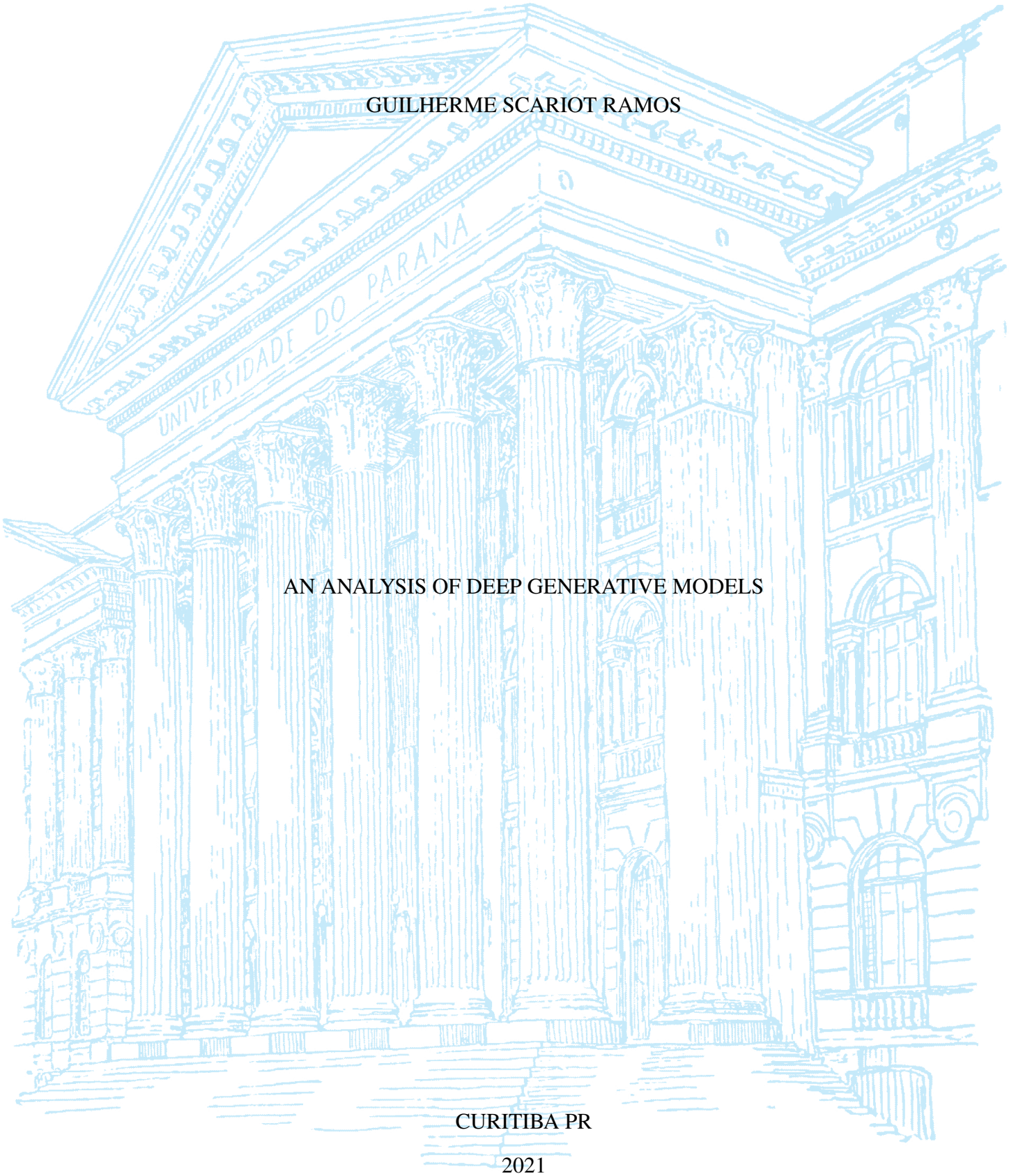
UNIVERSIDADE FEDERAL DO PARANÁ

GUILHERME SCARIOT RAMOS



AN ANALYSIS OF DEEP GENERATIVE MODELS

CURITIBA PR

2021

GUILHERME SCARIOT RAMOS

AN ANALYSIS OF DEEP GENERATIVE MODELS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: David Menotti Gomes.

CURITIBA PR

2021

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Graduação II

Título do Trabalho: AN ANALYSIS OF DEEP GENERATIVE MODELS

Autor(es):

GRR 20163086 None: GUILHERME SCARIOT RAMOS

GRR _____ Nome: _____

GRR _____ Nome: _____

Apresentação: Data: 27 / 02 / 2021 Hora: 9:00 Local: VIRTUAL-meet / DINF / UFPR

Orientador: DAVID MENOTTI GOMES

Membro 1: VALTER LUÍS ESTEVAM JUNIOR

Membro 2: GABRIEL SALOMON ANICETO

(nome) (assinatura)

AVALIAÇÃO – Produto escrito		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Conteúdo	(00-40)	36	36	36	36
Referência Bibliográfica	(00-10)	04	04	04	04
Formato	(00-05)	02	02	02	02
AVALIAÇÃO – Apresentação Oral					
Domínio do Assunto	(00-15)	15	15	15	15
Desenvolvimento do Assunto	(00-05)	02	02	02	02
Técnica de Apresentação	(00-03)	01	01	01	01
Uso do Tempo	(00-02)	01	01	01	01
AVALIAÇÃO – Desenvolvimento					
Nota do Orientador	(00-20)	14	**********	**********	14
NOTA FINAL		**********	**********	**********	**75**

Pesos indicados são sugestões.

Conforme decisão do colegiado do curso de Ciência da Computação, a entrega dos documentos comprobatório de trabalho de graduação 2 deve deve respeitar os seguintes procedimentos: Orientador deve abrir um processo no Sistema Eletrônico de Informações (SEI – UFPR); Selecionar o tipo: Graduação: Trabalho Conclusão de Curso; informar os interessados: nome do aluno e o nome do orientador; anexar esta ata escaneada e a versão final do pdf da monografia do aluno.; Tramita o processo para CCOMP (Coordenação Ciência da Computação).

