phi, \mathbf{\Phi})$, the resulting formulas are presented in Eq. (2.29). Notably, all the representations of samples here are obtained by feeding them into a pre-trained VGG-16 classifier [27], similar to what is done in FID and KID.

2.4.7 Density and Coverage

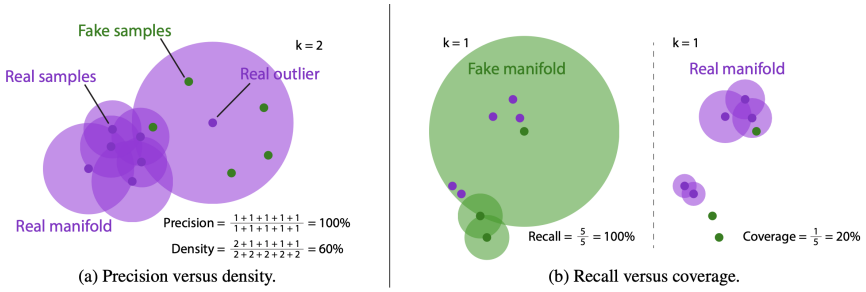


Figure 2.10: Overview of metrics. (a) illustrates the advantage of density against precision on a real manifold, on which a real outlier dramatically influences the accuracy of the precision, while density, a more robust metric, is capable of standing against the negative impact of the real outlier. (b) illustrates the advantage of coverage against recall on a fake manifold, on which a fake outlier tricks the recall by covering all the real samples in one circle. While the coverage uses the real manifold to avoid this fake outlier as the real samples are less likely with outliers. Notably, the real and fake samples are identical across subplots in graph (b). k indicates the k-nearest neighbors. Cited from [64]

Density and Coverage [64] are new evaluation metrics aiming to assess the fidelity and diversity of the generated samples. According to the paper, these new metrics resolve the issues that the latest precision and recall metrics [62] can not handle. For example, they cannot

detect a match between two identical distributions (a precision of 1 would not appear), are susceptible to outliers, and involve arbitrary choices for evaluating hyperparameters.

$$\text{density} := \frac{1}{kM} \sum_{j=1}^{M} \sum_{i=1}^{N} 1_{Y_j \in B(X_i, \text{NND}_k(X_i))}, \qquad \text{coverage} := \frac{1}{N} \sum_{i=1}^{N} 1_{\exists j \text{ s.t. } Y_j \in B(X_i, \text{NND}_k(X_i))}. \tag{2.30}$$

Density is defined in Eq. (2.30) where k indicates the k-nearest neighbors. Density favors samples in areas where real samples are densely clustered, reducing susceptibility to outliers. For instance, the issue of overestimating precision (100%) in Fig. (2.10) is addressed by using the density measure (60%). It is important to know that density does not have an upper limit of 1 and it may exceed 1 depending on the density of real samples surrounding the fake ones. On the other hand, coverage is defined as the percentage of real samples whose manifolds cover at least 1 fake sample and has a fixed interval between 0 and 1.

Chapter 3
Implementation

3.1 Datasets

Initially, our primary focus was on choosing datasets that contained the most readily available fingerprint images in live conditions. As the subject expanded to encompass fingerprint-to-fingerprint transformations, we gradually began to engage in the collection of spoof fingerprint images.

3.1.1 Data Composition

We list the information of all the datasets that were used for live fingerprint synthesis in Tab. (3.1), and fingerprint-to-fingerprint transformation in Tab. (3.2).

Table 3.1: Overview of datasets used for live fingerprint synthesis. The rule of names is: SENSOR-CONDITION-(Optional) SUBSET.

Name	Condition	Amount
S1-L	Live	8622
S2-L	Live	45043
S2-L-1	Live	22609
S2-L-2	Live	22434
S3-L	Live	37138
S4-L	Live	9163
S5-L	Live	9149

Due to the privacy issues, we are not allowed to present the real fingerprints either in live or in spoof conditions in the thesis for most datasets. We are not allowed also to present the real spoof material and mold names. We are however able to present some specific generated

Table 3.2: Overview of datasets used for fingerprint-to-fingerprint transformation. The rule of names is: SENSOR-CONDITION. The spoof images are made of different materials and molds.

Name	Condition	Amount
S6-L	Live	22434
S6-SM1	Spoof Material 1	3000
S6-SM2	Spoof Material 2	3000
S6-SM3	Spoof Material 3	3000
S7-L	Live	14993
S7-SM4	Spoof Material 4	3000

fingerprints here. In that case, models trained on dataset S3-L are allowed to print out the generated results.

3.1.2 Data Pre-processing

Data pre-processing involves cleaning, transforming, and organizing raw data into a structured format that is easily understood by algorithms. This process typically includes handling missing values, converting categorical variables into numerical ones, scaling and normalizing data, and even feature engineering.

In the domain of image processing and computer vision, data pre-processing serves as a crucial step for preparing image data before it is ingested into machine learning models. Images often require pre-processing to enhance their quality, remove noise, and standardize their properties, ensuring a more accurate and robust analysis. Common pre-processing techniques for images include resizing, normalization, data augmentation, and gray-scale conversion. Additionally, more advanced methods such as histogram equalization, edge detection, and feature extraction may be employed to highlight specific characteristics within images.

In order to match the image format with the deep generative models, the original data is first converted from .bmp format into .jpg. Likewise, in general a machine learning model has specific regulation of the pixel range. Hence the input images need to be normalized in a corresponding way, typically ranging from 0 to 1, or -1 to 1.

Considering the consistency of the available datasets, we propose to control all input images to a size of 128x128, using internal padding in the networks, to make the model building easier and more focused. In other cases, for some input images of bigger size, e.g., 192x192, we use a cropping method to extract 128x128 pixels of them.

In addition, considering the potential gains that data enhancement may bring, we do use pytorch's own data augmentation functions to flip, rotate, color enhance, sharpen, etc. on the real data during the training of the model. However, considering that our ultimate goal is to generate synthetic fingerprints similar to the real dataset, changes in color, brightness and contrast may result in a higher diversity of the generated fingerprints, which reduces the similarity of the synthetic fingerprints to some extent. And since we still compare the original dataset with the synthetic dataset when evaluating synthetic fingerprints, without using the augmented data, this makes the scores of each of the measures negatively affected by it.

3.2 Live Fingerprint Synthesis

The first goal of the thesis is to generated live fingerprint images in high fidelity. As written in the first chapter, there have been a few of publications that researched into fingerprint synthesis with either traditional or machine learning methods. We aim to first fulfill fingerprint synthesis with existing deep generative models such as GANs. Besides, we aim to try more implementations in terms of fingerprint synthesis using diffusion models. Our implementations demonstrated the feasibility of the goal. Amongst all the experiments we have done, the WGAN-GP and DDPMs obtained relatively satisfactory results. We therefore present more details below.

3.2.1 WGAN-GP

We adopt the DCGAN architecture for our generative and critic networks from Alec Radford et al. [25].

Table 3.3: Typical model structure of WGAN-GP model.

Model Name	WGAN-GP
Input	Padded to 128x128
$\boldsymbol{\lambda}$	10.0
Batch Size	128
Epochs	1000
Loss	Wasserstein-GP
Optimizer	Adam
Learning Rate	2e-4
$\boldsymbol{\beta_1}$	0.5
$\boldsymbol{\beta_2}$	0.9

The generator network contains four convolutions. Let $b8r256$ denote a $8 \times 8 \times 256$ Dense-BatchNorm-LeakyReLU-Reshape layer. dk denotes a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. R_k denotes a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer. uk denotes a 3×3 UpSampling-Convolution-BatchNorm-LeakyReLU layer with k filters, stride 1 and up sampling 2. The last convolution has a tanh activation function instead of a LeakyRelu. The network consists of:

$$b8r256 - u128 - u64 - u32 - u1$$

The critic network contains six convolutions, which aim to score the similarity between the real and generated image. Let Ck denote a 5×5 Convolution-LeakyReLU layer with k filters and stride 2. Let Cdk denote a 5×5 Convolution-Dropout-LeakyReLU layer with k filters and stride 2. After the last layer, we flatten the result to produce a 1-dimensional output. We do not use InstanceNorm for the first C64 layer. We use leaky ReLUs with a slope of 0.2 and a dropout value of 0.3. The network consists of:

$$C16 - Cd32 - C64 - Cd128 - C256 - Cd512$$

For all the experiments, we set $\lambda = 10$ in $L_{WGAN-GP}$ Eq. (2.3). We use the Adam solver with a batch size of 128. All networks were trained from scratch with a learning rate of 0.0001. A typical setting of WGAN-GP is shown in the Tab. (3.3).

3.2.2 Vanilla DDPM

We initially implemented a plain version of DDPM called Vanilla DDPM in the thesis, in which the deep learning model only consists of a typical U-Net structure within a type of layer normalization and a sigmoid linear unit (SiLU) activation function.

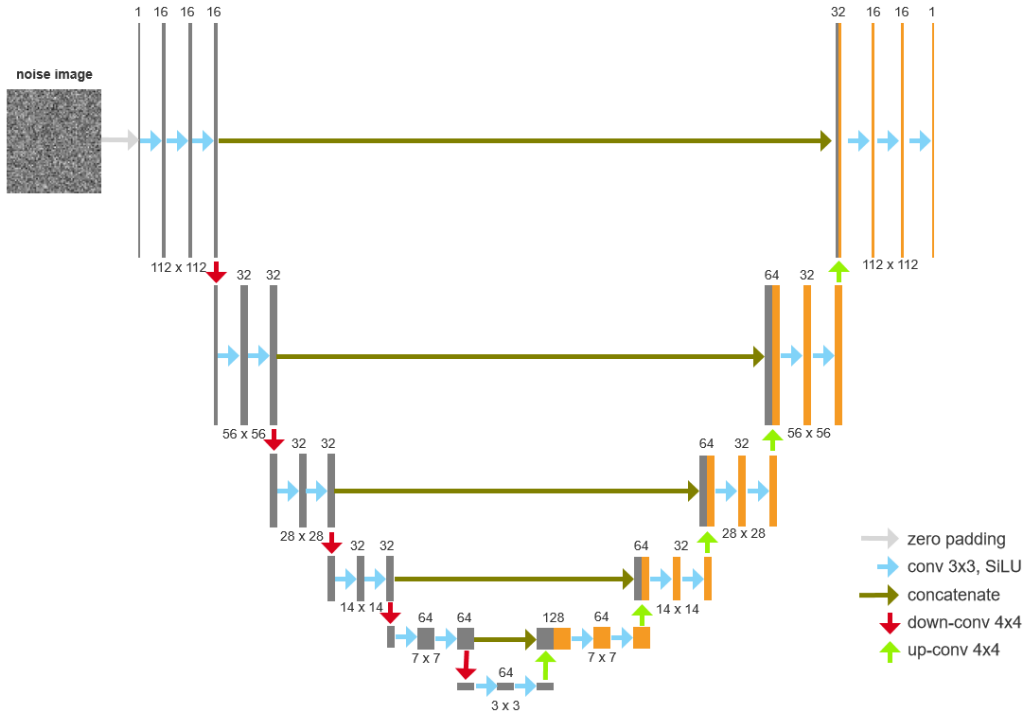


Figure 3.1: The U-Net structure used in vanilla DDPM. This is only a brief picture depicting a rough structure of U-Net. In the code, there exist many more trivial modules in between.

As shown in Fig. (3.1), this is a typical U-Net that can first down-sample the input padded image from 112x112 to 3x3, and then up-sample it back to 112x112. The output image is the estimation of the noise at that time step.

Multiple setups of kernel size and the number of channels have been tried when training the vanilla DDPM. A typical setting of vanilla DDPM is shown in the Tab. (3.4).

3.2.3 DDPM

After achieving good fitting results with the vanilla DDPM, we decided to increase the complexity of the model on top of it, while introducing various practical modules such as at-

Table 3.4: Typical model setting of vanilla DDPM.

