ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

Department of Electrical, Electronic, and Information Engineering “Guglielmo Marconi”-DEI

DEGREE PROGRAM

Electronic Technologies For Big-Data And Internet Of Things

THESIS

In

91250 DEEP LEARNING M

THESIS TITLE

A Comprehensive Survey on Generative Adversarial Networks

CANDIDATE:	Supervisor:
Sharma Manish Kumar	Prof. Andrea Asperti
Matricola:	
0000936053	

Academic Year 2022/23

Table Of Contents

List Of Figures

List Of Tables

ABSTRACT

Generative Adversarial Networks (GANs) are a class of neural network architectures that have been used to generate a wide variety of realistic data, including images, videos, and audio. GANs consist of two main components: a generator network, which produces new data, and a discriminator network, which attempts to distinguish the generated data from real data. The two networks are trained in a competitive manner, with the generator trying to produce data that can fool the discriminator, and the discriminator trying to correctly identify the generated data. Since their introduction in 2014, GANs have been applied to a wide range of tasks, such as image synthesis, image-to-image translation, and text-to-image synthesis. GANs have also been used in various fields such as computer vision, natural language processing, and speech recognition. Despite their success, GANs have several limitations and challenges, including mode collapse, where the generator produces only a limited number of distinct samples, and instability during training. Several methods have been proposed to address these challenges, including regularization techniques, architectural modifications, and alternative training algorithms. Overall, GANs have proven to be a powerful tool for generating realistic data, and research on GANs is an active area of study in the field of machine learning. This survey paper aims to provide an overview of the GANs architecture and its variants, applications and challenges, and the recent developments in GANs.

Keywords: GANs (Generative Adversarial Networks), Generator, Discriminator, Data, Network, Function

CHAPTER 1

INTRODUCTION

In AI and machine learning, there are two fundamental methods: supervised learning and unsupervised learning. The key distinction between the two methods is made by the titles of the two methods, namely, supervised learning and unsupervised learning. The algorithm gives in to the intended result. Contrarily, unsupervised learning trains the algorithm on unlabelled datasets to find latent patterns in the data that helped it solve grouping or association issues.

Generative Adversarial Networks (GANs) are a class of deep learning models that can generate new, previously unseen examples of data. They have been used for a wide variety of tasks, including image synthesis, image-to-image translation, and text-to-image synthesis. In this survey, we will provide an overview of the current state of research in GANs, including their architecture, training techniques, and applications.

GANs were first introduced by Ian Goodfellow and his colleagues in 2014. The basic idea behind GANs is to train two neural networks, a generator, and a discriminator, in an adversarial manner. The generator produces new examples of data, and the discriminator tries to determine whether each example is real or fake. Through this process, the generator learns to produce more realistic examples, and the discriminator learns to better distinguish between real and fake examples. The architecture of a GAN typically consists of a generator network and a discriminator network. The generator network maps a random noise vector to an example of data, while the discriminator network takes an example of data and outputs a probability of whether it is real or fake. There are several techniques used to train GANs, including the original GAN algorithm, variations such as Wasserstein GANs, and recent advances in training stability. Wasserstein GANs, for example, use the Wasserstein distance between the real and generated distributions instead of the Jensen-Shannon divergence used in the original GAN algorithm.

GANs have been applied to a wide variety of tasks, including image synthesis, image-to-image translation, and text-to-image synthesis. In image synthesis, for example, a GAN can be used to generate new images of a particular object or scene, such as faces or landscapes. In image-to-image translation, a GAN can be used to translate an image from one domain to another, such as from a sketch to a photograph. Despite their successes, GANs still have several limitations. One of the main limitations is that GANs can be difficult to train, and are prone to mode collapse, where the generator produces only a small subset of the possible outputs. Another limitation is that GANs can be sensitive to the choice of architecture and hyperparameters. In conclusion, GANs are a powerful class of deep learning models that have been used for a wide variety of tasks. They have shown great promise in generating new, previously unseen examples of data, but also have several limitations. Future research will focus on addressing these limitations and developing new and improved GAN architectures and training techniques. Alan Turing established his now-famous Turing Test for artificial intelligence in a 1950 article [56] he wrote exploring the idea of a thinking machine. Since then, AI has consistently been in the news. Research into artificial intelligence (AI) advanced quickly during the past decade. Each year, thousands of AI-related academic papers are released to the public.

Both supervised and unsupervised learning are fundamental methodologies utilised in the fields of artificial intelligence and machine learning. Supervised learning, on the other hand, makes use of labelled datasets in order to train the classification algorithm, which ultimately results in the desired conclusion. This is the crucial distinction that exists between the two methods, which is clearly characterised by their names. In contrast, unsupervised learning makes use of datasets that have not been labelled in order to train the algorithm. This allows the algorithm to identify hidden patterns in the data that assisted it in solving grouping or association issues. Supervised learning is still the most popular approach in artificial intelligence and machine learning, despite the substantial costs associated in labelling the data. Although unsupervised learning has a reduced cost need for datasets, it is not as effective as supervised algorithms in the majority of applications. Many scholars have trust in unsupervised learning due to the fact that the process of unsupervised learning is comparable to the way in which a human infant acquires knowledge. If all went according to plan, a perfect unsupervised learning model would be able to grasp the latent relationships in the dataset (like cat photographs) and differentiate between them with only a few samples of training data. The most promising approach in unsupervised learning is known as the generative model, and it is often based on Markov chains, maximum likelihood, and approximation inference. The world was first introduced to Generative Adversarial Networks (GAN) [32] by Ian Goodfellow et al. in 2014, and it quickly rose to become the most prominent figure in the area of machine learning because to its impressive performance on the MNIST and CIFAR-10 benchmarks. This strong model enabled many other researchers to move forward in a variety of different areas. When compared to other generating models, such as the Boltzmann machine or the Variational Autoencoder, the GAN does not require Markov chains or variational limits, and the design of generative functions is subject to just a limited number of constraints [30].

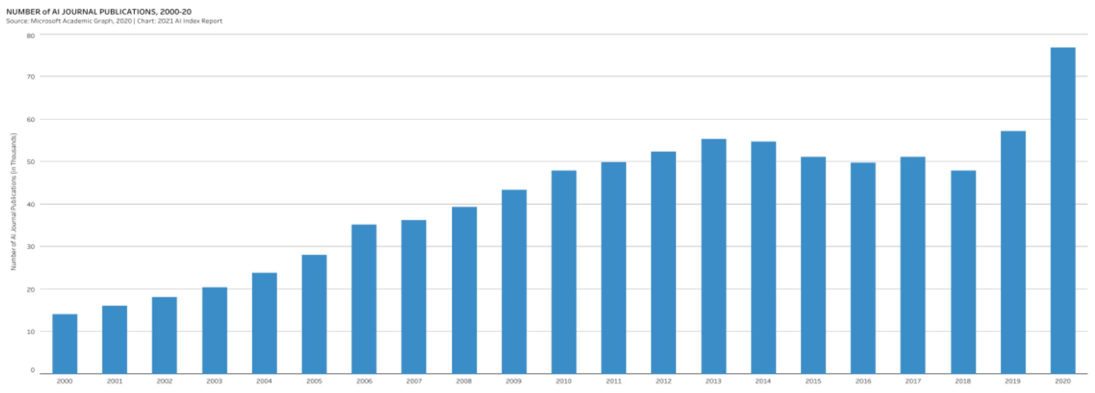


Figure 1.1: 2000-2020 Number of AI Journal Publication: from 2021 Stanford AI Index Report [37].

This research includes a wide range of GAN architectures in addition to the original GAN models as well as the current training GAN models. Some examples of these models include vanishing gradient and mode collapse. In addition, this research offers the previously appropriate answer that corresponds to many problems as a guideline for those individuals who are interested in obtaining a holistic perspective about GANs.

Even if a substantial amount of research and studies have been conducted to satisfy GAN models in a variety of domains, there is still a large amount of room for the assessment of GANs. GANs provide a difficult evaluation problem due to the fact that they are implicit generative models that lack a model likelihood function [18]. Lucic et al. (2018) noted that there are presently no standard processes for systematic evaluation, and they considered that quantitative evaluation is the most significant obstacle among all of the possible problems [55]. The manual visual fidelity of the obtained findings is used in qualitative assessments, which is a process that is both subjective and inefficient when applied to a large number of samples. Quantitative procedures, on the other hand, are subjective and open to interpretation, whereas quantitative measurements are objective and less influenced by human variables [80]. The third chapter provides an overview of a variety of assessment metrics that are often applied in a variety of research. The purpose of this study is to investigate the concepts that lie behind the assessment methodologies and to get an understanding of the most appropriate contexts in which to use the metrics.

GAN has many applications across many different fields, and it has achieved a great deal of success in the field of computer vision. Some examples of GAN's applications include plausible image generation, image to image translation, text to image translation, photo editing, face ageing, super-resolution, video prediction, and the generation of three-dimensional objects. The general artificial neural network (GAN) has not been as successful in other areas, such as natural language processing, synthetic time series, and semantic segmentation; nonetheless, it is still in the process of being developed and has a great deal of untapped potential.

Extensive surveys are carried out in an effort to either demonstrate the viewer uses of GANs or explain the trends that is currently occurring. Reference [68] primarily focused on the difficulties that the majority of GANs encountered and suggested a new taxonomy to organise potential solutions. The link between the various GANs and how they have developed is illustrated in Reference [34], which can be found here. In addition, topics such as transfer learning and reinforcement learning, two other types of machine learning algorithms, are also covered. A taxonomy of GANs based on their architectures and loss functions was provided in Reference [75]. The major focus of this taxonomy was on three aspects: 1) generate quality, 2) generated diversity, and 3) stable training. The details of GAN variations and how they worked were detailed in reference [40], which can be found here. In addition, the framework combination of GAN and Autoencoder was analysed, along with the applications it may be used for. The article cited in the reference (34), which examines numerous variations of GAN, also provides a detailed examination of several applications in other fields. In addition to that, we cover the challenges and solutions associated with GAN training. The GANs that are employed in computer vision and image classification were the primary topic of discussion in the survey [39]. Work [18, 89, and 55] examine traditional assessment measures for the purpose of comparing GANs and address questions regarding how these metrics should be evaluated. Goodfellow et al. [67] presents a novel architecture for GANs and also proposes some training techniques that might be applicable to the framework of GANs. The issue that Wang et al. [75] investigated is one that really piques my interest among all of these surveys. This work benefited from the classification of the most common variations of GANs, which will assist researchers in selecting the appropriate architecture and loss function for a particular job. In this study, we will explore the benefits of generative adversarial networks (GANs), as well as their limitations and potential enhancements, and offer various milestones of GANs and assessment metrics. The majority of the GAN survey studies that are relevant to this study are included below and it presents the list of GAN variations and assessment metrics that were looked at for this work.

Structure of this this paper:

1. We will present the original GAN and a few key related studies. We'll show how things have changed and improved. There will be an introduction to certain GAN applications.

2. Different assessment metrics for GANs will be discussed and contrasted.

3. There will be discussion of GAN stabilisation strategies and GAN barriers.

4. A summary of GAN variations and assessment metrics; talk about upcoming studies in the area.

Summary of reviewed surveys of GANs, evaluations, and techniques.

1 Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions [68] ,2021
2 A Review on Generative Adversarial Nets: Algorithms, Theory, and Applications [34], 2020
3 Generative Adversarial Networks: A Survey and Taxonomy [75], 2019
4 What Generative Adversarial Networks and Their Variants Work: An Overview [40], 2019
5 A Survey on Generative Adversarial Networks [42], 2018
6 Comparative Study on Generative Adversarial Networks [39] ,2018
7 Pros and Cons of GAN Evaluation Measures [18] ,2018
8 An Empirical Study on Evaluation Metrics OF Generative Adversarial Networks[80] ,2018
9 A Survey on Generative Adversarial Networks [17] ,2021
10 Are GANs Created Equal? A Large-Scale Study [55] ,2017
11 Improved Techniques for Training GANs [67] ,2017
12 A Note on The Evaluation of Generative Models [72] ,2016

Summary of reviewed GAN variants and evaluation methods.

1 A kernel two-sample test [33] ((MMD)),2012
2 Generative Adversarial Networks [32] (GAN) ,2014
3 Conditional Generative Adversarial Nets [60] (CGAN) ,2014

4 Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks [26] (LAPGAN) ,2015
5 Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets[22] (InfoGAN), 2016
6 Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks [63](DCGAN), 2016
7 Improved Techniques for Training GANs [67] IS, 2016
8 Wasserstein GAN [21] (WGAN) ,2017
9 Improved Training of Wasserstein GANs [35] (WGAN GP) ,2017
10 Progressive Growing of GANs for Improved Quality, Stability, and Variation [46] (PROGAN), 2017
11 GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium [38] (FID) 2017
12 Generative adversarial networks for diverse and limited data [36] (m-IS) , 2017
13 Are gans created equal? a large-scale study [56] (Precision, Recall and F1 score), 2017
14 Assessing generative models via precision and recall [66] (PRD) ,2018
15 Geometry score: A method for comparing generative adversarial net-works [48] (GS) , 2018
16 Demystifying MMD GANs [16] (KID), 2018
17 Self-Attention Generative Adversarial Networks [80] (SAGAN), 2019
18 Large Scale GAN Training for High Fidelity Natural Image Synthesis [20] (BigGAN), 2019
19 A Style-Based Generator Architecture for Generative Adversarial Networks [47] (StyleGAN), 2019
20 Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks [83] (CycleGAN)2020
21 Designing Two-Dimensional Local Attention Mechanisms for Generative Models citedaras2020your. (YLGAN) , 2020

CHAPTER 2

GENERATIVE ADVERSARIAL NETWORKS AND RELATIVE STUDIES

2.1 Autoencoder

Autoencoder is a type of deep learning neural network architecture used for unsupervised learning. It consists of two main parts: an encoder and a decoder. The encoder compresses the input data into a lower-dimensional representation, known as the encoding, which captures the essential features of the input. The decoder then uses this encoding to reconstruct the original input. The goal of the autoencoder is to learn the mapping between the input and its reconstructed output such that the difference between the original input and the reconstructed output is minimized. This is done by training the network with the input data and updating the weights of the network to reduce the reconstruction error. Autoencoders have a variety of applications, including dimensionality reduction, denoising, and generation of new data. They are also commonly used as building blocks for more complex deep learning models, such as Variational Autoencoders and Generative Adversarial Networks.

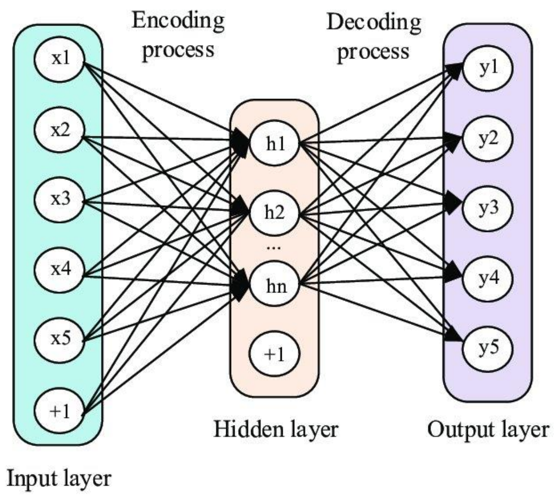


Figure 2.1: Autoencoder structure.

In 1955, Hinton et al. presented the concept of autoencoder, which has since been studied for decades. Next, Nonlinear Principal Component Analysis [52] was presented by Mark A. Kramer. Its primary purpose is to discover and eliminate nonlinear connections in data so that the dimensionality of the data may be reduced by excluding superfluous details. The Encoder networks (input layer) and Decoder networks (output layer) of a conventional Autoencoder use feedforward neural networks to perform their functions (output layer). Figure 2.1 depicts the finished construction. This latent representation, sometimes called a bottleneck, is created when the encoder reduces the high-dimensional input data to its smallest possible size. The decoder makes an effort to learn how to accurately recreate the bottleneck from the original input data. In Autoencoder, the reconstruct loss is quantified by calculating the L2-norm of the Euclidean distance between neighbouring pixels.

$$\text{Reconstruct Loss} = \|\text{Input} - \text{Reconstruct Output}\|^2$$

Unlike typical compression techniques, the encoder is not reducing data size. In contrast to principal component analysis, the encoder may pick up on nonlinear associations as well as linear ones. In a perfect world, an encoder would be able to figure out the input's hidden characteristics. The AE has overfitting issues when trying to reconstruct output from input, and the latent characteristics are often erratic, making it unsuitable for generating new data. Denoising and dimension reduction are two of Autoencoder's most well-known uses. The structure of the Variational Autoencoder [50] is strikingly similar to that of the Autoencoder. While AE can only recover data, VAE may regularise the latent representation and create new data.

2.2 Variational Autoencoder (VAE)

A Variational Autoencoder (VAE) is a type of deep learning model that combines the concepts of autoencoders and variational inference. An autoencoder is a neural network that learns to reconstruct its input, whereas variational inference is a statistical method for approximating complex distributions [78]. The key idea behind a VAE is to introduce a latent variable, or encoding, into the autoencoder architecture. The encoder network takes in an input, x, and maps it to a lower dimensional representation, z, in the latent space. The decoder network then takes this encoding and maps it back to the original input space.

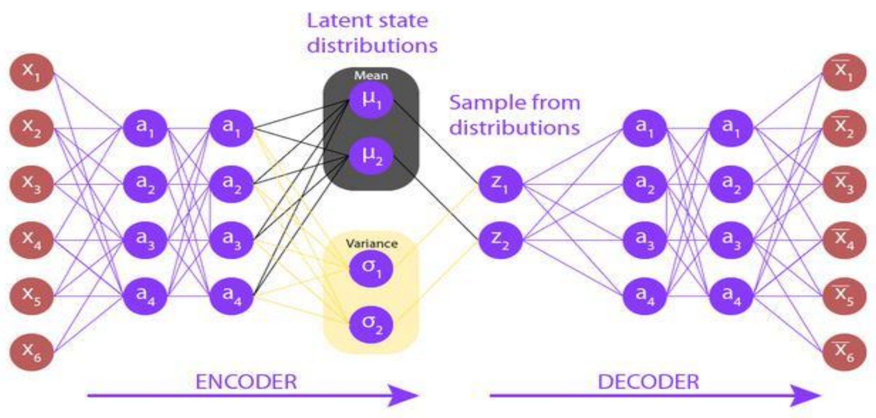


Figure 2.2: Variational Autoencoder Structure

The crucial difference between a traditional autoencoder and a VAE is the addition of a probabilistic interpretation of the encoding, z. In a VAE, the encoding is treated as a random variable with a prior distribution.

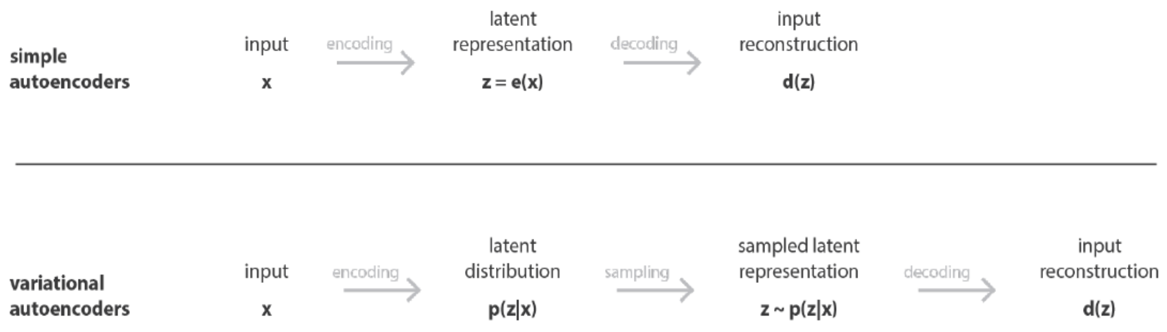


Figure 2.3: Simple Vs Variational Autoencoder

The encoder network then learns the parameters of a posterior distribution that describes the relationship between the input and the encoding. The objective of the VAE is to maximize the likelihood of the observed data, x, given the encoding, z, while ensuring that the encoding is representative of the underlying data distribution. This is achieved through the use of a variational lower bound, which balances the reconstruction error of the autoencoder and the KL-divergence between the prior and posterior distributions. VAEs are commonly used in generative modelling, where they can be trained to generate new data samples that are similar to the original training data. This is accomplished by sampling from the prior distribution in the latent space and then passing the encoding through the decoder network to obtain a reconstructed sample in the original space. Overall, Variational Autoencoders provide a powerful and flexible framework for deep generative models, allowing for the exploration and manipulation of complex, high-dimensional data distributions.

The structure of Variational Autoencoder [50] is very similar to Autoencoder. Unlike AE, which focuses on data reconstruction, VAE may regularise the latent representation and provide novel data. The VAE encoded the latent qualities as a probability distribution, whereas the Autoencoder assigned deterministic values to the latent variables (stored two parameters: mean and variance of Gaussian distribution). The data we utilise (x) is fed into a model with a latent variable (z). Bayesian formula allows us to compute the latent distribution p(z|x) and obtain p (x). However, obtaining p is often computationally difficult (x). For this reason, we must use a substitute distribution q to stand in for the true one. We then use the Kullback-Leibler divergence to quantify the gap between the two distributions and search for the smallest possible value. The following is a representation of the loss function:

$$L(x, \hat{x}) + \sum_j KL(q_j(z|x) \| p((z)) \tag{2.1}$$

2.3 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GANs) are a class of deep learning models that are capable of generating new, previously unseen examples of data. They have been used for a wide variety of tasks, including image synthesis, image-to-image translation, and text-to-image synthesis. In this survey, we will provide an overview of the current state of research in GANs, including their architecture, training techniques, and applications. GANs were first introduced by Ian Goodfellow and his colleagues in 2014. The basic idea behind GANs is to train two neural networks, a generator and a discriminator, in an adversarial manner. The generator produces new examples of data, and the discriminator tries to determine whether each example is real or fake. Through this process, the generator learns to produce more realistic examples, and the discriminator learns to better distinguish between real and fake examples. The architecture of a GAN typically consists of a generator network and a discriminator network. The generator network maps a random noise vector to an example of data, while the discriminator network takes an example of data and outputs a probability of whether it is real or fake. There are several techniques used to train GANs, including the original GAN algorithm, variations such as Wasserstein GANs, and recent advances in training stability. Wasserstein GANs, for example, use the Wasserstein distance between the real and generated distributions instead of the Jensen-Shannon divergence used in the original GAN algorithm. GANs have been applied to a wide variety of tasks, including image synthesis, image-to-image translation, and text-to-image synthesis. In image synthesis, for example, a GAN can be used to generate new images of a particular object or scene, such as faces or landscapes. In image-to-image translation, a GAN can be used to translate an image from one domain to another, such as from a sketch to a photograph.

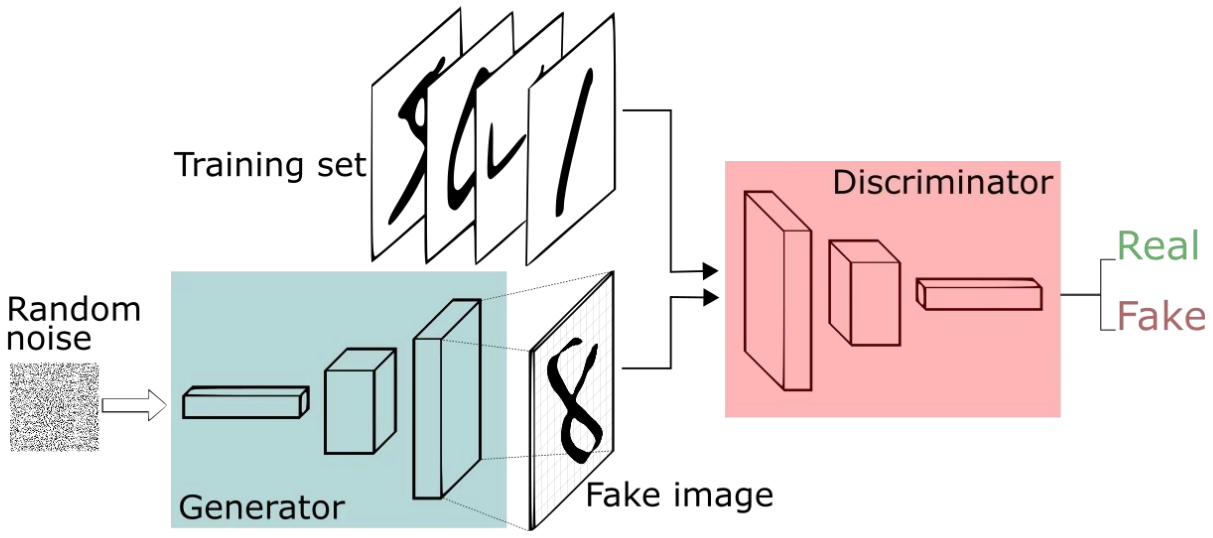


Figure 2.4: Generative Adversarial Network Structure

Despite their successes, GANs still have several limitations. One of the main limitations is that GANs can be difficult to train, and are prone to mode collapse, where the generator produces only a small subset of the possible outputs. Another limitation is that GANs can be sensitive to the choice of architecture and hyperparameters [78].