RESUMO

Modelos generativos podem ser usados para produzir novas amostras de dados, os quais seguem uma certa distribuição de probabilidade. Até recentemente, modelos generativos implementados como redes neurais eram incapazes de aprender representações de dados que possuem distribuições altamente complexas, como conjuntos de dados contendo milhares de fotos. O método generativo-adversário para o treinamento de redes neurais foi proposto logo após o aumento no uso de redes neurais profundas para a classificação de imagens e, desde então, recebeu muitas melhorias, as quais eventualmente levaram ao desenvolvimento de arquiteturas em grande escala que são capazes de produzir imagens fotorrealísticas de alta resolução. Em vez de simplesmente memorizar o conjunto de dados, as Redes Adversariais Generativas (Generative Adversarial Networks - GANs) são capazes de produzir amostras de dados completamente novas que não estão presentes no conjunto de dados real. Este trabalho apresenta o método de treinamento adversarial e o embasamento teórico por trás dele, que empresta conceitos da teoria da informação e da estatística. Melhorias na formulação original também são discutidas e analisadas - isso inclui a GAN Convolucional Profunda (Deep Convolutional GAN - DCGAN); o uso de uma aproximação da distância Wasserstein-1/Distância do Earth Mover como uma função de custo na Wasserstein GAN (WGAN); o uso do método Gradient Penalty como uma melhoria para GANs de Wasserstein com clip de peso; e o uso de técnicas de normalização, principalmente o método de Normalização Espectral com a SNGAN. Embora existam alguns artigos publicados envolvendo GANs aplicadas a tarefas como geração de som, o foco deste trabalho é a síntese de imagens por meio do uso do método generativo-adversário por meio de aprendizagem não supervisionada. Uma vez que para a maioria das arquiteturas os valores da função de custo não podem ser usados como uma medida da qualidade visual subjetiva das amostras, muitos métodos para quantificar a qualidade das amostras geradas foram propostos, como o Inception Score (IS) e o Fréchet Inception Distance (FID), os quais são independentes da arquitetura da rede geradora. A base teórica para ambos os métodos é discutida neste trabalho, juntamente com os detalhes de implementação. Ambos os métodos são então aplicados à avaliação das implementações da DCGAN, WGAN com Grandient Penalty (WGAN-GP) e da SNGAN, as quais foram treinadas no conjunto de dados CelebA. Limitações de ambas as medidas IS e FID são apresentadas, e um caso interessante encontrado durante os experimentos práticos - onde a medida FID é apenas fracamente correlacionada com a qualidade da amostra - é analisado e discutido. A SNGAN é treinada no conjunto de dados CIFAR10, e os resultados mostram a variabilidade de ambas as medidas de quantificação quando aplicadas a amostras provenientes de geradores treinados em conjuntos de dados diferentes, ao mesmo tempo que mostra uma variabilidade significativa relacionada ao número de amostras. A SNGAN também é treinada em uma versão reduzida do novo conjunto de dados Flickr-Faces-HQ (FFHQ), e seus resultados e medidas são analisadas, um resultado que parece ainda não ter sido publicado na literatura.

Palavras-chave: Redes Adversárias Generativas. Aprendizagem Profunda. Redes Neurais. Modelo Generativo.

ABSTRACT

Generative models can be used to produce new data points that follow a certain probability distribution. Up until recently, generative models implemented as neural networks were unable to learn representations of data with highly complex distributions, such as datasets containing thousands of photos. The generative adversarial framework for training neural networks was proposed shortly after the surge in use of deep neural networks for image classification, and has since then received many improvements, which eventually led to the development of large-scale architectures that are able to produce high-resolution and photorealistic images. Instead of simply memorizing the dataset, Generative Adversarial Networks (GANs) are able to produce completely new data points that are not present in the real dataset. This work presents the adversarial training framework and the theoretical background behind it, which borrows concepts from information theory and statistics. Improvements to the original formulation are also discussed and analyzed – this includes the Deep Convolutional GAN (DCGAN); the use of an approximation of the Wasserstein-1/Earth Mover's Distance distance as a loss function in the Wasserstein GAN (WGAN); the use of the Gradient Penalty method as an improvement for weight-clipped Wasserstein GANs; and the use of normalization techniques, most notably the Spectral Normalization method in the SNGAN. Although there has been some published research involving GANs applied to tasks such as sound generation, the focus of this work is image synthesis through the use of the generative adversarial framework through unsupervised learning. Since for most architectures the loss function values cannot be used as a measure of subjective visual quality of the samples, many architecture-agnostic methods for quantifying the quality of the generated samples have been proposed, such as the Inception Score (IS) and the Fréchet Inception Distance (FID). The theoretical background for both methods is discussed in this work, together with the implementation details. Both methods are then applied to the evaluation of implementations of the DCGAN, WGAN with Gradient Penalty (WGAN-GP) and the SNGAN, which were trained on the CelebA dataset. Limitations of both IS and FID measures are presented, and an interesting edge case found during the practical experiments – where the FID measure is only weakly correlated with sample quality – is analyzed and discussed. The SNGAN is trained on the CIFAR10 dataset, and the results show the variability of both quantifying measures when applied to samples coming from generators trained on different datasets, while also showing a significant variability related to the number of samples. The SNGAN is also trained on a downsampled version of the new Flickr-Faces-HQ (FFHQ) dataset, and its results and measures are analyzed, a result which seems to have not yet been published.

Keywords: Generative Adversarial Networks. Deep Learning. Neural Networks. Generative Model.

CONTENTS

1 INTRODUCTION

One of the most astonishing recent results in the field of machine learning, and arguably computer science research in general, is the ability to produce completely new, previously unseen, visually coherent images that seem to mimic reality to a degree where a viewer can often be confused about whether a sample is real or generated. Although the concepts, theory and architectures developed for this task are not limited to the image domain, most of the recent progress and research focus so far in this field seem to be related to image generation. It is not entirely clear as to why this is the case, but it has been suggested that convolutional layers might somehow make the task easier in the visual domain (see Chapter 4), with no known analogs in other domains, such as text and sound generation. The focus of this work is related specifically to generative models that try to learn the underlying probability distributions behind a certain image dataset, in order to produce new images that are similar to it. Since our objective is to produce new, realistic images, one may question how researchers can measure results produced by a certain generative model architecture. Unlike tasks such as image classification, where the dataset has expected class labels and, therefore, an error metric can be directly computed, the task of generating new samples comes with no easy way of quantifying how results from a model differ from other models. The objective of this work is to introduce the theory behind recent advancements in the field of deep generative modeling, present and implement different model architectures, and to quantify the generated results based on recent metrics that have been proposed, while also analyzing the strengths and issues for each of those quantitative metrics, and how quantitative metrics might be correlated with different choices of loss functions.