Model Name	Vanilla DDPM
Image Size	Padded to 112x112
Time Embedding	Sinusoidal Embedding
Time Schedule	Linear Schedule
Time Steps	1000
Noise Level	[1e-5, 1e-2]
Batch Size	64
Learning Rate	1e-4
Epochs	500
Loss	MSE
Optimizer	Adam

tention, ResNet, etc. Thanks to the meticulous annotations provided by HuggingFace [1], we therefore were able to efficiently build up our model based on their comprehensively fundamental DDPM. As shown in Fig. (3.2), the new DDPM structure in the thesis remains most parts of the standard one propose.

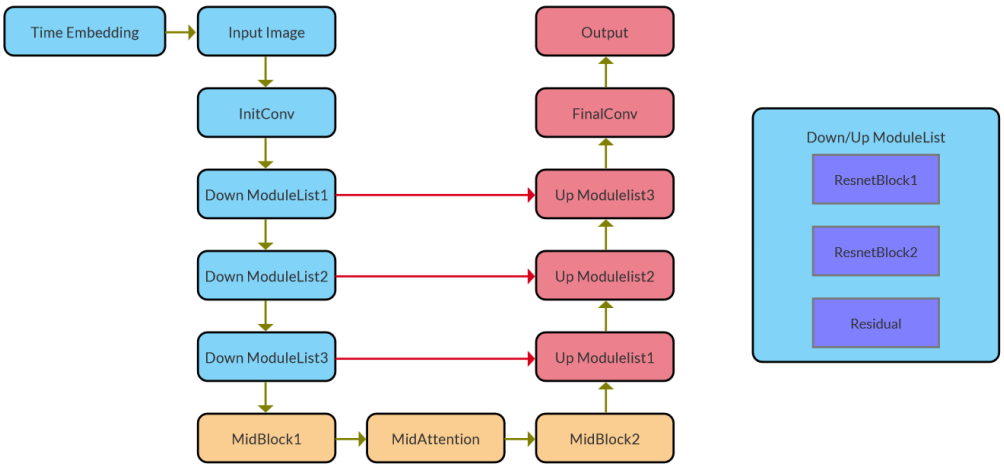


Figure 3.2: The U-Net structure used in DDPM. In order for the model to process more details of the input images, compared to vanilla DDPM, this DDPM deploys a variety of modules such as attention, ResNet, etc. to enrich the complexity of the model while maintaining a small number of model parameters.

3.3 Fingerprint-to-Fingerprint Transformation

Here we are going to explain the network architecture and training details for cycleGAN and cycleWGAN-GP.

[1] Credit to https://github.com/huggingface/blog/blob/main/annotated-diffusion.md

Table 3.5: Typical model setting of DDPM.

Model Name	DDPM
Image Size	Padded to 128x128
Time Embedding	Sinusoidal Embedding
Time Schedule	Cosine Schedule
Time Steps	1000
Noise Level	[1e-5, 1e-2]
Batch Size	64
Learning Rate	1e-4
Epochs	500
Loss	Huber
Optimizer	Adam

3.3.1 CycleGAN

- Network Architecture

  We adopt the architecture for our generative networks from Jun-Yan Zhu et al. [32] who have shown impressive results for style transfer, object transfiguration, season transfer, photo generation from paintings and photo enhancement.

  The generator network inputs different size training images which are padded to 128×128. This network contains three convolutions, six residual blocks [48], two fractionally-strided convolutions with stride $\frac{1}{2}$, and one convolution that maps features to grayscale. Similar as Jun-Yan Zhu et al. [32], we use instance normalization [65]. Below, we follow the naming convention used in the Johnson et al.'s Github repository.

  Let $c7s1-k$ denote a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. dk denotes a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. R_k denotes a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer. uk denotes a 3×3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride 12.

  The network with 6 residual blocks consists of:

  $$c7s1-64, d128, d256, R256, R256, R256, R256, R256, R256, u128, u64, c7s1-3$$

  The discriminator networks contains four convolutions, which aim to classify whether the image is real or fake.

  Let Ck denote a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. After the last layer, we apply a convolution to produce a 1-dimensional output. We do not use InstanceNorm for the first C64 layer. We use leaky ReLUs with a slope of 0.2.

  The discriminator architecture is:

  $$C64 - C128 - C256 - C512$$

- Training details

  We follow the same guidelines as Jun-Yan Zhu et al. [32]. We replace the negative log likelihood objective by a least-squares loss [66] for L_{GAN} Eq. (2.2). This loss have shown being more stable during training and generates higher quality results. For all the experiments, we set $\lambda = 10$ in Equation 3. We use the Adam solver [26] with a batch size of 1. All networks were trained from scratch with a learning rate of 0.0002. We keep the same learning rate for the first 100 epochs and linearly decay the rate to zero over the next 100 epochs.

Table 3.6: Typical model structure of cycleGAN model.

Model Name	cycleGAN
Input	Padded to 128x128
λ_{cycle}	10.0
$\lambda_{identity}$	0.5
Batch Size	1
Epochs	500
Loss	Wasserstein
Optimizer	Adam
Learning Rate	2e-4
β_1	0.5

3.3.2 CycleWGAN-GP

- Network Architecture

  We adopt the architecture for our generative networks from Jun-Yan Zhu et al. [32] who have shown impressive results for style transfer, object transfiguration, season transfer, photo generation from paintings and photo enhancement.

  The generator network inputs different size training images which are padded to 128×128. This network contains three convolutions, six residual blocks [48], two fractionally-strided convolutions with stride $\frac{1}{2}$, and one convolution that maps features to grayscale. Similar as Jun-Yan Zhu et al. [32], we use instance normalization [65]. Below, we follow the naming convention used in the Johnson et al.'s Github repository.

  Let $c7s1-k$ denote a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. dk denotes a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. R_k denotes a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer. uk denotes a 3×3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride 12.

  The network with 6 residual blocks consists of:

$$c7s1-64, d128, d256, R256, R256, R256, R256, R256, R256, u128, u64, c7s1-3$$

The discriminator networks contains four convolutions, which aim to classify whether the image is real or fake.

Let Ck denote a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. After the last layer, we apply a convolution to produce a 1-dimensional output. We do not use InstanceNorm for the first C64 layer. We use leaky ReLUs with a slope of 0.2.

The discriminator architecture is:

$$C64 - C128 - C256 - C512$$

- Training details

We follow the same guidelines as Jun-Yan Zhu et al. [32]. We replace the negative log likelihood objective by a least-squares loss [66] for L_{GAN} Eq. (2.2). This loss have shown being more stable during training and generates higher quality results. For all the experiments, we set $\lambda = 10$ in Equation 3. We use the Adam solver [26] with a batch size of 1. All networks were trained from scratch with a learning rate of 0.0002. We keep the same learning rate for the first 100 epochs and linearly decay the rate to zero over the next 100 epochs.

Table 3.7: Typical model structure of cycleWGAN-GP model.

Model Name	cycleWGAN
Input	Padded to 128x128
λ_{GP}	10.0
λ_{cycle}	10.0
$\lambda_{identity}$	0.5
Batch Size	1
Epochs	500
Loss	Wasserstein
Optimizer	Adam
Learning Rate	2e-4
β_1	0.5

Chapter 4
Result

4.1 Live Fingerprint Synthesis

In this section, we are going to present all the results we obtained during training a variety of deep generative models. Due to the privacy and IP issues, unfortunately, there will not be any real fingerprint images showing up from the private datasets collected by the company, and most generated fingerprint images will not appear in the thesis as well, which means that inevitably we will have to present and discuss our results mostly in numerical tables and figures.

Table 4.1: Evaluation of synthetic fingerprint images generated by 5 different models trained on one large common dataset S2-L with around 45,000 images in total. (C): Using ConvNext Block instead of ResNetBlock. (A): Deploying data augmentation while training. The up arrows indicate larger values are better, and vice versa. vDDPM means vanilla DDPM. The DDPM-080323 outperforms all the other models as it achieved the best performance across all the metrics except the IS which may be of the least interest.

Model name	IS↑	FID↓	KID↓	Precision↑	Recall↑	Density↑	Coverage↑
WGAN-GP-130223	2.53	16.43	0.0136	0.51	0.42	0.37	0.32
vDDPM-170223	**2.74**	15.99	0.0137	0.59	0.50	0.45	0.35
DDPM-080323	2.51	**15.78**	**0.0134**	**0.69**	**0.53**	**0.63**	**0.41**
DDPM-170323 (C)	2.23	42.67	0.0497	0.65	0.56	0.60	0.34
DDPM-250323 (A)	2.42	21.18	0.0211	0.53	0.60	0.39	0.29

As mentioned in the implementation, we experimented plenty of methods to achieve live fingerprint synthesis. As can be seen in Tab. (4.1), following the directions of the arrows, we achieved satisfactory results by using WGAN-GP, and the best scores by using a model called DDPM-080323 which is a standard version of DDPM in combination with U-Net, ResNet, attention architectures, etc. The highest Inception Score (IS) of 2.74 was obtained by WGAN-GP-130223, whereas the lowest FID of 15.78 and KID of 0.0134 plus the highest

Precison, Recall, Density and Coverage (0.69, 0.53, 0.63 and 0.41 respectively) were provided by DDPM-0803223. Interestingly, either utilizing the ConvNext Block or the data augmentation would eventually deteriorate the results. Given all these scores we have achieved so far, it has become significantly difficult for human beings to differentiate the authenticity of the synthetic fingerprint images generated by either WGAN-GP or DDPM.

Table 4.2: Overview of the Inception Score (IS). Left: IS of the real fingerprint datasets. Right: IS of the synthetic fingerprint datasets denoted by model names. A higher IS indicates the fingerprints are more diverse, while it might also indicate there exist more fingerprint images in low quality.

Dataset	IS↑
S1-L	2.29
S2-L	2.92
S3-L	**3.30**
S4-L	3.02
S5-L	3.11

Model Name	IS↑
WGAN-GP-130223	2.53
vDDPM-170223	**2.74**
DDPM-080323	2.51
DDPM-170323 (C)	2.23
DDPM-250323 (A)	2.42

As shown in the Tab. (4.2), the highest IS for real fingerprint datasets and generative models are obtained by S3-L and vDDPM-170223 respectively. We collected all the IS in an isolated table. The reason is that IS does not require the knowledge of the real fingerprint datasets. In other words, the IS only evaluates how many different classes the synthetic fingerprints can be classified into. This may sound a bit absurd as fingerprint denotes only one-class dataset, while we manage to make use of all the existing evaluation methods and IS has been quite significant for many other tasks in image generation. In fact, IS plays the least important role in our evaluation, whereas we may still be beneficial from it if a synthetic fingerprint dataset obtains a IS of 1.5 for example.

Table 4.3: Evaluation of the baseline of all the metrics by hypothesizing one of the real datasets is synthetic. S2-L-1 and S2-L-2 are two real datasets used for the same sensor (indicating the same image configurations but different local details). S2-L-2-1 and S2-L-2-2 contain 10,000 different fingerprint images separately, both collected from S2-L-2. As these metrics are neither built for fingerprints nor perfect for evaluating synthetic data, these results obtained by only real fingerprint datasets formulate the baseline for the evaluation of synthetic fingerprints.

Real-A	Real-B	FID↓	KID↓	Precision↑	Recall↑	Density↑	Coverage↑
S2-L-1	S2-L-2	5.93	0.0015	0.57	0.61	0.43	0.42
S2-L-2-1	S2-L-2-2	**1.40**	**0.00**	**0.86**	**0.86**	**1.00**	**0.87**
S2-L	S1-L	69.24	0.0633	0.24	0.11	0.12	0.06
S2-L	S3-L	154.68	0.1729	0.02	0.04	0.01	0.01
S2-L	S4-L	51.32	0.0515	0.55	0.57	0.42	0.29
S2-L	S5-L	46.59	0.0499	0.56	0.59	0.45	0.31

In order to have a better insight of the metrics, when interpreting the evaluation of the generated images, we recommend to refer the baseline of the metrics in Tab. (4.3) calculated by replacing the synthetic fingerprint images with the real ones. For example, through this experiment, we have the knowledge of what IS the real dataset can obtain, and also the FID between two real datasets. The Inception Score ranging from 2.29 to 3.30, the FID ranging

from 1.40 to 154.68 and the rest scores obtained by real datasets deliver us a message about what our expectations should be in terms of reaching different levels of live fingerprint synthesis. It is worth noting that for the remaining four metrics: Precision, Recall, Density, and Coverage (PRDC), they all range from 0 to 1, except for density, which ranges from 0 to infinity.