Generative adversarial networks (GAN) have gained popularity in the disciplines of machine learning, computer vision, and related areas since its introduction by Goodfellow- low et al. in 2014 [32]. Two components, a generator G and a discriminator D, make up the Generative Adversarial Networks, a zero-sum game. To trick the discriminator, the generator must accurately reproduce the training data's underlying distribution. The discriminator's task is to determine if its input comes from the training data or the output data. In this adversarial system, both G and D are taught at the same time, with the latter attempting to maximise the former based on the objective statement below.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.2}$$

where the pdata variable contains the actual data and the pz variable contains the data that was created. If all goes according to plan, we will end up with a generator that is able to create data that is so lifelike that neither the discriminator nor a person could tell them apart.

As was covered in the first chapter of this book, GAN training is difficult due to the fact that GAN is based on a zero-sum game and therefore requires a generator and a discriminator to establish Nash equilibrium in order to converge. The most typical problem that arises when training a GAN is referred to as mode collapse, and it is brought on because the generator tends to have trouble learning the underlying distribution of the dataset well. The model will not converge if the discriminator gets to the point where it is so powerful that the vanishing gradient of the generator. [29, 51] Are only two of the many publications that have been written on the subject of the controversy surrounding GAN training. The distance between the real data distribution and the produced data distribution has to be measured in order to evaluate a GAN. Because it is hard to precisely predict the true data distribution (otherwise, all AI issues would be straightforward to answer), it might be difficult to determine which model is superior to the others.

2.4 Adversarial Autoencoder (AAE)

Adversarial Autoencoder (AAE) is a generative deep learning model that combines autoencoder architecture with adversarial training. Autoencoders are neural networks designed to learn a compact representation of the input data in an unsupervised manner, while adversarial training involves using a discriminator network to distinguish between the generated and real samples. In the case of AAE, the autoencoder is used to learn a compact representation of the input data, while the discriminator network is used to ensure that the generated samples are indistinguishable from the real samples. The autoencoder is trained to reconstruct the input data from the compact representation, while the discriminator is trained to distinguish between the real and generated samples. The discriminator network serves as a regularizer for the autoencoder, ensuring that the learned representation is not only compact but also meaningful. The architecture of an AAE consists of three parts: the encoder network, the decoder

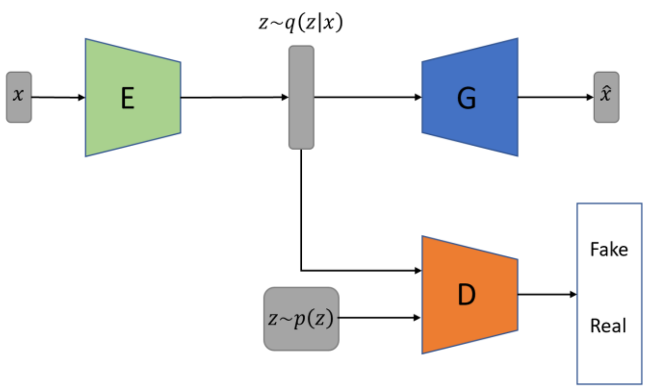


Figure 2.5: Adversarial Autoencoder Structure

network, and the discriminator network. The encoder network takes the input data and maps it to a low-dimensional representation, known as the code. The decoder network takes the code and maps it back to the original input data. The discriminator network takes the code and classifies it as either real or generated. The training process of an AAE involves two objectives. The first objective is to train the autoencoder to reconstruct the input data from the code, while the second objective is to train the discriminator to distinguish between the real and generated codes. The two objectives are in conflict with each other, as the autoencoder tries to generate codes that look like real codes, while the discriminator tries to distinguish between the real and generated codes. In conclusion, Adversarial Autoencoder is a generative deep learning model that combines autoencoder architecture with adversarial training. It uses the autoencoder to learn a compact and meaningful representation of the input data, while the discriminator network serves as a regularizer to ensure that the generated samples are indistinguishable from the real samples. This approach results in a more robust and generalizable representation compared to traditional autoencoders.

In many ways, the Adversarial Autoencoder [57] is conceptually analogous to the Variational Autoencoder. In order to regularise the latent representation, the AAE makes use of adversarial loss derived from the GAN, whereas the VAE makes use of KL divergence. The encoder functions similarly to a generator, producing fictitious data

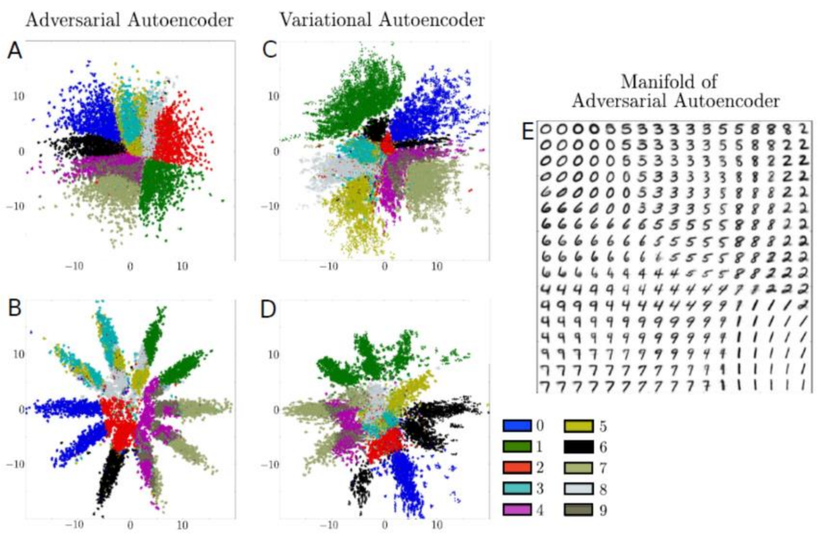


Figure 2.6: Comparison between AAE and VAE on MNIST. The hidden layer z represent image for compare: (a)(c) 2-D Gaussian. (b)(d) a mixture of 10 2-D Gaussians. (e). AAE generates sample [57].

in an effort to trick the discriminator. In addition, the discriminator will be able to tell the difference between the input that was created by the encoder (false data) and the input that was generated by the previous input distribution p(z) (real data). The author of the first study ran a number of comparison experiments between AAE and VAE in the original paper. When it comes to matching a two-dimensional Gaussian distribution to a mixture of ten two-dimensional Gaussian distributions on the MINST dataset, the results shown in Figure 2.6 show that the AAE is able to create a more underlying representation than the VAE does. The capabilities of the Adversarial Autoencoder might be expanded to include semi-supervised and unsupervised settings, with impressive outcomes possible in each case. Anomaly detection, which was first presented by Samet Akcay et al. [10], has been one of the most common applications of AAE in recent years.

2.5 Conditional GAN (CGAN)

Conditional GAN (Generative Adversarial Network) is a variant of the traditional GAN architecture that allows the generation of images conditioned on class labels or any other information. The generator network in a conditional GAN is trained to generate images that correspond to specific class labels or conditions provided as input. The discriminator network is trained to distinguish between the generated images and the real images, but it also receives the same condition information as input. In a conditional GAN, the generator takes both a random noise vector and a condition as input, and generates an image that corresponds to the provided condition. The discriminator receives both the image and the condition as input and outputs a scalar that indicates whether the image is real or fake. The objective of the GAN is to train the generator to produce images that are indistinguishable from real images by the discriminator, while also training the discriminator to correctly classify the generated images as fake.

Conditional GANs have been used for various applications such as generating images of specific objects, synthesizing realistic images for data augmentation, and generating images from text descriptions. The use of condition information allows for the generation of high-quality images that are more controllable compared to traditional GANs. Conditional GAN is a sort of GAN that was introduced by Mehdi Mirza and Simon Osindero in 2014 [60]. It is distinguished by the fact that it makes use of the conditional setting. Every single GAN might be converted into a conditional model, in which the output of both the generator and the discriminator is dependent on some type of auxiliary information y. (such as class labels or data).

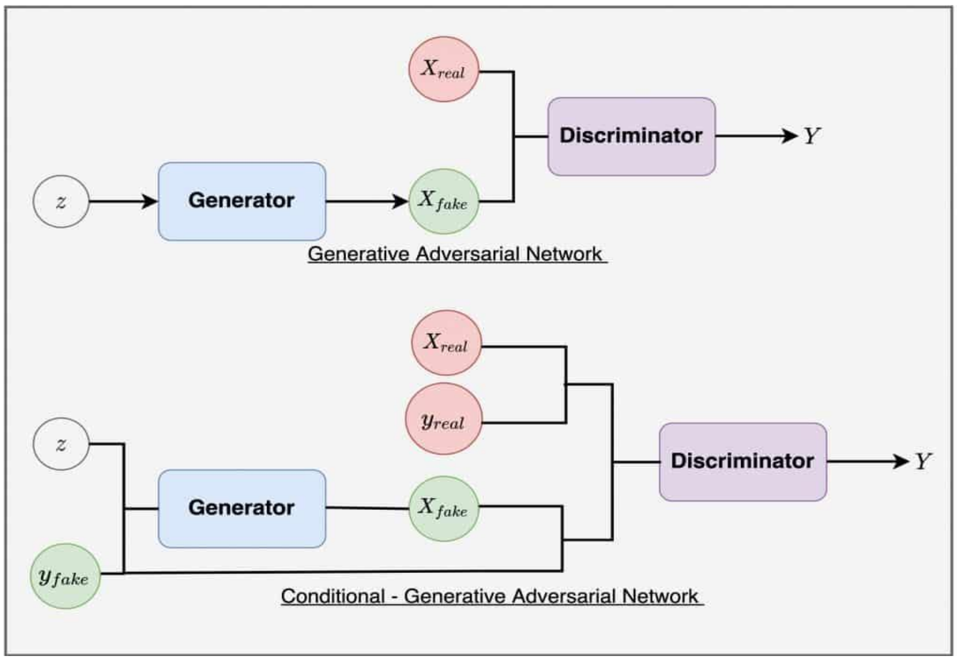


Figure 2.7: Structure of a Conditional GAN.

Figure 2.7 displays a basic CGAN structure. The preceding graphic demonstrates that the generator was responsible for subsequent processing of the new information after it had been first associated with latent space. In addition to that, the process of discrimination in the discriminator makes use of extra information. The structure of the Conditional GAN is simply an expansion of the structure of the vanilla GAN, and we can also see that the objective function that is shown below is very similar to the objective function that is shown for the vanilla GAN, with the exception that all of the data distributions are conditioned on the value of y.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x|y)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z|y)))] \tag{2.3}$$

2.6 LapGAN

LapGAN (Laplacian Generative Adversarial Networks) is a type of Generative Adversarial Network (GAN) that generates images by utilizing the Laplacian Pyramid Convolutional Neural Network (Lap-CNN) structure. LapGAN is a combination of GAN and Laplacian Pyramid that enables the network to learn the features at different scales and generate high-resolution images. In traditional GANs, the generator generates an image and the discriminator evaluates the image to see if it is real or fake. In LapGAN, the generator generates the image in the Lap-CNN structure and the discriminator uses the Laplacian Pyramid to evaluate the image in different scales. The Laplacian Pyramid is a multi-scale representation of an image, which allows the LapGAN to focus on different levels of details in the image. LapGAN operates in two stages: training and generation. In the training stage, the LapGAN is trained on a dataset of real images. The generator uses the Lap-CNN structure to generate fake images, and the discriminator evaluates the images to determine if they are real or fake. The generator and discriminator are trained using the adversarial loss function. The training process continues until the generator is able to produce high-resolution images that can fool the discriminator. In the generation stage, the trained LapGAN generates new images by starting from a random noise vector. The generator uses the Lap-CNN structure to transform the random noise into a high-resolution image, which is then evaluated by the discriminator. The generated images are compared to the real images in the dataset to evaluate the quality of the generated images.

LapGAN has several advantages over traditional GANs. Firstly, it generates high-resolution images by utilizing the multi-scale representation of the Laplacian Pyramid. Secondly, it is capable of generating images that have a high level of detail, as it focuses on different levels of details in the image. Thirdly, LapGAN is more computationally efficient compared to traditional GANs, as it uses a smaller number of parameters. In conclusion, LapGAN is a promising tool for generating high-resolution images. It combines the strengths of GANs and Laplacian Pyramid to generate images that have a high level of detail and quality.

In 2015, researchers from New York University and the Facebook AI Research (FAIR) team came up with the idea for the Laplacian pyramid of Adversarial Networks (LAPGAN) [26]. They produced a more powerful variation of the GAN by integrating the Laplacian pyramid [12] representation with CGAN. This variant of the GAN was able to create pictures that were more realistic than the original GAN. The Laplacian pyramid is a linear representation that may be inverted at numerous different scales. The following are the two basic steps involved in LP.

1. Creating a new image d(I) with a size that is half that of the original one requires first smoothing the original picture I by convolving it with a low-pass filter (such as the Gaussian filter), and then down sampling (operation d(.)) it by two to generate the new image.
2. Up sampling (operation u(.)) the new picture d(I) by inserting zeros into each row and column, and then passing it through the same filter, will produce an expanded image u(I) with the same dimensions as the first one. Last but not least, we have two pictures

taken from neighboring levels that have been subtracted one pixel at a time to produce the detail image L(I) provided by

$$L(I) = I - u(I) \tag{2.4}$$

The procedures described above can be carried out in a recursive manner an infinite number of times, K, to produce a picture of extremely modest proportions (leq8.8). The following table lists the values for the coefficient h at each level k of the Laplacian pyramid:

$$h_k = L_k(I) = I_k - u(I_{k+1}) \tag{2.5}$$

In order to reconstruct the LP, you must first invert the procedures described above and then extract the detailed picture from its higher level using the formula:

$$I_k = u(I_{k+1}) + h_k \tag{2.6}$$

The image with its full resolution may be recovered once the image h has been upsampled many times and added to neighboring images with a finer degree of detail.

Sampling and training are the two distinct components that make up LAPGAN. The LAPGAN sampling process involves training a collection of generative convert models (G0–GK) on the CGAN in order to provide an accurate representation of the LP coefficients hk at each level. The 2.6th equation will be rewritten as shown below.

$$\tilde{I}_k = u(\tilde{I}_{k+1}) + \tilde{h}_k + G_k(z_k, u(\tilde{I}_{k+1})) \tag{2.7}$$

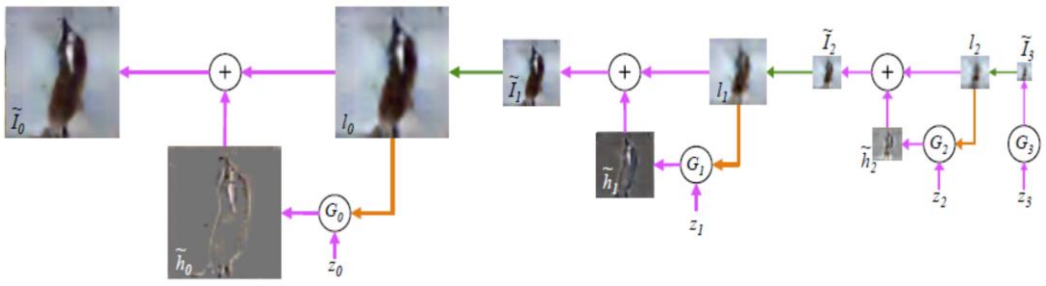


Figure 2.8: Sampling procedure: upsampling(green), conditioning variable(orange). Image from [26].

At the final level GK, IK+1 is reduced to zero in order to produce a residual picture IK by the application of the noise vector zK = GK (zK). Figure 2.8 depicted a sampling technique that took advantage of four generative models with the K parameter set to three in order to sample a 64 x 64 picture. The discriminator Dk always takes hk, hk, and lk as inputs when it is being trained. This is part of the training method. Whereas lk is input into the generator Gk in order to generate either a genuine or a created sample hk with an equal chance of occurrence.

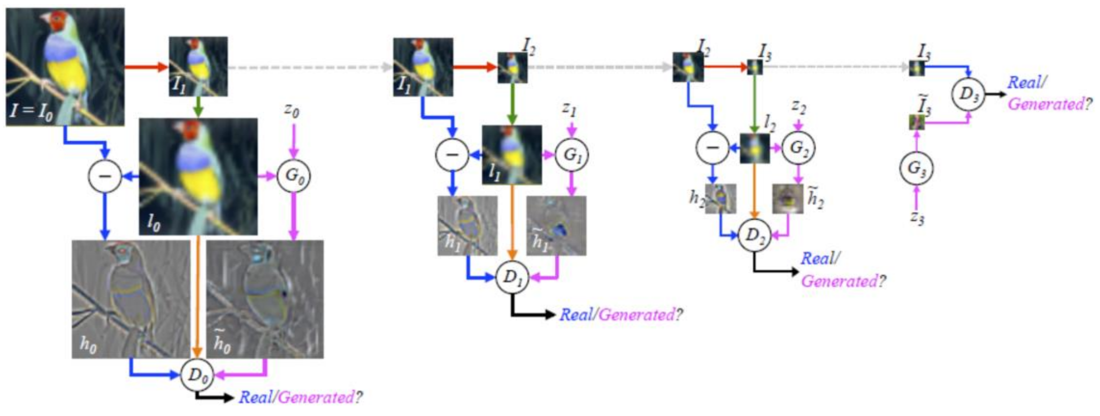


Figure 2.9: LAPGAN training procedure: downsampling(red), upsampling(green), real case(blue), generated case(magenta). Image from [26]. Figure 2.8 depicted a four-level Laplacian pyramid training process, which began with a 64 x 64 picture and culminated in an 8 x 8 image. The key concept of LAPGAN is to separate the generation into a series of subsequent refinements, with the goal of preventing overfitting by allowing each pyramid to be trained in isolation. Experiments carried out by LAPGAN on the CIFAR-10, STL, and LSUN datasets showed that both models obtained much greater log-likelihood than the vanilla GAN. In addition, there were 15 human volunteers who took part in the studies, and they correctly identified around 40% of the LAPGAN-generated samples as the genuine photograph.

2.7 Information Maximizing GAN(InfoGAN)

InfoGAN is a type of generative adversarial network (GAN) that aims to maximize the information about the factors of variation in the data, as opposed to only trying to generate realistic samples. InfoGAN does this by adding an auxiliary task to the standard GAN architecture to encourage the generator to produce outputs that are interpretable with respect to specific latent factors. By doing so, InfoGAN can discover and exploit the disentangled structure in the data, allowing the generator to generate data that can be modified along specific dimensions, such as style, class, or even continuous attributes.

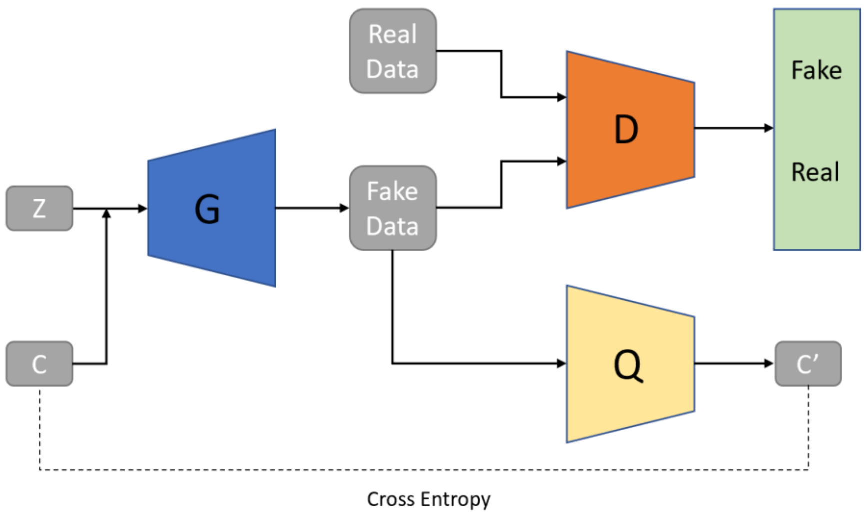


Figure 2.10: InfoGAN structure.

Information Maximizing GAN (InfoGAN) is a type of Generative Adversarial Network (GAN) that aims to disentangle the underlying factors of variation in the data by maximizing the mutual information between a subset of the latent code and the generated samples. In a traditional GAN, the generator and discriminator networks learn to generate and distinguish synthetic images from real images, respectively. The generator network takes a random noise vector as input and maps it to a synthetic image. The discriminator network takes both synthetic and real images as input and outputs a scalar indicating the probability that the image is real.

In InfoGAN, the random noise vector is split into two parts: a structured part and a non-structured part. The structured part is called the latent code, and it consists of a subset of the noise variables that control the specific factors of variation in the data. The non-structured part is a random noise that adds variability to the generated images. The goal of InfoGAN is to maximize the mutual information between the latent code and the generated images.The InfoGAN architecture consists of the generator network, discriminator network, and an additional network called the Q-network. The Q-network maps the generated images to the latent code values and estimates their distributions. The Q-network is trained to maximize the mutual information between the latent code and the generated images. In summary, InfoGAN aims to learn interpretable representations of the data by maximizing the mutual information between a subset of the latent code and the generated samples. The InfoGAN architecture consists of the generator, discriminator, and Q-network, and it can be used for various tasks such as image generation, representation learning, and control.

InfoGAN [13] was developed by Xi Chen and his colleagues in 2016, and it is an extension of the vanilla GAN that is based on information theory. In addition to the random noise Z, they suggested using a generator for the latent code C as another additional input. Thus the generator output becomes G(z,c) (z,c). Where the capital letter C is aimed towards the structured aspects of the genuine data distribution. In order to prevent the generator from ignoring the new information by discovering that PG(x c) = PG(x), there has to be a high level of mutual knowledge between c and G(z,c). As a result, the cost function of CGAN may be reformed as shown in the following example.

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c;G(z,c)),\ \lambda \geq 0 \tag{2.8}$$

It may be challenging to maximise the mutual information I(c ; G(z,c)) due to the fact that the posterior distribution P (c|x) is sometimes difficult to get. In order to make an approximation of the posterior distribution P (c|x), an auxiliary classifier called Q is constructed. With the exception of the output layer, the Q and the discriminator share all of the layers. Through the use of the approach of variational information maximisation, a lower bound for the mutual information known as LI (G,Q) is determined; hence, the equation 2.8 may be expressed as follows.

$$\min_{G,Q} \max_D V_{InfoGAN}(D,G,Q) = V(D,G) - \lambda L_I(G,Q),\ \lambda \geq 0 \tag{2.9}$$

This unsupervised variation of the GAN can learn interpretable representations of the datasets it is given, and it is not too difficult to train it.

2.8 Deep Convolution GAN (DCGANs)

Deep Convolutional Generative Adversarial Networks (DCGANs) are a type of Generative Adversarial Networks (GANs) that use convolutional neural networks for generating new images. The architecture of DCGANs is designed to address the challenges faced by traditional GANs in generating high-quality images. DCGANs consist of two neural networks – the generator and the discriminator. The generator network is responsible for creating new images, while the discriminator network evaluates the authenticity of the generated images. The generator network in DCGANs consists of a series of deconvolutional layers, batch normalization layers, and ReLU activation layers. The input to the generator is a random noise vector, which is fed through the network and transformed into an image. The output of the generator is a generated image, which is then fed into the discriminator network.

The discriminator network in DCGANs consists of a series of convolutional layers, batch normalization layers, and ReLU activation layers. The input to the discriminator is either a real image or a generated image from the generator network. The output of the discriminator is a prediction of whether the input image is real or fake. The two networks are trained simultaneously in a zero-sum game framework, where the goal of the generator is to produce images that the discriminator cannot distinguish from real images, and the goal of the discriminator is to correctly identify fake images generated by the generator. The training process involves updating the weights of the generator and discriminator networks based on the output of the discriminator.

DCGANs have been found to be effective in generating high-quality images that are almost indistinguishable from real images. This has led to their use in a variety of applications, including image generation, image editing, and image super-resolution. Deep Convolutional GAN was first presented by Alec Radford and colleagues [63] in the year 2015, and its introduction is widely regarded as an important milestone in the history of GAN. The author suggested that a Multi-layer Perceptron (MLP) structure be replaced with a Convolutional Neural Network (CNN) structure in GANs in order to construct a more stable architecture for training and obtain outstanding picture creation performance. This was done in order to achieve these goals. In addition to it, a number of different methods have been suggested as potential solutions to stabilise the training process.

- In both the generator and the discriminator, replace the spatial pooling layer with strided convolutions. This will allow the generator and the discriminator to learn their own spatial downsampling.
- Applying the Batch Normalization technique, which might normalise layer inputs to have zero mean and unit variance, on both the generator and the discriminator is a good idea.

- The Tanh function is used in the output layer of the generator, whereas the Rectified Linear Units (ReLU) activation is used in the other levels. LeakyReLU activation is utilised across all layers in Discriminator.

Following the publication of DCGAN, the majority of newly developed GANs pertaining to image processing began utilising the same design in the generator and using DCGAN as a baseline.

2.9 Wasserstein Generative Adversarial Network (WGAN)

Wasserstein Generative Adversarial Network (WGAN) is a type of generative adversarial network (GAN) that uses the Wasserstein distance metric to measure the difference between the generated and real data distributions. This distance metric is more stable compared to the traditional GAN, where the Jensen-Shannon (JS) divergence is used as the loss function. GAN was introduced to address the instability problems that arise in GANs when using the JS divergence as a loss function. These instability problems manifest in the form of mode collapse, where the generator only produces a limited number of different outputs, and difficulty in convergence, where the generator and discriminator become stuck in a never-ending loop of improved accuracy. The WGAN solves these problems by using the Wasserstein distance metric, which measures the earth mover's distance between two distributions. This metric measures the amount of work needed to transform one distribution into the other. The Wasserstein distance is calculated as the infimum (minimum) of the expected value of the cost of moving all the mass from one distribution to another.

