Generating images using a Deep Convolutional GANs

Author: Kai Deng

Supervisor: Serhiy Yanchuk

A thesis submitted in partial fulfilment of the
requirements for the degree of
MSc Mathematical Modelling and Machine Learning

School of Mathematical Sciences,
University College Cork,
Ireland

September 2024

Declaration of Authorship

This report is wholly the work of the author, except where explicitly stated otherwise. The source of any material which was not created by the author has been clearly cited.

Date: 25/08/2024

Signature: Kai Deng

Acknowledgements

I would like to thank...

Abstract

A brief overview of the thesis...

Contents

List of Figures

Chapter 1

Introduction

Generative Adversarial Networks (GANs) have emerged as a powerful class of generative models that revolutionize generative modeling by framing it as a game between two networks: a generator network that produces synthetic data from noise and a discriminator network that distinguishes between the generated data and real data [1]. These networks, introduced in 2014, have found applications in various fields, including materials science, radiology, and computer vision [2], [3], [4]. For example, CycleGAN has been effectively applied in the medical field, notably in medical imaging tasks. It has enhanced liver lesion classification through GAN-based synthetic medical image augmentation, surpassing traditional data augmentation methods in sensitivity and specificity [5]. Abdal et al. (2019) demonstrated the effectiveness of StyleGAN in tasks such as image deformation, style transfer [6]. GANs have gained significant attention in the computer vision community due to their ability to generate data without explicitly modeling the probability density function [3].

The purpose of this paper is to gain a comprehensive understanding of Generative Adversarial Networks (GANs). To achieve this, I will utilize the Animal Faces-HQ dataset, which comprises 16,130 high-quality images with a resolution of 512×512 pixels, to train a standard GAN model specifically designed for generating realistic cat pictures.

The Structure of this thesis.

Chapter 2

History Models of Image Generate

Deep Boltzmann Machines

Deep Boltzmann Machines (DBMs) are sophisticated generative models that excel in learning intricate data representations in an unsupervised manner. Unlike traditional feedforward networks, DBMs are structured with multiple layers of stochastic, binary latent variables, where each layer captures progressively abstract features of the input data. This architecture is characterized by fully connected layers, with no direct connections between units within the same layer, which enables DBMs to effectively model complex dependencies inherent in the data [7]. The undirected nature of the connections in DBMs allows for the propagation of uncertainties across layers, enhancing their capability in feature learning and unsupervised pre-training [7][8].

However, the training of DBMs presents significant challenges, primarily due to their reliance on Markov Chain Monte Carlo (MCMC) methods, such as Gibbs sampling, for approximate inference. These methods are essential for estimating gradients and the intractable partition function, which is vital for updating model parameters. The computational overhead introduced by MCMC methods often results in slow convergence and increased complexity, particularly when applied to large-scale datasets [9]. The requirement

for numerous sampling steps to achieve equilibrium further complicates the training process, making DBMs less accessible compared to other generative models, such as Variational Autoencoders (VAEs).

To mitigate these training difficulties, DBMs often utilize layer-wise pre-training, which initializes model parameters before the entire network is fine-tuned. This approach has been shown to enhance the efficiency of the learning process by providing a sensible initialization for the weights and variational inference [10]. Despite these strategies, the inherent complexity associated with MCMC methods continues to pose challenges in scaling DBMs effectively, particularly in comparison to more straightforward generative models like VAEs [9]. Recent advancements have explored alternative training methodologies, including the use of specialized hardware systems to improve the efficiency of sampling tasks, thereby addressing some of the computational hurdles associated with traditional training approaches [9].

Variational Autoencoders

Variational Autoencoders (VAEs) are generative models in machine learning that excel in tasks like image generation and data representation by learning the underlying structure of data through a latent space.

The architecture of VAEs consists of an encoder that maps input data into a latent space, where the latent variables are typically assumed to follow a Gaussian distribution. This assumption simplifies the learning process, as it allows for the use of techniques such as the reparameterization trick, which enables backpropagation through stochastic layers [11]. The decoder then reconstructs the input data from the latent variables, ensuring that the model captures the essential features of the data distribution. The loss function of a VAE combines reconstruction loss with a Kullback-Leibler (KL) divergence term, which regularizes the latent space and encourages the model to learn a smooth and continuous representation [12].

In comparison to DBMs, VAEs offer several advantages, including simpler training dynamics and better scalability. DBMs often require complex sampling methods, such as Gibbs sampling, which can be computationally

intensive and slow to converge. In contrast, VAEs can be trained using standard gradient descent methods, making them more accessible for practitioners [11][13]. Furthermore, the structured latent space of VAEs not only allows for efficient sampling but also supports various applications, such as image generation, anomaly detection, and data imputation, where the ability to interpolate between data points is crucial [14][15].

However, VAEs face limitations in modeling discrete data as they require back-propagation through hidden units, which poses challenges in handling such data types effectively [16]. Despite this limitation, VAEs have been extensively used to represent high-dimensional complex data by learning a low-dimensional latent space in an unsupervised manner [17].