Although practical applications of deep generative modeling are not the focus of this work, it is worth mentioning a few of them to further justify the study of this new and fascinating research field. Deep generative modeling has been successfully applied for image super-resolution [1], noise removal [2] and inpainting [3]. When too little data is available for training other neural networks, deep generative models have also been used for data augmentation, with a common example being providing new synthetic data for medical image classifiers [4]. These models have been also used for conditional image synthesis, such as generating an image that is described by an input text (text-to-image) [5], and generating realistic portraits of people when only a set of edges are given [6].

1.1 GENERATIVE AND DISCRIMINATIVE MODELS

In the field of statistical classification, a discriminative classifier is a classifier that models the posterior $p(y|\boldsymbol{x})$ directly – that is, given an input $\boldsymbol{x}$, it tries to directly map x to class labels. The classic example of such a classifier are linear regression classifiers. In contrast, generative classifiers try to model the joint distribution $p(x, y)$, and the probability $p(y|x)$ can be then calculated via Bayes chain rules. Examples of generative classifiers are Naïve Bayes classifiers. The term **generative models**, as used in this work and other recent publications, might differ from the standard statistical classification and textbook terminology. As we shall see in the following chapters, neural network-based generative models usually take noise (often called **latent variables**, a term that comes from the probabilistic graphical models literature) that follow a certain probability distribution, such as a Gaussian distribution, which is fed-forward through the network, transforming it into a new image sample. To avoid confusion, it is important to make clear that the term *model* is loosely used in this work and also in the generative modeling

literature, but usually refers to a neural network architecture and its learned parameters, which are able to transform random input noise into new data that resembles the original training data in a one-shot manner. This can be contrasted with other methods proposed in the literature, such as auto-regressive models, with an example being the PixelCNN [7], where each pixel is generated based on a neighbourhood of other pixels.

1.2 APPROACHES TO GENERATIVE MODELING

The proposal of Generative Adversarial Networks (GANs) in 2014 by Goodfellow et al. [8] has sparked great interest in the field of generative modeling, with the original paper alone having over 25 thousand citations as of late 2020. Despite the fact that it has clearly taken over as the main approach to generative modeling, it is useful to inspect older proposals for it, which will enable us to understand basic statistical concepts, how GANs fit into the big picture of this research area and, more importantly, what are their advantages over older model proposals.

Up until recently, generative modeling had remained as a relatively fringe topic within statistics and machine learning research, much like neural networks had been an almost forgotten subject within the artificial intelligence research community until the last decade. After the publication of AlexNet in 2012 [9] and its subsequent success over traditional image classification methods used in competitions, researchers realized that relatively simple network architectures were able to outperform older classification methods that were based on hand-crafted feature extractors, such as the Scale Invariant Feature Transform, coupled with classifiers such as the Support Vector Machine.

For generative modeling, where the task is to learn the probability distributions underlying a dataset, the same concept was found to be true – relatively simple neural network architectures were able to produce samples that had a higher subjective visual quality then those produced by older, heavily theory-based statistical methods. A prime example of this are Fully Visible Belief Networks (FVBNs) [10], which falls into the class of the so-called explicit density models. This model uses the principle of Maximum Likelihood Estimation (MLE), which tries to choose the parameters for the model in such a way that the likelihood that the model assigns to the training data is maximized [11]. GANs notoriously do not necessarily fall under the maximum likelihood category, and although this produces an inferior likelihood estimator, it is paradoxically able to produce better visual results. Unlike GANs, which can produce results in a parallel, one-shot manner, explicit density models such as FVBNs generate samples step by step, which leads to a high running time, making it unpractical for real-time use. Other than FVBNs, another model proposed that is highly worth mentioning are Variational Autoencoders (VAEs) [12], which is a explicit density method that maximizes the log-likelihood of the density function by maximizing a lower bound called the Evidence Lower Bound (ELBO). Other than GANs, Variational Autoencoders seem to be the major and most-cited proposal that is able to produce images with a considerable high degree of visual quality.

Although the GAN adversarial concept and its training algorithm can be implemented with any kind of feed-forward neural network such as a simple Multilayer Perceptron (MLP), virtually all successful architectures within the image domain seem to implement Convolutional Neural Networks (CNNs), most notably the highly cited Deep Convolutional Generative Adversarial Network (DCGAN) [13] and its variations. Even if GANs have limited real-world applicability as of today, having an algorithm that is able to master and capture visual concepts within an

extremely high dimensional dataset shows how far artificial intelligence research has come, and might give us a foresight about what is possible to accomplish with neural networks in the future. Given enough computing time and power, GANs are able to produce the most realistic images ever produced by any kind of generative model, and thus will be the main focus of this study.

2 INFORMATION THEORY

Before introducing Generative Adversarial Networks, it is useful to discuss some topics on the field of information theory, which will help clarify cost functions used in neural networks and also the theoretical framework for this generative modeling method. We will discuss the discrete case of these concepts, although continuous analogs do exist.

2.1 INFORMATION AND ENTROPY

The information content of a certain event occurring measures, roughly speaking, the amount of surprise contained in it. An event with low probability to occur gives us a higher quantity of information, and vice-versa. This concept is particularly useful for finding optimal codings when transmitting data, by allocating events with higher probability to smaller message sizes, but it is also a foundational concept for comparing different probability functions. Consider a random variable $\boldsymbol{X}$. We will now introduce some useful elementary definitions.

Definition 2.1.1 *The information content, or simply information of a given random event $\boldsymbol{X} = x$ is defined as the inverse of the log-probability of this event occurring:*

$$\mathrm{I}(x) = \log \frac{1}{p(x)} = -\log p(x). \tag{2.1}$$

Definition 2.1.2 *The entropy of a random variable $\boldsymbol{X}$ is defined as the expected value of the information content of $\boldsymbol{X}$, which can be interpreted as the average value of information or surprise for possible outcomes over this random variable:*

$$\mathrm{H}(X) = \mathbb{E}\left[\mathrm{I}(X)\right] = \sum_x p(x)\mathrm{I}(x) = -\sum_x p(x)\log p(x). \tag{2.2}$$

The measurement units for entropy are **bits** for a logarithm with basis 2, and **nats** for the natural logarithm. When talking about probability distributions directly, where x is sampled from a distribution p, we might also use the notation:

$$\mathrm{H}(p) = -\sum_x p(x)\log p(x). \tag{2.3}$$

2.2 CROSS-ENTROPY AND KULLBACK-LEIBLER DIVERGENCE

Before discussing cross-entropy in a more general sense, let us consider the context of its inception, which is related to communication and message encodings. Consider two probability distributions, p and q. These distributions define how likely a word is to appear in a message, given the same word list (i.e. the same random variable). Both p and q might model these percentages differently, based on different vocabulary usages for reach sending or receiving end.