The WGAN architecture is similar to that of a traditional GAN. The generator network takes a random noise vector as input and generates a sample of the target data distribution. The discriminator network takes both real and generated samples as inputs and outputs a probability value that indicates the authenticity of the sample. The loss function used in the WGAN is the Wasserstein distance between the real and generated data distributions. This loss function is calculated using the earth mover's distance, which is estimated using a criterion called the Kantorovich-Rubinstein duality. The loss function is optimized using the gradient descent algorithm. In conclusion, the Wasserstein GAN is a type of GAN that uses the Wasserstein distance metric to measure the difference between the generated and real data distributions.

The WGAN architecture is similar to that of a traditional GAN, but the loss function is more stable, resulting in better convergence and more diverse output.

Martin Arjovsky and colleagues [12] offered to utilise Earth-Mover (EM) or Wasserstein-1 Distance as a replacement for the original cost function. This would mean that the discriminator would no longer be used as a binary classifier to evaluate whether the input was phoney or real. The following is the definition of the EM distance:

$$W(P_r, P_g) = \inf_{\gamma \in \prod(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2.10}$$

where the notation (Pr,Pg) designates the sets of all joint distributions (x,y) whose marginals are, respectively, Pr and Pg. The inf (infimum) is notoriously difficult to solve. Because of this, the Kantorovich-Rubinstein duality [74] may be utilised to reformulate the aforementioned function to:

$$\sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_\theta}[f(x)] \tag{2.11}$$

where sup upper bound over all the 1-Lipschitz function. Continuous and differentiable, the EM distance may be used to assess the degree to which two distributions, in which there is either no or just a very small amount of overlap, differ from one another (gradient is zero). In this tricky zero-sum game, the generator may be adequately taught even if the discriminator always comes out on top. This has the potential to alleviate the vanishing gradients problem that arises during GAN training and improve learning stability.

2.10 Wasserstein GAN with Gradient Penalty (WGAN-GP)

Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) is a variant of the traditional Generative Adversarial Network (GAN) that uses a new loss function to stabilize the training process and generate high-quality samples.WGAN-GP is based on the Wasserstein distance, which is a measure of the difference between two probability distributions. Unlike the traditional GAN loss function, which is based on cross-entropy, the Wasserstein distance is easier to optimize, leading to a more stable and effective training process. The WGAN-GP algorithm consists of two main components: a generator network and a discriminator network. The generator network creates fake samples, while the discriminator network attempts to distinguish between real and fake samples. The two networks are trained in a minimax fashion, where the generator tries to create samples that fool the discriminator, and the discriminator tries to correctly classify the samples.

In the WGAN-GP algorithm, the discriminator network is trained to approximate the Wasserstein distance between the real and fake distributions. To ensure that the generator creates high-quality samples, a gradient penalty term is added to the loss function to encourage the discriminator to be differentiable and avoid mode collapse. The gradient penalty term measures the gradient norm of the discriminator network and penalizes it if it becomes too steep, leading to poor generalization and poor sample quality. This term helps to keep the generator and discriminator networks in a balanced state, leading to a stable and effective training process.

Overall, WGAN-GP is a promising alternative to traditional GANs, as it can lead to better sample quality and stability. It is also more computationally efficient than other variants of GANs, making it a popular choice for many applications.

Despite the fact that WGAN exhibited considerable increases in stability during GAN training, it is still possible for it to fail to converge because of its weight clipping on the critic (discriminator). In order to make sure that the Lipschitz requirement is followed, Gulrajani and colleagues [35] suggested including a gradient penalty in the original critic loss function. The goal is as follows:

$$L = \mathbb{E}_{\tilde{x}\sim P_g}[\mathbf{D}(\tilde{x})] - \mathbb{E}_{x\sim P_r}[\mathbf{D}(x)] + \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(\| \nabla_{\hat{x}} \mathbf{D}(\hat{x})\|_2 - 1)^2] \tag{2.12}$$

When the random samples are represented by xˆ ∼ P (ˆx). When compared to WGAN, WGAN-GP is more stable and has the potential to converge quicker on ImageNet, LSUN, and CIFAR-10 datasets thanks to the use of Adam hyperparameters.

2.11 Progressive Growing of GANs (PROGAN)

Progressive Growing of Generative Adversarial Networks (PROGAN) is a deep learning architecture that was introduced to generate high-quality synthetic images. It is a type of Generative Adversarial Network (GAN) that allows the generator and discriminator networks to grow progressively in complexity during the training process.In traditional GANs, the generator and discriminator networks have a fixed architecture and are trained simultaneously.

This can lead to instability in the training process and can result in poor quality images. PROGAN addresses this issue by starting with a very simple generator and discriminator, and gradually increasing the complexity of these networks over time.

The progressive growing process in PROGAN can be divided into several stages:

- Start with low-resolution images: The first stage of training begins with very low-resolution images, such as 4x4 or 8x8 pixels. This makes it easier to train the networks and to avoid overfitting.
- Grow the generator and discriminator: As the training progresses, the complexity of the generator and discriminator networks is gradually increased by adding more layers and neurons. This process is repeated until the desired image resolution is reached.
- Fine-tuning: After the desired resolution is reached, the generator and discriminator networks are fine-tuned to improve their performance.
- Synthesize high-resolution images: Finally, the trained networks can be used to generate high-resolution synthetic images.

One of the key benefits of PROGAN is that it allows the generator and discriminator to be trained in a stable and controlled manner. This results in the generation of high-quality images that are indistinguishable from real images. Additionally, the progressive growing process enables the networks to learn a hierarchy of features, from simple edge and texture patterns at lower resolutions to complex object shapes and structures at higher resolutions. Overall, PROGAN is a promising technique for generating high quality synthetic images and has been used in various applications such as computer vision, gaming, and video production.The DCGAN was able to attain a certain amount of success in the creation of huge size images. However, it is still lacking in terms of its capacity to generate images with a high quality. The reason for this is because if the resolution is good enough, the discriminator will be able to distinguish the artificially created image from a real one with ease. In addition to this, because of this, there would be an increased likelihood of mode collapse occurring while training with high-resolution samples. In 2018, the NVIDIA team led by Kerras et al. announced PROGAN [46], an algorithm that could create images with a resolution of 1024 by 1024 while still requiring less time for training (in terms of time). The Laplacian pyramid was utilised to handle the dataset in the optimal implementation of PROGAN. This allowed for the training of the generator and discriminator to begin at a lower resolution (4 x 4). After a compelling result was obtained in the previous level of training, the resolution was increased to begin a new cycle of practice in the following level.

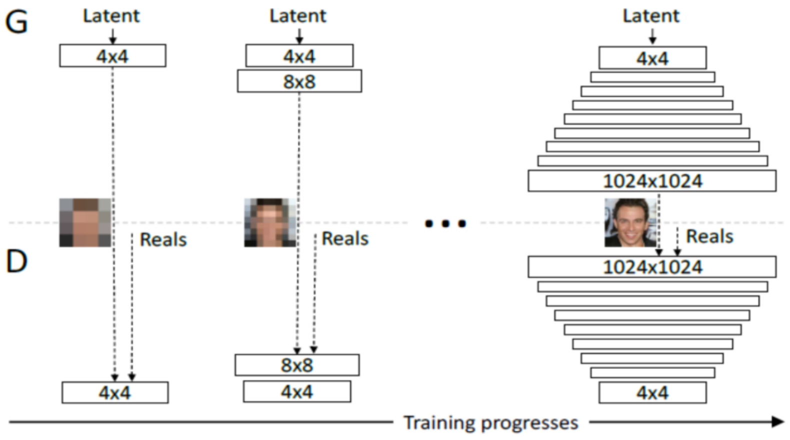


Figure 2.11: Sample of PROGAN training steps. Image from [46].

This allowed for the training of the generator and discriminator to begin at a lower resolution (4 x 4). After a compelling result was obtained in the previous level of training, the resolution was increased to begin a new cycle of practice in the following level. In the meanwhile, the weights from the previous level will be preserved, but they will not be locked. In addition, a new weight that varies linearly from 0 to 1 is introduced in order to fade the transition to the new layer in networks in a seamless manner. Figure provides an instance of this training approach. In this fashion, the networks will continue to add a new layer with resolution that is twice as high until they meet the output size objective. The PROGAN models were trained on the CIFAR-10 dataset (image size 32 31), the LSUN dataset (image size 256 256), and the CelebA-HQ dataset (image size 1024 1024), using the loss function of the WGAN-GP. Only pixelwise normalization was implemented on the generator while the training procedure was being carried out. The introduction of several methods that do not rely on any specific loss function to improve the training stability and substantially decrease the training time of GAN by anywhere from two to six times faster, depending on the final output resolution, is a milestone in the history of GAN. These methods were introduced by PROGAN.

2.12 Self-Attention GAN (SAGAN)

Self-Attention Generative Adversarial Network (SAGAN) is a type of Generative Adversarial Network (GAN) that was introduced in 2018. SAGAN is an improvement over traditional GANs in terms of generating high-quality images and preserving fine details. It accomplishes this by incorporating self-attention mechanisms into the generator and discriminator network.

A self-attention mechanism refers to the ability of the network to focus on specific parts of an image and adjust the weights of the neurons in those parts accordingly. This allows the network to pay more attention to important details in the image and produce more realistic outputs. The generator in a SAGAN network consists of multiple self-attention blocks that are interleaved with traditional convolutional layers. The self-attention blocks take the feature map from the previous layer as input and apply self-attention mechanisms to generate a new feature map. The generator also uses a residual connection, where the input feature map is directly added to the output feature map, to preserve fine details in the image. The discriminator network in a SAGAN network also uses self-attention mechanisms to evaluate the realism of the generated image. The discriminator network consists of multiple self-attention blocks that are interleaved with traditional convolutional layers. The self-attention mechanisms in a SAGAN network are trained to attend to different regions of the image depending on the task at hand. During training, the generator network is trained to generate images that fool the discriminator network into thinking that they are real, while the discriminator network is trained to distinguish real images from fake ones.

In conclusion, SAGAN is a powerful GAN architecture that has shown great promise in generating high-quality images. The use of self-attention mechanisms allows the network to focus on specific parts of the image and adjust the weights of the neurons in those parts, accordingly, leading to better image quality and preservation of fine details.

The conventional CNN-based GANs frequently struggle when it comes to capturing structural patterns or trivial feature alterations (for example, human faces with the eyes and nose in the incorrect positions), and they also find it challenging to learn from multi-class data. The Self-Attention model [14] is similar to the Attention model in that it shares fundamental ideas and the majority of its mathematical techniques. It is an attention module that was created to associate complimentary characteristics across vast distances in a sample, which can assist the generator in producing a picture with finer and more realistic detail.

Zhang, Metaxas, and others [80] came up with the idea for the SAGAN, which incorporated a Self-Attention mechanism into the Convolutional GAN model. This was done in order to facilitate the learning of dependencies that cut across spatial regions of an image without compromising the computational or statistical efficacy of CNNs. The non-local neural network was modified to function as an attention module in both the generator and the discriminator in order to assist in capturing the global discrepancies and correlations that exist between any particular pixel and other places.

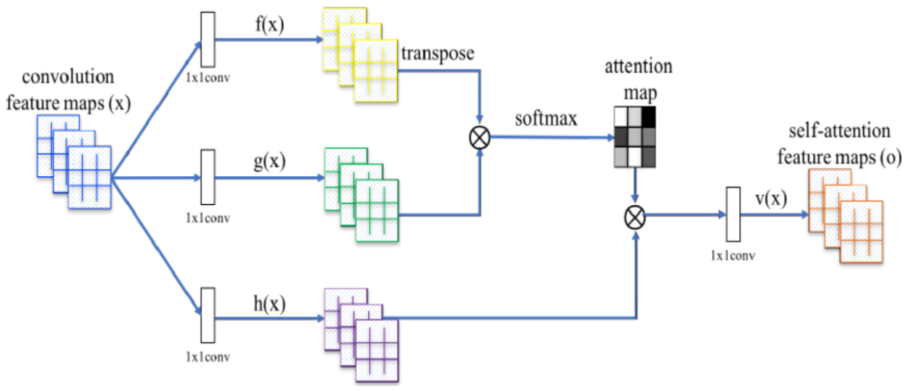


Figure 2.12: Self-Attention module of SAGAN. Image from [80].

Figure 2.12 provides a visual representation of the operational procedure of an attention module. Initially, the preprocessed image characteristics X (size Channels Width Height) were split, with the help of a notion from the Transformer [15], into three different feature spaces. These feature spaces were as follows: Key f (x) = W_f X, Query g(x) = W_g X, and Value h(x) = W_hX. After then, the Key and Query were used to create an attention map with a dimension of N by N, where N represents the width times the height.

$$\beta_{i,j} = softmax(f(x_i)^\top g(x_j)) \quad (2.13)$$

$$\mathbf{o}_j = W_v(\sum_{i=1}^{N} \beta_{i,j} h(x_i)) \quad (2.14)$$

In the equation that was just presented, the value of the parameter $\beta_{i,j}$ indicates the degree to which the model should pay attention to a specific location within the matrix. The column vector o represents the output of the attention layer that has a size of N, and the variables W_f,W_g,W_h,W_v represent the 1.1 convolution weight learning filter. In addition to this, an attention layer scale multiplier is applied to the output o of the attention layer, and this result is then added back to the original input.

$$y = x_i + \gamma \mathbf{o}_i \quad (2.15)$$

γ is set to start from 0 to allow the model to rely on the cues in the local regions at first, then increasing gradually to allow the attention module to focus on other regions.

2.13 Large Scale GAN (BigGAN)

Large Scale Generative Adversarial Network (BigGAN) is a deep learning-based architecture for generating high-resolution and high-quality images. It is designed to overcome the limitations of previous GAN models, which were unable to generate high-resolution images due to their limited computational resources. The BigGAN architecture consists of two parts: the generator and the discriminator. The generator network is responsible for generating images, while the discriminator network is responsible for classifying images as either real or fake. The two networks are trained in an adversarial manner, with the generator trying to generate images that are indistinguishable from real images, and the discriminator trying to identify fake images. BigGAN uses a deep convolutional neural network (DCNN) architecture, which is composed of multiple layers of convolution, batch normalization, activation functions, and upsampling layers. The generator network is trained to generate high-resolution images, while the discriminator network is trained to distinguish real images from fake images.One of the key features of BigGAN is its hierarchical structure, which allows it to generate high-resolution images. The hierarchical structure of BigGAN is based on the idea that images can be decomposed into smaller and smaller parts, starting from low-resolution images and then gradually increasing the resolution. This decomposition process is performed by the upsampling layers in the generator network.

Another key feature of BigGAN is its use of truncated normal distribution, which is a variant of the normal distribution that has a limited range. This truncation helps to reduce the number of mode collapses that occur in GAN training, which is a common problem in GANs. BigGAN also uses a class-conditional loss function, which allows the network to generate images of a specific class or type. The class-conditional loss function takes the class labels as input and trains the generator network to generate images of a specific class or type.In conclusion, BigGAN is a powerful generative model that is capable of generating high-resolution and high-quality images. Its hierarchical structure and truncated normal distribution help to reduce mode collapses and improve the quality of the generated images, while its class-conditional loss function allows it to generate images of a specific type or class.

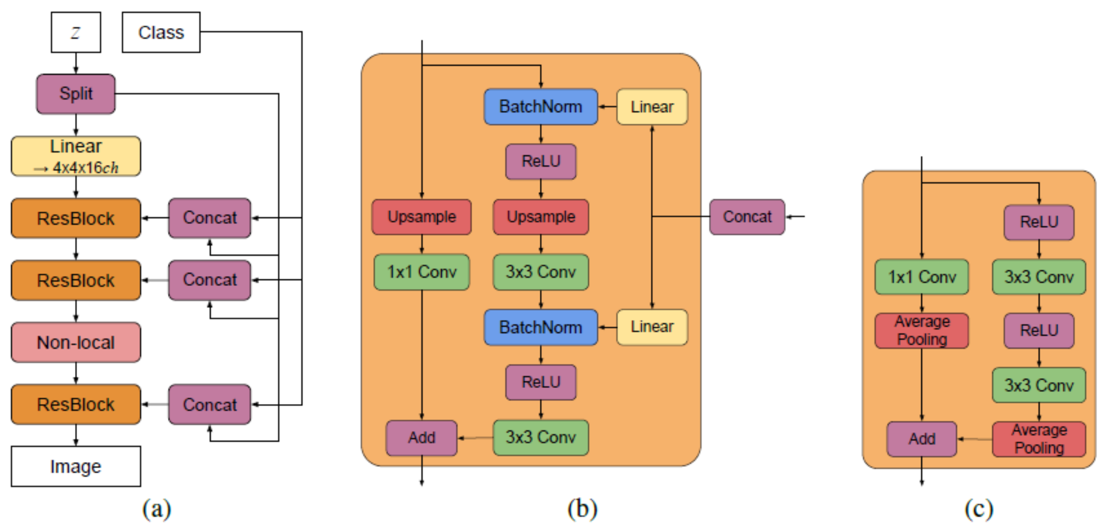


Figure 2.13: (a) A typical architectural layout for BigGAN's G; details are in the following tables. (b) A Residual Block (ResBlock up) in BigGAN's G. (c) A Residual Block (ResBlock down) in BigGAN's D. figure from [10]

BigGAN, which was first suggested in 2018 by Andrew Brock et al. [11] and then presented at ICLR, accomplished very good results when it was tested on the ImageNet dataset. They recommended using SAGAN as their starting point and illustrating how scaling up GAN in combination with other training strategies might improve the model's overall performance. In addition, the implementation of Spectral Norm in the generator underwent various iterations of adjustment. The performance of the model, which used a different latent distribution for sampling than was used in training, could be improved through the use of a truncation trick known as truncating the z vector (z N (0,I)) by resetting the value whose magnitude is outside of the desired range. This would produce a higher result in FID. The truncation approach, at its core, involves making a choice between the picture quality and the image diversity available. By reducing the sampling range, the photos will be of higher quality; however, this will come at the expense of a reduced diversity of samples. The following are some more detailed advances on training and design.

• The Self-Attention module that is available through SAGAN has been implemented in order to teach feature mapping using hinge loss.

• To reduce the computational and memory expenses, shared embedding was utilised in the generator rather than a distinct layer for each embedding in BatchNorm. This was done in order to save space.

• a distinct difference in the rate of update between the discriminator and the generator, with the discriminator updating at a pace that is twice as fast as the generator. • the weights of the generator were first averaged before the discriminator evaluated generated samples.

• Orthogonal Initialization and Skip-z were both utilized in this process.

2.14 StyleGAN

By adding a fully linked Mapping network to the generator (Synthesis network), StyleGAN expands on this structure by creating a "style" vector (w ∈ W) from the standard seed (z ∈ Z). The authors claim that Adaptive Instance Normalization (AdaIN), which specialises this vector perlayer, results in a behaviour like style-transfer. In order to improve the clarity of the output information, a little amount of noise is also introduced to each block of the Synthesis network.

Figure 2.14 depicts StyleGAN’s whole organisational structure.[78]

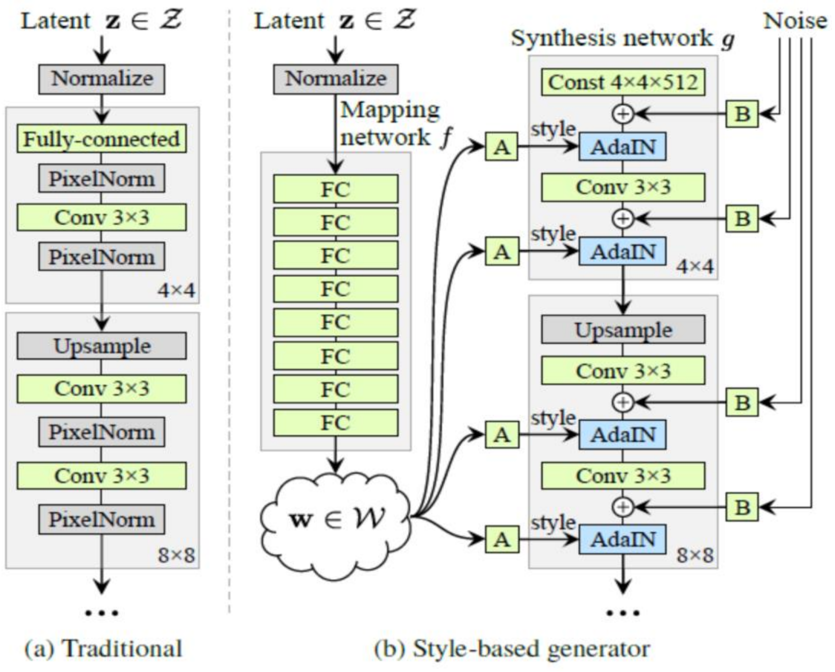


Figure 2.14: the structure of traditional Generator(a) and StyleGAN generator(b). Image from [47].

A GAN consists of two parts: a generator network and a discriminator network. The generator's goal is to generate fake images that look like real images, while the discriminator's goal is to distinguish real images from fake images. During training, the two networks are trained together in a zero-sum game, where the generator tries to generate images that the discriminator can't distinguish from real images, and the discriminator tries to correctly identify the fake images. In StyleGAN, the generator network uses a novel architecture that separates the generation of image structure and image style. The image structure is generated by a mapping network that takes a random noise vector as input and outputs a intermediate latent representation[78]. This intermediate representation is then transformed by a series of convolutional and upsampling layers to produce the final image.The image style is generated by using adaptive instance normalization (AdaIN) layers, which modify the intermediate latent representation based on the style information. The style information is derived from the input noise vector, and it can be manipulated to generate different styles of images. One of the key advantages of StyleGAN is its ability to generate high-resolution images. It can generate 1024x1024 pixel images, which is significantly higher resolution than other GANs at the time of its release. Additionally, StyleGAN also has a flexible structure that can be adapted to generate different types of images by changing the architecture of the generator and discriminator networks. StyleGAN (Style-based Generative Adversarial Networks) is a type of GAN that can generate high-quality images with fine-grained control over attributes such as age, gender, and facial expression. In the paper, the authors highlight the use of StyleGAN to generate realistic human faces with controllable attributes, which has applications in fields such as art, entertainment, and advertising [8].

Overall, StyleGAN is a state-of-the-art generative model that has been widely used in a variety of applications, such as generating photorealistic images for computer graphics, creating synthetic data for machine learning models, and producing art. The NVIDIA team (Tero Karras et al.) [47] came up with a novel generator design for GANs in 2019, and they based it on their previous work called PROGAN. First, the conventional input layer was switched out for one that was comprised of a learnt constant. After that, an intermediate latent space was made, which contains input latent code z Z that has been normalised and non-linearly mapped through the new input layer. This space was made since an intermediary latent space was needed. The latent component is separated from the variation in the intermediate latent space, which is denoted by the notation w W. A learnt affine transform was used to construct the style y = (ys,yb), which was then worked on by adaptive instance normalisation (AdaIN), which may be described as follows:

The NVIDIA team (Tero Karras et al.) [47] came up with a novel generator design for GANs in 2019, and they based it on their previous work called PROGAN. First, the conventional input layer was switched out for one that was comprised of a learnt constant. After that, an intermediate latent space was made, which contains input latent code z Z that has been

normalised and non-linearly mapped through the new input layer. This space was made since an intermediary latent space was needed. The latent component is separated from the variation in the intermediate latent space, which is denoted by the notation w W. A learnt affine transform was used to construct the style y = (ys,yb), which was then worked on by adaptive instance normalisation (AdaIN), which may be described as follows:

$$AdaIN(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i} \quad (2.16)$$

The Gaussian noise is fed to each layer of the Laplacian pyramid networks after the convolution process in order to create stochastic variation. As a result, the detail in the generated image (such as curls of hair or skin pores), will be finer while the major aspect (such as pose or identity) will not be affected.

The conventional GAN is unable to separate the finer details of a picture because the features are intertwined with one another, which prevents it from doing so. In addition to these two methods for determining how to quantify disentanglement, the intermediate space may make it possible for Style- GAN to relax some of the constraints placed on entanglements.

1. The length of the perceptual path; the perceptually-based pairwise image distance serves as the metric foundation for the calculation that determines the difference in weights between the VGG16 em-beddings.

2. Linear separability: To separate latent space into binary classes, an auxiliary classifier that has the same architecture as the discriminator is trained and used. The performance of the classifier was then evaluated with a linear support vector machine.

In StyleGAN, the performance of the algorithm was evaluated using a pair of different loss functions. On the CelebA-HQ dataset, the style-based generator WGAN-GP was employed, while on the FFHQ dataset, non-saturating loss with R1 regularisation was utilised. In both cases, the style-based generator obtained approximately 20% improvement over the traditional generator in terms of FID metrics.