Noise Contrastive Estimation

Noise-Contrastive Estimation (NCE) is a statistical method employed in machine learning, particularly for estimating parameters of unnormalized probabilistic models. The fundamental principle of NCE is to recast the maximum likelihood estimation problem into a binary classification task. In this framework, the model is trained to differentiate between actual data samples and artificially generated noise samples, which are drawn from a known distribution [18]. This transformation simplifies the estimation process and circumvents the computational challenges associated with calculating the intractable partition function, a common hurdle in traditional maximum likelihood estimation methods for unnormalized models [19].

The methodology of NCE involves the introduction of noise samples, which serve as a baseline for comparison against real data. The model is trained to assign higher probabilities to genuine data samples while assigning lower probabilities to noise samples. This approach effectively enables the model to learn the underlying data distribution without the necessity of explicit normalization [20]. The binary classification framework utilized in NCE allows for the application of standard logistic regression techniques, enhancing both the tractability and computational efficiency of the training process [21]. Moreover, NCE has been shown to be particularly advantageous

in large-scale models, such as energy-based models and word embeddings, where the computational burden of normalization can be substantial [22].

Despite its advantages, NCE is not without limitations. The choice of noise distribution is critical to the performance of the model; poorly selected noise distributions can lead to suboptimal parameter estimates and slow convergence rates [19]. Additionally, while NCE simplifies the training of unnormalized models, it may not perform as effectively in scenarios where the noise distribution is challenging to define or when dealing with highly complex data distributions. The empirical observations regarding the importance of noise distribution have been formalized in various studies, underscoring the need for careful consideration in its selection [19].

Chapter 3

Theoretical Background

Objective function of GAN

Generative Adversarial Networks (GANs) consist of a generator G and a discriminator D, both implemented using artificial neural networks. The parametrization of GANs involves defining the network structure of these components and initializing their weights and biases [23]. The success of GANs relies on balancing the training of these two networks, where the G aims to produce samples $G(z)$ that mimic real data distributions $p_{data}(x)$ by input noise z, while the D learns to differentiate between real and generated samples [24]. The training process involves iteratively updating the weights and biases of the networks through adversarial training, where the generator tries to deceive the discriminator, and the discriminator aims to accurately classify samples [25]. The objective function for GAN has the following form:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]. \quad (3.1)$$

It describes the game process between the Generator (G) and the Discriminator (D). Specifically, the formula defines a minimax game between the Discriminator and the Generator.

- $D(G(z))$ is the output of the discriminator for the data $G(z)$ generated

by the generator. Indicating the probability that the discriminator believes that the generated data comes from the real data distribution.

- $D(x)$**:** is the output of the discriminator for the real data x. Indicating the probability that the discriminator believes that the real data x comes from the real data distribution.
- $x \sim p_{data}(x)$**:** means that sample x is drawn from the true data distribution p_{data}.
- $z \sim p_z(z)$**:** means that sample z is drawn from the fake data distribution p_z.
- $\mathbb{E}_x \sim p_{data}(x)[\log D(x)]$**:** means taking the average value of $\log D(x)$ for all samples x on the true data distribution $p_{data}(x)$.
- $\mathbb{E}x \sim p_z(z)[\log(1 - D(x))]$**:** means taking the average value of $\log(1 - D(G(z)))$ for all samples z on the noise distribution $p_z(z)$.

D(G(z)) is the output of the discriminator for the data G(z) generated by the generator, indicating the probability that the discriminator believes that the generated data comes from the real data distribution. D(x) is the output of the discriminator for the real data x, indicating the probability that the discriminator believes that the real data x comes from the real data distribution.

$\mathbb{E}x \sim p_{data}(x)[\log D(x)]$ represents the average value of $\log D(x)$ for all samples x on the true data distribution $p_{data}(x)$. $x \sim p_{data}(x)$ means that sample x is drawn from the true data distribution p_{data}. $\log D(x)$ is the logarithm of the output of the discriminator D for the input x.

$$\max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]. \quad (3.2)$$

The formula (3.2) shows, the discriminator tries to maximize V(D, G). The formula

$$\min_G V(D,G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]. \quad (3.3)$$

shows, the generator tries to minimize V(D, G). This means that the generator tries to generate realistic data G(z) so that the discriminator cannot distinguish them from real data, that is: It is hoped that D(G(z)) is close to 1, so that $\log(1 - D(G(z)))$ is closer to negative infinity. This dynamic equilibrium drives the GAN framework towards generating outputs that closely resemble authentic data, fulfilling the objective of producing realistic data that is challenging to distinguish from real data.

$$\min_G V(D, G) = \mathbb{E}_{z \sim p_z(z)}[-\log(D(G(z)))]. \tag{3.4}$$

In real training process, the formula 3.3 will replace by 3.4. Since, when the discriminator D is strong, the gradient of the $log(1-D(G(z)))$ approaches zero, leading to slow generator training and diminished gradient update impact [26]. This approach ensures that the gradient of the logarithm is large, providing the generator with more effective gradient updates and helping to avoid the vanishing gradient issue [27].

The GAN network parameters play a crucial role in determining the quality and diversity of the generated samples $G(x)$ [23]. The optimization process in GANs typically involves minimizing a min-max function to ensure the G produces samples that are indistinguishable from real data [24]. Maintaining this balance during training is essential for achieving high-quality sample generation [23]. The D's role is to provide feedback to the generator by acting as a critic, guiding the G towards producing more realistic samples [25].

