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: 02/08/2024

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

Related Work

Deep generative models

Deep generative models, such as Deep Boltzmann Machines (DBMs), are a significant area of research in machine learning. These models are characterized by providing parameterized specifications of probability distribution functions and are typically trained by maximizing the log-likelihood function [7]. DBMs have shown success in extracting deep hierarchical representations of input data, although they often face challenges due to intractable likelihood functions, necessitating multiple approximations of the likelihood gradient [8].

One approach to addressing the challenges in training deep generative models is through techniques like deep tempering, which leverages properties of models like Deep Belief Networks (DBNs) to enhance ergodicity by sampling from deeper levels of the latent variable hierarchy [9]. Additionally, methods like reweighted wake-sleep have been proposed to improve the generative performance of deep models by capturing high-level abstractions and enhancing generalization capabilities [10].

Furthermore, the training of DBMs can be optimized by centering binary variables, which has been shown to improve generative performance and stabilize learning processes [11]. Additionally, the use of mean-field inference in Gaussian Restricted Boltzmann Machines has been explored to meet the in-

creasing demand for analyzing computational algorithms for RBMs in various fields [12].

In conclusion, the development and training of deep generative models like DBMs involve addressing challenges related to intractable likelihood functions, gradient approximations, and model optimization techniques. By leveraging properties of these models, exploring novel training algorithms, and optimizing learning processes, researchers aim to enhance the generative performance and generalization capabilities of deep generative models.

Generative Stochastic Networks

Generative machines have emerged as a solution to the challenges associated with training Deep Boltzmann Machines (DBMs) [13]. One notable example is Generative Stochastic Networks (GSNs), which have been developed to generate samples without explicitly representing the likelihood function. GSNs can be trained using exact backpropagation, eliminating the need for approximate methods required by DBMs. These networks are based on learning the transition operator of a Markov chain to estimate the data distribution. Furthermore, the concept of generative machines has been extended by eliminating Markov chains in generative stochastic networks, enhancing their efficiency and effectiveness [14].

Generative Neurosymbolic Machines represent another advancement in generative models, combining distributed and symbolic representations to support structured symbolic components and density-based generation. This approach leverages the benefits of both types of representations to enhance the overall performance of generative models.

In summary, generative machines like GSNs and Generative Neurosymbolic Machines offer innovative solutions for generating samples without explicitly modeling the likelihood function, thereby overcoming the complexities associated with training DBMs. These advancements in generative models pave the way for more efficient and effective sample generation in machine learning applications.

Variational Autoencoders

Variational Autoencoders (VAEs) have gained significant attention in the field of deep learning due to their ability to learn latent representations of complex data. Kingma and Welling, along with Rezende et al., introduced a stochastic back-propagation rule that enables training VAEs by back-propagation through a Gaussian distribution [15]. This approach pairs a generative network with a discriminative model to perform approximate inference, allowing VAEs to learn to encode data into a low-dimensional latent space and decode it back to the original data [16].

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 [15]. 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].

In summary, VAEs offer a powerful framework for deep generative modeling, enabling the creation of latent representations of data that can be decoded back to the original form. While they excel in modeling continuous data, challenges persist in effectively modeling discrete data due to the nature of back-propagation through hidden units.

Generative Stochastic Networks

Noise Contrastive Estimation (NCE) is a technique used in training generative models by distinguishing between data and noise samples [18]. It has been applied in various fields such as speech recognition and language modeling due to its ability to handle large vocabularies efficiently [19]. NCE addresses the computational challenges posed by traditional methods like softmax by transforming the estimation problem into a binary classification task [20]. By discriminating between observed data and artificially generated noise, NCE enables the estimation of non-normalized models effectively [21].

One of the key limitations of NCE is the need to evaluate and backpropagate two probability densities, one for the noise distribution and the other

for the model distribution [20]. Despite this limitation, NCE has proven to be a highly effective approach for unsupervised representation learning using deep networks [22]. The method has also been extended to flow models for energy-based models, where the update is based on noise contrastive estimation [23].