2.15 CycleGAN

CycleGAN is a Generative Adversarial Network (GAN) model developed for the purpose of unpaired image-to-image translation. In traditional GANs, the generator and discriminator networks are trained on a paired dataset, meaning that the input and output images are related in some way. However, in many real-world scenarios, such paired datasets are not available, and CycleGAN was developed to address this problem. CycleGAN is a GAN architecture that is used to learn mappings between two different image domains, without the need for paired training data. The model is trained to learn two mappings, one from domain X to domain Y, and the other from domain Y to domain X [3]. The generator network in CycleGAN consists of two encoders and two decoders, where the encoders are used to extract features from the input images and the decoders are used to generate the output images. The discriminator network in CycleGAN is used to distinguish between the generated images and the real images. CycleGAN has been used for a variety of applications such as style transfer, image-to-image translation, and object transfiguration [3]. The core idea behind CycleGAN is to use two generators and two discriminators, one for each direction of image translation. For example, if you wanted to translate an image of horses to zebras, you would have a generator network that takes an input image of horses and produces an output image of zebras. The corresponding discriminator network would evaluate the authenticity of the generated image of zebras. Similarly, another generator and discriminator network would translate images of zebras back to horses. This second generator and discriminator are used to ensure that the output image of horses is consistent with the original input image of horses.

The main training objective of CycleGAN is to minimize the difference between the generated image and the target image, while also ensuring that the translated image is consistent with the input image. The loss function for CycleGAN consists of two parts:

Adversarial Loss: This is the standard GAN loss that ensures that the generated image is indistinguishable from the target image.

Cycle Consistency Loss: This loss ensures that the translated image is consistent with the original image. For example, if an image of horses is translated to zebras and then translated back to horses, it should be close to the original image of horses.

The CycleGAN architecture allows for a wide range of image-to-image translations, including translating images between different domains, such as horses to zebras, or converting photographs to paintings. Additionally, because the model is trained on unpaired datasets, it can handle large amounts of data that do not need to be paired.

Overall, CycleGAN is a powerful model for image-to-image translation that has found applications in many areas, such as artistic style transfer, image colorization, and even medical imaging.

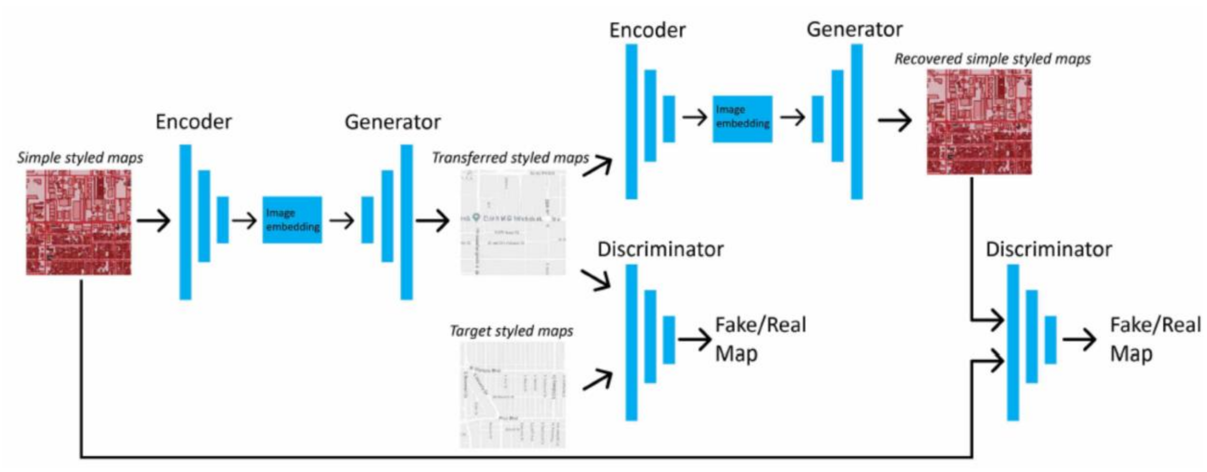


Figure 2.15: Structure of CycleGAN. Image from [45].

An architecture for unpaired image-to-image translation with a cycle consistency loss was first presented by Jun-Yan Zhu et al. [83] in the year 2017, and it has since been given the name CycleGAN. The goal of the image-to-image translation is to become familiar with the mapping that exists between pictures coming from domain X and images coming from domain Y. The vast majority of models that came before CycleGAN were supervised and heavily relied on paired pictures from many domains. The new design of CycleGAN does not require any paired training data, which results in a reduction in the overall cost of training. CycleGAN has two generators, G: X Y and F: Y X, as well as two discriminators: DY, which differentiates y from G(X), and DX, which separates x from F. Both generators and discriminators are introduced at the same time (Y). Figure 2.15 provides an illustration of the structure. Two different generators, G and F, were trained at the same time, and then a cycle consistency loss was introduced to promote F (G(x)) x and G(F (y)) y. This ensures that the trained versions of G and F will not be in conflict with one another. The following exposition will detail the goal function for mapping G: X to Y and DY.

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))] \tag{2.17}$$

While the generator G works to produce images in the manner of domain Y, the discriminator DY examines both the created images and the real samples in an effort to determine which is which. F and DX are likewise rivals in this regard and compete with one another. And the negative log likelihood in the goal function 2.17 has been swapped out for a least-squares loss [58] in order to attain higher levels of stability and performance in the training process. In addition to this, adversarial loss is utilised in both generators in order to prevent mode collapse and to stabilise the training process. The following is a complaint regarding the lack of cycle consistency:

$$\mathcal{L}_{cyc}(G,F) = \mathbb{E}_{x\sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim p_{data}(y)}[\|G(F(y)) - y\|_1] \tag{2.18}$$

The full objective function is:

$$\mathcal{L}_{GAN}(G,F,D_X,D_Y) = \mathcal{L}_{GAN}(G,D_Y,X,Y) + \mathcal{L}_{GAN}(F,D_X,X,Y) + \lambda\mathcal{L}_{cyc}(G,F) \tag{2.19}$$

Where aim to solve:

$$G^*, F^* = \arg\min_{G,F}\min_{D_X,D_Y} \mathcal{L}_{GAN}(G,F,D_X,D_Y) \tag{2.20}$$

Encoder, Transformer, and Decoder are the three components that make up the generator that CycleGAN uses. This generator is somewhat comparable to an autoencoder. Every single layer of the generator has its own dedicated ReLU activation and instance normalisation component. PatchGAN [43], which is computationally efficient and allows the discriminator to focus on shallow features, is where the discriminator network was transplanted from (textures or surface color). In addition, in order to lessen the likelihood of the model oscillating, the method proposed by Shrivastava et al. [69] was utilised. This technique calls for the discriminator to be fed a history of the images it has previously generated, and a pool that is capable of storing fifty years' worth of history was established. In the end, an Adam solver with a batch size of 1 and a value of 10 was configured to train at a learning rate of 0.0002 for the first 100 epochs and then linearly declined to zero for the next 100 epochs. This procedure was repeated until the learning rate reached zero. The findings obtained by CycleGAN are persuasive, and the picture quality is exceptional, especially when compared to those obtained by other paired image translation models. The CycleGAN isn't particularly excellent at handling changes in geometry, in contrast to the generator model, which concentrates mostly on aesthetic alterations.

2.16 Your Local GAN

However, there are certain restrictions: they are computationally and statistically inefficient [24]. The usual dense attention layer used in SAGAN and BigGAN might aid the GAN model in learning long-range relationships. Recent research [71] demonstrates that most attention models have a propensity to focus on local neighbourhoods, which helps to explain SAGAN and BigGAN's limitations. The Sparse Transformer [24] introduced the sparse attention layers, which might offset the drawbacks but were made for one-dimensional input. With the addition of a new local layer of sparse attention for two-dimensional data and the information-theoretic framework of Information Flow Graphs (IFGs) [27], YLGAN [25] updated SAGAN.

Instead of using standard operation in SAGAN to output the attention matrix

As AX;Y [a; b] = f(xa)^T g(xb), YLGAN proposed to split attention by adding the binary mask Mi 2 f0; 1gN_N to calculate a new matrix Ai X;Y where Ai X;Y [a; b] =

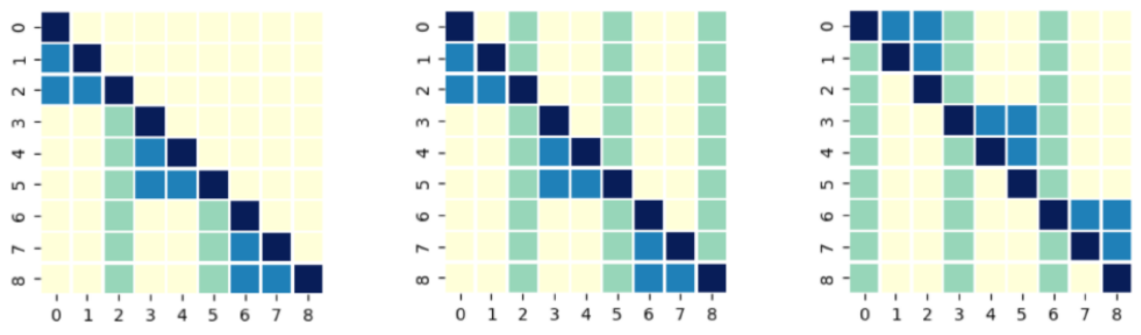
$$\begin{cases} A_{X,Y}[a,b], & M^i[a,b]=1 \\ -\infty, & M^i[a,b]=0 \end{cases}$$

. $-\infty$ means this specific position will not contribute to

(2.21)

the result, which is essential for cutting down the amount of time spent computing.

As an extension of the xed pattern from [24], two sparse patterns called Left to Right (LTR) and Right to Left (RTL) were developed to manage two-dimensional structures of images while supporting complete information of the related IGFs. These patterns are abbreviated as LTR and RTL, respectively. Figure 2.16 demonstrates the results of applying a two-step sparsification procedure to these designs. In addition, the process known as ESA (Enumerate, Shift, Apply) was used in order to be aware of the locale in two dimensions. First, the Manhattan distance of pixels from the zero location (0; 0) is enumerated.



(a) Attention masks for Fixed Pattern

(b) Attention masks for Left To Right (LTR) pattern.

(c) Attention masks for Right To Left (RTL) pattern.

Figure 2.16: YLGAN two-step sparsi_cations: dark blue color indicates of use both masks; Light blue color use _rst masks; green color use second masks;yellow color indicates sparsity. Image from [24].

Next, the indices of any given one-dimensional sparsi cation are shifted to match the Manhattan distance instead of the reshape enumeration. Finally, the new one-dimensional sparsi cation pattern is applied to the one-dimensional version of the reshaped image. Although the ESA technique can cause the output to become more distorted as a whole, according to resolution 128 128, using it is not a problem. In the studies, YLG-SAGAN scored 15.94 FID and 57.22 inception score while reducing training time by about 40 percent in comparison to SAGAN.

Other interesting studies have come up with various types of GANs all having different usages, one such study presents us with Pix2pix

This significant use of GANs is in image-to-image translation, which is the process of converting an image from one domain to another. Pix2Pix, which is introduced by Isola et al. (2017), is one of the most well-known image-to-image translation models that use GANs. The Pix2Pix model uses a U-Net-like generator network and a PatchGAN discriminator that only evaluates image patches instead of the whole image [2]. The objective function used in the Pix2Pix model can be formulated as follows:

$$L(G, D) = E(x,y)\sim p(x,y)[\log(D(x,y))] + E(x,z)\sim p(x,z)[\log(1 - D(x, G(x,z)))] + \lambda E(x,z)\sim p(x,z)[L1(y, G(x,z))]$$

where G is the generator network, D is the discriminator network, x and y are images from domains X and Y, z is a random noise vector, and λ is a hyperparameter that balances the GAN and L1 loss terms. The first term represents the adversarial loss, the second term represents the generator's ability to fool the discriminator, and the third term represents the L1 loss between the generated image and the ground truth image [2].

Several applications of GANs have been proposed and implemented, such as image generation, style transfer, image super-resolution, text-to-image synthesis, and speech synthesis, among others. GANs have also been used for data augmentation, which involves generating new data points from existing data to improve the accuracy of a machine learning model. Additionally, adversarial training, which involves training a model using both real and fake data generated by a GAN, has been shown to improve the robustness of machine learning models against adversarial attacks [5].

In particular, GANs have been used to translate images from one domain to another, such as converting a daytime scene to a nighttime scene or translating images of horses into zebras. These applications have important practical implications, such as enabling the creation of photo-realistic images for architectural visualization, or improving the quality of medical imaging for diagnosis and treatment [7].

One interesting outcome from the paper "CADA-GAN: Context-Aware GAN with Data Augmentation [9] that can be added to the effectiveness of this paper is of the proposed CADA-GAN in improving the visual quality of generated images.

In their experiments, the authors compared CADA-GAN with several state-of-the-art GAN models on three benchmark datasets, namely CIFAR-10, CelebA, and LSUN Bedroom. The results showed that CADA-GAN outperformed the baseline models in terms of various evaluation metrics, including Inception Score (IS), Fréchet Inception Distance (FID), and Learned Perceptual Image Patch Similarity (LPIPS)[9].

For instance, on the CelebA dataset, CADA-GAN achieved an IS of 8.26 and an FID of 10.39, which were significantly better than the baseline models. Moreover, the authors conducted a user study to evaluate the visual quality of the generated images, and the results showed that CADA-GAN received higher scores in terms of realism and diversity compared to the baseline models [9].

This outcome highlights the potential of CADA-GAN as a promising approach for generating high-quality images in various applications, such as image synthesis, style transfer, and image editing [9].

CHAPTER 3

DIFFERENT EVALUATION METRICS FOR GANS

This section focuses on the many methods that are already in use to quantitatively evaluate the performance of GAN models and the various efforts that have been made to obtain high-quality measurements in a variety of settings.

3.1 Inception Score (IS)

The Inception Score (IS) is a popular evaluation metric used to evaluate the quality of generative models, particularly generative adversarial networks (GANs). It is a composite metric that considers both the diversity and the quality of the generated samples. The Inception Score is based on the idea that high-quality generated images should have high activations in well-defined classes in the intermediate layers of a trained classifier network, referred to as an "Inception network". The Inception network is trained on real images to recognize objects and their attributes, such as the presence of faces, animals, textures, etc.

The IS is calculated as follows:

- First, the Inception network is used to classify the generated images into different classes. For each generated image, the network outputs a probability distribution over the classes, which represents the confidence of the network in classifying the image into each class.
- The average of the KL-divergence between the predicted class distribution and a uniform distribution is computed over all generated images. This measures the diversity of the generated images and penalizes the models that generate samples that belong to a single class or have a low entropy in the predicted class distribution.
- The final Inception Score is the exponential of the average KL-divergence, which is used as a normalization step to ensure that the score is always positive. The score ranges from 1 to infinity, with a higher score indicating higher-quality and more diverse generated images.

In summary, the Inception Score provides a single scalar value that summarizes the quality and diversity of the generated images. It is a widely used evaluation metric in the field of generative models and is considered to be a good indicator of the overall performance of a model.

However, it has been noted that the Inception Score is not a perfect evaluation metric and may not always align with human judgments of the quality and diversity of generated images.

There is a high potential for a correlation between the manual evaluation and the metric that was proposed by Salimans et al. [67] in 2016. The IS is based on two desired properties: i. the conditional label distribution p(yjx) calculated by fitting the generated image to an inception model should have low entropy for delity, and ii. the marginal label distribution Rp(yjx = G(z)) dz should have a high entropy for high diversity. Both of these properties should be met for the IS to be successful. The IS makes use of the Inception v3 Network, which is a deep neural architecture for classification that was trained on ImageNet [13]. ImageNet is a dataset that consists of 1.2 million RGB pictures from 1000 different classes.

$$IS(G) = \exp(\mathbb{E}_{p_g}[D_{KL}(p(y|\mathbf{x})\|p(y))]) \quad (3.1)$$

The KL divergence between the conditional label distribution p(yjx) and the marginal label distribution p is denoted by the notation DKL(p(yjx)kp(y) (y). If these two conditions were met by the model, then it would return a significant value for the IS. A high number of samples, such as 50,000, are recommended for use in the original research paper's implementation of the measure [67]. When analysing the effectiveness of the algorithms, the IS can act as a kind of general summary for the analysis.

Given a sample of size x, the empirical marginal label distribution takes into account the likelihood of being assigned the class label y. (i).

$$\hat{p}(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x^{(i)}) \quad (3.2)$$

N is the number of the synthetic data. The IS can be written under the empirical marginal label as below.

$$IS(G) \approx \exp(\frac{1}{N}\sum_{i=1}^{N} D_{KL}(p(y|\mathbf{x}^{(i)})\|\hat{p}(y))) \quad (3.3)$$

Research conducted by Barratt and Sharma (2018) examines the IS in great detail and identifies two primary issues: i. suboptimalities and ii. issues with the widespread utilisation of IS [13].

The first category of problems arises from a hypersensitivity to the natural element of chance that exists in the weights of the network.

The second category of problems consists of a variety of distinct restrictions. The dataset that is utilised in the training of the Inception network will have an effect on the IS. As a result, the first IS that was trained on the ImageNet network must to be utilised for the ImageNet generators. Another common problem is that the IS is unable to handle mode collapse failure when the generator generates the same outputs repeatedly and the discriminator stays in a local optimum solution [64]. This is a situation that occurs when the generator produces the same outputs again.

3.2 Fréchet Inception Distance (FID)

Fréchet Inception Distance (FID) is a commonly used evaluation metric in deep learning, particularly in generative models such as Generative Adversarial Networks (GANs). The FID measures the difference between two probability distributions, one generated by the model and the other from real data. The FID score is based on the Fréchet distance, which is a metric that measures the distance between two probability distributions. The Fréchet distance is calculated by comparing the features of two distributions in a high-dimensional feature space. The FID score is calculated as the Fréchet distance between the generated and real feature distributions, which is expected to be lower when the generated images are more similar to the real images.

To calculate the FID score, the following steps are performed:

- Compute activations of a pre-trained Inception V3 network on real and generated images. The activations are essentially the embeddings of the images in a high-dimensional feature space
- Compute the mean and covariance of the activations of real images.
- Compute the mean and covariance of the activations of generated images.
- Calculate the Fréchet distance between the two distributions by using the formula for the distance between two multivariate Gaussian distributions. The formula takes into account the means and covariances of both distributions.
- The final FID score is the square root of the Fréchet distance between the two distributions. The FID score is unitless, and a lower score indicates that the generated images are more similar to the real images.

The FID is considered to be a more robust metric than other commonly used metrics like the accuracy or mean squared error, as it takes into account both the similarity in the mean and covariance of the feature distributions. Additionally, the FID score can be computed quickly and is computationally efficient compared to other evaluation metrics.

Heusel et al.(2018) presented the Fréchet Inception Distance (FID), which is an advance on the IS since it takes into account real-world data in addition to synthetic samples [37]. This was accomplished by introducing the Fréchet Inception Distance (FID). The fundamental idea behind the FID is to determine the gap that exists between the probabilities of real-world samples, denoted by pw(:), and synthetic samples, denoted by p. (:). It is assumed that real and synthetic embedded data both follow a multidimensional Gaussian, with the embeddings of the generated data having a distribution of N(m;C) and the embeddings of the actual data following N. This assumption is based on the fact that real and synthetic embedded data are assumed to have the same shape N(mw;Cw). It is feasible to assess the degree of resemblance that exists between any two probability distributions by entering the distributions into the Multivariate Normal Fréchet Distance.

A low FID value indicates that the distributions are more concentrated. During the training process for the inception model, the values of the mean and covariance matrices are obtained.

$$d^2((\mathbf{m}, \mathbf{C}), (\mathbf{m}_w, \mathbf{C}_w)) = \|\mathbf{m} - \mathbf{m}_w\|_2^2 + Tr(\mathbf{C} + \mathbf{C}_w - 2(\mathbf{C}\mathbf{C}_w)^{\frac{1}{2}}) \tag{3.4}$$

performance of FID and IS under a variety of various sorts of sounds, and conclude that FID is better suited to deal with disturbances than IS. In contrast to the IS, the FID is able to determine whether or not the model produces only a single picture for each class [55]. This was demonstrated by Lucic et al., who noted that the FID is capable of detecting intra-class mode dropping. IS measures fidelity and variety, whereas FID estimates the distance between produced and real distributions, hence the values of these two statistics reflect different things when compared to one another. In order to provide a fair comparison between IS and FID, IS needs to be transformed into distance form. Because Heusel et al. demonstrated that IS has an upper bound, it is easy to compute the distance for IS as the difference between IS and the upper bound, which is equal to the size of the samples multiplied by m. This is because IS has an upper bound. On the other hand, FID is incapable of addressing the overfitting issue as well as memorization [77]. There are constraints placed on the generator and the features due to the fact that FID additionally makes use of a pre-trained inception network. Despite this, there is a possibility that the assumption of a Gaussian distribution is not guaranteed.

3.3 Modified Inception Score (m-IS)

Modified Inception Score (m-IS) is a variant of the traditional Inception Score (IS) metric for evaluating the performance of generative models in image synthesis. It was introduced as a solution to the limitations of the original IS, which did not take into account the diversity of the generated images. In simple terms, the IS measures the quality and diversity of the generated images by considering two key components: The probability assigned by a pre-trained classifier to the ground-truth class for each generated image. The entropy of the predicted class probabilities for each generated image However, the original IS had a few limitations, such as being sensitive to the choice of the pre-trained classifier, being prone to overfitting to the training data, and not taking into account the relative distances between the generated images and their closest training images. The m-IS addresses these limitations by incorporating a new component into the IS calculation: the relative distances between the generated images and their closest training images. The m-IS is defined as:

$$m\text{-}IS = exp(E(KL(p(y|x) \,||\, p(y))) - E(KL(p(y|x) \,||\, p(y)\_c)) \quad (3.5)$$

Where x is a generated image, y is the class predicted by the classifier, p(y|x) is the predicted class probabilities for x, p(y) is the marginal probability of class y, and p(y)_c is the marginal probability of class y for the closest training image to x. In essence, the m-IS takes into account not only the quality of the generated images (as measured by the entropy of the predicted class probabilities), but also their diversity (as measured by the relative distances to the closest training images). This makes the m-IS a more robust metric for evaluating generative models, as it considers both quality and diversity in a single score. In order to characterize diversity within a particular category, the modified Inception Score present by Gurumurthy et al. combines a cross-entropy style score -p(y|xi) log(p(y|xj)) where x_js are the samples with the same class as x_i derived from the outputs of the inception model [36]. The updated inception score with the crossentropy is as the equation below.

$$IS(G) = \exp(\mathbb{E}_{x_i}[\mathbb{E}_{x_j}[D_{KL}(P(y|x_i) \| P(y|x_j))]]) \quad (3.6)$$

The m-IS is capable of measuring both the diversity inside the intra-class sample and sample fidelity.

3.4 Precision and Recall Distance (PRD)

Precision and Recall Distance (PRD) is a method used to evaluate the performance of classification models. PRD is used to assess the accuracy of a model's predictions in comparison to the actual results. The precision and recall metrics are combined to give a single measure of accuracy, known as the PRD score. Precision measures the proportion of true positive predictions (correctly classified instances) in relation to the total number of positive predictions (true positives and false positives). It answers the question: "Out of all the positive predictions, how many were actually correct?" Precision is calculated as follows:

$$\text{Precision} = \text{True Positives} / (\text{True Positives} + \text{False Positives}) \tag{3.7}$$

Recall, on the other hand, measures the proportion of true positive predictions in relation to the actual number of positive instances. It answers the question: "Out of all the actual positive instances, how many were correctly classified?" Recall is calculated as follows:

$$\text{Recall} = \text{True Positives} / (\text{True Positives} + \text{False Negatives}) \tag{3.8}$$

The PRD score is a combination of precision and recall and is calculated as the Euclidean distance between the points (precision, recall) and (1, 1) in the 2-dimensional plane. A PRD score of zero indicates perfect accuracy, while a score of one or more indicates poor accuracy. The PRD score is calculated as follows:

$$\text{PRD} = \sqrt{(1 - \text{Precision})^2 + (1 - \text{Recall})^2} \tag{3.9}$$

The PRD score is particularly useful when the precision and recall metrics cannot be used to fully evaluate the accuracy of the model. For example, when the data set is unbalanced, with a high number of positive instances and a low number of negative instances, precision and recall may give conflicting results. In such cases, the PRD score can provide a more comprehensive evaluation of the model's accuracy. In conclusion, the Precision and Recall Distance (PRD) score is a metric used to evaluate the accuracy of classification models. The PRD score combines the precision and recall metrics to give a single measure of accuracy. A PRD score of zero indicates perfect accuracy, while a score of one or more indicates poor accuracy. An approximation to accuracy, recall, and F1 was introduced by Lucic et al. as a supplemental metric [55]. This was done in an effort to overcome the disadvantages that are associated with the IS and FID. The degree of accuracy of the generated samples is measured by the precision, which is the fraction of the created distribution that overlaps with the actual distribution. The recall is the proportion of the actual distribution that is covered by the distribution that was created, and it is a measure of variety.

F1 score is a measurement of how well accuracy and recall are balanced against one another. They point out that the information retrieval system (IS) is deficient in recall since it permits partial modes of the data distribution but penalises the model for partial classes, whereas the fuzzy inference system (FID) is capable of capturing both precision and recall.