Optimal Discriminator

The training process of Generative Adversarial Networks (GANs) is inherently adversarial. The generator endeavors to create increasingly realistic fake samples to deceive the discriminator, while the discriminator seeks to more accurately distinguish between real and fake samples. To understand how this process works, in the following sections, I will describe the GAN training process from the perspectives of data distribution and mathematical formulation.

Distribution Angle, the name is not good

a simaple diagram show how distribution change in GAN training

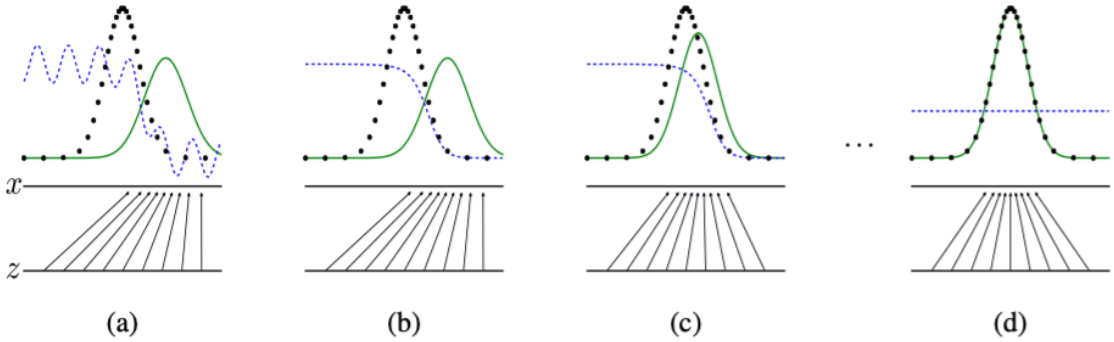


Figure 3.1: Diagram of GAN training process
Source: [28]

- **Green Curve:** Represents the distribution of generated samples. Initially, the distribution of generated samples may differ significantly from the real samples. As training progresses, the generated samples' distribution gradually approaches that of the real samples.
- **Black Dots:** Represent the distribution of real samples. These points remain unchanged throughout the training process and represent the target distribution.
- **Blue Line:** Represents error or noise. Initially, the error is large and manifests as a wavy sine curve. As training progresses, the error gradually decreases, and the blue line becomes increasingly flat.
- **Lines Arranged Below (Labeled as x and z):** Represent the distribution of samples in the latent space. In GAN training, samples from the latent space are mapped to the data space through the generator, producing corresponding samples.

During the training process of the GAN model, the generator and the discriminator fight against each other. The generator tries to produce realistic samples to fool the discriminator, while the discriminator strives to

distinguish real samples from generated samples. Initially, the generator generates samples with poor quality and large errors, similar to the blue sine wave in Figure (a). However, as training proceeds and the generator continues to improve, the error in generated samples gradually decreases, similar to the blue lines in figures (b) and (c) becoming flat. At the same time, the distribution of real samples and generated samples gradually becomes consistent, and finally reaches the optimal state in Figure (d). At this moment $p_{data}(x) = p_g(x)$, the samples generated by the generator are almost the same as the real samples, and the error is minimized. During the entire process, the sample distribution gradually converges from the initial noise and discrete state to a state that highly coincides with the real distribution, reflecting a significant improvement in the quality of the generated samples.

Formula Angle

1. Problem Setup

In a Generative Adversarial Network (GAN), the objective is to train a generator G and a discriminator D to generate data that resembles the real data distribution. The objective function to be maximized is given by:

$$V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{3.5}$$

2. Rewriting the Objective Function

First, the objective function is rewritten in integral form:

$$V(D, G) = \int p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x})\, d\mathbf{x} + \int p_{\mathbf{z}}(\mathbf{z}) \log(1 - D(G(\mathbf{z})))\, d\mathbf{z}. \tag{3.6}$$

By changing variables $\mathbf{x}' = G(\mathbf{z})$, the second term can be rewritten as an integral over the generated data distribution $p_g(\mathbf{x})$:

$$\int p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))\, d\mathbf{x}. \tag{3.7}$$

Thus, the objective function becomes:

$$V(D,G) = \int \left[p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))\right] d\mathbf{x}. \tag{3.8}$$

3. Deriving the Optimal Discriminator

To find the optimal discriminator D^*, it needs to take the derivative of the objective function with respect to $D(\mathbf{x})$ and set it to zero.

Let:

$$f(D(\mathbf{x})) = p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})). \tag{3.9}$$

Taking the derivative with respect to $D(\mathbf{x})$:

$$\frac{d}{dD(\mathbf{x})} f(D(\mathbf{x})) = \frac{p_{\text{data}}(\mathbf{x})}{D(\mathbf{x})} - \frac{p_g(\mathbf{x})}{1 - D(\mathbf{x})}. \tag{3.10}$$

Setting the derivative to zero:

$$\frac{p_{\text{data}}(\mathbf{x})}{D(\mathbf{x})} = \frac{p_g(\mathbf{x})}{1 - D(\mathbf{x})}. \tag{3.11}$$