Definition 2.2.1 *The cross-entropy $H(p, q)$ over distributions p and q is defined as:*

$$\mathrm{H}(p|q) = -\sum_x p(x) \log q(x) \tag{2.4}$$

$$= -\mathbb{E}_{x\sim p}[\log q(x)]. \tag{2.5}$$

Within the communication theory context, this could be interpreted as the average amount of information sent over a channel given an encoding that follows p or q, and given a message that follows p or q. Considering this simple hypothetical example, one can imagine a 2x2 matrix representing the four possible scenarios, and how it would be possible to pick the best encoding for each pair of users of this communication channel.

For the generative modeling context, where we are often interested in approximating a distribution q to the original p, the concept of cross-entropy gives us a way to measure of distance between these two probability distributions.

Definition 2.2.2 *The Kullback-Leibler (KL) Divergence D_{KL} of p with respect to q is defined as:*

$$D_{KL}(p|q) = \mathrm{H}(p|q) - \mathrm{H}(p|p) \tag{2.6}$$

$$= -\sum_x p(x) \log q(x) + \sum_x p(x) \log p(x) \tag{2.7}$$

$$= \sum_x p(x) \log \frac{p(x)}{q(x)}. \tag{2.8}$$

The KL divergence is an asymmetric measure that can be used to compute how two probability distributions are different from each other. When trying to learn the probability distribution underlying a certain dataset through maximum likelihood estimation, it is straightforward to see that this is similar to starting from some arbitrary probability distribution and minimizing the KL divergence between the two distributions.

2.3 CROSS-ENTROPY AS A LOSS FUNCTION

Cross-entropy is often used as a loss function in machine learning for problems such as binary classification and multiclass or multinomial classification. Let us consider the binary case – for GANs, as we shall see in the next chapter, a discriminator is trained to output the probability that a sample is real, which falls under a binary classification problem. Consider now that a random variable $\boldsymbol{Y}$ can have outcomes $\{y_1, y_2\}$ when measured, with y_1 and y_2 being the probability for each class of our binary classification problem. Calculating the cross-entropy for this case, we have:

$$\mathrm{H}(p|q) = -\sum_{y\in\boldsymbol{Y}} p(y) \log q(y) \tag{2.9}$$

$$= -p(y_1) \log q(y_1) - p(y_2) \log q(y_2). \tag{2.10}$$

Considering that this is a binary classification problem, we know that y_1 and y_2 fall under a Bernoulli distribution – that is, $p(y_1) = 1 - p(y_2)$ and $q(y_1) = 1 - q(y_2)$. Let us rename $p(y_1) = y$ and $q(y_1) = \hat{y}$. We then have:

$$\mathrm{H}(p|q) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}), \tag{2.11}$$

which is the expression for the binary cross-entropy loss function that will be extensively used throughout this work. We can interpret y as following the real class distribution p, and its value is known a priori – that is, $y \in \{0, 1\}$ since we know for sure the class it belongs to. We can then understand $\hat{y}$ as being our class prediction probability following our classifier's current data distribution q.

Starting from Chapter 3, we will use the natural language notation $\texttt{BCE}(\hat{y}\ \texttt{is}\ y)$ for binary cross entropy with expected labels y and output prediction $\hat{y}$. The reasoning for this is the fact that it makes the mathematical notation easier to understand when applied in optimization contexts. This notation is defined below only for further reference purposes:

$$\texttt{BCE}(\hat{y}\ \texttt{is}\ y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{2.12}$$

When considering a batch size of m samples, the binary cross entropy is defined as the average over the number of samples:

$$\texttt{BCE}_m(\hat{y}\ \texttt{is}\ y) = \frac{1}{m} \sum_{(\hat{y},y)} \Big(- y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \Big). \tag{2.13}$$

2.4 JENSEN-SHANNON DIVERGENCE

Another important similarity metric when comparing two probability distribution is the Jensen-Shannon divergence. Unlike the Kullback-Leibler divergence, this is a symmetric measure. This is achieved by introducing a third measure m into the calculation, which is the arithmetic mean of two distributions p and q, and comparing them directly to m.

Definition 2.4.1 *The Jensen-Shannon divergence D_{JS} is defined as:*

$$D_{JS}(p|q) = \frac{1}{2} D_{KL}(p|m) + \frac{1}{2} D_{KL}(q|m), \tag{2.14}$$

where m is a mixture such that $m = (p + q)/2$.

3 GENERATIVE ADVERSARIAL NETWORKS

The generative adversarial framework, in its basic formulation, consists of training a generator G against a discriminator D through a minimax game [8]. Both G and D are implemented as neural networks. Let us assume that G outputs data following a probability distribution p_g, and that the dataset which we are trying to learn has an underlying distribution p_{data}. The generator takes as input the latent variables $\boldsymbol{z}$, which follow a distribution p_z, and output $\boldsymbol{x}$ according to the current p_g. The discriminator outputs a single value $D(\boldsymbol{x})$, which represents the probability that a sample $\boldsymbol{x}$ came from p_{data} instead of p_g. The generator's goal is to learn p_{data} in order to fool the discriminator, which is concurrently trained to become better at discerning real from fake samples. We can mathematically represent this game as the following expression over a value function $V(D, G)$ [8]:

$$\min_G \max_D V(D, G) \tag{3.1}$$

where $V(D, G)$ is defined as:

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

One can immediately see that this is a sum of cross-entropies and, as well we will see in further sections, is actually a lower bound to the Jensen-Shannon divergence measure between p_{data} and p_g (the actual bound value can be found in Chapter 5). This is also closely related to the reverse KL divergence between the two distributions – defined as $D_{KL}(p_g|p_{data})$, remembering that this measure is not symmetric. As we will see in Section 3.1, there is a high correlation between these three different measures.