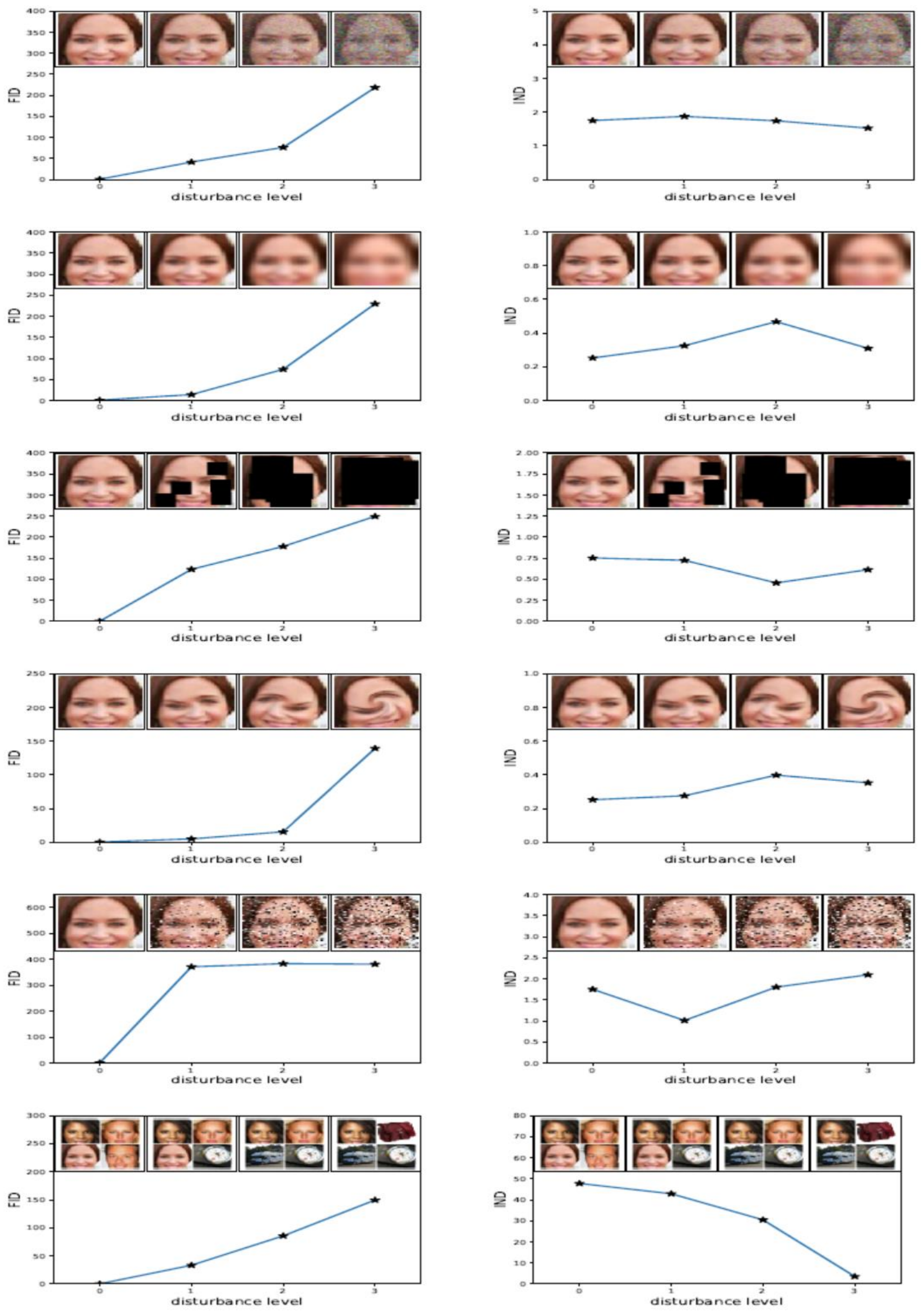
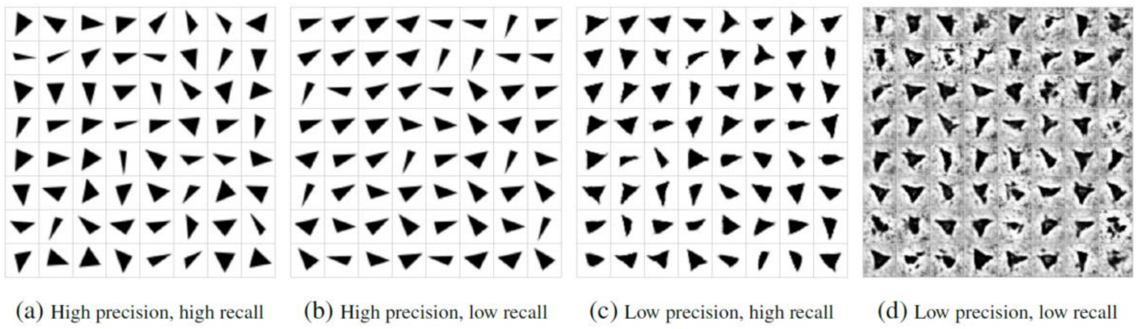


Figure 3.1: Left: FID and right: IND (Max{IS}:m - IS) under different disturbances: first row: Gaussian noise, second row: Gaussian blur, third row: implanted black rectangles, fourth row: swirled images, fifth row. salt and pepper noise, and sixth row: the CelebA dataset contaminated by ImageNet images. FID shows consistency with the disturbance levels whereas IND fluctuates and even decreases, which doesn't match the increase in disturbance level [38].

a metric of the balance between precision and recall. They point out that IS fails to reflect recall since it allows the partial modes of the data distribution but penalizes the model for partial classes, whereas FID can capture precision and recall.



(a) High precision, high recall
(b) High precision, low recall
(c) Low precision, high recall
(d) Low precision, low recall

Figure 3.2: Samples are generated Gray-scaled triangles which are from a low-dimensional manifold 3 embedded in Rd×d. (a) high recall and precision, (b) high precision and low recall(is not as variable as a), (c) low precision, but high recall(can not maintain the convexity), (d) low precision and recall [55].

They recommend building a data manifold to improve the calculation efficiency of the distances from samples to the manifold. This may be done by developing a data manifold. The effectiveness of the generative model may be evaluated based on the distance that separates it from the manifold. When establishing what constitutes a good generative model, one of the examples that demonstrates precision and recall is shown in figure 3.2. In this particular scenario, a well-executed generative model is able to deal with rotation, translation, changes in angle size, and so on. Precision and recall are both dependent on the distance from the samples to the manifold. This distance is defined as the squared Euclidean distance between a sample and G(z), where z is a latent representation of sample x determined using gradient descent [55, 18]. Precision is the fraction of synthetic instances that have a distance to the manifold that is below a certain threshold, and recall is the percentage of test samples that have an L2 distance to G(z) that is below a given threshold [18].

However, the strategy cannot be implemented on data sets of a higher complexity than that. When comparing the distributions of P and Q, a better measure called Accuracy and Recall Distance includes precision and recall making the comparison [66]. The distributions are utilised by Sajjadi et al. in order to define precision and recall. The objective of the strategy that has been suggested is to generalise the total variation distance between the reference distribution (P) and the learnt distribution (Q) while maintaining accuracy and recall.

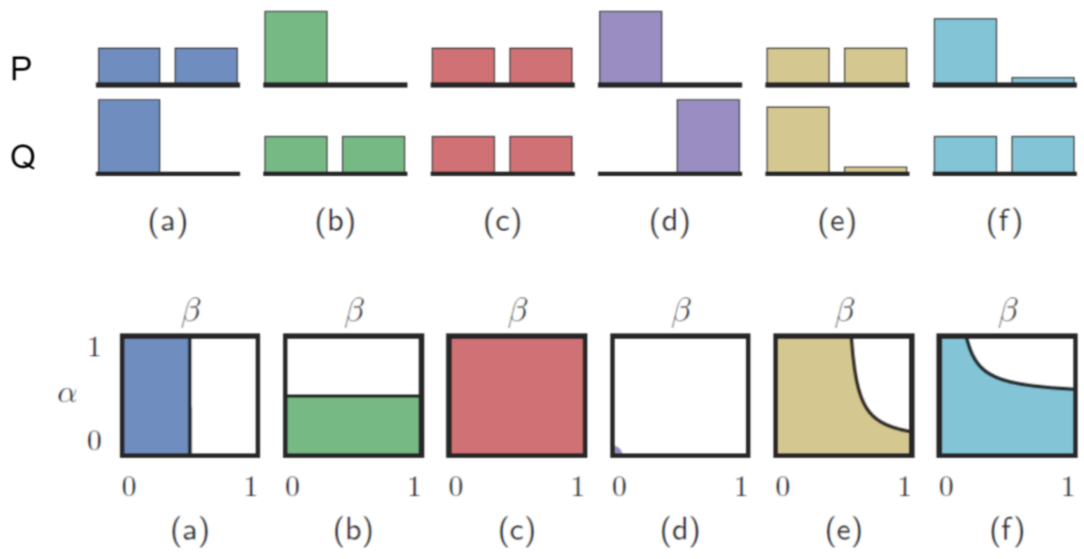


Figure 3.3: Left: IS and FID varies as adding and dropping classes Middle: PRD curves are generated under different amounts of classes inˆQ. Notice that the increase in the amount of classes after five will cause a loss in the precision.Right: it shows the recall increases as a class added to the Q [66].

Definition 1. For α,β ∈ (0, 1], the probability distribution Q has precision α at recall β w.r.t P if there exist distribution μ, νP and νQ such that [65]

$$P = \beta\mu + (1 - \beta)\nu_P \text{ and } Q = \alpha\mu + (1 - \alpha)\nu_Q \qquad (3.10)$$

where the νP means the non-overlapping part of P with Q and νQ is the irrelevant part of Q. Figure 3.3 displays the pairs of precision and recall that PRD(Q,P) contains under several different situations. The predicted values for P and Q are arrived at by employing a pre-trained Inception network. In order to make the problem more manageable, they combine P and Q using a mini-batch k-means clustering technique. If it is unable to create samples from a cluster that contains many samples from the real-world distribution, then the recall will be affected.

On the other hand, the precision will be lowered if it produces samples from a cluster that contains only a few genuine samples.

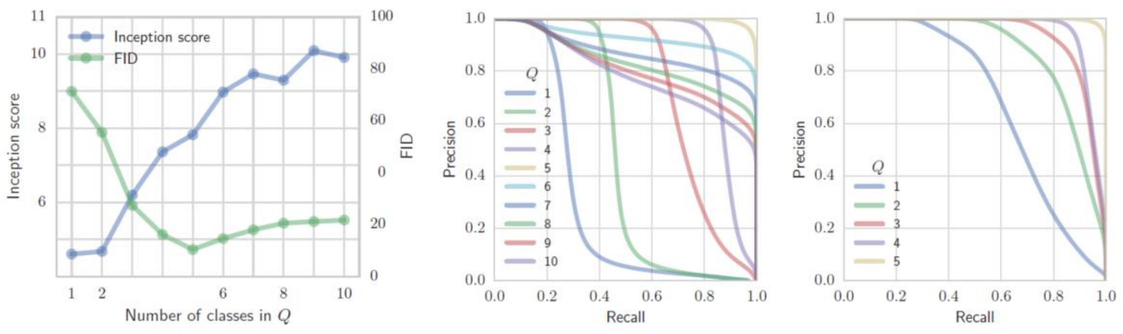


Figure 3.4: Left: IS and FID varies as adding and dropping classes Middle: PRD curves are generated under different amounts of classes inˆQ. Notice that the increase in the amount of classes after five will cause a loss in the precision Right: it shows the recall increases as a class added to the Q [66].

The effectiveness of IS and FID, in addition to the suggested method, is compared in figure xx. In the first trial (Figure3.4 Left and Figure3.4 Middle), the only CIFAR-10 classes that are included in P are the top five. In the second experiment, P is adjusted such that it contains samples from all classes (Figure3.4 Right). IS shows a rising tendency when more classes are added to Q, whereas FID initially shows a decreasing trend and then an increasing trend after the number of courses exceeds five. FID is able to manage the addition and removal of classes, but it cannot attribute to the adding or deleting of modes simply based on the FID score. In contrast, PRD curves are able to detect mode collapse (low recall) and poor quality (low accuracy) of produced samples. Both of these issues might affect the precision of the data.

In the year 2019, Simon et al. provide a new technique that is based on Sajjad's approach. This algorithm extends precision and recall from finite support to arbitrary distributions and correlates precision and recall with type I and type II errors of likelihood ratio classifiers [61]. [53] Kynka-Anniemi et al. present an improved precision and recall metric, which determines precision and recall by using the kth closest neighbour in the feature space.

3.5 Maximum Mean Discrepancy (MMD)

Maximum Mean Discrepancy (MMD) is a distance metric that measures the difference between two probability distributions. It is used to compare two sets of data to determine whether they are from the same distribution. MMD is a popular metric in the field of machine learning and computer vision, especially for applications involving generative models, such as generative adversarial networks (GANs).The idea behind MMD is to find the maximum difference between the mean embeddings of the two data sets in a reproducing kernel Hilbert space (RKHS). An RKHS is a high-dimensional feature space where data points can be transformed into vectors, and similarity between data points can be measured by computing inner products between their feature vectors. The MMD distance between two data sets is defined as the maximum difference between the mean feature vectors of the two data sets.

Formally, let X and Y be two sets of data points, each with n samples. The MMD between X and Y is defined as:

$$\text{MMD}(X, Y) = || E[f(X)] - E[f(Y)] || \tag{3.11}$$

where f is a feature mapping function that transforms the data points into vectors in the RKHS, and E[f(X)] denotes the expectation operator. The feature mapping function f is typically chosen as a kernel function, such as the radial basis function (RBF) kernel or the polynomial kernel.

MMD has several attractive properties, including being a well-defined and flexible metric that can handle non-linear relationships between data points. It also has a convenient and efficient computational procedure, which makes it suitable for large-scale applications. Furthermore, MMD can be used to determine the goodness-of-fit of a generative model, by comparing the distribution of its generated samples with the distribution of the target data. In conclusion, MMD is a powerful metric for comparing two data distributions, and has a wide range of applications in machine learning and computer vision. It provides a way to measure the difference between two sets of data points in a high-dimensional feature space, and has attractive properties such as flexibility, computational efficiency, and suitability for large-scale applications.

Maximum Mean Discrepancy is designed to compare samples from two prob- ability distributions with a distance between the expectations of a function F in a reproducing kernel Hilbert space (RKHS) H [33].

$$\text{MMD}(F, P, Q) = EX{\sim}P\,[f(X)] - EY{\sim}Q[f(Y)]\,H \tag{3.12}$$

where F is a group of functions denoted by f. In addition to being used for GAN training, MMD may also be utilised to evaluate the performance of GAN models. When the MMD value is low, it implies that the distributions of synthetic and actual data are more similar to one another.

If X and X independently follow distribution P while Y and Y independently follow distribution Q, then MMD will incorporate characteristic kernels in such a way that it will be possible to calculate the squared population MMD and its unbiased estimator. This is because the kernel functions k: XX R will be continuous in this case [33].

RKHS allows for the application of a wide variety of kernel functions, including Gaussian and Laplace kernels, amongst others.

$$\mathrm{MMD}^2[\mathcal{F}, P, Q] = \mathbb{E}_{X,X'}[k(X,X')] - 2\mathbb{E}_{X,Y}[k(X,Y)] + \mathbb{E}_{Y,Y'}[k(Y,Y')] \quad (3.13)$$

The empirical unbiased estimator of MMD2[F; P; Q] is:

$$\mathrm{MMD}^2[\mathcal{F}, P, Q] = \frac{1}{m(m-1)}\sum_{i=1}^{m}\sum_{j\neq i}^{m} k(X,X') - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=i}^{n} k(X,Y) + \frac{1}{n(n-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n} k(Y,Y') \quad (3.14)$$

To resolve manual annotation in evaluation, Bounliphone et al. exploit the joint asymptotic distribution of the MMDs and a hypothesis testing i.e. relative similarity test for model selections [19]. Under the assumption that Px 6= Py, Px 6= Pz,E[k(xi; xj)] < 1, E[k(xi; xj)] < 1 and E[k(xi; yj)] < 1, the joint asymptotic distribution converges to a Gaussian:

$$\sqrt{m}\left(\begin{pmatrix}\mathrm{MMD}^2(\mathcal{F},X,Y)\\ \mathrm{MMD}^2(\mathcal{F},X,Z)\end{pmatrix} - \begin{pmatrix}\mathrm{MMD}^2(\mathcal{F},P_x,P_y)\\ \mathrm{MMD}^2(\mathcal{F},P_x,P_z)\end{pmatrix}\right) \xrightarrow{d} \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}\sigma_{XY}^2 & \sigma_{XYXZ}\\ \sigma_{XYXZ} & \sigma_{XZ}^2\end{pmatrix}\right) \quad (3.15)$$

Where X, Y, and Z are independent and identically distributed from the reference distribution Px, the candidate reference Py, and the candidate reference Pz, respectively. Random variables z follow Pz as random variables. The combined asymptotic distribution is determined by the inequalities between Px,Py and Px,Pz. The null hypothesis states that Px is more similar to Py than to Pz, namely MMD2(F,X,Y) MMD2 (F ,X,Z). The alternative hypothesis is that MMD2(F,X,Y) is greater than MMD2 (F ,X,Z). MMD2(F,X,Y) MMD2(F,Z) is a sequence of rotation and integration of the joint asymptotic distribution [18]. Kernel Inception Distance (KID) was developed by Bi'nkowski et al. to quantify the dissimilarity between two distributions using the squared MMD with the polynomial kernel, k(x,y) = (d1 xT y +1)3, where d is the dimension. In contrast to FID, KID is a nonparametric form with an unbiased estimator, meaning that fewer samples are required for KID to converge to the theoretical value [16] than for FID.

3.6 Wasserstein Critic

Wasserstein critic, also known as Wasserstein distance or Earth Mover's Distance, is a measure of distance between two probability distributions. It is used in the field of generative adversarial networks (GANs) to evaluate the quality of the generated samples. The Wasserstein critic compares the distribution of the generated samples to the target distribution. The Wasserstein distance is calculated as the amount of "work" required to move the generated samples to match the target distribution. This "work" is represented as the minimum amount of probability mass that needs to be moved from one distribution to the other. The Wasserstein critic uses the concept of a "ground metric," such as the Euclidean distance, to calculate the amount of work required to move the probability mass from one distribution to the other. The ground metric is used to determine the cost of moving a unit of probability mass from one place to another. The Wasserstein distance is then calculated as the minimum cost of transforming one distribution into the other.

The Wasserstein critic has several advantages over other evaluation methods. For example, it is able to handle high-dimensional data and multi-modal distributions, which makes it a good fit for evaluating the quality of generated samples in GANs. It is also less sensitive to the effects of mode collapse, which is a common issue in GANs where the generator only produces a small number of different samples.

In conclusion, the Wasserstein critic is a useful evaluation method for GANs, as it provides a measure of the quality of the generated samples based on the amount of work required to match the target distribution. This allows for a more accurate evaluation of the generator's performance and can be used to guide improvements in the training process. The Wasserstein metric Earth-Mover (EM) distance is used to compare two distributions Pr, Pg : Prob(X). EM distance is the cheapest way to convert Pr to Pg. Equation 2.10 infimum is unsolvable. Kantorovich-Rubinstein duality (Equation 2.11) approximates Wasserstein distance. K-Lipschitz can replace the 1-Lipschitz function, f: X → R. The supremum is intractable. Lipschitz continuous function is used to approximate [21].

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2) \tag{3.16}$$

where dY and dX are given X and Y metrics. If K exists for any x1,x2, f : X → Y is Lipschitz continuous. A parameterized function family fww, where w represents weights in a set of all possible weights W, is assumed to be K-Lipschitz. Solve Equation 3.12. Dimitrakopoulos et al. suggest Wasserstein Inception Distance (WInD), an extension of FID that integrates Wasserstein distance, to evaluate GAN models. They drop the Gaussian distribution assumption for Inception embeddings.

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_\theta}[f_w(x)] \tag{3.17}$$

Instead, a multimodal Gaussian Mixture Model (GMM) models embedding distribution. GMM Equation 3.13 [28].

$$P_{GMM}(x) = \sum_{j=1}^{K} \pi^j \mathcal{N}(\mu^j, \Sigma^j) \tag{3.18}$$

where the π1,π2,...,πK are weights which sum up to one, and μ1,μ2,...,μK , and Σ1, Σ2,..., ΣK are the means and covariances of Gaussian kernels. The Expectation- Maximization algorithm (EM) is used to estimate the parameters of a GMM. The Wasserstein distance which measures distances between Finite Mixture Models, is de- fined as Equation 3.14.

$$d_{EMD}(x, g) = \frac{\sum_{j=1}^{J} \sum_{k=1}^{K} f^{jk} d^{jk}}{\sum_{j=1}^{J} \sum_{k=1}^{K} f^{jk}} \tag{3.19}$$

where f jk is non-negative number that means the flow from kernel j to kernel k. djk denotes ground distances between kernel j and kernel k [28].

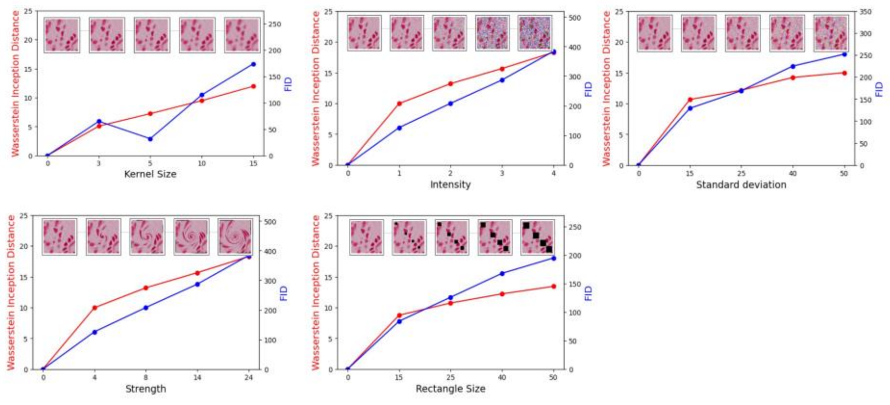


Figure 3.5: 1st Row Left: Blur. 1st Row Middle: Salt & Pepper. 1st Row Right: White Gaussian Noise. 2nd Row Left: Swirl. 2nd Row Middle: Occlusion [28].

In the study, the BBBC038v1 cell microscope imaging dataset's WInD and FID are compared with various disturbances (Figure 3.5). As the disturbance level rises, WInD rises as well. According to Dimitrakopoulos et al., the distributional flexibility of WInD makes it perform better than FID. If the dataset has a different distribution than expected, FID may have a tendency to consider the incorrect or poorly created samples to be realistic under the Gaussianity assumption. However, WInD can address the same distribution problem [28]. The high sample and time complexity, however, is the Wasserstein distance's drawback [18].

3.7 Geometry Score (GS)

Geometry Score (GS) is a mathematical measure used to evaluate the similarity between two shapes. It is widely used in computer vision, image processing, and pattern recognition fields to compare and match shapes. The score is expressed as a numerical value between 0 and 1, where 0 represents no similarity, and 1 represents complete similarity.

The Geometry Score is based on the concept of shape invariance, meaning that the score should remain unchanged even if the two shapes are transformed in different ways. The score takes into account several aspects of the shape, such as size, orientation, and shape features.

The calculation of the Geometry Score involves finding the overlap between the two shapes and then normalizing the overlap based on the size of each shape. The overlap is calculated using mathematical algorithms that compare the pixel values in the two shapes and determine which pixels are common between them. This overlap value is then divided by the total number of pixels in the two shapes to determine the Geometry Score.

The Geometry Score is an important tool for many computer vision applications, including object recognition, tracking, and classification. For example, in object recognition, the Geometry Score can be used to compare the shape of an object in an image to a database of known objects and determine the closest match. In tracking, the score can be used to evaluate the similarity between the current shape of an object and its previous shape, helping to track its movement over time.

In conclusion, the Geometry Score is a mathematical measure that provides a numerical representation of the similarity between two shapes. It is widely used in computer vision and image processing applications, where it provides a reliable and efficient method for comparing and matching shapes. Geometry Score (GS) is a tool developed by Khrulkov and Oseledets [48] to statistically assess the quality and variety of topology-based produced samples. GS examines how comparable the underlying manifolds of real samples and synthetic samples are (Figure 3.6).Simplicial complexes, which have a vertex, an edge joined by two vertices, a triangle, and a tetrahedron, are simpler approximations of manifolds. A convex hull with k+1 vertices is a k-dimensional simplex. How to build a simplicial complex with a fix is shown in Figure 3.7 .

The homology, which is defined as the number of holes of different diameters, is summarised by the persistence barcode (Figure3.8). How long the persistence interval may last is a key component of the persistence barcode. For instance, the status of one hole is dominant in the set, H1, in Figure3.8 because it lasts longer than the status of two holes.

They introduce the homology's Relative Living Time (RLT). A persistence codebar with a persistence parameter encompassing the range [0, max] may be constructed given a dataset X and a few options for landmarks L. The picked landmark at random because to their reduced number, points facilitate computation. The Betti Kth number of persistence intervals having beneath the number given a fixed the size of k.