Solving this equation:

$$p_{\text{data}}(\mathbf{x})(1 - D(\mathbf{x})) = p_g(\mathbf{x})D(\mathbf{x}), \tag{3.12}$$

$$p_{\text{data}}(\mathbf{x}) - p_{\text{data}}(\mathbf{x})D(\mathbf{x}) = p_g(\mathbf{x})D(\mathbf{x}), \tag{3.13}$$

$$p_{\text{data}}(\mathbf{x}) = p_{\text{data}}(\mathbf{x})D(\mathbf{x}) + p_g(\mathbf{x})D(\mathbf{x}), \tag{3.14}$$

$$p_{\text{data}}(\mathbf{x}) = D(\mathbf{x})(p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})), \tag{3.15}$$

$$D(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{3.16}$$

4. Optimal Discriminator Formula

Therefore, the optimal discriminator D^* is given by:

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{3.17}$$

- When $p_{\text{data}}(x)$ is much larger than $p_g(x), D_G^*(x) \approx 1$, indicating that the data point is almost certainly from the real data.
- When $p_{\text{data}}(x)$ is much smaller than $p_g(x), D_G^*(x) \approx 0$, indicating that the data point is almost certainly from the generated data.
- When $p_{\text{data}}(x)$ is close to $p_g(x), D_G^*(x) \approx 0.5$, indicating that the data point has a 50

5. Verifying the Optimal Discriminator

To verify that this D^* maximizes the objective function, substitute D^* back into the objective function:

$$V(D^*, G) = \int \left[p_{\text{data}}(\mathbf{x}) \log \left(\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right) + p_g(\mathbf{x}) \log \left(1 - \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right) \right] d\mathbf{x}. \tag{3.18}$$

Since:

$$1 - D^*(\mathbf{x}) = 1 - \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} = \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}, \tag{3.19}$$

substituting this in:

$$V(D^*, G) = \int \left[p_{\text{data}}(\mathbf{x}) \log \left(\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right) \right] d\mathbf{x}. \tag{3.20}$$

This objective function represents the negative of the cross-entropy, which is maximized when $D(\mathbf{x}) = D^*(\mathbf{x})$. The cross-entropy is minimized by maximizing the similarity between the true distribution and the predicted distribution, so its negative is maximized at the same point.

6. Conclusion

Through the above derivation, it has shown that given the generator G, the optimal form of the discriminator D is:

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{3.21}$$

This demonstrates that the optimal discriminator D^* outputs the probability that the input data comes from the real data distribution. This formula provides a theoretical foundation for training GANs, guiding the updates to the generator G so that its generated data gradually approaches the real data distribution.

Measuring performance of GAN

Fréchet Inception Distance (FID) is a metric commonly used in Generative Adversarial Network (GAN) models to quantify the dissimilarity between two image distributions [29]. It measures the distance between the distributions of real images and generated images, providing a numerical assessment of the quality of generated images. FID has gained prominence in evaluating the performance of GANs due to its ability to capture both the quality and diversity of generated images [30].

A lower FID value indicates that the distribution of the generated images is closer to that of real images, reflecting higher quality and diversity in the generated images [31]. Specifically, a FID value below 10 is considered to represent very high-quality generated images, while values between 10 and 50 indicate good quality, and values above 50 suggest average or poor quality [31].

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}) \tag{3.22}$$

$$\mu_r = \frac{1}{N}\sum_{i=1}^{N} f(x_i), \quad \Sigma_r = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - \mu_r)(f(x_i) - \mu_r)^T \tag{3.23}$$

$$\mu_g = \frac{1}{M}\sum_{i=1}^{M} f(G(z_i)), \quad \Sigma_g = \frac{1}{M}\sum_{i=1}^{M}(f(G(z_i)) - \mu_g)(f(G(z_i)) - \mu_g)^T \quad (3.24)$$

- μ_r **and** μ_g**:** are the feature means of the real and generated images, respectively.
- Σ_r **and** Σ_g**:** are the feature covariance matrices of the real and generated images, respectively.
- **Tr:** represents the trace (the sum of the diagonal elements of the matrix).

Why Not Accuracy

Accuracy is not suitable for evaluating GANs because accuracy is an indicator of classification tasks, which is used to measure the prediction accuracy of the model in classification tasks, and cannot measure the quality and diversity of generated data. In generation tasks, there is no clear "correct answer" and the generated data has no "real label", so it is impossible to directly compare the correspondence between the generated data and a real sample.