In the context of training large vocabulary neural language models, NCE has been shown to be a sampling-based technique that offers speed improvements [24]. Researchers have explored different strategies to enhance the training algorithms, including investigating noise contrastive estimation and diagonal contexts for further speed improvements [25]. Additionally, the use of noise-contrastive estimation has been linked to self-supervised tasks in state-of-the-art methods [26].

In conclusion, Noise Contrastive Estimation (NCE) is a valuable technique for training generative models efficiently, particularly in scenarios with large vocabularies. While it has some limitations related to the evaluation of multiple probability densities, researchers continue to explore and extend NCE to address various challenges in unsupervised representation learning.

Predictability minimization

Predictability minimization is a technique that involves training two neural networks competitively to encourage the hidden units of the neural network to be independent of each other. Unlike Generative Adversarial Networks (GANs), where the competition is the sole training criterion, predictability minimization uses a regularizer to promote independence among the hidden units [27].

In the context of predictive coding, predictability minimization aims to reduce prediction errors by minimizing the mismatch between incoming sensations and predictions established through experience. This process involves deviance processing, which is part of an inference process where prediction errors are minimized at different levels of the auditory hierarchy [28].

Regularization techniques play a crucial role in predictability minimization by penalizing model complexity to prevent overfitting. These methods

typically involve balancing prediction errors on training data against regularization to optimize the model's performance [29]. Regularization functions often impose constraints on the model to ensure smooth transitions and prevent over-parameterization, thus aiding in minimizing prediction errors [30].

Overall, predictability minimization, through the use of regularization techniques and competitive training of neural networks, aims to enhance the independence of hidden units and reduce prediction errors by optimizing model complexity and promoting smoother transitions within the model.

Chapter 3

Theoretical Background

Generative Adversarial Nets

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 [31]. 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 [32]. 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 [33].

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

The formula 3.1 is the objective function of Generative Adversarial Networks (GANs). 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).

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

The formula 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 [34]. 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 [35].

The GAN network parameters play a crucial role in determining the quality and diversity of the generated samples $G(x)$ [31]. The optimization process in GANs typically involves minimizing a min-max function to ensure the G produces samples that are indistinguishable from real data [32]. Maintaining this balance during training is essential for achieving high-quality sample generation [31]. The D's role is to provide feedback to the generator by acting as a critic, guiding the G towards producing more realistic samples [33].

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

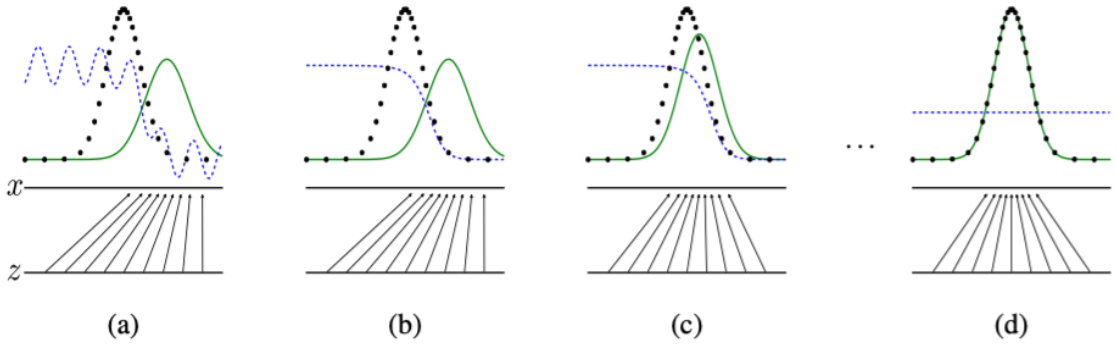


Figure 3.1: Diagram of GAN training process
Source: [36]

- **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 maxi-

mizing 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 [37]. 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 [38].