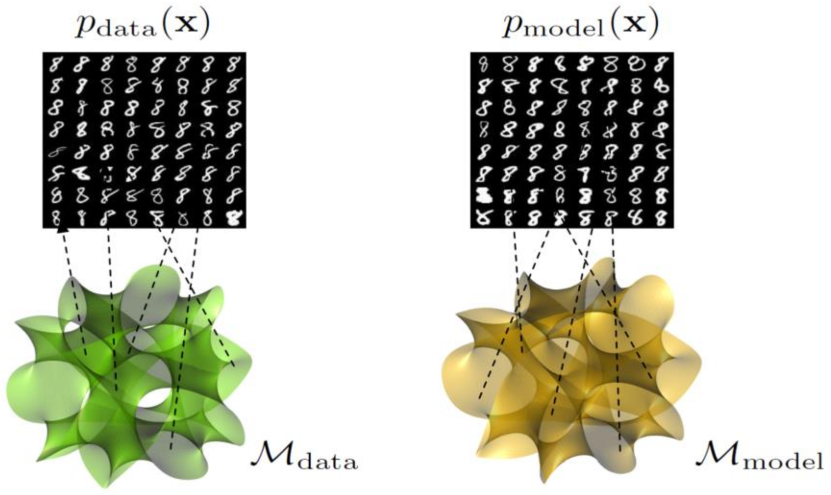

$p_{\text{data}}(\mathbf{x})$
$p_{\text{model}}(\mathbf{x})$
$\mathcal{M}_{\text{data}}$
$\mathcal{M}_{\text{model}}$

Figure 3.6: The Manifold Hypothesis states that high dimensional data like natural images can be Fitted into low dimensional data manifold Mdata. The idea can be employed to assess the generated sample by comparing the topological properties of the manifold Mdata and Mmodel [48].

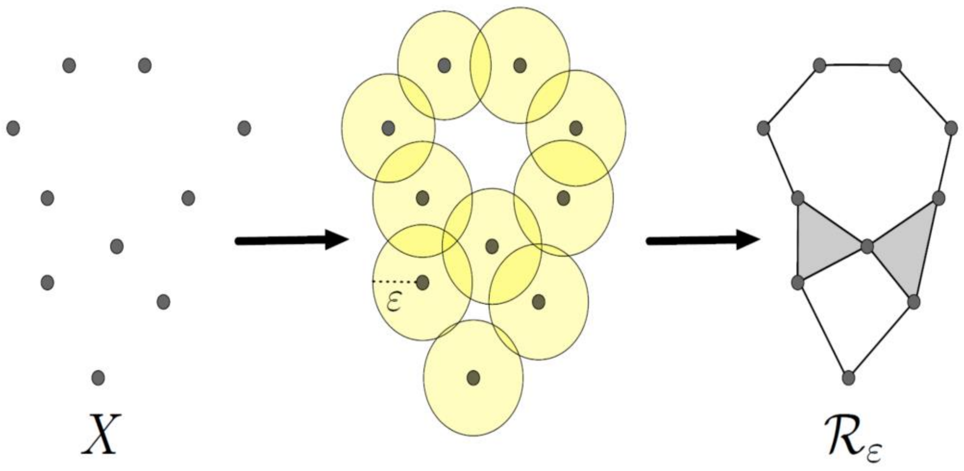


Figure 3.7: In order to construct a simplicial complex on X, with a fixed as the radius centered at each vertex, a k-dimensional simplex is added to the simplicial complex R if k + 1 pairwise intersections occur [48].

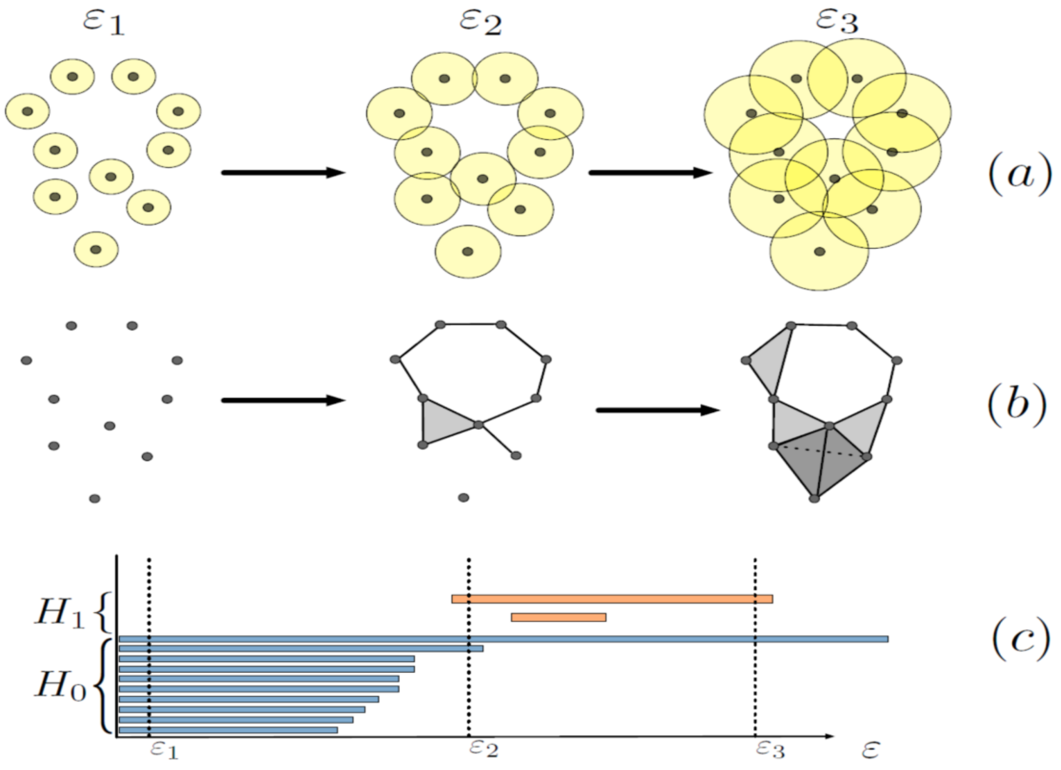


Figure 3.8: The simplicial complexes are generated under different values of proximity parameters. (a) Left: no intersections among the balls when $\epsilon = \epsilon 1$ Middle: one loop (hole) and a filled triangle pop up as there are pairwise intersections $\epsilon = \epsilon 1$. When increases, there is a small loop at the bottom of the data points which will disappear soon as keep increasing to

the point where the 4 balls below intersect with each other. Right: as reaches 3, the loop is smaller. (c) is the persistence barcode which displays the change in H0 and H1 as the increases. [48].

The outcome demonstrates that the predominant numbers of holes match the actual samples (Figure 3.8). The comparison of WGAN and WGAN-GP on producing digits further demonstrates the applicability of the suggested GS metric for assessing performance (Figure 3.8). In comparison to WGAN, WGAN-distribution GP's of the 1-dimension holes is closer to the genuine distribution. The topological character of GS, however, makes it insensitive to directions or embedding distances inside the manifold [14]. Instead of just utilising GS, Khrulkov and Oseledets advise using GS together with FID. Topology Distance (TD), which Horak et al. propose in 2020 and claims is a potent substitute for present metrics like FID, KID, and GS, is comparable to the concept of GS. In situations involving pixel noise capture, patch mask, and patch exchange, TD performs better than FID and KID. Additionally, compared to GS, TD is more consistent with the various types of perturbations [41].

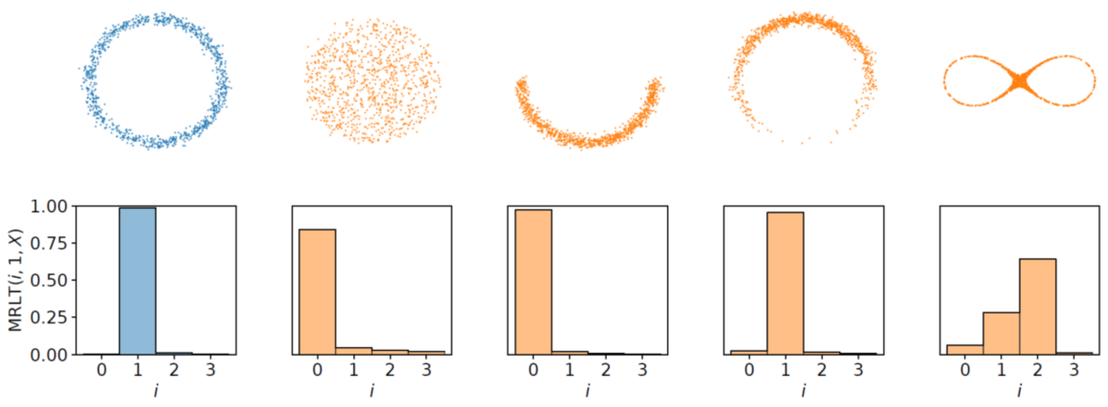


Figure 3.9: MRLT calculated for different 2D dataset. MRLT manages to compute

the number of 1-dimensional holes existing in the dataset. [48].

CHAPTER 4

TRAINING CHALLENGES

Generative Adversarial Networks (GANs) are a type of neural network that consists of two sub-networks: a generator and a discriminator. The generator is tasked with generating synthetic samples that resemble real data, while the discriminator tries to distinguish between real and synthetic samples. Both sub-networks are trained simultaneously, with the generator's objective being to fool the discriminator and the discriminator's objective being to correctly classify real and synthetic samples. Training GANs can be challenging due to a number of problems that can arise, including:

Mode Collapse: In this scenario, the generator collapses and generates only a limited number of samples that represent the same mode, rather than producing diverse samples. This occurs when the generator finds an optimal solution that generates synthetic samples that are all similar and fool the discriminator. To avoid this problem, it is recommended to use different regularization techniques and update the generator with more frequent feedback. Instability: GAN training can be highly unstable due to the nature of the loss function, which is non-convex and requires both sub-networks to be trained simultaneously. This instability can lead to oscillations, vanishing gradients, and other optimization problems. To mitigate this issue, it is recommended to use a balanced generator/discriminator ratio and monitor the convergence criteria for both sub-networks.

4.1 Gradient Vanishing

Gradient vanishing is a problem that occurs in Generative Adversarial Networks (GANs) when the gradients of the generator's loss function become too small to update the parameters effectively during training. This problem can lead to slow training and poor convergence of the GAN model. To understand gradient vanishing in GANs, let's first define the basic components of a GAN: Generator (G): The generator takes in random noise and generates fake data samples. Discriminator (D): The discriminator takes in both real and fake data samples and tries to distinguish between them.

Adversarial loss: The generator is trained to generate data samples that can fool the discriminator, and the discriminator is trained to correctly distinguish between real and fake data samples. The adversarial loss is the sum of the discriminator's loss and the generator's loss.

The generator's loss function can be defined as:

$L_G = -\frac{1}{m} \sum_{i=1}^{m} log(D(G(z^{(i)})))$ (4.1)

where m is the batch size, $z^{(i)}$ is a random noise vector, and $D(G(z^{(i)}))$ is the output of the discriminator on the fake data generated by the generator.During backpropagation, the gradient of the generator's loss function with respect to the generator's parameters is calculated. This gradient is used to update the generator's parameters using an optimization algorithm such as stochastic gradient descent (SGD).

However, if the discriminator is too good at distinguishing between real and fake data samples, the gradient of the generator's loss function can become very small. This is because the discriminator's output becomes very close to either 0 or 1, causing the log function to approach negative infinity. This leads to the vanishing gradient problem, where the gradient becomes too small for the generator to learn effectively.One way to mitigate the gradient vanishing problem is to use a different loss function for the generator. One such loss function is the Wasserstein distance, which measures the distance between the distribution of real data and the distribution of fake data generated by the generator. The Wasserstein distance can be defined as:

$W(\mathbb{P}r, \mathbb{P}g) = \sup{\Vert f \Vert_L \leq 1} \mathbb{E}{x \sim \mathbb{P}r}[f(x)] - \mathbb{E}{x \sim \mathbb{P}_g}[f(x)]$

where $\mathbb{P}_r$ is the distribution of real data, $\mathbb{P}_g$ is the distribution of fake data generated by the generator, and f is a 1-Lipschitz function. The generator's loss function using the Wasserstein distance can be defined as:

$L_G = -W(\mathbb{P}_r, \mathbb{P}_g)$ (4.2)

During training, the discriminator is also updated to maximize the Wasserstein distance. This is done by minimizing the negative Wasserstein distance:

$L_D = -W(\mathbb{P}_r, \mathbb{P}_g)$ (4.3)

The gradients of the Wasserstein distance are generally more stable and avoid the vanishing gradient problem. Before the publication of the vanilla GAN by Goodfellow et al., the vast majority of generative models estimated the distribution of datasets through the use of Maximum Likelihood Estimation (MLE). In order to maximise the efficacy of the MLE's goal function

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{N} p(x_i|p)$$ (4.4)

is the same as minimizing the KL-divergence

$$D_{KL}(\mathbb{P}_r \| \mathbb{P}_g) = \int_x p_r(x) \log \frac{p_r(x)}{p_g(x)} \, dx$$ (4.5)

The KL-divergence is famously asymmetric: DKL(Pr|Pg) 6= DKL(Pg|Pr) which means when DKL(Pr|Pg) —0, DKL(Pg|Pr) could still be very large, vice versa. The Jensen-Shannon(JS) divergence is symmetrical and based on KL-divergence, defined as follows

$$D_{JS}(\mathbb{P}_r \| \mathbb{P}_g) = D_{KL}(\mathbb{P}_r \| \frac{\mathbb{P}_r + \mathbb{P}_g}{2}) + D_{KL}(\mathbb{P}_g \| \frac{\mathbb{P}_r + \mathbb{P}_g}{2})$$ (4.6)

When the discriminator is at optimal, the objective function of the generator can be transformed as follow

$$\min_G V(D^*, G) = 2D_{JS}(\mathbb{P}_r \| \mathbb{P}_g) - 2\log 2$$ (4.7)

Therefore, limiting JS-divergence is the same as optimising GAN for maximum performance. And in extreme conditions (or early stages of training), when Pr and Pg do not overlap or the overlap part is negligible, the JS-divergence will equal log2, which means the gradient will be 0, and the generator will learn very slowly or even not at all. This is because the JS-divergence will be equal to the log2 of the difference between Pr and Pg. In the first publication of the GAN algorithm, Goodfellow and colleagues advocated use a different gradient step for G in order to circumvent the issue of vanishing gradients.

$$\nabla_\theta[log(1 - D(G(z)))] \rightarrow \nabla_\theta[-\log D(G(z))]$$ (4,8)

Remarked by Arjovsky et al. [11] For D* = $\frac{\text{Pr}}{Pg\theta 0+Pr}$ be the optimal discriminator, with axed value θ_0, equation 4.5 can be transformed as follow.

$$\mathbb{E}_{z\sim p(z)}[-\nabla_\theta \log D^*(g_\theta(z))|_{\theta=\theta_0}] = \nabla_\theta[D_{KL}(\mathbb{P}_{g\theta}\|\mathbb{P}_r) - 2D_{JS}(\mathbb{P}_{g\theta}\|\mathbb{P}_r)]|_{\theta=\theta_0} \tag{4.9}$$

The first problem of the above equation is that JS-divergence is in the wrong direction, which means they push the distributions to be different, and the gradient might be unstable. Second problem is, KL-divergence is reversed which means it is not equiva- lent to MLE. In the case of DKL(Pg   Pr), when Pg (x) → 0, Pr(x) → 1, the penalty of generator could not generate real enough samples Pg (x)log PPgr ((xx)) → 0; when Pg (x) →1, Pr(x) → 0, the penalty of generator generate unreal samples Pg (x)log Pg (x) → +∞.The generator will suffer huge from generator unreal sample thus it will tend to gen- erate samples with higher quality but less diversity, which could cause another issue called mode collapse. Some works that focused on replace cost function, like WGAN, WGAN-GP, LSGAN [57], BEGAN [15] are quite effective on eliminate gradient vanishing problem and performance improvement.

Overall gradient vanishing is a common problem in GANs that can lead to slow training and poor convergence of the model. Using alternative loss functions such as the Wasserstein distance can mitigate this problem and improve the stability of the training process.

4.2 Mode Collapse

Mode collapse is a phenomenon that can occur in Generative Adversarial Networks (GANs) during the training process. It happens when the generator network produces limited and repetitive samples, resulting in a loss of diversity and quality in the generated images. In a GAN, the generator and discriminator networks are trained simultaneously through a min-max game. The generator network tries to create realistic images that can fool the discriminator network, while the discriminator network tries to distinguish real images from the generated ones. If the discriminator network becomes too good, it can sometimes overpower the generator and force it to produce a limited set of outputs. This is because the generator learns to produce images that are similar enough to fool the discriminator but lack the necessary diversity to cover the entire image space.

Mode collapse can also happen if the generator network is not powerful enough to capture the full complexity of the data distribution. In this case, the generator produces a limited set of images that are representative of only a few modes of the underlying distribution.

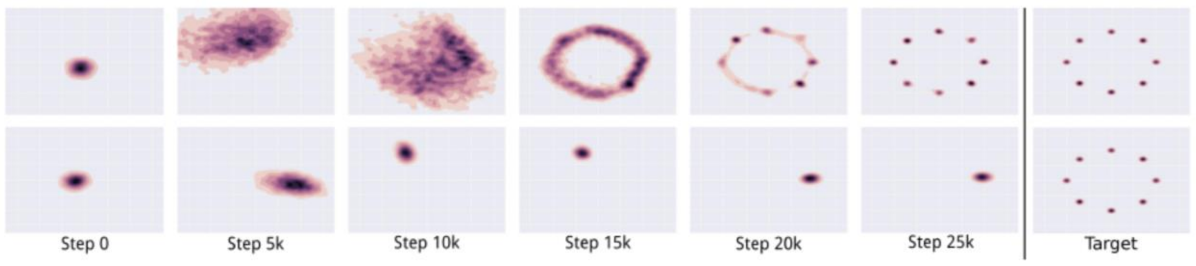


Figure 4.1: A mode collapse example on a toy 2D Gaussian dataset. Image from [59].

A generator that has collapsed to a certain parameter value and can only create a single mode is said to have collapsed in mode. This is shown in the following passage. There is no such mechanism to modulate the generator for more diverse output when the collapse is imminent because the discriminator processes all inputs independently, the gradients of the discriminator are unable to cooperate, and the gradient associated with z approaches $\frac{\partial J}{\partial z} \approx 0$, which encourages the generator to approach this single point. This is because the discriminator processes all inputs independently (collapse). In actual actuality, total mode collapse occurred only very infrequently, but partial mode collapse (when G only learnt a tiny subset of all modes) occurred far more frequently. There are a few strategies and approaches that may be utilised to prevent mode collapse, the most obvious of which is to enable the discriminator to accept numerous inputs. This method is referred to as minibatch discrimination and can be found in [67]. In addition, the purpose of the discriminator [12, 63] and the generator [76, 59] might potentially be altered to assist address the collapse problem, as shown by various published research. To mitigate mode collapse, several techniques have been proposed. One popular method is to use a variation of the GAN architecture, such as a Wasserstein GAN (WGAN), which provides a more stable training process and better gradient flow. Another approach is to introduce diversity-promoting objectives, such as the maximum mean discrepancy (MMD), to encourage the generator to produce a wider range of outputs.Overall, mode collapse is a significant challenge in GAN training, and researchers continue to explore new techniques to address this issue and improve the quality and diversity of generated images.

4.3 Instability

Generative Adversarial Networks (GANs) are a popular class of generative models used in machine learning for synthesizing realistic data. They consist of two networks: a generator and a discriminator. The generator tries to produce fake data that can fool the discriminator, while the discriminator tries to distinguish between the real and fake data. The training process involves updating the generator and discriminator alternately until the generator produces data that is similar to the real data. However, GANs can suffer from instability issues during training, resulting in poor performance or failure to converge. There are several reasons for this instability, such as vanishing gradients, mode collapse, and oscillations. Let's discuss these issues in more detail.

Vanishing Gradients:

The generator and discriminator are trained using a loss function that measures the difference between the real and fake data. The gradients of this loss function are used to update the parameters of the networks. However, in some cases, the gradients can become very small or zero, making it difficult to update the network weights. This is known as the vanishing gradients problem.

Vanishing generator gradients in original GAN

The original value function suggested in the GAN paper is

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{data}} [\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right) \tag{4.10}$$

where the dependence on discriminator, generator parameters θD, θG have been explicitly included. Since the discriminator wants to maximize this as a function of θD, a good discriminator will output 1 on real samples and 0 on fake samples. Remember that D is a probability and the output of a sigmoid, so DθD (s) = σθD (s), where s is the discriminator input (s = GθG (z)); the vanishing gradient/saturation referred to when the discriminator is good originate from the sigmoid. From the chain rule,

$$\nabla_{\theta_G} \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] = \mathbb{E}_{z \sim p(z)} \left[\frac{1}{1 - \sigma_{\theta_D}(s)} (-\sigma'_{\theta_D}(s)) \Big|_{s = G_{\theta_G}(z)} \nabla_{\theta_G} G_{\theta_G}(z) \right] \tag{4.11}$$

The difference in behavior comes from the product of the first two terms in the two cases. In the case above, σ/(s) → 0 as σ → 0, while the fraction remains finite. For a generator function log(Dθ (Gθ (z))), the fraction diverges and the two balance to give nonvanishing gradient. Figure 4.2 plots log(1 − σ(s)) vs. log(σ(s)), as a function of s, and we see the vanishing gradient/saturation in the region s → −∞ (σ → 0).

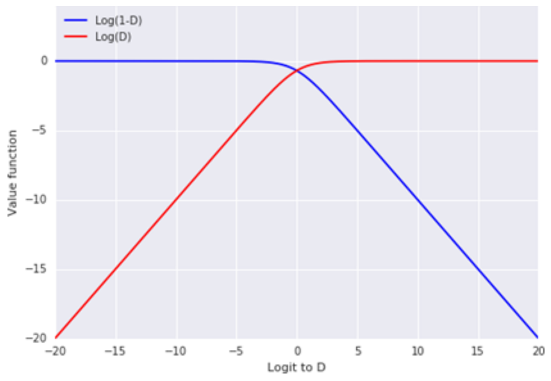


Figure 4.2: Comparing two generator value functions, log(1 − σ(s)) and log(σ(s)), as a function of the logit value s to discriminator.

The vanishing gradients problem can occur when the discriminator becomes too good at distinguishing between the real and fake data. In this case, the gradients from the discriminator to the generator become very small, and the generator fails to produce diverse samples. This results in a collapse of the generator, where it produces only a few samples that are similar to each other. To address these challenges further, a new class of GANs called Transformer-based GANs (T-GANs) have emerged. T-GANs replace the traditional convolutional layers with self-attention layers, which allow the generator to model long-range dependencies in the input data [1].

		NC subjects			MCI subjects			Params	GFLOPs
Method	Venue	PSNR	SSIM	NMSE	PSNR	SSIM	NMSE		
3D Transformer-GAN [84]	MICCAI'21	*24.818*	*0.986*	*0.0212*	*25.249*	**0.987**	*0.0231*	*76M*	**20.78**
3D CVT-GAN [85]	MICCAI'22	**24.893**	**0.987**	**0.0182**	**25.275**	**0.987**	**0.0208**	**16M**	*23.80*

Table 4.1: The results comparison of Transformer-based GANs for PET reconstruction on NC subjects and MCI subjects' datasets [1].

"The self-attention mechanism in transformers can be formulated as follows: given a sequence of input vectors X = {x1, x2, ..., xn}, the self-attention module computes a set of attention weights α = {α1, α2, ..., αn}, where αi represents the importance of xi with respect to the other input vectors [1]. The attention weights are computed as follows:

$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{n} \exp(e_j)}$$

where α_i is the energy associated with the ith element of the input sequence, defined as a dot product between a query vector q and a key vector k:

$$e_i = q \cdot k_i$$

The query, key, and value vectors are learned parameters of the self-attention module and are computed from the input vectors using linear transformations. The output of the self-attention module is a weighted sum of the value vectors, where the weights are given by the attention weights [1]:

$$\text{SelfAttention}(X) = \sum_{i=1}^{n} \alpha_i v_i$$

Self-attention mechanism is a technique that helps to improve the quality of generated images by focusing on important regions of the image during the generation process. It allows the model to selectively attend to different parts of the input image and weigh the importance of each part based on its relevance to the generation task [6].

This mechanism has been applied in various GAN architectures such as "Self-Attention GAN" (SAGAN), "BigGAN", and "StyleGAN". These models have shown significant improvements in image generation quality and diversity [6].

4.4 Evaluation Metrics

Evaluation of GANs is a challenging task since there is no objective metric to evaluate the quality of synthetic samples. Common evaluation methods include visual inspection and perceptual similarity measures, but these can be subjective and difficult to quantify. To address this issue, recent studies have proposed alternative metrics such as the Fréchet Inception Distance (FID) and Inception Score (IS). Further this will be discussed in the next chapter.

4.5 Data Distribution

The quality of synthetic samples generated by GANs heavily depends on the quality of the training data. If the training data is biased or limited, the generator may produce samples that do not represent the full range of the data distribution. This can be addressed by using more diverse training data or data augmentation techniques.

4.6 Computational Cost

GAN training can be computationally expensive, requiring significant resources such as GPU clusters and large amounts of memory. This can be a barrier for researchers and practitioners with limited resources. However, recent advances in GAN architecture and training techniques have addressed this problem to some extent by reducing the complexity and training time of

GAN models. The goal of GAN optimization is to locate the global Nash equilibrium of a zero-sum game, which is also known as a minimax game. In this game, both the generator and the discriminator want to minimise their cost function by locating a point (D,G) where JD for the discriminator is at a minimum for D and JG for the generator is at a minimum for G. Nash equilibrium is already a difficult issue to solve; however, solving it with GANs is significantly more difficult due to the objective function being non-convex and the parameters having a large dimension and being continuous. In addition, gradient-based training, which is a local optimization method that has a tendency to identify local Nash equilibria and fails to converge, is the most popular training strategy employed in GANs training. This technique was developed in the 1980s. (A modification to D in order to decrease JD may result in an increase in JG, and a modification to G in order to decrease JG could also result in an increase in JD.) [67, 30, 31]. In training, GANs also have issues with mode collapse and gradient vanishing, both of which are common concerns.