The following is a result for a standard GANs model with high accuracy but generate low quality images.

```
1900 [D loss: 9.02800047697383e-06 | D accuracy: 100.0] [G loss: 0.0005307616665959358] [FID: -1.49196970922936e+87]
2/2 [==============================] - 0s 28ms/step
1901 [D loss: 7.511995590903098e-06 | D accuracy: 100.0] [G loss: 0.0006589822005480528] [Epoch time: 0.51 seconds]
2/2 [==============================] - 0s 29ms/step
1902 [D loss: 9.099765065911924e-06 | D accuracy: 100.0] [G loss: 0.0008528590551577508] [Epoch time: 0.48 seconds]
2/2 [==============================] - 0s 29ms/step
1903 [D loss: 7.715634183114162e-06 | D accuracy: 100.0] [G loss: 0.0008156942203640938] [Epoch time: 0.48 seconds]
2/2 [==============================] - 0s 29ms/step
1904 [D loss: 7.386817742371932e-06 | D accuracy: 100.0] [G loss: 0.0005978870904073119] [Epoch time: 0.49 seconds]
2/2 [==============================] - 0s 29ms/step
1905 [D loss: 1.5066923879203387e-05 | D accuracy: 100.0] [G loss: 0.0014342099893838167] [Epoch time: 0.48 seconds]
2/2 [==============================] - 0s 29ms/step
1906 [D loss: 1.9851730939990375e-05 | D accuracy: 100.0] [G loss: 0.0007973920437507331] [Epoch time: 0.49 seconds]
2/2 [==============================] - 0s 29ms/step
1907 [D loss: 1.692915657258709e-05 | D accuracy: 100.0] [G loss: 0.0009673223830759525] [Epoch time: 0.49 seconds]
2/2 [==============================] - 0s 29ms/step
1908 [D loss: 1.5433080079674255e-05 | D accuracy: 100.0] [G loss: 0.0010331417433917522] [Epoch time: 0.49 seconds]
2/2 [==============================] - 0s 29ms/step
1909 [D loss: 3.74487547105673e-06 | D accuracy: 100.0] [G loss: 0.0008501751581206918] [Epoch time: 0.52 seconds]
```

Figure 3.2: Training accuracy

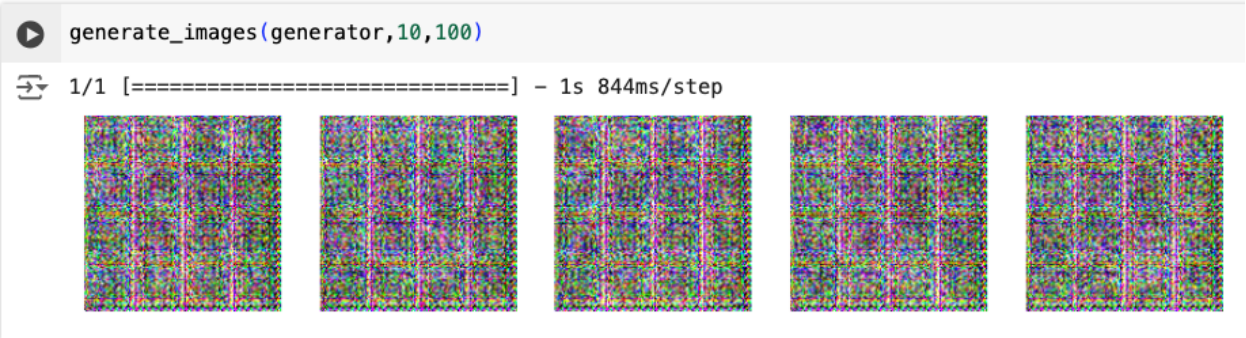


Figure 3.3: Generate images

FID (Fréchet Inception Distance) quantifies the difference between generated data and real data by comparing their distribution in the Inception network feature space. FID takes into account the overall distribution of generated data and real data, and can reflect the quality and diversity of generated data. A low FID value indicates that the distribution of generated data is very close to the distribution of real data, that is, the generated data is both realistic and diverse, so FID is a more suitable indicator for evaluating the performance of generative models.

Chapter 4

Experiments

Model select

Approximately a decade has passed since Goodfellow introduced Generative Adversarial Networks (GANs), during which numerous variants of GAN models have been developed. For the purpose of this study, I have selected seven distinct GAN models for examination and minimal implementation: Standard GANs, Conditional GANs, Auxiliary Classifier GANs, Cycle GANs, Domain Transfer Network GANs, Coupled GANs, and Style GANs. The primary criterion for selection is training time, with only one model chosen for further in-depth analysis. Due to the extensive training time required, I ultimately selected Standard GANs for detailed study.

Model Structure

In this step, I compared two types of standard GANs. The first model utilized dense layers, while the second model employed convolutional layers. The structure of the generator and discriminator was modified accordingly, while all other hyperparameters were kept constant. Both models were trained on the MNIST dataset for 1000 epochs. Upon comparing the generated images, it was observed that the GAN with the convolutional neural network (CNN) architecture outperformed the one with the dense layer architecture

in terms of image quality. Consequently, I selected the standard GAN model implemented with convolutional layers for further experiments.

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 256)	25856
leaky_re_lu_22 (LeakyReLU)	(None, 256)	0
batch_normalization_18 (BatchNormalization)	(None, 256)	1024
dense_31 (Dense)	(None, 512)	131584
leaky_re_lu_23 (LeakyReLU)	(None, 512)	0
batch_normalization_19 (BatchNormalization)	(None, 512)	2048
dense_32 (Dense)	(None, 1024)	525312
leaky_re_lu_24 (LeakyReLU)	(None, 1024)	0
batch_normalization_20 (BatchNormalization)	(None, 1024)	4096
dense_33 (Dense)	(None, 784)	803600
reshape_6 (Reshape)	(None, 28, 28, 1)	0

Total params: 1493520 (5.70 MB)
Trainable params: 1489936 (5.68 MB)
Non-trainable params: 3584 (14.00 KB)