Considering the original formulation from a game-theoretical perspective, let us analyze this function in two parts. Firstly, the discriminator will try to maximize $D(x)$ when $\boldsymbol{x}$ comes from the real dataset, and will also try to maximize the quantity $1 - D(G(z))$ when x comes from the latent variables $\boldsymbol{z}$ being fed through the generator. In order to do that, it has to decrease $D(G(z))$, which is the probability of the discriminator guessing that a fake sample is real. On the other hand, the generator will try to minimize the probability of the discriminator being right on real samples (first term), and will also try to minimize $1 - D(G(z))$ when a sample comes from the generator. In order to do that, is has to increase $D(G(z))$, which is the probability of the discriminator guessing that a fake sample is real. In practice, it has been found by the original authors that, early in training, the discriminator is good enough to discern fake from real samples, and we would have $\log(1 - D(G(z)))$ close to 0, which results in gradient saturation and makes the training process harder. To avoid that, the authors suggest to abandon the relatively elegant expression as seen in Equation 3.2, and to simply try to maximize $D(G(z))$ directly when updating the generator.

3.1 COST FUNCTIONS

Before introducing the training algorithm itself, let us introduce and discuss the possible cost functions for both generator and discriminator. For the original minimax game over the same

value function $V(D, G)$, we can define the discriminator's cost function $J^{(D)}$ over the model parameters $\boldsymbol{\theta}^{(D)}$ and $\boldsymbol{\theta}^{(G)}$ as:

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\mathbb{E}_{x\sim p_{data}}\left[\log\ D(\boldsymbol{x})\right] - \mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(\boldsymbol{z}))\right)\right] \tag{3.3}$$

$$= -\texttt{BCE}_m(\boldsymbol{x}\ \texttt{is real}) - \texttt{BCE}_m(G(\boldsymbol{z})\ \texttt{is not real}). \tag{3.4}$$

Notice that this is a sum of two binary cross-entropy functions, which are usually used when training an image classifier with a single sigmoid output, and can be rewritten as such. Equation 3.4 is written in something resembling natural language and can be interpreted more easily – the discriminator is clearly trying to discern between $\boldsymbol{x}$ and $G(\boldsymbol{z})$ by maximizing the probability that samples from the real dataset are detected as real, and also maximizing the probability that generated samples are detected as fake.

To be more precise and in accordance with the BCE definition (Equation 2.12), we explicitly define $D(\boldsymbol{x}) \stackrel{\text{def}}{=} (\boldsymbol{x}\ \texttt{is real})$ and $(1 - D(G(\boldsymbol{z}))) \stackrel{\text{def}}{=} (G(\boldsymbol{z})\ \texttt{is not real})$.

One of the advantages of our natural language notation is that the probability $1 - D(G(\boldsymbol{z}))$ can be inverted, and we can then also define $D(G(\boldsymbol{z})) \stackrel{\text{def}}{=} (G(\boldsymbol{z})\ \texttt{is real})$.

When considering the original **minimax** game, where the game is played over the same value function, it follows that:

$$J^{(G)} = -J^{(D)}. \tag{3.5}$$

Since we are not sampling from p_{data} for the generator update, as we will see in Algorithm 1, this effectively means that:

$$J^{(G)} = \mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(\boldsymbol{z}))\right)\right] \tag{3.6}$$

$$= \texttt{BCE}_m(G(\boldsymbol{z})\ \texttt{is not real}). \tag{3.7}$$

This could be interpreted as – when training the generator, minimize the probability that a fake sample is classified as not real.

Some implementations have the discriminator loss normalized by $\frac{1}{2}$, since it is trained with a minibatch containing real samples, and another batch of the same size containing fake samples, as we will see in section 3.2. As previously mentioned, since we are trying to avoid gradient saturation, it might be a good idea to use an heuristic approach and set a different cost function for the generator. Maximizing $D(G(z))$ inside $\log(1 - D(G(z)))$ can be achieved by simply writing it as $\log D(G(z))$ and flipping (once again) the sign of $J^{(G)}$, resulting in our alternate cost function for the generator, which is often called the **non-saturating heuristic**:

$$J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\mathbb{E}_{z\sim p_z}\left[\log D(G(\boldsymbol{z}))\right] \tag{3.8}$$

$$= -\texttt{BCE}_m(G(\boldsymbol{z})\ \texttt{is real}). \tag{3.9}$$

This could be interpreted as – when training the generator, maximize the probability that a fake sample is detected as being real. Equation 3.9 directly gives us a way to train a non-saturating GAN with standard binary cross-entropy functions on popular machine learning frameworks – by flipping the expected label for a generated sample to `real`. This is explained in further detail in Section 3.4.

Another cost function worth mentioning, considering the more general generative modeling context, is the possibility of setting a **maximum likelihood** maximization cost function for GANs, as seen in Goodfellow et al. [11], which can be shown to be the equivalent of minimizing $D_{KL}(p_{data} | p_g)$ [14] under the assumption that the discriminator is optimal. This is accomplished by setting the generator cost as $-\mathbb{E}_{z \sim p_z} exp(\sigma^{-1}(D(G(z))))$ [11], where σ is the logistic sigmoid function. In practice, and considering that the KL divergence is asymmetrical, GANs as previously formulated in the minimax form seem to perform the minimization of $D_{KL}(p_g | p_{data})$ instead [11], which is often called the reverse KL divergence. This minimization also seems to be closely related to the minimization of the Jensen-Shannon divergence, as can be seen in the empirical results presented in a paper by Nowozin et al. [15]. The minimax formulation decreases the probability that the generator will produce a sample that has low probability density assigned in the original data distribution, and has thus been theorized as an explanation as to why GANs produce visual outputs with higher aesthetic qualities, as opposed to maximum likelihood based methods, which will try to smooth the densities between modes, producing non-sharp outputs.

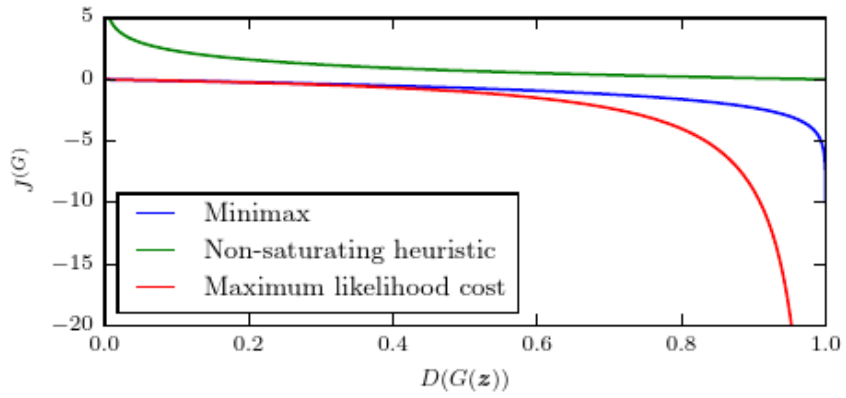


Figure 3.1: Comparison between different loss functions by Goodfellow et al. [11].