As we always emphasize, we are not only assess the quality of the synthetic fingerprint images, but also assess in which ways we should make use of them.

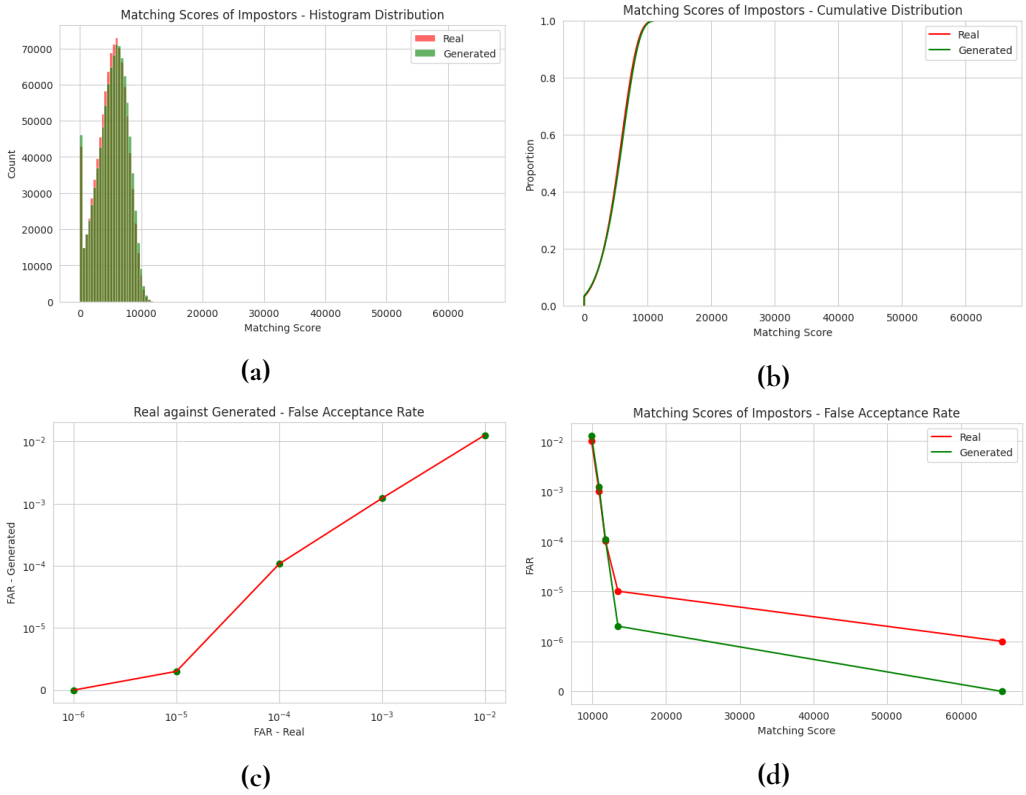


Figure 4.1: Evaluation of the synthetic fingerprints generated by WGAN-GP-130223 using the fingerprint matching algorithm provided by Precise Biometrics. (a) presents the histogram of the matching scores for both real and generated fingerprints. A higher score indicates the two fingerprints contain more structures overlapped or connected. (b) presents the cumulative distribution of the matching scores. (c) and (d) depict a picture where the FAR of the generated ones beat the FAR of the real ones.

Apart from using all those metrics created for evaluating the quality of synthetic data, we also want to explore some methods specific for evaluating fingerprints. In the field of fingerprint recognition, False Acceptance Rate (FAR) and False Rejection Rate (FRR) seem to be very suitable for testing the performance of fingerprint authentication. Likewise, in order to find out whether these generated fingerprints are unique or not, we employed the fingerprint matching algorithm from the company. As shown in Fig. (4.1) and (4.2), the histograms of (a) and (b) clearly depict the strong consistency between real and generated fingerprints in terms of the score distributions. The charts (c) and (d) show the comparison of FARs between real and generated ones. The table of FARs is given in Tab. (4.4).

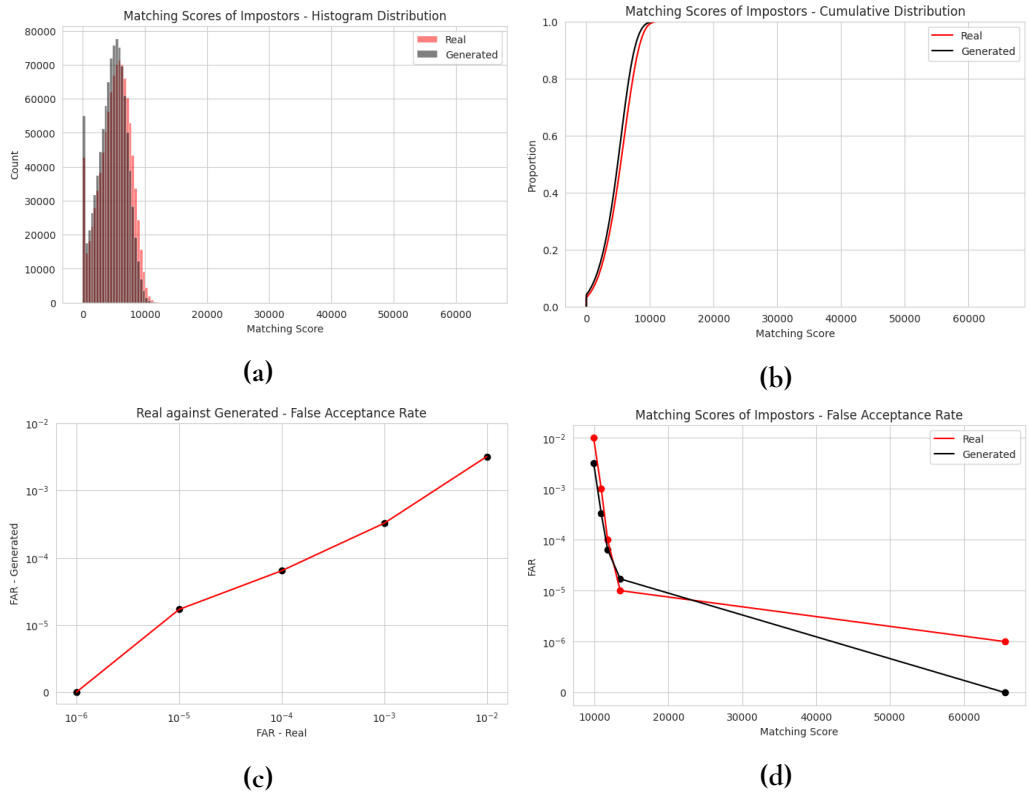


Figure 4.2: Evaluation of the synthetic fingerprints generated by vDDPM-170223 using the fingerprint matching algorithm provided by Precise Biometrics. (a) presents the histogram of the matching scores for both real and generated fingerprints. A higher score indicates the two fingerprints contain more structures overlapped or connected. (b) presents the cumulative distribution of the matching scores. (c) and (d) depict a picture where the FAR of the generated ones beat the FAR of the real ones.

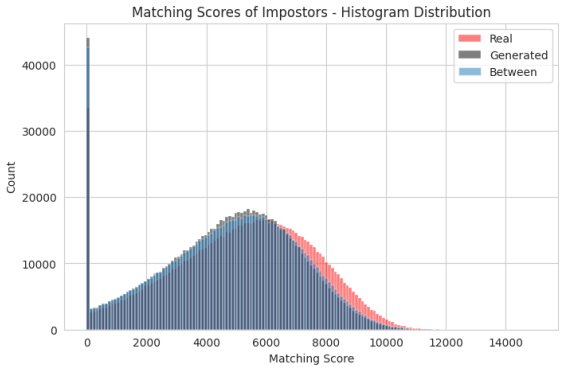


Figure 4.3: A zoom-in graph of histograms obtained by vDDPM-170223. This is a zoom-in version of the sub-figure (a) above. The "Between" histogram is calculated by matching one fingerprint from the real dataset and the other fingerprint from the generated dataset.

Table 4.4: Overview of FARs. The FAR of real is first calculated by setting the threshold of X. The matching scores equal or larger than X would be considered as accepted. If multiple samples share the same value of X, the FAR of real would be slightly larger than the FAR, otherwise they would be identical. Once all the thresholds of X are found, they will be used for calculating the FAR of generated ones.

FAR	X	FAR - Real	FAR - WGAN-GP-130223	FAR - vDDPM-170223
1e-6	65535	1e-6	0	**0**
1e-5	13473	1e-5	**2e-6**	1.7e-5
1e-4	11770	1e-4	1.07e-4	**0.64e-4**
1e-3	10908	1.002e-3	1.217e-3	**0.327e-3**
1e-2	9877	1.0002e-2	1.2606e-2	**0.3209e-2**

Notably, all the FARs were calculated between impostors, which means none of these pairs are supposed to be matched. By taking the FAR of real as standards so as to choose suitable thresholds **X** for each FAR level, we were able to compute the FAR of generated samples.

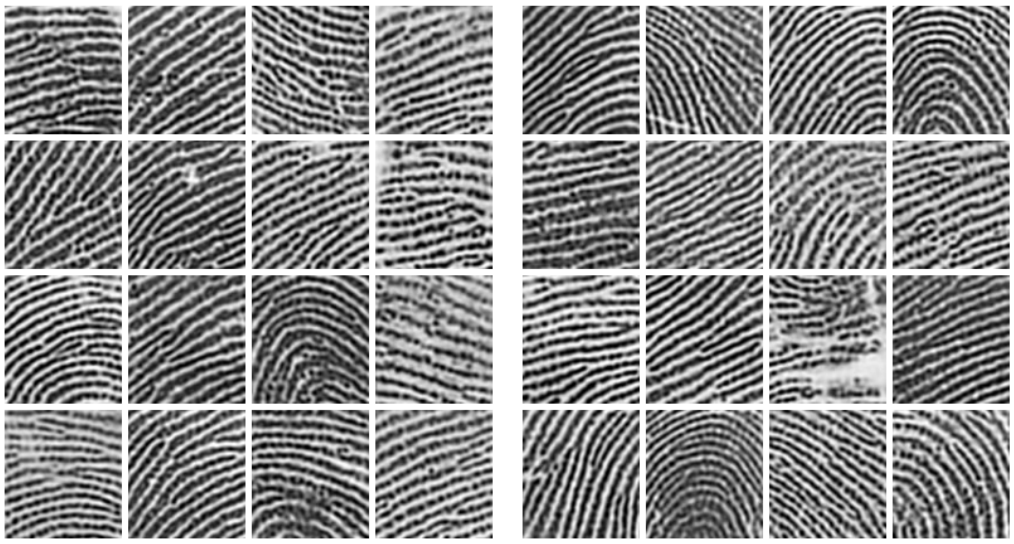


Figure 4.4: Generated live fingerprints at the best epoch from DDPM-050423 trained on S3-L

Fig. (4.4) presents some examples of synthetic live fingerprint images at the best epoch from DDPM-050423, and Fig. (4.5) visualizes the change of synthetic live fingerprint images during training.