(a) Generator with dense layer

Layer (type)	Output Shape	Param #
dense_38 (Dense)	(None, 6272)	633472
leaky_re_lu_28 (LeakyReLU)	(None, 6272)	0
reshape_8 (Reshape)	(None, 7, 7, 128)	0
batch_normalization_24 (BatchNormalization)	(None, 7, 7, 128)	512
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 128)	262272
leaky_re_lu_29 (LeakyReLU)	(None, 14, 14, 128)	0
batch_normalization_25 (BatchNormalization)	(None, 14, 14, 128)	512
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 64)	131136
leaky_re_lu_30 (LeakyReLU)	(None, 28, 28, 64)	0
batch_normalization_26 (BatchNormalization)	(None, 28, 28, 64)	256
conv2d (Conv2D)	(None, 28, 28, 1)	3137

Total params: 1031297 (3.93 MB)
Trainable params: 1030657 (3.93 MB)
Non-trainable params: 640 (2.50 KB)

(b) Generator with convelution layer

Figure 4.1: The structure for generator

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_23 (Dense)	(None, 512)	401920
leaky_re_lu_17 (LeakyReLU)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_24 (Dense)	(None, 256)	131328
leaky_re_lu_18 (LeakyReLU)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_25 (Dense)	(None, 1)	257

Total params: 533505 (2.04 MB)
Trainable params: 533505 (2.04 MB)
Non-trainable params: 0 (0.00 Byte)

(a) Discriminator with dense layer

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu_39 (LeakyReLU)	(None, 14, 14, 64)	0
dropout_6 (Dropout)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_40 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_7 (Dropout)	(None, 7, 7, 128)	0
flatten_3 (Flatten)	(None, 6272)	0
dense_47 (Dense)	(None, 1)	6273

Total params: 80769 (315.50 KB)
Trainable params: 80769 (315.50 KB)
Non-trainable params: 0 (0.00 Byte)

(b) Discriminator with convelution layer

Figure 4.2: The structure for discriminator

(a) Images generate by dense structure

(b) Images generate by convelution structure

Figure 4.3: Model performance 1000 epochs

Explore More Layers

In this section, I explored the impact of changing convolutional layers in the generator and discriminator of a GAN.

Firstly, I trained the basic convolutional GAN model three times and recorded the FID scores. Secondly, based on the basic structure, I gradually added convolutional layers to the generator, from one to three layers, while keeping the discriminator structure fixed. I trained the model three times for each configuration and recorded the FID scores.

Finally, using the base GAN with three additional convolutional layers in the generator, I gradually added convolutional layers to the discriminator, from one to three layers, while keeping the generator structure fixed. I trained the model three times for each configuration and recorded the FID scores.

Upon comparing, I found that increasing the number of layers in either the generator or the discriminator alone worsened the model's performance. This imbalance disrupts the dynamic equilibrium between the generator and discriminator in the GAN framework. However, simultaneously increasing the layers in both the generator and the discriminator improved the model's performance. This finding aligns with the concept of maintaining a balance between the generator and discriminator during training, as highlighted in

the literature [32]. The importance of this balance is crucial for the effective operation of GANs, ensuring stable learning and improved image quality [33].

Table 4.1: FID scores for different structure

	1st training	2nd training	3rd training
Basic structure	68.27	76.33	59.49
Add 1 convolution layer in generator	77.25	81.14	78.46
Add 2 convolution layer in generator	90.14	85.96	121.42
Add 3 convolution layer in generator	151.55	147.53	284.91
Add 3 convolution layer in generator Add 1 convolution layers in discriminator	70.72	72.97	70.93
Add 3 convolution layer in generator Add 2 convolution layers in discriminator	37.41	42.45	31.43
Add 3 convolution layer in generator Add 3 convolution layers in discriminator	36.75	35.87	34.72

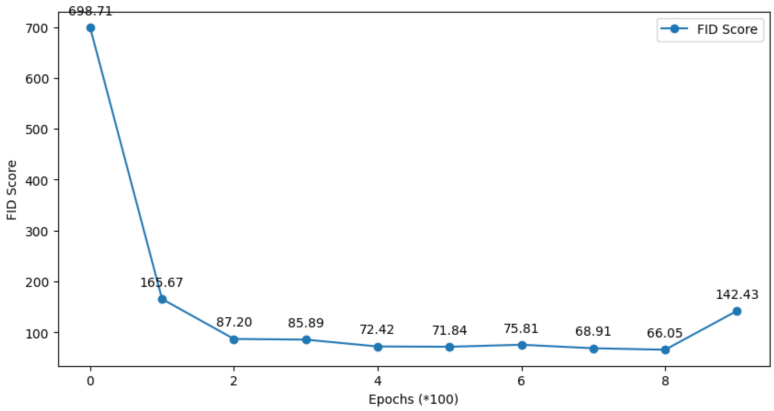
(a) Images generate by basic structure

(b) Images generate by both generator and discriminator are add 3 layers

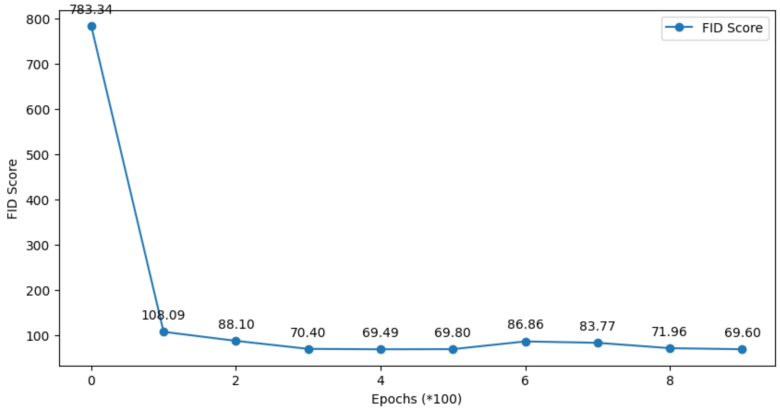
Figure 4.4: The output images for different GAN structure