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 [39]. 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 [39].

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

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

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

- 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

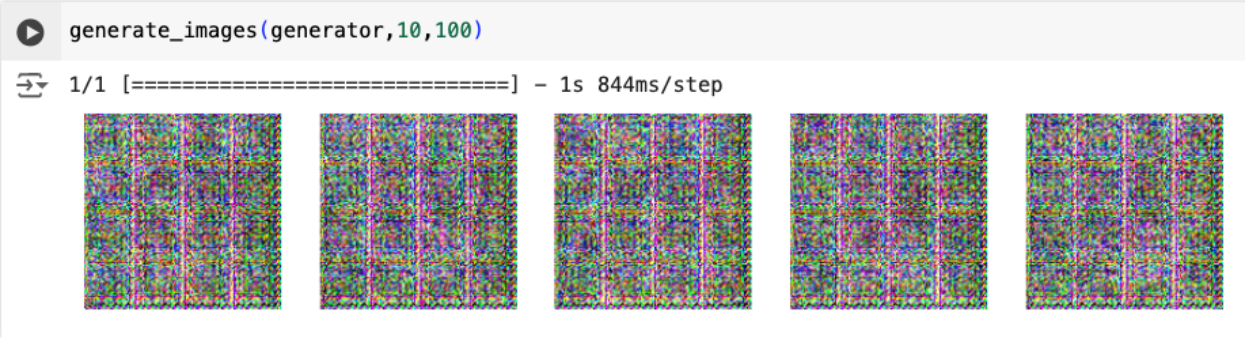


Figure 3.3: Generate images

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 [40]. The importance of this balance is crucial for the effective operation of GANs, ensuring stable learning and improved image quality [41].

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

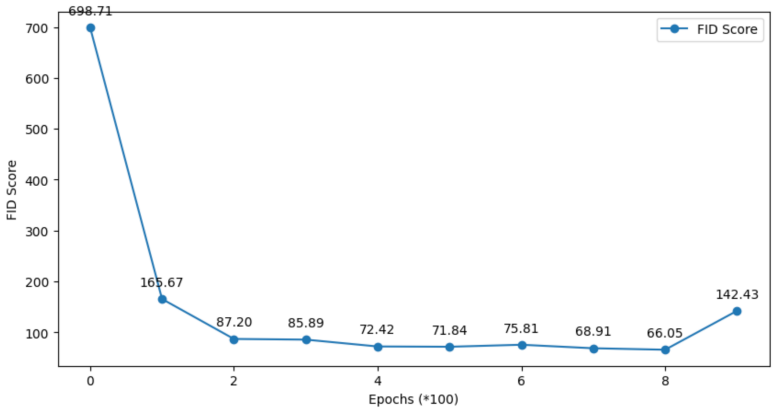
(a) Images generate by basic structure

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

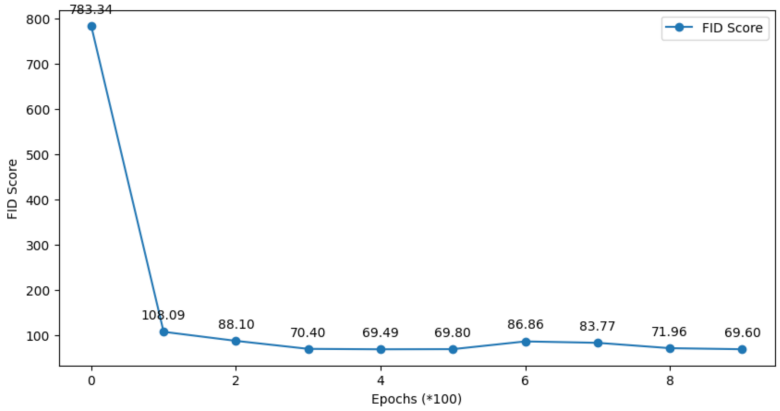
Figure 4.4: The output images for different GAN structure