4.7 Oscillations:

In some cases, the generator and discriminator can oscillate between different states, resulting in unstable training. This can occur when the discriminator is too weak, and the generator produces samples that are too diverse, making it difficult for the discriminator to distinguish between them. As a result, the discriminator becomes too strong, and the generator produces less diverse samples. This creates a feedback loop, where the generator and discriminator oscillate between different states.

To mitigate these instability issues, several techniques have been proposed, such as:

1. Adding noise to the input of the discriminator to prevent it from becoming too strong.
2. Using gradient penalties to encourage the gradients to remain non-zero.
3. Adding regularization to the generator to promote diversity in the generated samples.
4. Using different loss functions that are more robust to instability.

In conclusion, instability is a common issue in GANs that can result in poor performance or failure to converge. Vanishing gradients, mode collapse, and oscillations are some of the causes of instability. Several techniques have been proposed to mitigate these issues, but GANs still remain a challenging area of research. Overall, GANs are powerful tools for generating synthetic data that can be used in a variety of applications. However, the training process can be challenging due to the problems mentioned above, and it requires careful tuning and experimentation to achieve optimal results.

CHAPTER 5

TRAINING CHALLENGES

Generative Adversarial Networks (GANs) are a type of machine learning models that are designed to generate new data that is similar to a given dataset. Training GANs is a complex process, and there are several techniques that can be used to improve their performance. In this article, we will explain in detail the training techniques for GANs.

5.1 Adversarial Loss

The main idea behind GANs is to train two neural networks simultaneously: a generator and a discriminator. The generator generates new data, and the discriminator tries to distinguish between the generated data and the real data. The adversarial loss is used to train the generator and the discriminator simultaneously. The adversarial loss is computed based on the difference between the predictions of the discriminator for the real and generated data. The generator tries to minimize the adversarial loss, while the discriminator tries to maximize it.

5.2 Gradient Penalty

The gradient penalty is a regularization technique used to prevent the discriminator from becoming too powerful. In GANs, the discriminator is trained to maximize the adversarial loss. However, if the discriminator becomes too powerful, it can easily distinguish between the generated and real data, making it difficult for the generator to learn. The gradient penalty is a way to regularize the discriminator and prevent it from becoming too powerful.

5.3 Mini-Batch Discrimination

Mini-batch discrimination is a technique used to improve the diversity of the generated data. In GANs, the generator can generate similar data, which can lead to mode collapse. Mode collapse is a situation where the generator produces a limited set of outputs, ignoring the rest of the data. Mini-batch discrimination is a way to overcome mode collapse by introducing diversity in the generated data. The technique involves introducing additional information about the batch of data that the discriminator receives. The discriminator can then use this information to distinguish between the generated data.

5.4 Label Smoothing

Label smoothing is a regularization technique used to prevent overfitting in the discriminator. In GANs, the discriminator is trained to distinguish between the generated and real data.

However, if the discriminator becomes too good at this task, it can overfit to the training data, leading to poor performance on new data. Label smoothing is a way to prevent overfitting by introducing noise in the labels of the training data.

5.5 Progressive Growing

Progressive growing is a technique used to improve the stability and performance of GANs. The technique involves gradually increasing the resolution of the generated images during training. The generator and the discriminator are trained on lower resolution images first and then gradually move to higher resolution images. This allows the generator to learn low-level features first and then build on them to learn high-level features.

5.6 Conditional GANs

Conditional GANs are a type of GANs that allow the generator to generate data based on a specific condition. The condition can be any type of input, such as text or images. This technique is useful when the task involves generating data based on a specific input, such as image-to-image translation or text-to-image generation.

In conclusion, the training of GANs is a complex process that requires several techniques to improve their performance. Adversarial loss, gradient penalty, mini-batch discrimination, label smoothing, progressive growing, and conditional GANs are some of the techniques that can be used to train GANs effectively.

5.7 Architecture Rebuild

Generative Adversarial Networks (GANs) have been used to generate realistic images, videos, and other media with impressive results. However, the architecture of GANs is often complex, and can be challenging to design and optimize for specific tasks. Architecture rebuild in GANs refers to the process of redesigning and refining the architecture of GANs to improve their performance on specific tasks. The architecture of a GAN typically consists of two components: a generator and a discriminator. The generator generates new data samples, while the discriminator distinguishes between real and fake data samples. The generator and discriminator are trained together in an adversarial setting, with the generator learning to produce more realistic data samples, and the discriminator learning to better distinguish between real and fake data.

The architecture of the generator and discriminator can have a significant impact on the performance of the GAN. A poorly designed architecture can lead to poor quality results, slow convergence, and other issues. Architecture rebuild in GANs can help address these problems by optimizing the design of the generator and discriminator. There are several approaches to architecture rebuild in GANs. One approach is to use different types of layers and activation functions in the generator and discriminator. For example, using convolutional layers in the generator can help it generate more realistic images, while using batch normalization in the discriminator can help it better distinguish between real and fake data.

Another approach is to adjust the number of layers and neurons in the generator and discriminator. Adding or removing layers and neurons can impact the complexity and capacity of the GAN, which can affect its performance on different tasks. Other approaches to architecture rebuild in GANs include using different loss functions, adding regularization techniques, and adjusting the learning rate and batch size. These techniques can help improve the stability and convergence of the GAN and can lead to better results.

The goal of training a GAN is to learn the underlying distribution of desired samples and create ones that are realistic. The conventional generator, on the other hand, may not be strong enough to learn complicated distributions. It is possible that GAN will become more powerful and stable with the assistance of the new architecture. CGAN [60] presented an enhanced conditional model that was designed to provide G and D with more information. CGAN was able to produce samples of higher quality thanks to the auxiliary information, although the training sample still has to be labelled. DCGAN [63] took the role of spatial pooling, which had the potential to degrade the quality of the samples generated. StackGAN [82], which is derived from CGAN, presented a sketch-refinement method as a means of partitioning the text-to-image conversion procedure into two steps. In the first step, a crude picture with a low resolution is produced, and in the second stage, an image with a greater resolution and more information is produced. The progressive network that was proposed by PROGAN [46] begins with a resolution of 4 by 4 and builds up to a picture with a resolution of 1024 by 1024. SAGAN [80] developed an attention mechanism to collect information from a wider range, which allowed it to create pictures with more realistic features. This was accomplished by collecting information from a wider range. It was proposed by InfoGAN [22] to include a latent code as another input of the generator and to maximise the amount of mutual information between the output of the discriminator and the latent code. The Laplacian pyramid was included into CGAN by LAPGAN [26]. It was able to create photos of a better resolution and a quality that was believable. StyleGAN [47] suggested a different generator structure as a means of untangling the latent space. CycleGAN [83] presented a novel structure that produced high-quality translated pictures with unpaired samples. This structure had G and D numbers that were twice.

In summary, architecture rebuild in GANs involves redesigning and refining the architecture of the generator and discriminator to improve the performance of the GAN on specific tasks. There are several approaches to architecture rebuild, including adjusting the layers, neurons, activation functions, loss functions, regularization techniques, and learning rate. By optimizing the architecture of the GAN, we can improve its ability to generate realistic data samples and achieve better performance on a range of applications.

5.8 Cost Function

The cost function in Generative Adversarial Networks (GANs) plays a crucial role in the optimization of the model. GANs are a type of deep learning algorithm that aims to generate synthetic data that resembles the real-world data. The GAN architecture is comprised of two networks, the generator and the discriminator. The generator produces the synthetic data, and the discriminator classifies it as either real or fake. The cost function in GANs is designed to ensure that the generator produces high-quality synthetic data that is indistinguishable from the real data. The cost function in GANs consists of two components: the generator loss and the discriminator loss. The generator loss measures the difference between the synthetic data generated by the generator and the real data. The discriminator loss measures the difference between the discriminator's classification of the synthetic data and the real data.

The generator loss is calculated using the cross-entropy loss function. The cross-entropy loss function measures the difference between the probability distribution of the real data and the synthetic data. The generator loss is used to update the generator's parameters, which allows it to produce better synthetic data that resembles the real data. The discriminator loss is also calculated using the cross-entropy loss function. The discriminator loss measures the difference between the probability distribution of the real data and the synthetic data generated by the generator. The discriminator loss is used to update the discriminator's parameters, which allows it to classify the synthetic data more accurately.

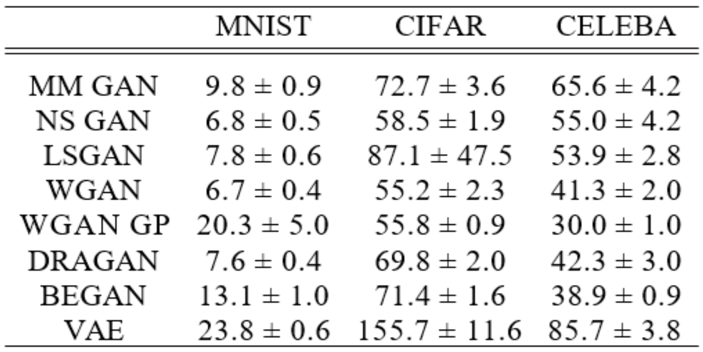
As discussed in Training problems, the KL-divergence and JS-divergence are defective for GANs. Some works focus on changing the objective function to encourage the model to convergence, and others focus on modifying the function with additional cost to enforce constraints. Table 4.1 lists the cost function of all the models we will discuss in this session. LSGAN [58] proposed to adopt Least square loss instead of the sigmoid cross-entropy loss, which could improve training stability and generate images with higher quality than the original GAN. Arjovsky et al. [12] proved that for an optimal D, Wasserstein distance between actual and generated distribution can be minimized by minimizing the value function for G. WGAN used Wasserstein distance which is smooth and can reflect the distance of two distributions even there is no overlap to replace the JS-divergence and solve the mode collapse and gradient vanishing problem.

Table 5.1: Discriminator and Generator Loss Function of each Model; MM GAN is the vanilla GAN, NS GAN is the vanilla GAN use alternative cost function in the original paper to improve the gradient signal.

	Discriminator Loss	Generator Loss
MM GAN	$-\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$-\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$-\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$-\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$-\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_D^{WGAN} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x}))\|_2 - 1)^2]$	$-\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LSGAN	$-\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$-\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x})-1)^2]$
DRAGAN	$\mathcal{L}_D^{GAN} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathbb{E}_{x\sim p_d}[\|x - AE(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - AE(\hat{x})\|_1]$	$\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - AE(\hat{x})\|_1]$

Because WGAN-GP [35] used gradient penalty to establish Lipschitz continuity, it was able to have a faster rate of convergence than WGAN. This was made possible by the fact that WGAN-GP introduced gradient penalty. BEGAN [15] moved away from using a discriminator in favour of an autoencoder and made use of the Wasserstein distance as a gauge of convergence. In addition, the introduction of a new hyperparameter is suggested as a means to achieve a balance between D and G throughout the training phase. DRAGAN [51] suggested including a gradient penalty in the original GAN, which would have brought both an increase in results and stability. The experimental results for the aforementioned functions are displayed in Table 4.2. These results were measured using FID while the architecture remained the same. It is difficult to determine which cost function is preferable to others due to the fact that certain cost functions may have worse results in one dataset but better results in another. In addition, if given sufficient processing power and an appropriate dataset, a subpar algorithm could be able to outperform superior ones.

Table 5.2: Best FID for each Model. Data from [55].



	MNIST	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	58.5 ± 1.9	55.0 ± 4.2
LSGAN	7.8 ± 0.6	87.1 ± 47.5	53.9 ± 2.8
WGAN	6.7 ± 0.4	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	155.7 ± 11.6	85.7 ± 3.8

The overall cost function in GANs is a combination of the generator loss and the discriminator loss. The goal of the cost function is to minimize the difference between the probability distribution of the real data and the synthetic data generated by the generator. As the generator produces better synthetic data that is more indistinguishable from the real data, the discriminator becomes better at classifying it. This creates a feedback loop, which allows the GAN to improve the quality of the synthetic data over time. In conclusion, the cost function in GANs plays a crucial role in the optimization of the model. It ensures that the generator produces high-quality synthetic data that is indistinguishable from the real data. The cost function is comprised of two components, the generator loss, and the discriminator loss, and it is designed to minimize the difference between the probability distribution of the real data and the synthetic data.

5.9 Optimization Methods

Generative Adversarial Networks (GANs) are a class of deep learning models used for generating realistic data, such as images, videos, and audio. GANs have two components: a generator network that creates fake data, and a discriminator network that tries to distinguish between real and fake data.

The optimization of GANs involves training both the generator and discriminator networks, which can be challenging. The following are some of the most popular optimization methods for GANs:

- A crucial iterative optimization approach in machine learning is referred to as Stochastic Gradient Descent (SGD). This method uses a gradient descent procedure to build an expectation of the gradient using just a minibatch of the available data. When used to huge datasets, it will prove to be more cost effective than gradient descent. In addition, SGD has the ability to ensure that convergence takes place for a convex or pseudo-convex cost function.

- Mini-batch stochastic gradient descent (SGD): This is a popular optimization method for training deep neural networks, including GANs. In mini-batch SGD, the weights of the generator and discriminator networks are updated using the gradients of the loss function computed on small batches of data.
- Adam optimization: This is a variant of mini-batch SGD that uses adaptive learning rates. Adam adjusts the learning rate for each parameter in the generator and discriminator networks based on the first and second moments of the gradients.

  Kingma et al. [49] propose Adam as a first-order gradient-based stochas- tic optimization method with adaptive learning rate. Adam is described as having these characteristics. It is suitable for non-convex optimization as well as problems with highly noisy and/or sparse gradients because to its high level of robustness and adaptability.
- Batch normalization is a technique used to normalize the inputs of a deep learning model. It is commonly used in Generative Adversarial Networks (GANs) to improve the training stability and convergence. In GANs, batch normalization is applied to the inputs of the generator and discriminator networks. The main idea is to normalize the inputs to have zero mean and unit variance across the training data. This helps to prevent the vanishing gradients problem and allows the networks to learn more efficiently. The batch normalization process involves computing the mean and variance of the inputs over a mini-batch of samples. Then, the inputs are normalized by subtracting the mean and dividing by the standard deviation. This is followed by a scaling and shifting operation, which allows the network to learn the optimal scale and bias for each layer. By applying batch normalization, the GANs can produce more stable and diverse outputs, and it is less sensitive to the choice of hyperparameters. This technique can also speed up the training process and reduce the number of training iterations required to achieve a good result. Batch Normalization [42] is a method that makes the optimization of deep neural networks simpler by lowering the amount of internal covariate shift. [42] It does this by i)normalizing each scalar feature individually with zero mean and variance of 1, and ii)producing estimates of the mean and variance of each activation for each mini-batch. This process is referred to as "whitening" [62, 78] the inputs of each layer. It's possible that using Batch Normalization will allow for a

faster learning rate while requiring less focus on the initialization. In addition to this, Batch normalisation may decrease the requirement for Dropout. In conclusion, batch normalization is an important technique for improving the training stability and convergence of GANs. It helps to prevent the vanishing gradients problem, and it can produce more stable and diverse outputs.

- Virtual batch normalization (VBN) is a technique used in Generative Adversarial Networks (GANs) to improve the stability and performance of the generator network. In standard batch normalization, the mean and variance of a batch of samples are calculated and used to normalize the inputs to a layer of the network. However, in GANs, each generated sample is unique and may not represent the same distribution as the original batch used for normalization. VBN addresses this problem by normalizing each generated sample with respect to a reference batch that is fixed and pre-computed. This reference batch is independent of the generator network and does not change during training. This allows the generator to learn a more stable and consistent mapping between the noise input and the generated samples' can also be used to facilitate network transfer between different data domains. By using a reference batch from a different domain, the generator can be trained to generate samples that are more representative of that domain. This technique has been shown to be effective in improving the performance and quality of GAN-generated images.

  The Virtual Batch Normalization (VBN) algorithm [67] was developed with the goal of eliminating the possibility that the output of a neural network for one input might be heavily reliant on the inputs of other networks in the same minibatch. It began by creating a reference batch that was normalised based on its own data, and then it repaired that batch before beginning training in order to normalise the input to the neural network. VBN is only utilised in generators due to the significant computational cost associated with executing forward propagation on two minibatch of data. VBN is only used in generators.

- Gradient Penalty is a technique used in Generative Adversarial Networks (GANs) to prevent the discriminator from becoming too powerful and overfitting the generator. It does this by adding a regularization term to the loss function of the discriminator, which penalizes it for having a steep gradient. This encourages the discriminator to have a smooth gradient, which in turn helps to stabilize the training process and improve the quality of the generated samples. The gradient penalty technique was first introduced in the Wasserstein GAN (WGAN) algorithm, which is a variant of GAN that uses the Wasserstein distance instead of the Jensen-Shannon divergence to measure the distance between the real and generated distributions. The gradient penalty is applied to the discriminator's loss function in WGAN by computing the norm of the gradient of its output with respect to the input samples and penalizing it if this norm deviates from a target value. The gradient penalty has several advantages over other regularization techniques such as weight decay or dropout. It is easy to implement, computationally efficient, and does not require tuning of hyperparameters. It has been shown to improve the stability of GAN training, reduce mode collapse, and improve the diversity and quality of the generated samples.

- Spectral normalization: Spectral Normalization is a technique used to normalize the weights of neural networks to reduce the instability of the model during training. It works by constraining the maximum singular value of each weight matrix in the network to be less than or equal to a pre-defined constant. This helps in stabilizing the training process by reducing the chance of exploding or vanishing gradients. In simpler terms, the spectral normalization technique ensures that the weights of the neural network are not too large or too small, which can lead to a lack of convergence or poor generalization. By keeping the weights within a certain range, the model is able to learn more efficiently and effectively. This technique is particularly useful in deep neural networks, which are prone to instability during training due to the complex interactions between layers. The Lipschitz constant is the sole configurable hyperparameter in the Spectral Normalization [61] normalisation approach, which is used to stabilise the discriminator training. By normalising the spectral norm of the weight matrix, it meets the Lipschitz constraint, which enables it to achieve a greater level of stability and a superior outcome. Overall, spectral normalization is a powerful tool for improving the performance and stability of neural networks, particularly in deep learning applications. It is often used in combination with other techniques such as batch normalization and weight decay to further enhance the effectiveness of the model.
- One-sided label smoothing: This is a technique used to prevent the discriminator from becoming too confident in its predictions. In one-sided label smoothing, the discriminator is trained to recognize real data as a range of values rather than a single value, which can prevent the generator from generating data that is too similar to the real data.

- A 1x1 convolution is a type of convolutional operation where the kernel size is 1x1. It is also called pointwise convolution since it acts on individual pixels or features. 1x1 convolutions are often used to adjust the depth or number of channels in the feature maps without changing their spatial dimensions. They are commonly used in deep neural networks to reduce the computational cost of the network and improve its accuracy. On the other hand, Network in Network (NiN) is a type of neural network architecture that uses a combination of 1x1 convolutions, global average pooling, and multiple fully connected layers to improve the accuracy of the model. In a NiN network, instead of using traditional convolutional layers, 1x1 convolutions are used to extract features and multiple fully connected layers are used for classification. This architecture helps to reduce the number of parameters in the network while improving the accuracy and speed of training. 1×1 Convolution or Network in Network is an operation to learn the representations from images proposed by Lin et al. [54]. It consists of a mlpconv layer and a global average pooling layer to generates feature maps.

5.10 Activation Function

An activation function is a mathematical function that is applied to the input of a neural network to introduce non-linearity in the output. This non-linearity helps the neural network to learn complex patterns in the data and make accurate predictions. Without an activation function, a neural network would just be a linear regression model that is unable to capture complex patterns in the data. Activation functions can be categorized into three types: sigmoid functions, rectified linear units (ReLU), and hyperbolic tangent functions. Sigmoid functions were widely used in the past, but now the most popular activation function is the ReLU function, which is simple and computationally efficient. The hyperbolic tangent function is another popular choice and is similar to the sigmoid function but with a different range. Activation functions take an input value and transform it into an output value, which is then used as the input for the next layer of the neural network. The output value is usually between 0 and 1 or -1 and 1, depending on the activation function used. The choice of activation function depends on the type of problem being solved and the architecture of the neural network.

Additionally some analysis also reveals that the training dynamics of GANs can be affected by the choice of activation function used in the generator and discriminator networks. Specifically, we find that using activation functions that are not smooth can lead to slower convergence of the GAN training, whereas smooth activation functions can result in faster convergence [4].

This highlights the importance of carefully selecting the activation functions for GANs and suggests that future research could explore the design of novel activation functions that are optimized for GAN training [4].

- Rectified Linear Unit (ReLU) is an activation function widely used in deep learning networks, particularly in convolutional neural networks. The ReLU activation function replaces all negative values in an input matrix with zero, while leaving positive values unchanged. The ReLU function is defined as follows:

$$f(x) = \max(0, x) \tag{5.1}$$

  where x is the input value. ReLU is a non-linear activation function that has become popular because of its simplicity and efficiency. It is faster to compute than other activation functions such as the sigmoid function or the hyperbolic tangent function.

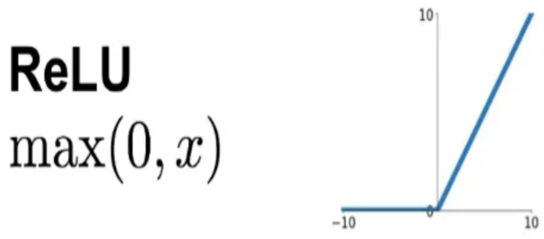


Figure 5.1: Representation of Rectified Linear Unit (ReLU)

ReLU is also less prone to the vanishing gradient problem, which can occur when training deep networks using other activation functions. The ReLU function is also sparsifying, meaning that it promotes sparse activations. This is because any input value that is less than or equal to zero is set to zero. As a result, ReLU helps to reduce the number of active neurons in a layer, which can lead to better performance in some cases. One downside of ReLU is that it can lead to "dead" neurons. If the input to a neuron is always negative, then the output of that neuron will always be zero[17]. This can happen if the weights of a neuron are initialized such that the input to the neuron is always negative. In this case, the neuron will never contribute to the output of the network, and its weights will not be updated during training. This is known as a dead neuron, and it can reduce the performance of the network. To mitigate the dead neuron problem, variants of ReLU have been proposed, such as leaky ReLU and parametric ReLU. Leaky ReLU adds a small positive slope to the negative values of the function, while parametric ReLU introduces a learnable parameter that controls the slope of the negative values.

In summary, ReLU is a popular activation function used in deep learning networks. Its simplicity and efficiency make it a good choice for large-scale networks, and its sparsity-promoting properties can lead to better performance. However, it can suffer from the dead neuron problem, which can be mitigated by using variants such as leaky ReLU or parametric ReLU.

- The Leaky ReLU (Rectified Linear Unit) activation function is an improved version of the popular ReLU function. The ReLU function is a non-linear activation function that is widely used in neural networks. It is defined as

$$f(x) = \max(0, x), \tag{5.2}$$

which means that it returns 0 if the input value is negative and returns the input value if it is positive[17]. The Leaky ReLU function, on the other hand, returns a small negative value for negative inputs. It is defined as

$$f(x) = \max(a*x, x), \tag{5.3}$$

where a is a small positive value that is usually set to 0.01. This means that for negative input values, the Leaky ReLU function returns a small negative value, which helps to avoid the "dying ReLU" problem that can occur with the ReLU function. The "dying ReLU" problem occurs when the ReLU function returns 0 for negative inputs.

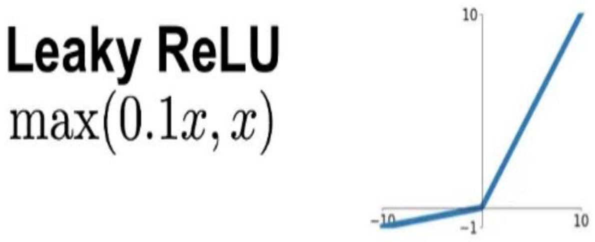


Figure 5.2: Representation of The Leaky ReLU (Rectified Linear Unit)

This can cause the gradient of the activation function to become zero, which can lead to the "vanishing gradient" problem during training. The vanishing gradient problem occurs when the gradients of the loss function become too small, making it difficult for the network to learn. The Leaky ReLU function helps to avoid these problems by providing a non-zero gradient for negative inputs. This allows the network to learn more effectively and improve its performance. In summary, the Leaky ReLU activation function is an improved version of the ReLU function that returns a small negative value for negative inputs. This helps to avoid the "dying ReLU" and "vanishing gradient" problems that can occur during training.

- The sigmoid activation function is a widely used non-linear function in deep learning and artificial neural networks. It takes an input value and squashes it into a range between 0 and 1. It is defined as:

  $$f(x) = 1 / (1 + \exp(-x)) \qquad (5.4)$$

  Here, the input x can be any real number, and the output f(x) always lies between 0 and 1.