Data Augmentation

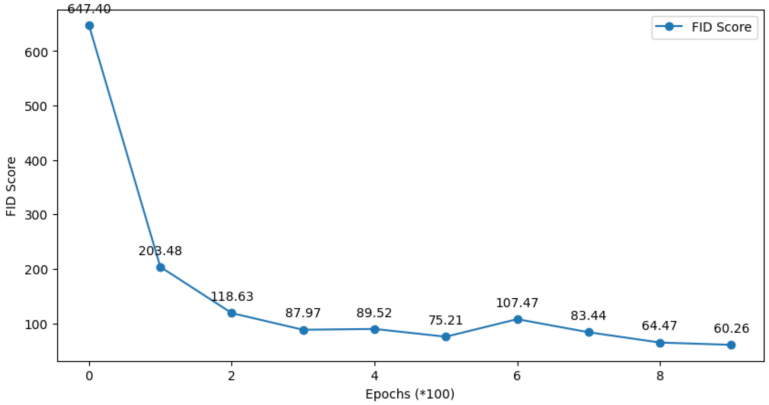
I conducted three training runs of the standard GAN model, both with and without data augmentation, and compared the FID scores. The results indicate that data augmentation generally led to lower optimization performance.



(a) Standard GAN without data augmentation 1

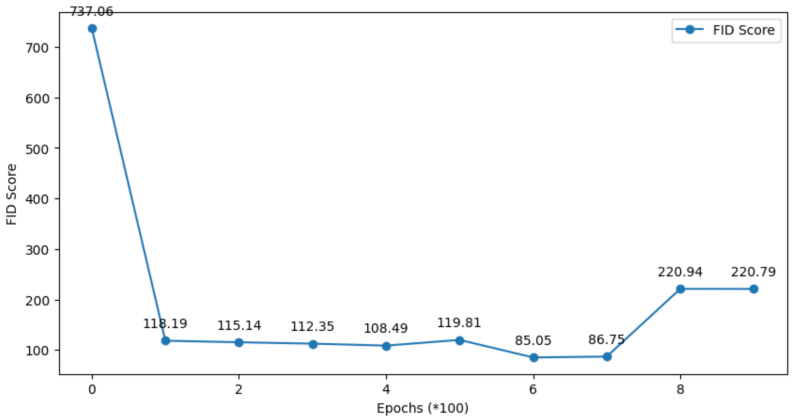


(b) Standard GAN without data augmentation 2

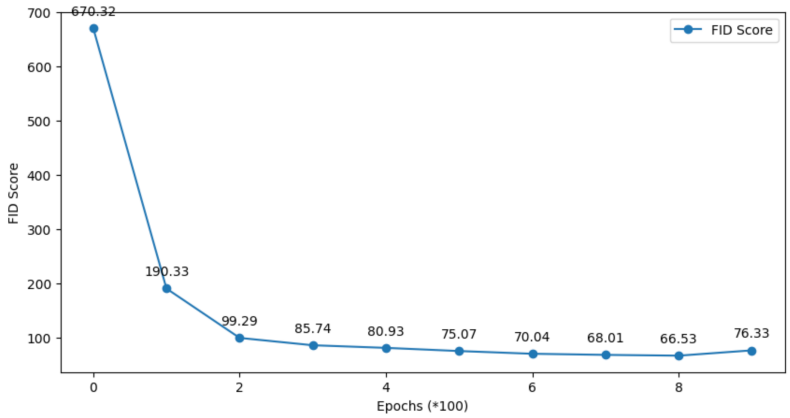


(c) Standard GAN without data augmentation 3

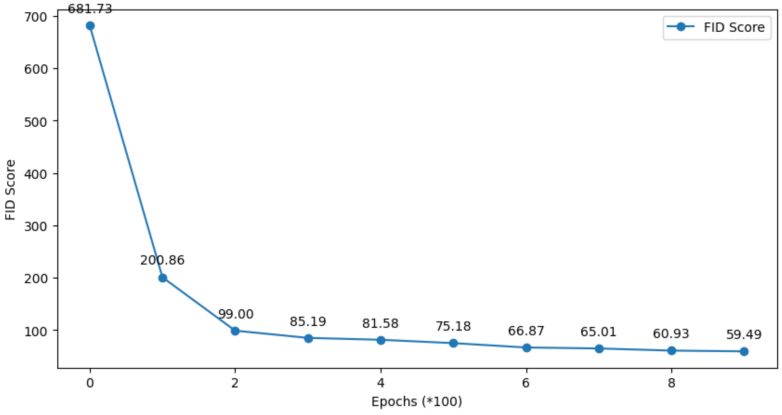
Figure 4.5: The FID scores for standard GAN without data augmentation



(a) Standard GAN with data augmentation 1



(b) Standard GAN with data augmentation 2



(c) Standard GAN with data augmentation 3

Figure 4.6: The FID scores for standard GAN with data augmentation

Apply New Dataset

Chapter 5

Discussion

In this chapter I provide a discussion and concluding remarks...