Figure 3.1 shows the relationship between the generator's loss function, $J^{(G)}$, and the output of the discriminator when a fake sample is used as input. During early training, when the discriminator is good at detecting fakes and $D(G(z)) \rightarrow 0$, the generator has a loss and thus gradient close to zero, which is the saturating scenario.

3.2 THE TRAINING ALGORITHM

Since we are training two neural networks at the same time, the training process is relatively less straightforward than training standard models such as an image classifier. The original algorithm alternates between k updates for the discriminator and one update to the generator. The algorithm presented below is adapted from the original [8] and assumes this is equal to one, as suggested by the authors:

Algorithm 1 GAN training algorithm.

for number of training iterations **do**
  update the discriminator:
    sample a minibatch of n real samples with labels 1 from p_{data}
    sample a minibatch of n fake samples with labels 0 from p_g by sampling noise from p_z and using it as input to the generator
    calculate $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ and **update** the discriminator
  update the generator:
    sample a minibatch of n fake samples with labels 0 from p_g by sampling noise from p_z and using it as input to the generator
    calculate $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ and **update** the generator
end for

Generative Adversarial Networks are, however, known as being hard to train. Algorithm 1, when implemented as simple neural networks on low-resolution datasets – such as the grayscale Toronto Face Dataset – can only produce some basic results that are not particularly impressive, even after fine tuning hyperparameters and using convolutional layers. This has been since then greatly improved by different network architectures, loss functions based on new theoretical backgrounds, and by a vast pool of knowledge gained from empirical research. Many of the popular methods used for training GANs, commonly known as GAN hacks, were first published in a 2016 paper by Radford et al. [13], together with a new architecture called Deep Convolutional Generative Adversarial Networks (DCGAN), which is the main subject of the next chapter and is also highly regarded as one of the first successful convolutional GANs.

3.3 IMPLEMENTATION CHALLENGES

As we have seen in Algorithm 1, not only are we training two neural networks, but the gradient of the generator is dependent on the output $D(G(\boldsymbol{z}))$ of the discriminator, regardless of the loss function being used. Implementations found on the web are often poorly documented and do not explicitly specify the cost function being used. There are many ways to implement Algorithm 1 with different deep learning libraries. We will mention two approaches to it – pure Keras, regardless of the backend being used; and TensorFlow 2.0 with gradient manipulations.

3.4 KERAS

Keras provides an easy, albeit hacky way to implement the non-saturating GAN. The GAN model, consisting of a generator and discriminator connected in a sequential order, can be defined as follows:

Algorithm 2 Keras GAN model definition.

Define the sequential discriminator model `D`, compile it by calling `D.compile(loss='binary_crossentropy', ...)`
Define the sequential generator model `G` and do not compile it
Define a third sequential model, `GAN`
Add `G` to `GAN` by calling `GAN.add(G)`
Set `D.trainable = False`
Add `D` to `GAN` by calling `GAN.add(D)`
Compile `GAN` by calling `GAN.compile(loss='binary_crossentropy', ...)`

Line 5 in Algorithm 2 is a hack that allows both discriminator and generator to be updated independently, as seen in the discriminator loop (Algorithm 1, Line 2) and generator loop (Algorithm 1, Line 6). We then add both `D` and `G` to a third model, `GAN`. The previously used hack enables us to update the generator `G` by training a third model, `GAN`, which will only affect `G`'s weights. The whole model can be thought as of a single sequential model, where both `G` and `D` are updated according to the gradients calculated considering the weights of each separate model but a single loss function, which is calculated as a function of the output of `D`.

The training algorithm can be then roughly defined as follows:

Algorithm 3 Keras non-saturating GAN training loop.

1: **for** batch **do**
2: Sample $\boldsymbol{x}$ (n real samples) with labels $y = 1$ (real)
3: Sample $\boldsymbol{x}^*$ (n fake samples) with labels $y^* = 0$ (fake)
4: Stack (2) and (3) in the same arrays as `stack(`$\boldsymbol{x}$`,` $\boldsymbol{x}^*$`)` and `stack(`y`,` y^*`)`
5: Train `D` by calling `D.train_on_batch(stack(`$\boldsymbol{x}$`,` $\boldsymbol{x}^*$`),stack(`y`,` y^*`))`
6: Generate latent variables $\boldsymbol{z}$ with labels $y_g = 1$ (real)
7: Train `G` indirectly by calling `GAN.train_on_batch(`$\boldsymbol{z}$`,` y_g`)`
8: **end for**

Line 6 in Algorithm 3 is what indirectly defines the cost function for the generator as seen in Eq. 3.8, that is, the non-saturating version of $J^{(G)} = -J^{(D)}$. The trick that allows this cost function to be defined is setting the labels y_g as 1 (real) for the generated outputs (as seen in Eq. 3.9), while feeding the latent variables $\boldsymbol{z}$ to our third model (`GAN`). Lines involving the generation of latent variables/noise were omitted, but implementations often use NumPy and its function `numpy.random.randn`$(\cdot)$ to sample from a standard normal distribution.

3.5 TENSORFLOW 2.0

Both TensorFlow 2.0 and PyTorch provide the ability to directly access the gradient tape and easily customize it. These gradients can be manipulated separately when there are multiple neural network models defined, which is especially useful when training GANs. Keras was also recently integrated into TensorFlow, and newer versions of it can be accessed only as a TensorFlow sub-module. What this means in practice is – it is possible to use Keras to define our network models in a simple and concise manner, while also being able to manipulate details such as custom gradients. This will become more relevant starting from Chapter 5, where we will use direct gradient operations.