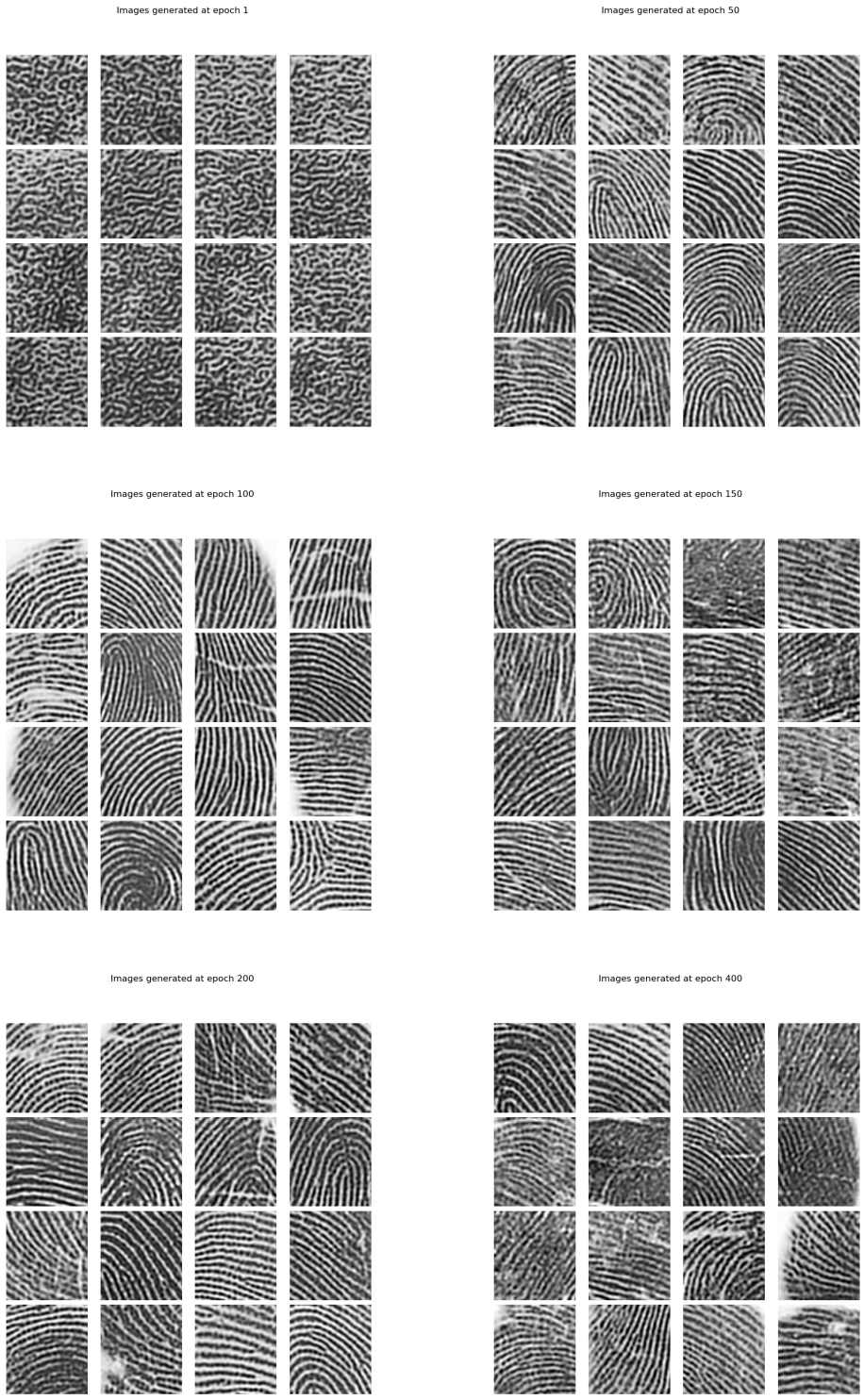


Figure 4.5: Generated live fingerprints at different epochs from DDPM-050423 trained on S3-L

4.2 Fingerprint-to-Fingerprint Transformation

In this section, we are going to present all the results we obtained during training a variety of deep generative models. As mentioned in the section above, unfortunately, there will not be any real fingerprint images showing up from the private datasets collected by the company, and most generated, spoof and live, fingerprint images will not appear in the thesis as well, which means that inevitably we will have to present and discuss our results mostly in numerical terms.

4.2.1 Model results with Spoof Material 1

For the first experiment, we use the cycleGAN architecture with the S6-L and S6-SM1 datasets. The model consist of two cycles: Live-Spoof-Live (LSL) and Spoof-Live-Spoof (SLS). Each cycle makes two transformations, resulting in two generated datasets respectably. In the LSL cycle, we get the Generated spoof (GS) and Cycle live (CL) fingerprints. The GS is the result of transforming the S6-L fingerprints to the S6-SM1 domain, and the CL is the result of transforming the GS fingerprints to the S6-L domain. This completes the LSL cycle. In the SLS cycle, we get the Generated live (GL) and Cycle spoof (CS) fingerprints. The GL is the result of transforming the S6-SM1 fingerprints to the S6-L domain, and the CS is the result of transforming the GL fingerprints to the S6-SM1 domain. This completes the SLS cycle.

The calculated metrics between the real live dataset (S6-L), and the four generated fingerprints datasets and the real spoof dataset (S6-SM1) are recorded in Tab. (4.5).

Table 4.5: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real live (RL)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S6-L and S6-SM1, respectively.

Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real live (RL)	22434	Real spoof (RS)	3150	2.58	**67.46**	0.37	0.58	0.21	0.19
Real live (RL)	22434	Gen spoof (GS)	22434	2.35	**64.61**	0.34	0.60	0.22	0.20
Real live (RL)	22434	Gen live (GL)	3150	2.53	27.44	0.77	0.42	0.73	0.52
Real live (RL)	22434	Cycle live (CL)	22434	2.65	134.03	0.04	0.57	0.02	0.03
Real live (RL)	22434	Cycle spoof (CS)	3150	2.60	168.53	0.01	0.07	0.01	0.01

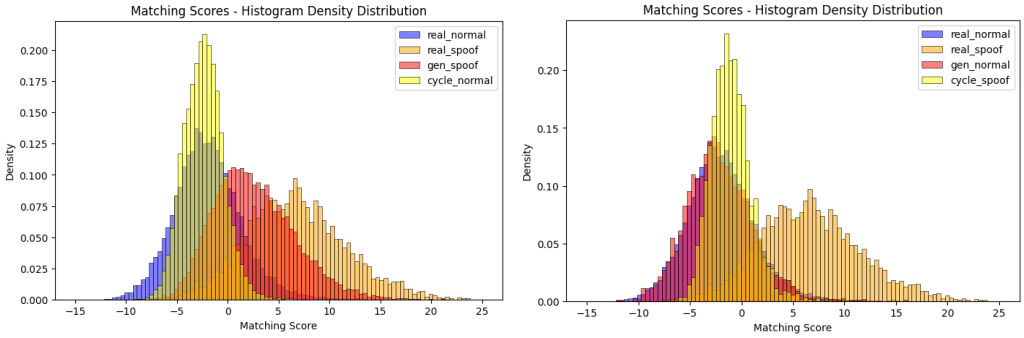
The calculated metrics between the real spoof dataset (S6-SM1), and the four generated fingerprints datasets and the real live dataset (S6-L) are recorded in Tab. (4.6). As in the previous subsection, it is very unfortunate that the resulting images can not be shown over here, we however believe that is very interesting to see how the metrics give us a glimpse in how consistent is the transformation between the two cycle networks.

Apart from using all those metrics created for evaluating the quality of synthetic data, we also want to explore some methods specific for evaluating fingerprints. In order to find out whether the generated fingerprints are similar to the domain target, we employed the fingerprint matching algorithm from the company. As shown in Fig. (4.6), (4.7) and (4.8), the histograms clearly depict the density distribution between real, generated and cycle finger-

Table 4.6: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real spoof (RS)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S6-L and S6-SM1, respectively.

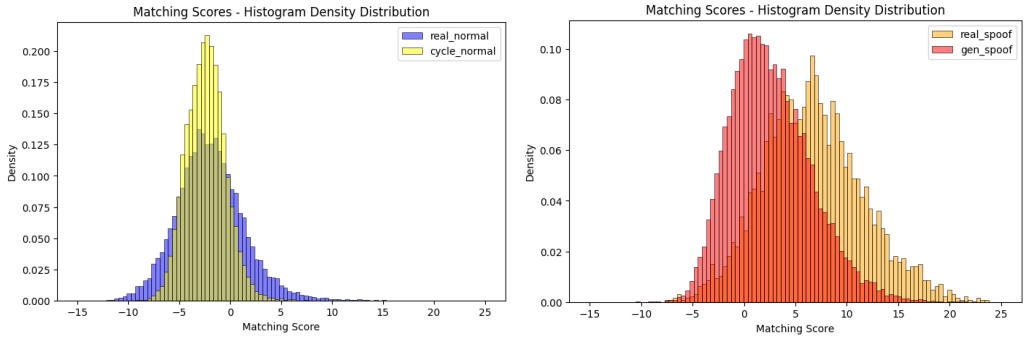
Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real spoof (RS)	3150	Real live (RL)	22434	2.97	**67.46**	0.58	0.37	0.42	0.32
Real spoof (RS)	3150	Gen live (GL)	3150	2.53	**81.53**	0.71	0.17	0.60	0.35
Real spoof (RS)	3150	Gen spoof (GS)	22434	2.35	47.28	0.75	0.46	0.84	0.53
Real spoof (RS)	3150	Cycle spoof (CS)	3150	2.60	132.43	0.07	0.09	0.02	0.01
Real spoof (RS)	3150	Cycle live (CL)	22434	2.65	135.10	0.17	0.36	0.07	0.04

prints in terms of the matching score. The goal of the model is to have the shape distributions of live vs live and spoof vs spoof similar to each other.



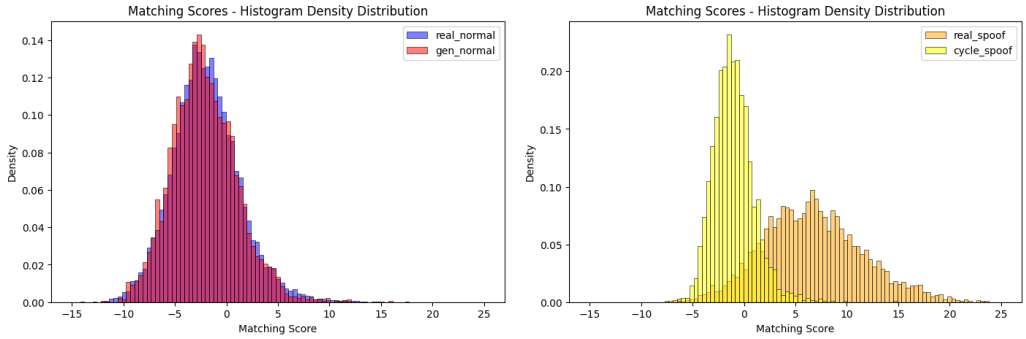
(a) **(b)**

Figure 4.6: Evaluation result with one of Precise Biometrics' spoof scoring functions on the real, generated and cycle fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Histogram density distribution of the Live-Spoof-Live cycle. (b) Histogram density distribution of the Spoof-Live-Spoof cycle.



(a) **(b)**

Figure 4.7: Evaluation result with one of Precise Biometrics' spoof scoring functions on the Live-Spoof-Live cycle for the real live (S6-L) and spoof (S6-SM1) fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Comparison of the histogram density distribution between Real live (RL) and Cycle live (CL) fingerprints. (b) Comparison of the histogram density distribution between Real live (RL) and Gen Spoof (GS) fingerprints.



(a) **(b)**

Figure 4.8: Evaluation result with one of Precise Biometrics' spoof scoring functions on the Spoof-Live-Spoof cycle for the real live (S6-L) and spoof (S6-SM1) fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Comparison of the histogram density distribution between Real live (RL) and Gen live (GL) fingerprints. (b) Comparison of the histogram density distribution between Real live (RL) and Cycle spoof (CS) fingerprints.

4.2.2 Model results with Spoof Material 2

For the second experiment, we use the cycleWGAN-GP architecture with the live (S6-L) and Spoof material 2 (S6-SM2) datasets. The model consist of two cycles: Live-Spoof-Live (LSL) and Spoof-Live-Spoof (SLS). Each cycle makes two transformations, resulting in two generated datasets respectably. In the LSL cycle, we get the Generated spoof (GS) and Cycle live (CL) fingerprints. The GS is the result of transforming the S6-L fingerprints to the S6-SM2 domain, and the CL is the result of transforming the GS fingerprints to the S6-L domain.

This completes the LSL cycle. In the SLS cycle, we get the Generated live (GL) and Cycle spoof (CS) fingerprints. The GL is the result of transforming the S6-SM2 fingerprints to the S6-L domain, and the CS is the result of transforming the GL fingerprints to the S6-SM2 domain. This completes the SLS cycle.

The calculated metrics between the real live dataset (S6-L), and the four generated fingerprints datasets and the real spoof dataset (S6-SM2) are recorded in Tab. (4.7).

Table 4.7: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real live (RL)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S6-L and S6-SM2, respectively.