Appendix A

Code

An appendix can be used for many things, including relevant code...

Bibliography

[1] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. 2017.

[2] Y. Jiang. Applications of generative adversarial networks in materials science. *Materials Genome Engineering Advances*, 2, 2024.

[3] Y. Xin, E. Walia, and P. Babyn. Generative adversarial network in medical imaging: a review. *Medical Image Analysis*, 58:101552, 2019.

[4] S. Kazeminia, C. Baur, A. Kuijper, B. Ginneken, N. Navab, S. Albarqouni, and A. Mukhopadhyay. Gans for medical image analysis. *Artificial Intelligence in Medicine*, 109:101938, 2020.

[5] M. Frid-Adar, I. Diamant, E. Klang, M. M. Amitai, J. Goldberger, and H. Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[6] R. Abdal, Y. Qin, and P. Wonka. Image2stylegan: how to embed images into the stylegan latent space? *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[7] K. Cho, T. Raiko, A. Ilin, and J. Karhunen. A two-stage pretraining algorithm for deep boltzmann machines. pages 106–113, 2013.

[8] G. Montavon. Learning feature hierarchies with centered deep boltzmann machines. 2012.

[9] S. Niazi, N. Aadit, M. Mohseni, S. Chowdhury, and Y. Qin. Training deep boltzmann networks with sparse ising machines. 2023.

[10] R. Salakhutdinov and G. Hinton. An efficient learning procedure for deep boltzmann machines. *Neural Computation*, 24:1967–2006, 2012.

[11] D. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12:307–392, 2019.

[12] B. Kiran, D. Thomas, and R. Parakkal. An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. *Journal of Imaging*, 4:36, 2018.

[13] C. Guo, J. Zhou, H. Chen, N. Ying, J. Zhang, and D. Zhou. Variational autoencoder with optimizing gaussian mixture model priors. *Ieee Access*, 8:43992–44005, 2020.

[14] Y. Varolgunes, T. Bereau, and J. Rudzinski. Interpretable embeddings from molecular simulations using gaussian mixture variational autoencoders. *Machine Learning Science and Technology*, 1:015012, 2020.

[15] S. Portillo. Dimensionality reduction of sdss spectra with variational autoencoders. 2020.

[16] P. Munjal, A. Paul, and N. Krishnan. Implicit discriminator in variational autoencoder. 2019.

[17] X. Bie, L. Girin, S. Leglaive, T. Hueber, and X. Alameda-Pineda. A benchmark of dynamical variational autoencoders applied to speech spectrogram modeling. 2021.

[18] L. Han, Y. Huang, and T. Zhang. Candidates vs. noises estimation for large multi-class classification problem. 2017.

[19] B. Liu, E. Rosenfeld, P. Ravikumar, and A. Risteski. Analyzing and improving the optimization landscape of noise-contrastive estimation. 2021.

[20] B. Damavandi, S. Kumar, N. Shazeer, and A. Bruguier. Nn-grams: unifying neural network and n-gram language models for speech recognition. 2016.

[21] M. Labeau and A. Allauzen. An experimental analysis of noise-contrastive estimation: the noise distribution matters. 2017.

[22] Y. Song. How to train your energy-based models. 2021.

[23] J. Parikh, T. Rumbell, X. Butova, T. Myachina, J. Acero, S. Khamzin, O. Solovyova, J. Kozloski, A. Khokhlova, and V. Gurev. Generative adversarial networks for construction of virtual populations of mechanistic models: simulations to study omecamtiv mecarbil action. *Journal of Pharmacokinetics and Pharmacodynamics*, 49:51–64, 2021.

[24] Y. Saito, S. Takamichi, and H. Saruwatari. Statistical parametric speech synthesis incorporating generative adversarial networks. *Ieee/Acm Transactions on Audio Speech and Language Processing*, 26:84–96, 2018.

[25] T. Miyato. Cgans with projection discriminator. 2018.

[26] G. Qi. Loss-sensitive generative adversarial networks on lipschitz densities. *International Journal of Computer Vision*, 128:1118–1140, 2019.

[27] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. On the effectiveness of least squares generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:2947–2960, 2019.

[28] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. 2014.

[29] T. Kynkäänniemi, T. Karras, M. Aittala, T. Aila, and J. Lehtinen. The role of imagenet classes in fréchet inception distance. 2022.

[30] Y. Xu, T. Wu, J. Charlton, and K. Bennett. Gan training acceleration using fréchet descriptor-based coreset. *Applied Sciences*, 12:7599, 2022.

[31] R. Xu, J. Wang, J. Liu, F. Ni, and B. Cao. Thermal infrared face image data enhancement method based on deep learning. 2023.

[32] D. Berthelot, T. Schumm, and L. Metz. Began: boundary equilibrium generative adversarial networks. 2017.

[33] H. Hyungrok, T. Jun, and D. Kim. Unbalanced gans: pre-training the generator of generative adversarial network using variational autoencoder. 2020.