Instead of defining `GAN` as a sequential model with `D` following `G` as seen in the Keras implementation, we can elegantly define `GAN` as a class inheriting from `tf.keras.Model` and override its `train_step` function, with `D` and `G` being contained within the `GAN` class. Algorithm 4 shows a snippet of the overridden `train_step` function when using TensorFlow 2.0 with `D` and `G` models defined by `tf.keras`, where `G_opt` and `D_opt` are any Keras-defined optimizer such as `tf.keras.optimizers.Adam` and `self.D_loss` is a user-defined loss function:

Algorithm 4 TensorFlow 2.0 GAN train_step override snippet. (Python)

```
 1: z = tf.random.normal(shape=(batch_size, self.latent_dim))
 2: G_z = self.G(z)
 3: stacked_x = tf.concat([x, G_z], axis=0)
 4: stacked_y = tf.concat([tf.ones((batch_size, 1)),
                           tf.zeros((batch_size, 1))], axis=0)
 5: with tf.GradientTape() as tape:
 6:   stacked_D = self.D(stacked_x)
 7:   loss_D = self.D_loss(stacked_y, stacked_D)
 8: grad = tape.gradient(loss_D, self.D.trainable_weights)
 9: self.D_opt.apply_gradients(zip(grad, self.D.trainable_weights))
10: z = tf.random.normal(shape=(batch_size, self.latent_dim))
11: y = tf.ones((batch_size, 1))
12: with tf.GradientTape() as tape:
13:   D_G_z = self.D(self.G(z))
14:   loss_G = self.G_loss(y, D_G_z))
15: grad = tape.gradient(loss_G, self.G.trainable_weights)
16: self.G_opt.apply_gradients(zip(grad, self.G.trainable_weights))
```

Note that, on Line 12, we will still have the gradients defined on the gradient tape as if we had only one sequential generator-discriminator network, but we still update only G's weights. To define the saturating loss function for the generator, we can either define a custom loss function `G_loss` directly – or use the standard binary cross-entropy function without flipping the expected labels (Line 11), and then multiplying the generator's loss by -1 (Line 14).

3.6 SATURATING GRADIENT COMPARISON

Figure 3.2 shows an attempt at analyzing the suggested gradient issues caused by using the original minimax/saturating loss function, using a subset of the CelebA dataset for 5 epochs, with each epoch having 943 steps. On the graph, the saturating loss results for the generator are multiplied by -1 for an easier comparison, since the saturating loss function leads to a negative number (as seen in Equation 3.7). The architecture used in this experiment was a simple convolutional architecture without label smoothing or batch normalization, following similar conditions similar to the original paper.

It is easy to notice that, early in training, the non-saturating G loss leads to a stronger gradient. Although more modern practices such as adding label noise seem to decrease such issues, the non-saturating loss function became standard practice due to its simplicity to implement, higher stability and faster convergence during training. One important fact to mention is that, in this architecture, images were normalized to the range $[0, 1]$, the generator is similar to a mirrored discriminator, and the generator's output has a sigmoid activation function. For the purpose of comparisons with other reports found in the literature it is worth mentioning that, when using the tanh activation function (used in architectures such as DCGAN), the generator usually produces a higher loss value.

3.7 LABEL CONDITIONING

One way to considerably improve the results produced by the generator is training the network with class labels. When trained this way, the network is often able to produce less ambiguous objects, a phenomenon called class-leakage by some authors. The improvements seem to be very

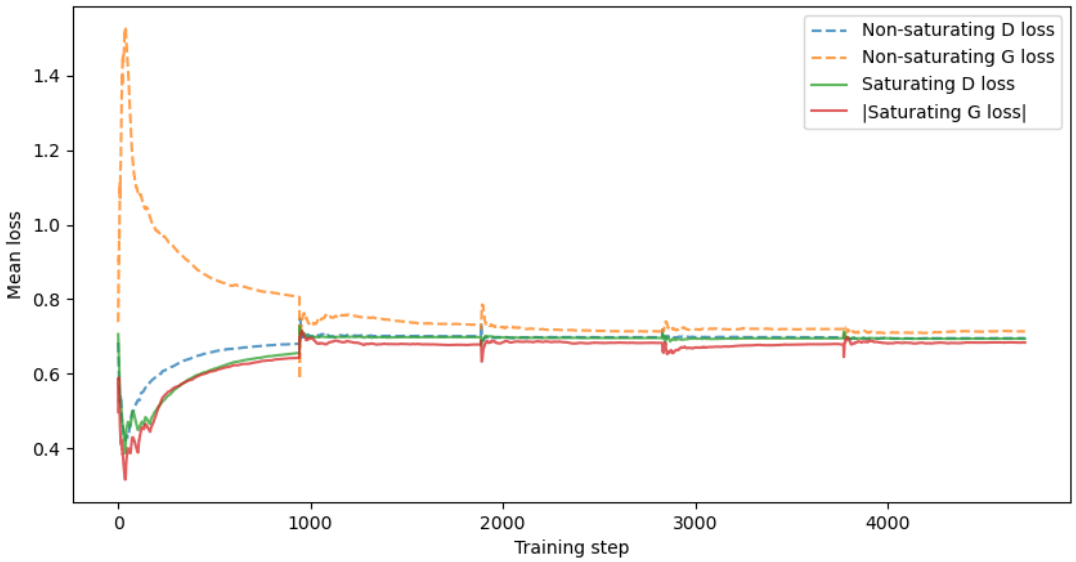


Figure 3.2: Comparison between different saturating and non-saturating functions on a convolutional GAN.

consistent between a large number of GAN architectures, to such an extent that this has become a separate sub-field of its own, where comparing class conditioned GANs to unsupervised GANs is avoided or explicitly mentioned. Due to this, it is important to state that the focus of this work are unsupervised GANs, although conditional GANs are briefly mentioned in later chapters.

3.8 LATENT SPACE INTERPOLATION

A remarkable property of Generative Adversarial Networks is the fact that they are able to learn latent space mappings in a way such that arithmetic can be done over the latent vectors z. Figure 3.3 shows a linear interpolation done between vector z_i and z_j. More sophisticated arithmetic can also be made with the latent vectors, such as addition and subtraction. A common example of this is adding a latent vector of a person without sunglasses to another vector, which is the latent vector of a person with sunglasses, resulting in a new third sample containing a new person with sunglasses, with this third person looking roughly like an interpolation between the initial two.

Figure 3.3: GAN-generated samples from a linear interpolation between latent vectors z_i and z_j, which are used as inputs to the network. Source: Donahue et al. [16], adapted.

4 DEEP CONVOLUTIONAL GAN

The Deep Convolutional Generative Adversarial Network (DCGAN) [13] was one of the first major breakthroughs after the publication of the original paper proposing the adversarial training framework, and remains as one of the most cited papers in this newly created field. It has a relatively simple architecture (Figure 4.1), and proposed some guidelines that might help when training the DCGAN or similar convolutional models. This includes replacing pooling layers with strided convolutions for the discriminator, and fractionally-strided convolutions for the generator; using batch normalization [17], using the ReLU activation function [18] for the generator and the Leaky ReLU activation function [18] for the discriminator with a slope of 0.2. The paper also recommends using the Adam optimizer [19] and provides hyperparameters – a learning rate of 0.0002 and a β_1 momentum term of 0.5.