Data Augmentation

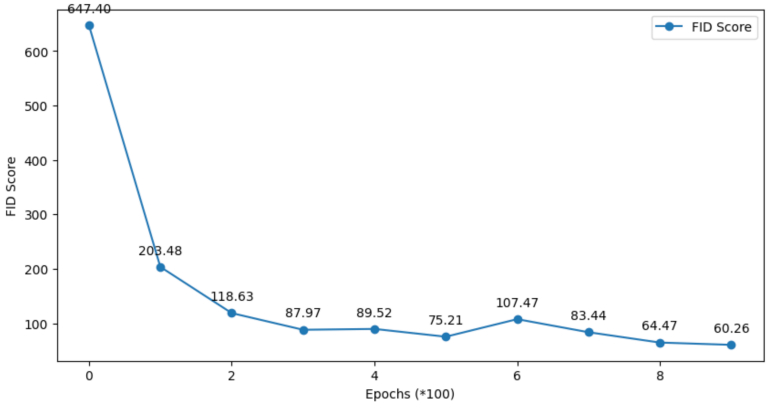
I conducted three training runs of the standard GAN model, both with and without data augmentation, and compared the FID scores. The results indicate that data augmentation generally led to lower optimization performance.



(a) Standard GAN without data augmentation 1

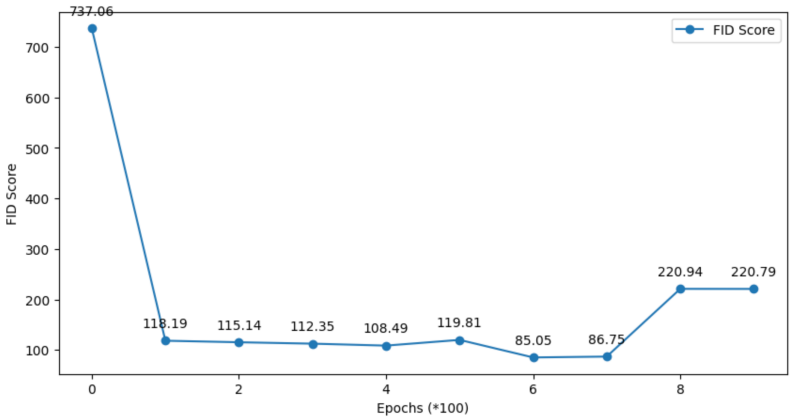


(b) Standard GAN without data augmentation 2

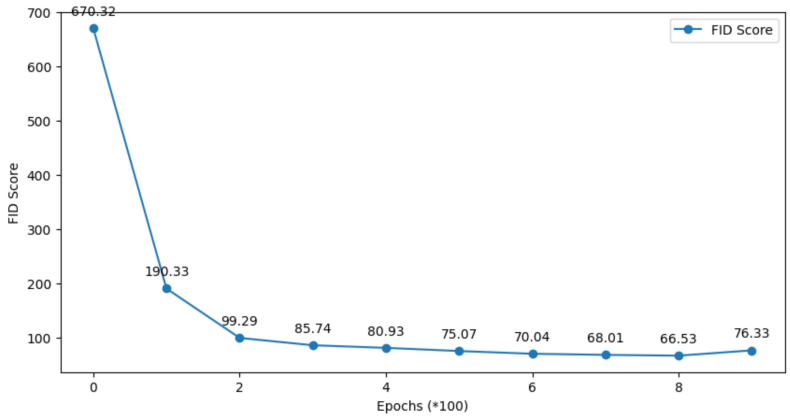


(c) Standard GAN without data augmentation 3

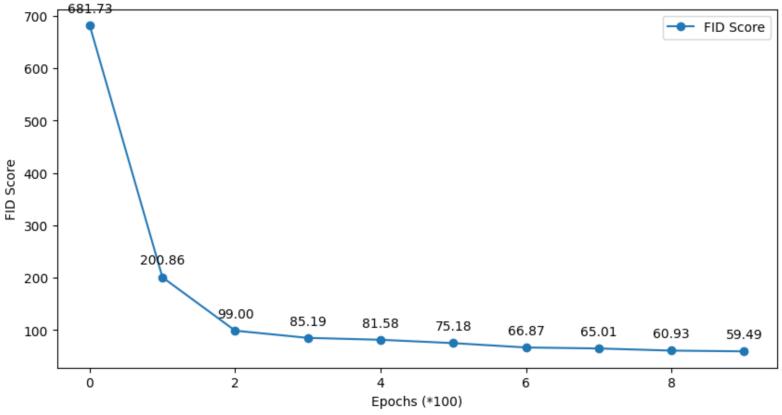
Figure 4.5: The FID scores for standard GAN without data augmentation



(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] H. Xu and Z. Ou. Joint stochastic approximation learning of helmholtz machines. 2016.

[8] K. Zhang and X. Chen. Large-scale deep belief nets with mapreduce. *Ieee Access*, 2:395–403, 2014.

[9] G. Desjardins. Deep tempering. 2014.

[10] J. Bornschein. Reweighted wake-sleep. 2014.

[11] J. Melchior. How to center binary deep boltzmann machines. 2013.

[12] C. Takahashi and M. Yasuda. Mean-field inference in gaussian restricted boltzmann machine. *Journal of the Physical Society of Japan*, 85:034001, 2016.

[13] G. Alain, Y. Bengio, L. Yao, J. Yosinski, É. Thibodeau-Laufer, S. Zhang, and P. Vincent. Gsns: generative stochastic networks. *Information and Inference a Journal of the Ima*, 5:210–249, 2016.

[14] L. Pan, D. Zhang, J. Moksh, L. Huang, and Y. Bengio. Stochastic generative flow networks. 2023.

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

[16] M. Sidulova. Conditional variational autoencoder for functional connectivity analysis of autism spectrum disorder functional magnetic resonance imaging data: a comparative study. *Bioengineering*, 10:1209, 2023.

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

[19] F. Liza and M. Grzes. Improving language modelling with noise contrastive estimation. *Proceedings of the Aaai Conference on Artificial Intelligence*, 32, 2018.