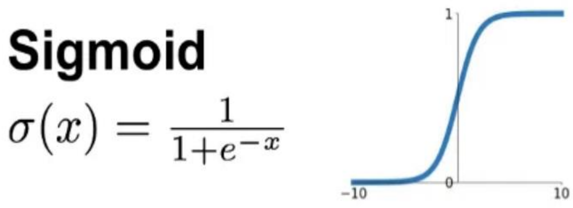


Figure 5.3: Representation of The sigmoid activation function

The sigmoid function is popular because it is easy to compute, and it provides a smooth gradient, which is necessary for backpropagation in neural networks. It is often used as an activation function in the output layer of a binary classification problem.

There are several advantages of using a sigmoid function as an activation function in neural networks:

1. Non-linearity: The sigmoid function is a non-linear function, which allows neural networks to learn complex relationships between inputs and outputs.
2. Easy to compute: The sigmoid function is a simple mathematical function that can be easily computed on a computer.
3. Smooth gradient: The sigmoid function provides a smooth gradient, which makes it easier to optimize the neural network using gradient descent.
4. Binary classification: The sigmoid function is commonly used in the output layer of a neural network for binary classification problems, where the output is either 0 or 1.

However, there are also some disadvantages of using a sigmoid function:

1. Vanishing gradient: The gradient of the sigmoid function becomes very small for very large or very small input values. This can lead to the problem of vanishing gradients, which makes it difficult to train deep neural networks.
2. Output range: The output of the sigmoid function is always between 0 and 1, which can cause problems if the output is required to be outside this range.

In summary, the sigmoid function is a useful activation function for neural networks, particularly in binary classification problems. However, it is important to be aware of its limitations and potential problems, such as vanishing gradients and output range constraints.

- The tanh (hyperbolic tangent) activation function is a non-linear mathematical function commonly used in artificial neural networks. It is similar to the sigmoid function but with a range of -1 to 1 instead of 0 to 1. The formula for the tanh function is:
  $\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$
  where e is the mathematical constant approximately equal to 2.71828 and x is the input to the function. The tanh function takes any real-valued number as input and returns a value between -1 and 1.

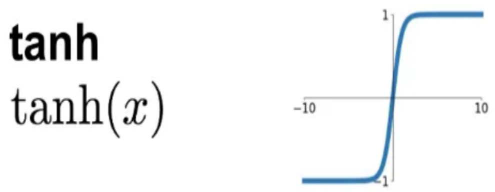


Figure : Representation of The tanh (hyperbolic tangent) activation function

When the input is close to zero, the output of the tanh function is also close to zero. As the input moves away from zero, the output approaches -1 or 1, depending on the sign of the input.
One of the main benefits of the tanh function is that it is symmetric around the origin. This means that if the input to the function is negated, the output will also be negated. This property can be useful in certain types of neural networks where the input can be positive or negative. Another advantage of the tanh function is that it is more sensitive to changes in the input compared to the sigmoid function. This can be helpful in certain applications where fine-grained distinctions need to be made between different inputs. However, one disadvantage of the tanh function is that it can suffer from the vanishing gradient problem, particularly when the input values are large. This means that the gradients can become very small, which can make it difficult to train the neural network.

In summary, the tanh activation function is a non-linear function that maps any real-valued input to a value between -1 and 1. It is useful in certain types of neural networks, but care must be taken to avoid the vanishing gradient problem.

- The Gaussian Error Linear Unit (GELU) is an activation function used in artificial neural networks. It was introduced in 2018 by Dan Hendrycks and Kevin Gimpel. The GELU function is a smooth approximation of the rectified linear unit (ReLU) function, which is commonly used in neural networks.The GELU function is defined as: $$\text{GELU}(x) = x\Phi(x)$$,where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. The GELU function is differentiable and monotonic, which makes it suitable for use in backpropagation algorithms. One of the advantages of the GELU function over the ReLU function is that it is smoother, which can help prevent overfitting. The ReLU function has a sharp corner at zero, which can cause problems during training. The GELU function, on the other hand, has a gradual transition from the linear region to the non-linear region, which can help prevent the network from getting stuck in local optima. Another advantage of the GELU function is that it has a mean of zero and a standard deviation of one, which can help improve the stability and convergence of the network. This is because the GELU function is based on the standard normal distribution, which is a well-behaved probability distribution. In summary, the Gaussian Error Linear Unit (GELU) is an activation function used in artificial neural networks. It is a smooth approximation of the rectified linear unit (ReLU) function and has several advantages over ReLU, including smoother behavior, better convergence properties, and a mean of zero and a standard deviation of one.
- Historical averaging is proposed by Goodfellow et al. [67] is a technique for improving the performance and stability of neural networks during training. The technique was first proposed by Ian Goodfellow in his 2016 paper "On large-batch training for deep learning: Generalization gap and sharp minima."
  The historical average can be denoted as follows:

$$\left\| \boldsymbol{\theta} - \frac{1}{t} \sum_{i=1}^{t} \boldsymbol{\theta}[i] \right\|^2 \quad (5.5)$$

In the preceding equation, θ[i] is the value of parameters at a particular time, i. This approach can improve the training stability of GANs too.

The basic idea behind historical averaging is to keep track of the past weights of the network during training and use the average of these past weights as the final weights of the network. Specifically, during training, the network's weights are updated using stochastic gradient descent (SGD) or a variant of it such as Adam or RMSProp. At the end of each epoch or mini batch, the current weights are added to a running average of the past weights. The final weights used for inference are the average of all the weights seen during training. The benefit of using historical averaging is that it helps to regularize the network and prevent overfitting. By averaging the weights over time, the final weights are less likely to be influenced by the noise in the training data or the specific mini-batches used for training. This can result in a network that generalizes better to new data and is less sensitive to the specific training set used. In addition to improving the generalization performance of the network, historical averaging can also help to stabilize the training process. The average weights tend to be smoother and more consistent than the individual weights seen during training, which can reduce the oscillations and instability that can occur during training. Overall, historical averaging is a simple and effective technique for improving the performance and stability of neural networks. It is easy to implement and can be used in combination with other regularization techniques such as dropout or weight decay.

In conclusion, the optimization of GANs is an ongoing research area, and many techniques have been developed to improve the stability and quality of generated data. The choice of optimization method depends on the specific application and the characteristics of the data being generated.

CHAPTER 6

CONCLUSION

In conclusion, Generative Adversarial Networks (GANs) have emerged as a powerful deep learning technique for generative modeling. GANs have proven to be effective in various applications, including image and video generation, style transfer, and data augmentation. The success of GANs can be attributed to their ability to learn the underlying data distribution and generate new samples that are similar to the real data. The survey of GANs presented in this paper provides an overview of the evolution of GANs, the challenges faced, and the progress made in addressing these challenges. The survey also highlights some of the key applications of GANs and the potential of GANs in solving real-world problems.

One of the key challenges of GANs is their instability during training, which can lead to mode collapse and poor quality outputs. However, recent advancements such as Wasserstein GANs, Progressive GANs, and StyleGANs have addressed these issues and have improved the stability and quality of GANs.

In addition to that, this study presents a variety of current metrics in order to evaluate the degree of similarity that exists between the synthetic samples and the actual samples. A decent measure should be able to compare the performance while also revealing the current faults that need to be improved. This article provides a wealth of examples that make use of supplemental metrics to paint a clearer picture of the performance being measured. For example, accuracy and recall separate the fidelity from the variety, which is something that FID does not support [66]. A number of different techniques, such as the Wasserstein Inception Distance [12] are proposed in addition to additional metrics in order to circumvent the inherent Gaussian limitation that is present in the FID. When assessing the efficacy of the models, researchers should think about utilising a variety of indicators to increase the likelihood of arriving at a convincing comparison. For a compelling demonstration, it is vital to have an understanding of the limits of the present measures, despite the fact that there is no general agreement over the standard metrics. Therefore, further research in the future is required to develop a generic metric that offers a comparison that is objective for all GANs.

Moreover, the survey also identifies the limitations of GANs, such as their high computational and memory requirements, and the difficulty of evaluating the performance of GANs.

However, these limitations can be addressed with further research and advancements in the field of GANs.

In summary, the survey paper on GANs presents an in-depth overview of the advancements and challenges of GANs in the field of deep learning. GANs have shown great potential in generative modeling, and with further research, GANs can be used to solve real-world problems in various domains. GANs are a promising deep learning technique, and their potential can be harnessed with continued research and development in the field.

BIBLIOGRAPHIES

[1] Dubey, Shiv Ram and Singh, Satish Kumar, (2023) “Transformer-based Generative Adversarial Networks in Computer Vision: A Comprehensive Survey”. arXiv preprint arXiv:2302.08641 [vol 1]. https://doi.org/10.48550/arXiv.2302.08641

[2] H. Hossam, E. Elgmmal and R. H. Elnabawy,(2022) "A Review of Generative Adversarial Networks Applications," 2022 4th Novel Intelligent and Leading Emerging Sciences Conference (NILES), Giza, Egypt, 2022, pp. 142-146, doi:10.1109/NILES56402.2022.9942383.

[3] P. Pradhyumna and Mohana,(2022) "A Survey of Modern Deep Learning based Generative Adversarial Networks (GANs)," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2022, pp. 1146-1152, doi: 10.1109/ICCMC53470.2022.9753782.

[4] Durr, Steven and Mroueh, Youssef and Tu, Yuhai and Wang, Shenshen,(2022) “Effective Dynamics of Generative Adversarial Networks” arXiv preprint :arXiv:2212.04580 [vol 1]. https://doi.org/10.48550/arXiv.2212.04580

[5] Afia Sajeeda, B M Mainul Hossain,(2022) “Exploring generative adversarial networks and adversarial training”. International Journal of Cognitive Computing in Engineering,[Vol 3], Pages 78-89, ISSN 2666-3074, https://doi.org/10.1016/j.ijcce.2022.03.002.

[6] Wenzel, Markus, (2022) “Generative Adversarial Networks and Other Generative Models” arXiv preprint : arXiv:2207.03887 [vol 1]. https://doi.org/10.48550/arXiv.2207.03887

[7] Dimitrios C Asimopoulos, Maria Nitsiou, Lazaros Lazaridis, George F Fragulis,(2022) “Generative Adversarial Networks: a systematic review and applications” .

SHS Web Conf. 139 03012 (2022) DOI: 10.1051/shsconf/202213903012

[8] Gilad Cohen, Raja Giryes, (2022) “Generative Adversarial Networks” corr/abs-2203-00667. Doi:10.48550/arXiv.2203.00667 [vol 1], https://doi.org/10.48550/arXiv.2203.00667

[9] Sofie Daniels, Jiugeng Sun, Jiaqing Xie, (2023) “CADA-GAN: Context-Aware GAN with Data Augmentation”. arXiv preprint arXiv:2301.08849 [vol 1], https://doi.org/10.48550/arXiv.2301.08849

[10] J. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, (2018) "Ganomaly: Semi-supervised anomaly detection via adversarial training," in 2018 IEEE International Conference on Image Processing (ICIP), Oct. 2018 [Vol 3], pp. 3318-3322. https://doi.org/10.48550/arXiv.1805.06725

[11] Arjovsky, M., & Bottou, L. (2017). Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862 [Vol 1]. https://doi.org/10.48550/arXiv.1701.04862

[12] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. In International Conference on Machine Learning (pp. 214-223) [Vol 3]. https://doi.org/10.48550/arXiv.1701.07875

[13] Barratt, S., & Sharma, R. (2018). A note on the inception score. arXiv preprint arXiv:1801.01973 [Vol 2]. https://doi.org/10.48550/arXiv.1801.01973

[14] Li, J., Zhou, W., Cao, L., & Liu, L. (2021). Quality Evaluation of GANs Using Cross Local Intrinsic Dimensionality. IEEE Transactions on Neural Networks and Learning Systems, 32(9), 4062-4075. https://doi.org/10.1109/TNNLS.2020.3043441

[15] Berthelot, D., Schumm, T., & Metz, L. (2018). BEGAN: Boundary Equilibrium Generative Adversarial Networks. arXiv preprint arXiv:1703.10717[Vol 4]. https://doi.org/10.48550/arXiv.1703.10717

[16] Bińkowski, M. et al. (2021) Demystifying MMD GANs, (pp. 2234-2244) [V1]. arXiv.org [Vol 1]. Available at: https://arxiv.org/abs/1801.01401 .

[17] Li, Bowen. Et al.(2021) A Survey on Generative Adversarial Networks. University of Dalware [Vol 1]. https://udspace.udel.edu/handle/19716/30449

[18] Borji, A. (2021). Pros and cons of GAN evaluation measures. arXiv preprint arXiv:2106.04568 [Vol 5]. https://doi.org/10.48550/arXiv.1802.03446

[19] Bounliphone, W., Belilovsky, E., Blaschko, B., Antonoglou, I., & Gretton, A. (2016). A test of relative similarity for model selection in generative models. arXiv preprint arXiv:1606.05908 [Vol 3]. https://doi.org/10.48550/arXiv.1511.04581

[20] Brock, Andrew, Jeff Donahue, and Karen Simonyan, (2019) "Large Scale GAN Training for High Fidelity Natural Image Synthesis." Proceedings of the 33rd International Conference on Neural Information Processing Systems(NeurIPS 2019), pp.5439-5450. https://dl.acm.org/doi/book/10.5555/3454287

[21] Peter J Burt and Edward H Adelson,(1987) The laplacian pyramid as a compact image code. In Readings in computer vision, pages 671–679. Elsevier, 1987[vol 1]. https://api.semanticscholar.org/CorpusID:8018433

[22] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan(2016): Interpretable representation learning by information maximizing generative adversarial nets. In Proceedings of the 30th International Conference on Neural Information Processing Systems, pages 2180–2188[vol 1], https://dl.acm.org/doi/10.5555/3157096.3157340

[23] Jianpeng Cheng, Li Dong, and Mirella Lapata(2016). Long short-term memory-networksfor machine reading. arXiv preprint arXiv:1601.0673 [vol 7]. https://doi.org/10.48550/arXiv.1601.06733

[24] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever (2019). Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509 [vol 1]. https://doi.org/10.48550/arXiv.1904.10509

[25] Daras, G. et al. (2020) "Your local gan: Designing two dimensional local attention mechanisms for generative models," IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), [Vol 1], pp. 14530–14540. Available at: https://doi.org/10.1109/cvpr42600.2020.01454 .

[26] Emily Denton, Soumith Chintala, Arthur Szlam (2015), and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. arXiv preprint arXiv:1506.05751,[Vol 1]. https://doi.org/10.48550/arXiv.1506.05751

[27] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright and K. Ramchandran, (2010) "Network Coding for Distributed Storage Systems," in IEEE Transactions on Information Theory 2010 , vol. 56, no. 9, pp. 4539-4551, Sept. doi: 10.1109/TIT.2010.2054295. https://ieeexplore.ieee.org/document/5550492

[28] P. Dimitrakopoulos, G. Sfikas, and Christophoros Nikou(2020). Wind: Wasserstein inception distance for evaluating generative adversarial network performance. In ICASSP 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3182–3186. https://researchr.org/publication/icassp-2020

[29] Soheil Feizi, Farzan Farnia, Tony Ginart, and David Tse (2017). Understanding GANs: the lqg setting. arXiv preprint arXiv:1710.10793, [Vol 2]. https://doi.org/10.48550/arXiv.1710.10793

[30] Ian Goodfellow (2016). Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160,[vol 4]. https://doi.org/10.48550/arXiv.1701.00160

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). Deep learning. MIT press, 2016. http://www.deeplearningbook.org

[32] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde- Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). Generative adversarial networks. arXiv preprint arXiv:1406.2661, [vol 1]. https://doi.org/10.48550/arXiv.1406.2661

[33] Arthur Gretton, Karsten Borgwardt, Malte J. Rasch, Bernhard Scholkopf, and Alexander J. Smola (2008). A kernel method for the two-sample problem, 2008. arXiv preprint 0805.2368[vol 1]. https://doi.org/10.48550/arXiv.0805.2368

[34] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye (2020). A review on generative adversarial networks: Algorithms, theory, and applications. arXiv preprint arXiv:2001.06937, [vol 1]. https://doi.org/10.48550/arXiv.2001.06937

[35] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville (2017). Improved training of Wasserstein gans. arXiv preprint arXiv:1704.00028, [vol 3]. https://doi.org/10.48550/arXiv.1704.00028

[36] Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla, and Venkatesh Babu Radhakrishnan (2017). Deligan : Generative adversarial networks for diverse and limited data, ArXiv preprint=1706.02071[Vol 1] . https://doi.org/10.48550/arXiv.1706.02071

[37] J¨org Hellwig and Thomas A. Collins (2021). Artificial Intelligence Index Report 2021

Chapter 1,[vol 1]. https://aiindex.stanford.edu/wp-content/uploads/2021/11/2021-AI-Index-Report_Master

[38] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (2018). Gans trained by a two time-scale update rule converge to a local nash equilibrium, arXiv:1706.08500 [Vol 6]. https://arxiv.org/abs/1706.08500

[39] Saifuddin Hitawala. Comparative study on generative adversarial networks (2018). arXiv preprint arXiv:1801.04271, [vol 1]. https://doi.org/10.48550/arXiv.1801.04271

[40] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon (2019). How generative adversarial networks and their variants work: An overview. ACM Computing Surveys (CSUR), 52(1):1–43, [vol 2], https://dl.acm.org/doi/10.1145/3301282

[41] Danijela Horak, Simiao Yu, and Gholamreza Salimi-Khorshidi (2020). Topology distance: A topology-based approach for evaluating generative adversarial networks, arXiv:2002.12054 [vol 1]. https://doi.org/10.48550/arXiv.2002.12054

[42] Sergey Ioffe and Christian Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. PMLR 37, [vol 1]. https://proceedings.mlr.press/v37/ioffe15.html

[43] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros (2017). Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1125–1134, https://ieeexplore.ieee.org/document/9741362

[44] Abdul Jabbar, Xi Li, and Bourahla Omar (2020). A survey on generative adversarial networks: Variants, applications, and training. arXiv preprint arXiv:2006.05132,[vol 1]. https://doi.org/10.48550/arXiv.2006.05132

[45] Yuhao Kang, Song Gao, and Robert E Roth (2019). Transferring multiscale map styles

using generative adversarial networks. International Journal of Cartography, 5(2-3):115–141, [vol 1]. https://cartogis.org/wpcontent/uploads/2019/07/2019_US_National_Report

[46] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen (2017). Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196, [VOL 3]. https://doi.org/10.48550/arXiv.1710.10196

[47] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks (2019). In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4401–4410, 2019. https://ieeexplore.ieee.org/xpl/conhome/1000147/all-proceedings

[48] Valentin Khrulkov, Ivan Oseledets, Proceedings of the 35th International Conference on Machine Learning, PMLR 80:2621-2629, (2018). https://proceedings.mlr.press/v80/khrulkov18

[49] Diederik P Kingma and Jimmy Ba. Adam, (2014), A method for stochastic optimization. arXiv preprint arXiv:1412.6980, [vol 1]. https://doi.org/10.48550/arXiv.1412.6980

[50] Diederik P Kingma and Max Welling (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, [vol 2]. https://doi.org/10.48550/arXiv.1312.6114

[51] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira (2017). On convergence and stability of gans. arXiv preprint arXiv:1705.07215, [vol 5]. https://doi.org/10.48550/arXiv.1705.07215

[52] Mark A Kramer (1991), Nonlinear principal component analysis using autoassociative neural networks. AICHE journal, 37(2):233–243. https://doi.org/10.1002/aic.690370209

[53] Tuomas Kynk¨a¨anniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila (2019). Improved precision and recall metric for assessing generative models, arXiv:1904.06991, vol 3]. https://doi.org/10.48550/arXiv.1904.06991

[54] Min Lin, Qiang Chen, and Shuicheng Yan(2013). Network in network. arXiv preprint arXiv:1312.4400, [vol 3]. https://doi.org/10.48550/arXiv.1312.4400

[55] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet (2018). Are gans created equal? a large-scale study, arXiv:1711.10337, [vol 4]. https://doi.org/10.48550/arXiv.1711.10337

[56] A. M. Turing, I (1950).Computing Machinery And Intelligence, Mind, Volume LIX, Issue 236, Pages 433–460, https://doi.org/10.1093/mind/LIX.236.433

[57] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey (2015). Adversarial autoencoders. arXiv preprint arXiv:1511.05644, [vol 2]. https://doi.org/10.48550/arXiv.1511.05644

[58] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang and S. P. Smolley (2017)."Least Squares Generative Adversarial Networks," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, pp. 2813-2821, doi: 10.1109/ICCV.2017.304. https://ieeexplore.ieee.org/document/8237566

[59] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein (2016). Unrolled generative adversarial networks. arXiv preprint arXiv:1611.02163, [vol 5]. https://doi.org/10.48550/arXiv.1611.02163

[60] Mehdi Mirza and Simon Osindero(2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, [vol 1]. https://doi.org/10.48550/arXiv.1411.1784

[61] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957, [vol 1]. https://doi.org/10.48550/arXiv.1802.05957

[62] Genevieve B Orr and Klaus-Robert M¨uller (2003). Neural networks: tricks of the trade. Springer,Pages 51-53, 113-132[vol 4]. https://link.springer.com/book/10.1007/3-540-49430-8

[63] Alec Radford, Luke Metz, and Soumith Chintala (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434,[vol 2]. https://doi.org/10.48550/arXiv.1511.06434

[64] Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, and Shakir Mohamed(2017). Variational approaches for auto-encoding generative adversarial networks, arXiv:1706.04987, [vol 2]. https://doi.org/10.48550/arXiv.1706.04987

[65] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1985). Learning internal representations by error propagation. Technical report, California University San Diego La Jolla Inst for Cognitive Science, 1985.

[66] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly (2018). Assessing generative models via precision and recall, arXiv:1806.00035, [vol 2]. https://doi.org/10.48550/arXiv.1806.00035

[67] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen (2016). Improved techniques for training gans. CoRR, abs/1606.03498, [vol 1]. https://doi.org/10.48550/arXiv.1606.03498

[68] Divya Saxena and Jiannong Cao (2021). Generative adversarial networks (gans) challenges, solutions, and future directions. ACM Computing Surveys (CSUR), 54(3):1–42, https://dl.acm.org/doi/10.1145/3446374

[69] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb (2017). Learning from simulated and unsupervised images through adversarial training. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2105–2119,[vol 1]. https://ieeexplore.ieee.org/xpl/conhome/1000147/all-proceedings

[70] Lo¨ıc Simon, Ryan Webster, and Julien Rabin (2019). Revisiting precision and recall definition for generative model evaluation, arXiv preprint arXiv:1905.05441,[vol 1]. https://doi.org/10.48550/arXiv.1905.05441

[71] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin (2019). Adaptive attention span in transformers. arXiv preprint arXiv:1905.07799, [vol 2]. https://doi.org/10.48550/arXiv.1905.07799

[72] Lucas Theis, A¨aron van den Oord, and Matthias Bethge (2015). A note on the evaluation of generative models. arXiv preprint arXiv:1511.01844, [vol 3]. https://doi.org/10.48550/arXiv.1511.01844

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). Attention is all you need. In Advances in neural information processing systems, pages 5988–6009, [vol 30]. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[74] C´edric Villani (2009). Optimal transport: old and new, volume 338. Springer.

[75] Zhengwei Wang, Qi She, and Tomas E Ward (2019). Generative adversarial networks: A survey and taxonomy. arXiv preprint arXiv:1906.01529, 2, [vol 6]. https://doi.org/10.48550/arXiv.1906.01529

[76] David Warde-Farley and Yoshua Bengio (2017). Improving generative adversarial networks with denoising feature matching. International Conference on Learning Representations, [vol 3]. https://openreview.net/forum?id=S1X7nhsxl

[77] R. Webster, J. Rabin, L. Simon and F. Jurie (2019), "Detecting Overfitting of Deep Generative Networks via Latent Recovery," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019, pp. 11265-11274,doi: 10.1109/CVPR.2019.01153. https://ieeexplore.ieee.org/document/8953411

[78] Asperti, A., Tonelli, V. Comparing the latent space of generative models. Neural Comput & Applic 35, 3155–3172 (2023). https://doi.org/10.1007/s00521-022-07890-2

[79] Simon Wiesler and Hermann Ney (2011). A convergence analysis of log-linear training.

Advances in Neural Information Processing Systems, 24:657–665, 2011, [vol 24]. Inproceedings NIPS2011_e836d813, https://proceedings.neurips.cc/paper/2011/file/e836d813fd184325132fca8edcdfb40e-Paper.pdf

[80] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Q (2018). Weinberger. An empirical study on evaluation metrics of generative adver- sarial networks. CoRR, abs/1806.07755, 2018[vol 2]. https://doi.org/10.48550/arXiv.1806.07755

[81] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena (2019). Self-attention generative adversarial networks. Proceedings of the 36th International Conference on Machine Learning, pages 7354–7363. PMLR, 2019[vol 97].pmlr-v97-zhang19d, https://proceedings.mlr.press/v97/zhang19d.html

[82] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas (2017). Stackgan: Text to photo-realistic image syn- thesis with stacked generative adversarial networks. In Proceedings of the IEEE international conference on computer vision, pages 5907–5915, 2017. https://ieeexplore.ieee.org/document/9011034

[83] J. -Y. Zhu, T. Park, P. Isola and A. A. Efros (2017), "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 2242-2251, doi: 10.1109/ICCV.2017.244. https://ieeexplore.ieee.org/document/8237506