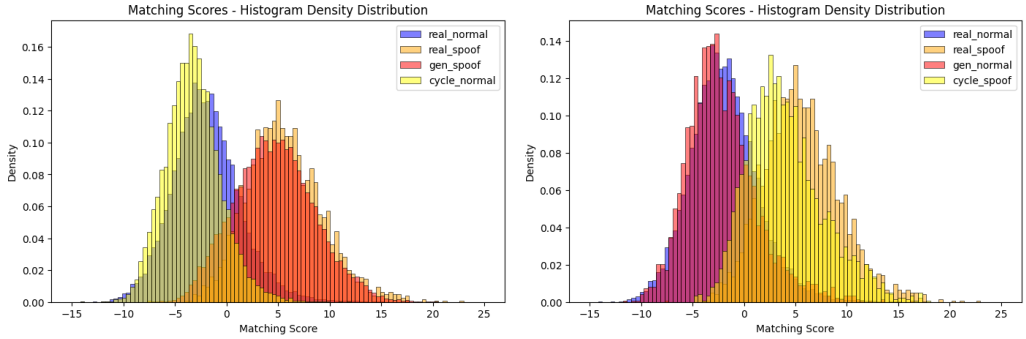
Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real live (RL)	22434	Real spoof (RS)	3065	2.49	**103.86**	0.44	0.33	0.30	0.09
Real live (RL)	22434	Gen spoof (GS)	22434	2.36	**106.80**	0.30	0.56	0.22	0.18
Real live (RL)	22434	Gen live (GL)	3065	2.50	33.22	0.73	0.32	0.62	0.41
Real live (RL)	22434	Cycle live (CL)	22434	2.97	9.18	0.70	0.74	0.55	0.55
Real live (RL)	22434	Cycle spoof (CS)	3065	2.65	89.32	0.37	0.26	0.23	0.09

The calculated metrics between the real spoof dataset (S6-SM2), and the four generated fingerprints datasets and the real live dataset (S6-L) are recorded in Tab. (4.8).

Table 4.8: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real spoof (RS)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S6-L and S6-SM2, respectively.

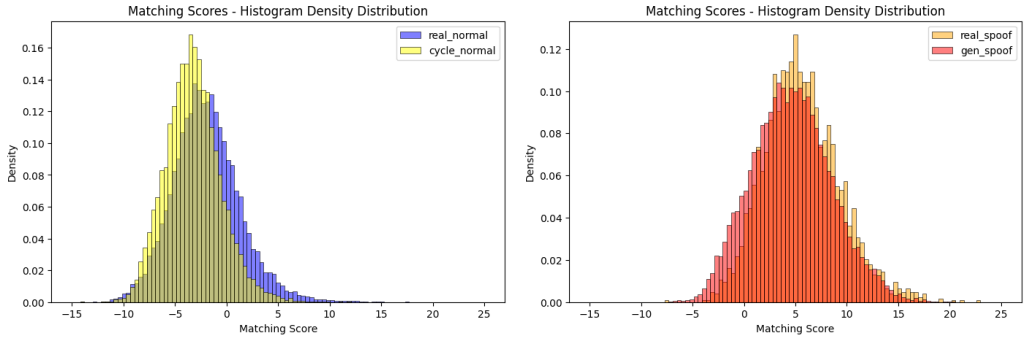
Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real spoof (RS)	3065	Real live (RL)	22434	2.97	**103.86**	0.33	0.44	0.18	0.11
Real spoof (RS)	3065	Gen live (GL)	3065	2.50	**105.99**	0.44	0.04	0.24	0.11
Real spoof (RS)	3065	Gen spoof (GS)	22434	2.36	23.34	0.58	0.53	0.52	0.51
Real spoof (RS)	3065	Cycle spoof (CS)	3065	2.65	13.94	0.86	0.79	0.96	0.78
Real spoof (RS)	3065	Cycle live (CL)	22434	2.97	121.03	0.32	0.32	0.17	0.11

As in the subsection 4.2.1, in order to find out whether the generated fingerprints are similar to the domain target, we employed the fingerprint matching algorithm from the company. As shown in Fig. (4.9), (4.10) and (4.11), the histograms clearly depict the density distribution between real, generated and cycle fingerprints in terms of the matching score. The goal of the model is to have the shape distributions of live vs live and spoof vs spoof similar to each other.



(a) **(b)**

Figure 4.9: Evaluation result with one of Precise Biometrics' spoof scoring functions on the real, generated and cycle fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Histogram density distribution of the Live-Spoof-Live cycle. (b) Histogram density distribution of the Spoof-Live-Spoof cycle.

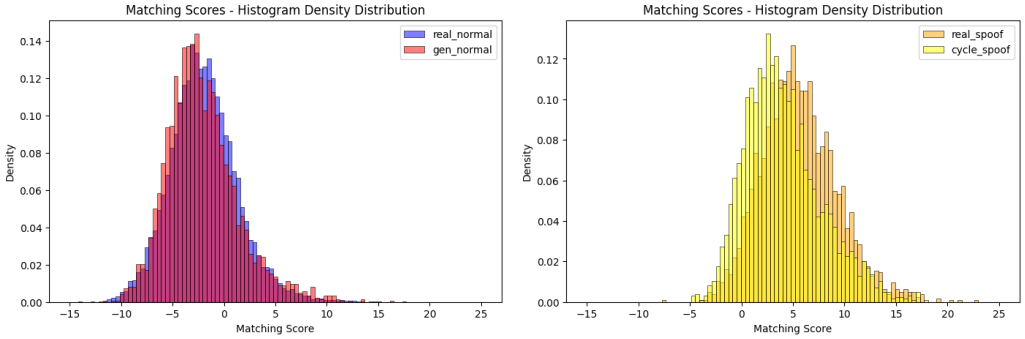


(a) **(b)**

Figure 4.10: Evaluation result with one of Precise Biometrics' spoof scoring functions on the Live-Spoof-Live cycle for the real live (S6-L) and spoof (S6-SM2) fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Comparison of the histogram density distribution between Real live (RL) and Cycle live (CL) fingerprints. (b) Comparison of the histogram density distribution between Real live (RL) and Gen Spoof (GS) fingerprints.

4.2.3 Model results with Spoof Material 3

For the third experiment, we use the cycleWGAN-GP architecture with the live (S6-L) and Spoof material 3 (S6-SM3) datasets. The model consist of two cycles: Live-Spoof-Live (LSL) and Spoof-Live-Spoof (SLS). Each cycle makes two transformations, resulting in two generated datasets respectably. In the LSL cycle, we get the Generated spoof (GS) and Cycle live (CL) fingerprints. The GS is the result of transforming the S6-L fingerprints to the S6-SM3 domain, and the CL is the result of transforming the GS fingerprints to the S6-L domain. This completes the LSL cycle. In the SLS cycle, we get the Generated live (GL) and Cycle spoof (CS) fingerprints. The GL is the result of transforming the S6-SM3 fingerprints to the



(a) **(b)**

Figure 4.11: Evaluation result with one of Precise Biometrics' spoof scoring functions on the Spoof-Live-Spoof cycle for the real live (S6-L) and spoof (S6-SM2) fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Comparison of the histogram density distribution between Real live (RL) and Gen live (GL) fingerprints. (b) Comparison of the histogram density distribution between Real live (RL) and Cycle spoof (CS) fingerprints.

S6-L domain, and the CS is the result of transforming the GL fingerprints to the S6-SM3 domain. This completes the SLS cycle.

The calculated metrics between the real live dataset (S6-L), and the four generated fingerprints datasets and the real spoof dataset (S6-SM3) are recorded in Tab. (4.9).

Table 4.9: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real live (RL)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S6-L and S6-SM3, respectively.

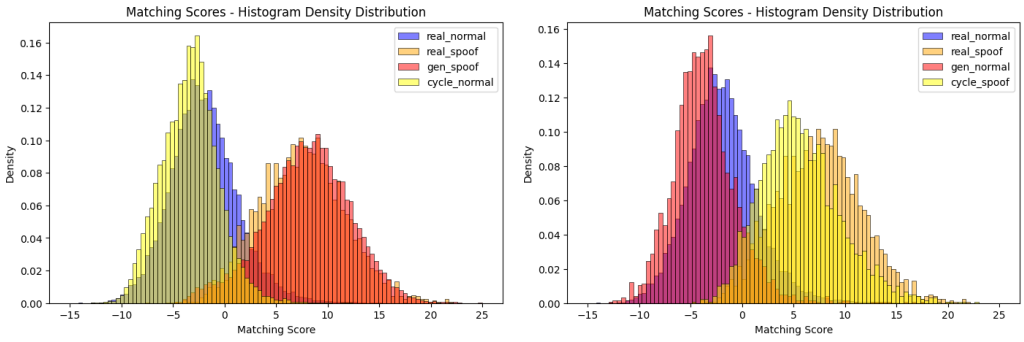
Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real live (RL)	22434	Real spoof (RS)	2993	2.10	**162.70**	0.52	0.39	0.45	0.05
Real live (RL)	22434	Gen spoof (GS)	22434	1.87	**167.54**	0.56	0.34	0.47	0.07
Real live (RL)	22434	Gen live (GL)	2993	2.48	35.46	0.71	0.31	0.60	0.41
Real live (RL)	22434	Cycle live (CL)	22434	2.85	7.69	0.71	0.70	0.60	0.56
Real live (RL)	22434	Cycle spoof (CS)	2993	2.26	159.08	0.35	0.35	0.24	0.04

The calculated metrics between the real spoof dataset (S6-SM3), and the four generated fingerprints datasets and the real live dataset (S6-L) are recorded in Tab. (4.10).

As in the subsection 4.2.1, in order to find out whether the generated fingerprints are similar to the domain target, we employed the fingerprint matching algorithm from the company. As shown in Fig. (4.12), (4.13) and (4.14), the histograms clearly depict the density distribution between real, generated and cycle fingerprints in terms of the matching score. The goal of the model is to have the shape distributions of live vs live and spoof vs spoof similar to each other.

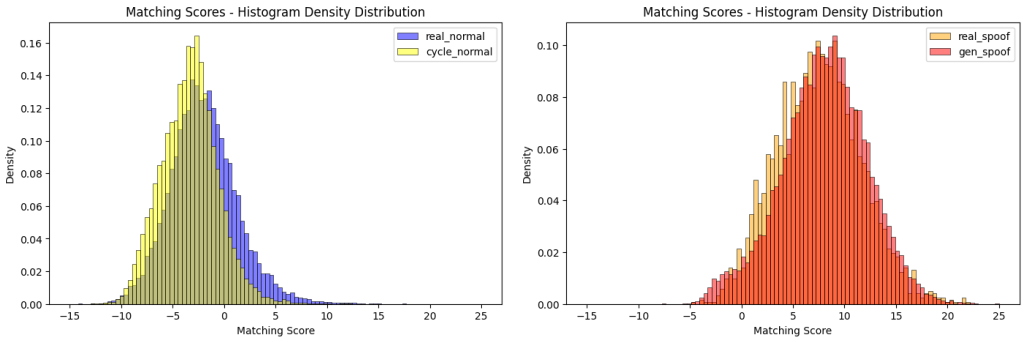
Table 4.10: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real spoof (RS)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S6-L and S6-SM3, respectively.

Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real spoof (RS)	2993	Real live (RL)	22434	2.97	**162.70**	0.39	0.52	0.18	0.04
Real spoof (RS)	2993	Gen live (GL)	2993	2.48	**168.57**	0.54	0.11	0.29	0.03
Real spoof (RS)	2993	Gen spoof (GS)	22434	1.88	16.67	0.66	0.53	0.73	0.57
Real spoof (RS)	2993	Cycle spoof (CS)	2993	2.26	12.50	0.81	0.85	0.79	0.76
Real spoof (RS)	2993	Cycle live (CL)	22434	2.85	174.96	0.40	0.27	0.20	0.03



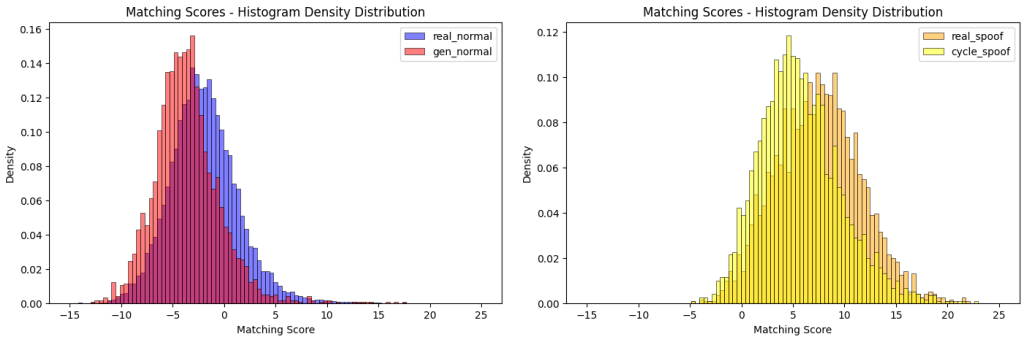
(a) **(b)**

Figure 4.12: Evaluation result with one of Precise Biometrics' spoof scoring functions on the real, generated and cycle fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Histogram density distribution of the Live-Spoof-Live cycle. (b) Histogram density distribution of the Spoof-Live-Spoof cycle.



(a) **(b)**

Figure 4.13: Evaluation result with one of Precise Biometrics' spoof scoring functions on the Live-Spoof-Live cycle for the real live (S6-L) and spoof (S6-SM3) fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Comparison of the histogram density distribution between Real live (RL) and Cycle live (CL) fingerprints. (b) Comparison of the histogram density distribution between Real live (RL) and Gen Spoof (GS) fingerprints.



(a) **(b)**

Figure 4.14: Evaluation result with one of Precise Biometrics' spoof scoring functions on the Spoof-Live-Spoof cycle for the real live (S6-L) and spoof (S6-SM3) fingerprint images. As the size of the datasets vary by a large margin, we choose to use the histogram density distribution for a better analysis. (a) Comparison of the histogram density distribution between Real live (RL) and Gen live (GL) fingerprints. (b) Comparison of the histogram density distribution between Real live (RL) and Cycle spoof (CS) fingerprints.

4.2.4 Model results with Spoof Material 4

For the last experiment, we use the cycleWGAN-GP architecture with the live (S7-L) and Spoof Material 4 (S7-SM4) datasets. The model consist of two cycles: Live-Spoof-Live (LSL) and Spoof-Live-Spoof (SLS). Each cycle makes two transformations, resulting in two generated datasets respectably. In the LSL cycle, we get the Generated spoof (GS) and Cycle live (CL) fingerprints. The GS is the result of transforming the S7-L fingerprints to the S7-SM4 domain, and the CL is the result of transforming the GS fingerprints to the S7-L domain. This completes the LSL cycle. In the SLS cycle, we get the Generated live (GL) and Cycle spoof (CS) fingerprints. The GL is the result of transforming the S7-SM4 fingerprints to the S7-L domain, and the CS is the result of transforming the GL fingerprints to the S7-SM4 domain. This completes the SLS cycle.

The calculated metrics between the real live dataset (S7-L), and the four generated fingerprints datasets and the real spoof dataset (S7-SM4) are recorded in Tab. (4.11).

Table 4.11: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real live (RL)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S7-L and S7-SM4, respectively.

Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real live (RL)	14139	Real spoof (RS)	2564	2.43	**102.86**	0.50	0.23	0.33	0.15
Real live (RL)	14139	Gen spoof (GS)	14139	2.31	**60.68**	0.54	0.55	0.55	0.48
Real live (RL)	14139	Gen live (GL)	2564	2.21	46.21	0.73	0.58	0.82	0.29
Real live (RL)	14139	Cycle live (CL)	14139	2.49	9.21	0.74	0.77	0.70	0.70
Real live (RL)	14139	Cycle spoof (CS)	2564	2.34	108.59	0.58	0.21	0.45	0.16

The calculated metrics between the real spoof dataset (S7-SM4), and the four generated fingerprints datasets and the real live dataset (S7-L) are recorded in Tab. (4.12).

Table 4.12: Evaluation on fingerprint-to-fingerprint transformation between real, generated and cycle fingerprint images with main focus in the *Real spoof (RS)* dataset. The original datasets are *Real live (RL)* and *Real spoof (RS)*. The generated datasets are *Gen spoof (GS)*, *Gen live (GL)*, *Cycle spoof (CS)* and *Cycle live (CL)*. The up arrows indicate larger values are better, and vice versa. The model was trained with a sample size of 1000 live and spoof images of the dataset S7-L and S7-SM4, respectively.

Dataset-A	Amount	Dataset-B	Amount	IS (of B)↑	FID↓	Pre.↑	Rec.↑	Den.↑	Cov.↑
Real spoof (RS)	2993	Real live (RL)	14139	2.47	**102.86**	0.23	0.50	0.13	0.14
Real spoof (RS)	2993	Gen live (GL)	2564	2.22	**105.03**	0.16	0.73	0.11	0.22
Real spoof (RS)	2993	Gen spoof (GS)	14139	2.31	40.69	0.52	0.48	0.41	0.40
Real spoof (RS)	2993	Cycle spoof (CS)	2564	2.34	7.79	0.93	0.90	1.04	0.90
Real spoof (RS)	2993	Cycle live (CL)	14139	2.49	115.04	0.15	0.42	0.08	0.09

Chapter 5
Discussion

5.1 Metrics

To begin with the discussion of the results here, we would like to first elaborate our opinions of the evaluation metrics. Starting from the Inception Score (IS), we prefer to interpret it as a metric to evaluate the diversity of the synthetic fingerprint images. This metric was originally invented by using a large image classifier that was trained on a huge database called ImageNet, in which there were up to 1000 classes of images. Typically, the range of IS is from 1 to 1000, which is determined by the number of classes in the dataset. Considering this feature, Inception Score may not be the best metric here to assess the quality of synthetic fingerprints, as we only have, in this case, one class in the dataset which is the fingerprint. As we have also done some experiments of evaluating the IS of the real fingerprint datasets (See in Tab. (4.3)), most of which obtain a score of around 3, which we believe helps to restrict the range of IS from 1 to 3. The IS seems to be more interesting for the task of live fingerprint synthesis rather than fingerprint-to-fingerprint transformation as the goals of them are different.

The FID and KID are both defined in a similar way to calculate the dissimilarity between two datasets. In our case, the FID score is considered as a very meaningful and the most important metric to evaluate the similarity between two distributions, since one of the prioritized features of synthetic fingerprints is similarity. The lower FID and KID interpret the more similar patterns between real ones and generated ones.

On top of that, the Precision, Recall, Density and Coverage (PRDC) are deemed to be important metrics to further elaborate the relationships between two distributions. As written in the chapter of methodology, by creating the manifolds of the two distributions, one is able to calculate the precision and recall to explain how much real samples are covered by the distribution of generated samples, and how much generated samples are covered by the distribution of real samples respectively. The density and coverage are deemed to be more robust and accurate than the precision and recall in terms of identifying the information of

the two distributions.

Another core feature of synthetic fingerprints is the uniqueness. In order to assess the uniqueness of synthetic fingerprints, we intended to use the company's fingerprint matching algorithm which would calculate the matching score of a pair of fingerprint images from the dataset, and then a False Acceptance Rate (FAR) would be reported for each dataset. By comparing the FAR of real and generated fingerprint datasets while taking the matching scores of "Between" distribution (See Fig. (4.3)) into account, we are able to evaluate the uniqueness of the synthetic fingerprints.

5.2 Live Fingerprint Synthesis

We have seen in previous chapters the worthy performance and potential of deep generative models in the topic of live fingerprint synthesis. However, some details involved in the training as well as image generation process have not been discussed and explained in a targeted manner. Therefore, we will make a comprehensive discussion of all the experiments we have done in the next step.

We initially chose to use the basic DCGAN mainly because previous studies had demonstrated its effectiveness. However, due to the relatively naive model configuration of DCGAN, it did not perform well on our dataset without much tuning. Also, part of the reason may be that we did not explore the depth of the network extensively. Although it is possible to achieve some degree of fingerprint synthesis using DCGAN, as we have been seeking more innovative and diverse generative models, based on our findings with WGAN-GP, we believe that the robustness and fitting ability of WGAN-GP will exceed that of traditional DCGAN. We therefore pivoted our focus toward WGAN-GP.

The truth is, WGAN-GP demonstrated astonishing learning capabilities from the very beginning, with the quality of its generated images being quite difficult to distinguish from real ones compared to DCGAN. We made numerous configurations about its model architecture and parameters afterward, hoping to obtain even higher-quality synthetic images. However, after conducting a multitude of experiments, we found that the training process of WGAN-GP usually converges within the first 50 epochs, i.e., the loss remains stable after 50 epochs, but it is capable of generating fingerprints in good quality around 300 to 500 epochs. This could be a good thing that the gradient optimization of its loss function is highly efficient, but it also presents a potential issue, namely that it might be difficult to further optimize WGAN-GP by exploring its model depth and structure.

Later on, we became more interested in trying the state-of-the-art deep generative models, the diffusion models. We implemented a few types of diffusion models which share very similar structure and configuration to each other. The first issue we found out about diffusion models was its excessive training time. We implemented both WGAN-GP and DDPMs with similar complexity, and the latter took much longer time (approximately 3 to 6 times) to train. In addition to that, DDPMs require a sampling process in 1000 steps to generate images, which again increases the time and computational cost of image generation. Nevertheless, along with the cost, it is worth noting that DDPMs achieve better results of fingerprint synthesis in almost all aspects, and it also avoids the problem of model collapse occurring in WGAN-GP.

As shown in the Tab. (4.2), the best IS of 2.74 was obtained by vDDPM-170223. Although

vDDPM-170223 achieves higher IS than WGAN-GP-130223 across all the metrics, it is beaten by DDPM-080323 again except for the IS which is considered to be of least interest. In the Tab. (4.1), the best results we achieved using WGAN-GP has a rather low FID value of 16.43. At the same time, DDPM-080323 has the lowest FID score of 15.78, which means that it has the highest similarity to the real fingerprint dataset among all other generative models. Interestingly, the FID score calculated between two real datasets used for the same sensor is 5.93 according to the table, which sort of inspires us about the minimum FID score we can get. As proposed in the previous paper [21], the state-of-the-art model, at that moment, generating 256x256 fingerprint images from the public datasets achieved a relatively low FID score of 70.5. Similarly, the smallest KID score is again obtained by DDPM-080323 as well. Speaking of the rest four metrics PRDC, WGAN-GP-130223 still have the potential to improved compared to the baseline of PRDC shown in Tab. (4.3), while DDPM-080323 amazingly outperforms even the real fingerprint datasets in terms of PRDC. Moreover, if we take a closer look at the Fig. (4.1), in sub-figure (a) the histogram of the generated one is located slightly on the right side of the histogram of the real one, which proves that the synthetic fingerprints generated by WGAN-GP-130223 match each other more easily than the real fingerprints in the real dataset. On the contrary, if we look at the sub-figure (a) in Fig. (4.2), a reverse situation happens to vDDPM-170223, which means based on the identical histogram of the real dataset, the synthetic fingerprints generated by vDDPM-170223 are clearly better than the real fingerprints, and much better than the synthetic fingerprints generated by WGAN-GP-130223. This finding is also demonstrated in sub-figure (c) of both figures where the FAR - generated of vDDPM-170223 is much better than the others.

In addition to calculating the FAR within each dataset, it is also crucial to verify if the fingerprint generators are doing copy work. As shown in Fig. (4.3), on top of the real and generated histograms, the added histogram of "Between" depicts a relatively similar distribution that has a high peak at the score of 0, and is highly overlapped with the other two distributions. Given the results obtained from these evaluations, these synthetic fingerprint data has proven its high quality and uniqueness.

5.3 Fingerprint-to-Fingerprint Transformation

We have seen in section 4.2, the performance of the cycleGAN and cycleWGAN-GP models.

In subsection 4.2.1, we focus mainly on the LSL cycle evaluation for RL-RS vs RL-GS in Tab. 4.5. These two comparisons should give us close IS and FID scores to consider GL a successful transformation. In this case, we obtain an IS of 2.58 and FID of 67.46 for RS-RL, and an IS of 2.35 and a FID of 64.61 for RL-GS. This appears to be a good range of similarity between them. This seems surprising as in Fig. 4.6a, the histogram distribution shows that the GS fingerprints are just in the middle of the distributions of RL and RS.