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

[21] T. Matsuda and A. Hyvärinen. Estimation of non-normalized mixture models and clustering using deep representation. 2018.

[22] P. Awasthi, N. Dikkala, and P. Kamath. Do more negative samples necessarily hurt in contrastive learning? 2022.

[23] R. Gao, E. Nijkamp, D. Kingma, Z. Xu, A. Dai, and Y. Wu. Flow contrastive estimation of energy-based models. 2020.

[24] W. Chen, D. Grangier, and M. Auli. Strategies for training large vocabulary neural language models. 2016.

[25] P. Baltescu and P. Blunsom. Pragmatic neural language modelling in machine translation. 2015.

[26] O. Chehab, A. Gramfort, and A. Hyvärinen. The optimal noise in noise-contrastive learning is not what you think. 2022.

[27] M. Li. Scaling distributed machine learning with the parameter server. 2014.

[28] F. Lecaignard, O. Bertrand, G. Gimenez, J. Mattout, and A. Caclin. Implicit learning of predictable sound sequences modulates human brain responses at different levels of the auditory hierarchy. *Frontiers in Human Neuroscience*, 9, 2015.

[29] H. Liu, K. Verspoor, D. Comeau, A. MacKinlay, and W. Wilbur. Optimizing graph-based patterns to extract biomedical events from the literature. *BMC Bioinformatics*, 16, 2015.

[30] B. Wang, J. Basart, and J. Moulder. Linear and nonlinear image restoration methods for eddy current nondestructive evaluation. pages 791–798, 1998.

[31] 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.

[32] 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.

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

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

[35] 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.

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

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

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

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

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

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