In Fig. 4.7, the distribution of the real and cycle live fingerprints are well aligned to each other. Nevertheless, from the metric evaluation in Tab. 4.5 we got a FID of 134.03 which means the features of the two datasets are different to each other. This also explains the shape difference between the distributions. This can also be corroborated in the SLS cycle in Tab. 4.6 and Fig. 4.8 where we get a FID score of 132.43 and mismatched distributions for the real and cycle spoof fingerprints.

In order to improve these scores, we implemented the Wasserstein loss with gradient

penalty to the cycleGAN model, called cycleWGAN-GP. In subsection 4.2.1 and 4.2.1, we get the results by running the model with the S6-L, S6-SM2 and S6-SM3 datasets. The RL-RS vs RL-GS metric comparison for the LSL cycle is shown in Tab. 4.7 for the S6-SM2 spoof dataset. We get satisfactory results with an IS of 2.49 and FID of 103.86 for RS-RL, and an IS of 2.36 and a FID of 106.80 for RL-GS. Not only that, in Fig. 4.9a, we can see how perfectly aligned in shape and mean are the RS and GS distributions. In a similar way, the RL-RS vs RL-GS metric comparison for the LSL cycle is shown in Tab. 4.9 for the S6-SM3 spoof dataset. We get again astonishing results with an IS of 2.10 and FID of 162.70 for RS-RL, and an IS of 1.87 and a FID of 167.54 for RL-GS. In Fig. 4.12a, we can see how perfectly aligned in shape and mean are the RS and GS distributions. We can agree that this model has achieved a successful transformation for these datasets.

As to further test the capabilities of the cycleWGAN-GP model, we used a more challenging spoof dataset (S7-SM4). As expected, the model struggled to make a clear distinction between the live and spoof domains as shown in Tab. 4.11. We get quite different results from the previous discussed datasets. The RS-RL have an IS of 2.43 and a FID of 102.86 and the RL-GS have an IS of 2.31 and a FID of 60.68, far worse results than with cycleGAN. We can agree that there is still work to be done for this type of spoof fingerprints.

An unexpected finding in the cycleWGAN-GP models was the behavior of the cycle generated fingerprints. As shown in Fig. 4.10, 4.11, 4.13 and 4.14, the cycle images have the characteristic of being less spoof than their real counterparts. Further evaluation is needed to find the changes the model made in the fingerprints to get these results.

In general, we can agree that the cycleWGAN-GP outperforms the cycleGAN model in generation quality and training capabilities. We need to take in account how spoofy the spoof fingerprints are, as there is a direct correlation between the spoofiness and the transformation quality. The higher the spoofiness in the fingerprint is, the better the transformation for the model becomes.

Chapter 6
Conclusion

We have now achieved satisfactory results in the task of live fingerprint synthesis and fingerprint-to-fingerprint transformation, and we have used a variety of evaluation metrics, i.e., Inception Score (IS), Fréchet Inception Distance (FID), Kernel Inception Distance (KID), Precision, Recall, Density and Coverage (PRDC), to assess the quality of the synthetic fingerprint images.

It is natural to suspect the effectiveness of all the metrics used since there does not exist a criterion that can perfectly evaluate the quality of synthetic images in all aspects, let alone fingerprints. When we were utilizing these methods to evaluate the fingerprints, these methods were also evaluated by us simultaneously. We hope that these metrics in some sense validate the quality of the synthetic fingerprints.

In the task of live fingerprint synthesis, both Generative Adversarial Networks (GANs) and diffusion models (e.g., DDPMs) showcase their incredible learning abilities. At the cost of computational power and time, DDPMs achieve the best performance by generating more unique, similar and random fingerprints.

In the task of fingerprint-to-fingerprint transformation, both Cycle Consistent Adversarial Network (cycleGAN) and Cycle Consistent Wasserstein Adversarial Network with Gradient Penalty (cycleWGAN-GP) showcase an incredible performance and creativity while maintaining a reasonable computational cost and training time.

The models have shown us the potential of generating paired images with high quality and fidelity while training with unpaired data.

Now if we look back to the goals we proposed in the beginning, here is the answers to them:

- Which deep generative models perform best in live fingerprints synthesis?

  WGAN-GP has shown its competence in terms of generating very similar synthetic fingerprint images that can hardly be distinguished by humans, while synthetic fingerprints with the best uniqueness, similarity and fidelity are generated by the state-of-the-art DDPMs.

- Which deep generative models can achieve fingerprint-to-fingerprint transformation in sufficient quality?

  CycleWGAN-GP have shown more stable and better results than cycleGAN. The cycle images generated by the former are as similar as possible to the real ones, while the latter removed the noisiness in the fingerprints resulting in untrainable images for the company. Another advantage is the robustness of the cycleWGAN-GP while training with uneven ratio of live and spoof fingerprints. In which case, the cycleGAN mode collapse during training.

  One main comment would be that the effectiveness of the transformation was highly correlated with how spoofy was the fingerprint. As higher spoofiness in the fingerprint, meant a better transformation for either model.

- How to measure the quality of generated fingerprints in practice and in statistics?

  The first step is to visually check the similarity between synthetic fingerprint images and real fingerprint images. Then we recommend to mainly calculate FID to evaluate the similarity, and FAR to evaluate the uniqueness. Overall, IS is not recommended, but PRDC can be considered as extra metrics to comprehend the results from wider perspectives.

We have proven the feasibility of generating synthetic fingerprints in high fidelity by using deep generative models. We believe there is still a great deal of further work that can be done in the future, for instance

- Extend the research of live fingerprint synthesis by applying the existing deep generative models to a wide range of live fingerprint datasets
- Extend the research of fingerprint synthesis by constructing deep generative models to generate fingerprint in more difficult capture conditions (such as cold or dry fingers) and a wider range of spoof materials.
- Implementation of a fingerprint database pipeline which will generate random unique fingerprints and their corresponding spoof variations for creating a complete database not controlled by GDPR policies.

References

[1] Roberto Cipolla et al. Studies in Computational Intelligence: 411. Springer Berlin Heidelberg, 2013. ISBN: 9783642286605.

[2] Chunfeng Lian et al. *Machine Learning in Medical Imaging. 12th International Workshop, MLMI 2021, Held in Conjunction with MICCAI 2021, Strasbourg, France, September 27, 2021, Proceedings.* Image Processing, Computer Vision, Pattern Recognition, and Graphics: 12966. Springer International Publishing, 2021. ISBN: 9783030875886.

[3] M. W. Mak and Jen-Tzung Chien. *Machine learning for speaker recognition.* Cambridge University Press, 2021. ISBN: 9781108428125.

[4] Nikolaos-Ioannis Galanis et al. *Machine Learning Meets Natural Language Processing - The Story so Far.* Vol. 627. IFIP Advances in Information and Communication Technology. 627. Springer International Publishing, 2021. ISBN: 978-3-030-79149-0.

[5] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG].

[6] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML].

[7] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[9] R. Cappelli et al. "Synthetic fingerprint-image generation". In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. Vol. 3. 2000, 471–474 vol.3. DOI: 10.1109/ICPR.2000.903586.

[10] R. Cappelli, D. Maio, and D. Maltoni. "Synthetic fingerprint-database generation". In: *2002 International Conference on Pattern Recognition*. Vol. 3. 2002, 744–747 vol.3. DOI: 10.1109/ICPR.2002.1048096.

[11] Jin Hu et al. "A synthetic fingerprint generation method and its implementation." In: *Journal of Software* 18.3 (2007), pp. 517–526.

[12] Jiali Cui et al. “An iris image synthesis method based on PCA and super-resolution”. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 4. 2004, 471–474 Vol.4. DOI: 10.1109/ICPR.2004.1333804.

[13] Sarvesh Makthal and Arun Ross. “Synthesis of iris images using Markov Random Fields”. In: *2005 13th European Signal Processing Conference*. 2005, pp. 1–4.

[14] R. Cappelli et al. “Fingerprint Image Reconstruction from Standard Templates”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.9 (2007), pp. 1489–1503. DOI: 10.1109/TPAMI.2007.1087.

[15] Mark Rahmes et al. “Fingerprint Reconstruction Method Using Partial Differential Equation and Exemplar-Based Inpainting Methods”. In: *2007 Biometrics Symposium*. 2007, pp. 1–6. DOI: 10.1109/BCC.2007.4430539.

[16] Arun Ross, Jidnya Shah, and Anil K. Jain. “From Template to Image: Reconstructing Fingerprints from Minutiae Points”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.4 (2007), pp. 544–560. DOI: 10.1109/TPAMI.2007.1018.

[17] Jianjiang Feng and Anil K Jain. “FM model based fingerprint reconstruction from minutiae template”. In: *Advances in Biometrics: Third International Conference, ICB 2009, Alghero, Italy, June 2-5, 2009. Proceedings 3*. Springer. 2009, pp. 544–553. ISBN: 978-3-642-01793-3.

[18] Jianjiang Feng and Anil K. Jain. “Fingerprint Reconstruction: From Minutiae to Phase”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.2 (2011), pp. 209–223. DOI: 10.1109/TPAMI.2010.77.

[19] Kai Cao and Anil K. Jain. “Learning Fingerprint Reconstruction: From Minutiae to Image”. In: *IEEE Transactions on Information Forensics and Security* 10.1 (2015), pp. 104–117. DOI: 10.1109/TIFS.2014.2363951.

[20] Peter Johnson, Fang Hua, and Stephanie Schuckers. “Texture Modeling for Synthetic Fingerprint Generation”. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2013, pp. 154–159. DOI: 10.1109/CVPRW.2013.30.

[21] Shervin Minaee and AmirAli Abdolrashidi. “Finger-GAN: Generating Realistic Fingerprint Images Using Connectivity Imposed GAN”. In: *CoRR* abs/1812.10482 (2018). arXiv: 1812.10482. URL: http://arxiv.org/abs/1812.10482.

[22] Kim Hakil et al. “Fingerprint Generation and Presentation Attack Detection using Deep Neural Networks.” In: Inha University, Department of Information and Communication Engineering, 100 Inha, Michuhol, Incheon, 22212, South Korea, 26718, 2019. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=inh&AN=18619188&site=eds-live&scope=site.

[23] Masud An-Nur Islam Fahim and Ho Yub Jung. “A Lightweight GAN Network for Large Scale Fingerprint Generation”. In: *IEEE Access* 8 (2020), pp. 92918–92928. DOI: 10.1109/ACCESS.2020.2994371.

[24] Chenhao Zhong, Pengxin Xu, and Longsheng Zhu. “A deep convolutional generative adversarial network-based fake fingerprint generation method”. In: *2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*. 2021, pp. 63–67. DOI: 10.1109/CEI52496.2021.9574508.

[25] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. DOI: 10.48550/ARXIV.1511.06434. URL: https://arxiv.org/abs/1511.06434.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[27] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[28] O. Striuk and Y. Kondratenko. "Adaptive Deep Convolutional GAN for Fingerprint Sample Synthesis." In: Petro Mohyla Black Sea National University, Intelligent Information Systems Department, Mykolaiv, Ukraine, 2021. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=inh&AN=21503542&site=eds-live&scope=site.

[29] Keivan Bahmani et al. "High Fidelity Fingerprint Generation: Quality, Uniqueness, And Privacy". In: *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, pp. 3018–3022. DOI: 10.1109/ICIP42928.2021.9506386.

[30] André Brasil Vieira Wyzykowski, Mauriécio Pamplona Segundo, and Rubisley de Paula Lemes. "Level Three Synthetic Fingerprint Generation". In: *CoRR* abs/2002.03809 (2020). arXiv: 2002.03809. URL: https://arxiv.org/abs/2002.03809.

[31] Ataher Sams, Homaira Shomee, and S. Rahman. "HQ-finGAN: High-Quality Synthetic Fingerprint Generation Using GANs". In: *Circuits, Systems, and Signal Processing* 41 (July 2022), pp. 1–16. DOI: 10.1007/s00034-022-02089-1.

[32] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. arXiv: 1703.10593 [cs.CV].

[33] Yanming Zhu, Xuefei Yin, and Jiankun Hu. *FingerGAN: A Constrained Fingerprint Generation Scheme for Latent Fingerprint Enhancement*. 2022. arXiv: 2206.12885 [cs.CV].

[34] Steven A. Grosz and Anil K. Jain. "SpoofGAN: Synthetic Fingerprint Spoof Images". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 730–743. DOI: 10.1109/TIFS.2022.3227762.

[35] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. DOI: 10.48550/ARXIV.1704.00028. URL: https://arxiv.org/abs/1704.00028.

[36] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *CoRR* abs/1812.04948 (2018). arXiv: 1812.04948. URL: http://arxiv.org/abs/1812.04948.

[37] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].

[38] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going Deeper into Neural Networks*. 2015. URL: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[39] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[40] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[41] Andrew L. Maas. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: 2013.

[42] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[43] Cédric Villani. "The Wasserstein distances". In: *Optimal Transport: Old and New*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 93–111. ISBN: 978-3-540-71050-9. DOI: 10.1007/978-3-540-71050-9_6. URL: https://doi.org/10.1007/978-3-540-71050-9_6.

[44] Furkan Luleci, F. Necati Catbas, and Onur Avci. *CycleGAN for Undamaged-to-Damaged Domain Translation for Structural Health Monitoring and Damage Detection*. 2022. arXiv: 2202.07831 [cs.LG].

[45] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG].

[46] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2020. DOI: 10.48550/ARXIV.2010.02502. URL: https://arxiv.org/abs/2010.02502.

[47] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[48] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[49] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV].

[50] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[51] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[52] Lilian Weng. *Attention? Attention!* 2018. URL: https://lilianweng.github.io/posts/2018-06-24-attention.

[53] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. DOI: 10.48550/ARXIV.2102.09672. URL: https://arxiv.org/abs/2102.09672.

[54] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. DOI: 10.48550/ARXIV.1606.03498. URL: https://arxiv.org/abs/1606.03498.

[55] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[56] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: (2017). DOI: 10.48550/ARXIV.1706.08500. URL: https://arxiv.org/abs/1706.08500.

[57] Maurice Fréchet. “Sur la distance de deux lois de probabilité”. In: *Comptes Rendus Hebdomadaires des Seances de L Academie des Sciences* 244.6 (1957), pp. 689–692.

[58] Mikołaj Bińkowski et al. *Demystifying MMD GANs*. 2021. arXiv: 1801.01401 [stat.ML].

[59] Qiantong Xu et al. *An empirical study on evaluation metrics of generative adversarial networks*. 2018. arXiv: 1806.07755 [cs.LG].

[60] Shuyue Guan and Murray Loew. “A novel measure to evaluate generative adversarial networks based on direct analysis of generated images”. In: *Neural Computing and Applications* 33.20 (May 2021), pp. 13921–13936. DOI: 10.1007/s00521-021-06031-5. URL: https://doi.org/10.1007%2Fs00521-021-06031-5.

[61] Frank J Massey Jr. “The Kolmogorov-Smirnov test for goodness of fit”. In: *Journal of the American statistical Association* 46.253 (1951), pp. 68–78.

[62] Tuomas Kynkäänniemi et al. *Improved Precision and Recall Metric for Assessing Generative Models*. 2019. DOI: 10.48550/ARXIV.1904.06991. URL: https://arxiv.org/abs/1904.06991.

[63] Mehdi S. M. Sajjadi et al. *Assessing Generative Models via Precision and Recall*. 2018. arXiv: 1806.00035 [stat.ML].

[64] Muhammad Ferjad Naeem et al. *Reliable Fidelity and Diversity Metrics for Generative Models*. 2020. DOI: 10.48550/ARXIV.2002.09797. URL: https://arxiv.org/abs/2002.09797.

[65] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. arXiv: 1607.08022 [cs.CV].

[66] Xudong Mao et al. *Least Squares Generative Adversarial Networks*. 2017. arXiv: 1611.04076 [cs.CV].

Chapter 7
Appendix

7.1 Algorithms

7.1.1 Improved WGAN Algorithm

The algorithm of improved WGAN is shown below.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 7.1: The training algorithm of WGAN-GP. Cited from [35]

7.1.2 Denoising Diffusion Probabilistic Models

The training and sampling algorithms of DDPMs are shown below.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 7.2: The training and sampling algorithms of DDPMs. In the training algorithm, every training sample will be fed into the model with the time embedding that is arbitrarily chosen from a uniform distribution, which means the model also learns how to predict noise at different time steps. In the sampling algorithm, a Gaussian noise sample has to go through the whole denoising process step by step. At each step, a small random noise is added again to enrich the diversity of generated images. Cited from [8]

7.1.3 Likeness Score

The algorithm of likeness score is shown below.

Algorithm 4 The definition of Likeness Score

Explanations

R corresponds to the real dataset; G corresponds to the generated dataset.
The Intra-Class Distance (ICD) set is a set of distances between any two points in the same dataset (R or G).
The Between-Class Distance (BCD) set is a set of distances between any two points from different datasets (R and G).
The Kolmogorov-Smirnov (KS) distance is the maximum distance between two distribution functions.

Steps

1: Calculate the ICD set of R $\{d_r\}$:
$$\{d_r\} = \left\{ \left\| x_i - x_j \right\|_2 \mid x_i, x_j \in R; x_i \neq x_j \right\}$$
2: Calculate the ICD set of G $\left\{d_g\right\}$:
$$\left\{d_g\right\} = \left\{ \left\| x_i - x_j \right\|_2 \mid x_i, x_j \in G; x_i \neq x_j \right\}$$
3: Calculate the BCD set of R and G $\left\{d_{r,g}\right\}$:
$$\left\{d_{r,g}\right\} = \left\{ \left\| x_i - y_j \right\|_2 \mid x_i \in R; y_j \in G \right\}$$
4: Calculate the KS distance between ICD and BCD sets:
$$s_r = KS\left(\{d_r\}, \left\{d_{r,g}\right\}\right)$$
$$s_g = KS\left(\left\{d_g\right\}, \left\{d_{r,g}\right\}\right)$$
where $KS(P, Q) = \sup_x |P(x) - Q(x)|$
5: Calculate the Likeness Score:
$$LS = 1 - DSI(\{R, G\}) = \max\left\{s_r, s_g\right\}$$

7.2 DDPM Structure

The listing below elaborates the model structure of the DDPM.

Listing 7.1: DownSampling

```
(downs): ModuleList(
(0): ModuleList(
  (0): ResnetBlock(
    (mlp): Sequential(
      (0): SiLU()
      (1): Linear(in_features=128, out_features=32, bias=True)
    )
    (block1): Block(
      (proj): Conv2d(20, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (norm): GroupNorm(8, 32, eps=1e-05, affine=True)
      (act): SiLU()
    )
    (block2): Block(
      (proj): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (norm): GroupNorm(8, 32, eps=1e-05, affine=True)
      (act): SiLU()
    )
    (res_conv): Conv2d(20, 32, kernel_size=(1, 1), stride=(1, 1))
  )
  (1): ResnetBlock(
    (mlp): Sequential(
      (0): SiLU()
      (1): Linear(in_features=128, out_features=32, bias=True)
    )
    (block1): Block(
      (proj): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (norm): GroupNorm(8, 32, eps=1e-05, affine=True)
      (act): SiLU()
    )
    (block2): Block(
      (proj): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (norm): GroupNorm(8, 32, eps=1e-05, affine=True)
      (act): SiLU()
    )
    (res_conv): Identity()
  )
  (2): Residual(
    (fn): PreNorm(
      (fn): LinearAttention(
        (to_qkv): Conv2d(32, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (to_out): Sequential(
          (0): Conv2d(128, 32, kernel_size=(1, 1), stride=(1, ...
```

```
      1))
43            (1): GroupNorm(1, 32, eps=1e-05, affine=True)
44          )
45        )
46        (norm): GroupNorm(1, 32, eps=1e-05, affine=True)
47      )
48    )
49    (3): Conv2d(32, 32, kernel_size=(4, 4), stride=(2, 2), ...
      padding=(1, 1))
50  )
```

Listing 7.2: MidBottle

```
1   (mid_block1): ResnetBlock(
2     (mlp): Sequential(
3       (0): SiLU()
4       (1): Linear(in_features=128, out_features=256, bias=True...
      )
5     )
6     (block1): Block(
7       (proj): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, ...
      1), padding=(1, 1))
8       (norm): GroupNorm(8, 256, eps=1e-05, affine=True)
9       (act): SiLU()
10    )
11    (block2): Block(
12      (proj): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, ...
      1), padding=(1, 1))
13      (norm): GroupNorm(8, 256, eps=1e-05, affine=True)
14      (act): SiLU()
15    )
16    (res_conv): Identity()
17  )
18  (mid_attn): Residual(
19    (fn): PreNorm(
20      (fn): Attention(
21        (to_qkv): Conv2d(256, 384, kernel_size=(1, 1), stride...
      =(1, 1), bias=False)
22        (to_out): Conv2d(128, 256, kernel_size=(1, 1), stride...
      =(1, 1))
23      )
24      (norm): GroupNorm(1, 256, eps=1e-05, affine=True)
25    )
26  )
27  (mid_block2): ResnetBlock(
28    (mlp): Sequential(
29      (0): SiLU()
30      (1): Linear(in_features=128, out_features=256, bias=True...
      )
31    )
32    (block1): Block(
33      (proj): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, ...
      1), padding=(1, 1))
34      (norm): GroupNorm(8, 256, eps=1e-05, affine=True)
35      (act): SiLU()
```

```
    )
    (block2): Block(
      (proj): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, ...
  1), padding=(1, 1))
      (norm): GroupNorm(8, 256, eps=1e-05, affine=True)
      (act): SiLU()
    )
    (res_conv): Identity()
  )
```

Listing 7.3: UpSampling

```
(ups): ModuleList(
  (0): ModuleList(
    (0): ResnetBlock(
      (mlp): Sequential(
        (0): SiLU()
        (1): Linear(in_features=128, out_features=128, bias=...
  True)
      )
      (block1): Block(
        (proj): Conv2d(512, 128, kernel_size=(3, 3), stride...
  =(1, 1), padding=(1, 1))
        (norm): GroupNorm(8, 128, eps=1e-05, affine=True)
        (act): SiLU()
      )
      (block2): Block(
        (proj): Conv2d(128, 128, kernel_size=(3, 3), stride...
  =(1, 1), padding=(1, 1))
        (norm): GroupNorm(8, 128, eps=1e-05, affine=True)
        (act): SiLU()
      )
      (res_conv): Conv2d(512, 128, kernel_size=(1, 1), ...
  stride=(1, 1))
    )
    (1): ResnetBlock(
      (mlp): Sequential(
        (0): SiLU()
        (1): Linear(in_features=128, out_features=128, bias=...
  True)
      )
      (block1): Block(
        (proj): Conv2d(128, 128, kernel_size=(3, 3), stride...
  =(1, 1), padding=(1, 1))
        (norm): GroupNorm(8, 128, eps=1e-05, affine=True)
        (act): SiLU()
      )
      (block2): Block(
        (proj): Conv2d(128, 128, kernel_size=(3, 3), stride...
  =(1, 1), padding=(1, 1))
        (norm): GroupNorm(8, 128, eps=1e-05, affine=True)
        (act): SiLU()
      )
      (res_conv): Identity()
    )
```

```
37        (2): Residual(
38          (fn): PreNorm(
39            (fn): LinearAttention(
40              (to_qkv): Conv2d(128, 384, kernel_size=(1, 1), ...
      stride=(1, 1), bias=False)
41              (to_out): Sequential(
42                (0): Conv2d(128, 128, kernel_size=(1, 1), stride...
      =(1, 1))
43                (1): GroupNorm(1, 128, eps=1e-05, affine=True)
44              )
45            )
46            (norm): GroupNorm(1, 128, eps=1e-05, affine=True)
47          )
48        )
49        (3): ConvTranspose2d(128, 128, kernel_size=(4, 4), ...
      stride=(2, 2), padding=(1, 1))
50      )
```

Master's Theses in Mathematical Sciences 2023:E24
ISSN 1404-6342

LUTFMA-3502-2023

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/