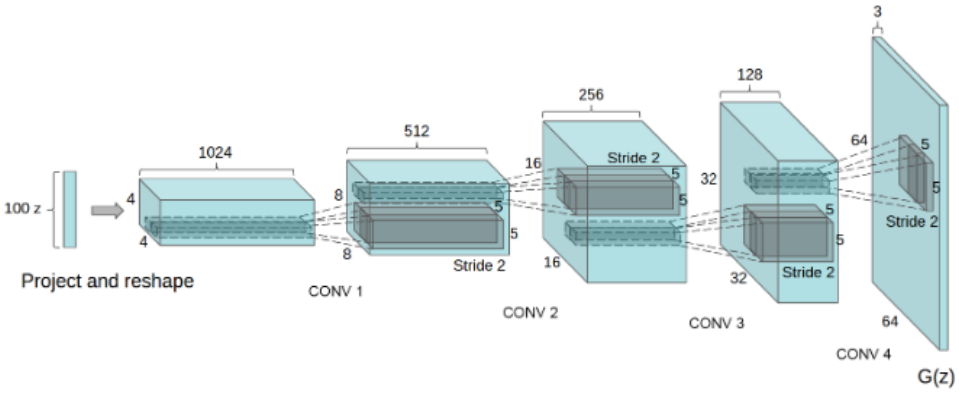


Figure 4.1: DCGAN generator architecture as seen in the 2016 paper [13].

4.1 CONVOLUTIONAL AND TRANSPOSE CONVOLUTIONAL LAYERS

Due to its success in practical applications, convolutional layers became one of the most important components in deep learning and computer vision in recent years. The research related to artificial neural networks based on how visual cortices in animals work date to the early 1980s [20], and they are perhaps the most notorious example of biologically-inspired concepts being successful in artificial intelligence. In the context of deep learning, convolutional layers actually apply an operation called cross-correlation. This layer has multiple filters or kernels, each usually having a square dimensionality. The kernel's weights are applied to the input using dot product, resulting in a single value. The kernel is then moved by a certain stride value, which usually has the same value for all dimensions, and this is applied until the entire input is covered. After this is done, an activation function and a pooling layer can be applied. By reducing dimensionality, pooling layers can reduce the impact of minor input changes such as rotations and shifts. A kernel can learn its weights by backpropagation, much like simpler neural networks containing only dense layers. In fact, a convolutional layer can be rewritten as a dense layer with shared weights.

Generative Adversarial Networks often require transforming lower-dimensional noise to higher dimensional data. For instance, a simple image generation application may use a latent vector of size 128, and output an image with size $64 \times 64 \times 3$. The way this can be accomplished is

by consecutive applications of upsampling layers. A rudimentary way of doing that is by having an upsampling layer that doubles the input size by performing a nearest-neighbour or bilinear interpolation. A more robust way of performing upsampling is by applying the convolution operation with **fractional stride** f, such as $f = 1/2$. This operation is sometimes called transpose convolution, or deconvolution, although the latter has its own different mathematical definition in the context of signal processing and some authors therefore argue that is technically incorrect. The way a stride with factor $f \leq 1$ can be achieved is by filling the input with zeroes and performing a convolution [21] with $f = 1$. On deep learning libraries, this operation is implemented through a layer that is typically called `Conv2DTranspose`.

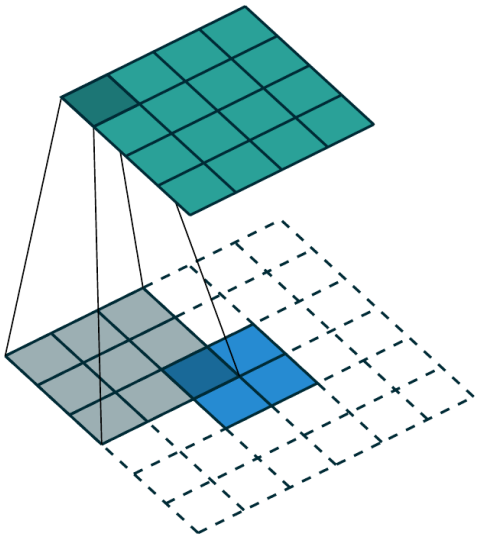


Figure 4.2: Visualization of a transpose convolution being applied to a 2×2 input (blue squares), resulting in a 4×4 output (green squares), by applying a standard convolution with zero padding (white squares). To perform a transpose convolution with a higher stride (for instance, 2), zero padding is also added in between the blue squares. Source: A guide to convolution arithmetic for deep learning by Dumoulin et al. [21].

4.2 THE CHECKERBOARD EFFECT

Successful convolutional neural network architectures used for image classification – such as Inception [22] and ResNet [23] – often have odd-numbered kernel sizes, such as 3×3 or 5×5. Emulating this way of building convolutional architectures to networks which use transpose convolutional layers may result in the production of image containing checkerboard patterns, as seen in Figure 4.3. Images containing this type of visual artifact may be less aesthetically pleasing when the pattern is clearly visible, but this kind of pattern also has implications to computer forensics. Even if the pattern is not easily perceptible to the human eye, some novel neural networks are able to distinguish between real and generated images by spotting artifacts such as the checkerboard pattern, or other patterns generated by convolutional networks.

A simple way of avoiding artifacts is to use convolutions where the kernel size is divisible by the stride. This apply to both the generator and discriminator – for the backward pass, the convolutions on the discriminator might cause the gradient to also produce the checkerboard pattern [24]. Another approach would be using standard upsampling instead of transpose convolutions. In this case, using nearest-neighbour is reported to provide better results than bilinear interpolation [24].

4.3 BATCH NORMALIZATION

A common guideline for training deep convolutional GANs is the use of batch normalization [17] in both generator and discriminator, except in the last layer of the generator and in the first layer of the discriminator [11]. Batch normalization improves the optimization process by subtracting the mean of a certain feature map x and then dividing it by its standard deviation, calculated over a minibatch. This may lead to unwanted side effects, particularly when using small batch sizes [11]. An alternative called virtual batch normalization [25] has been developed, but recent architectures seem to default to the standard batch normalization.

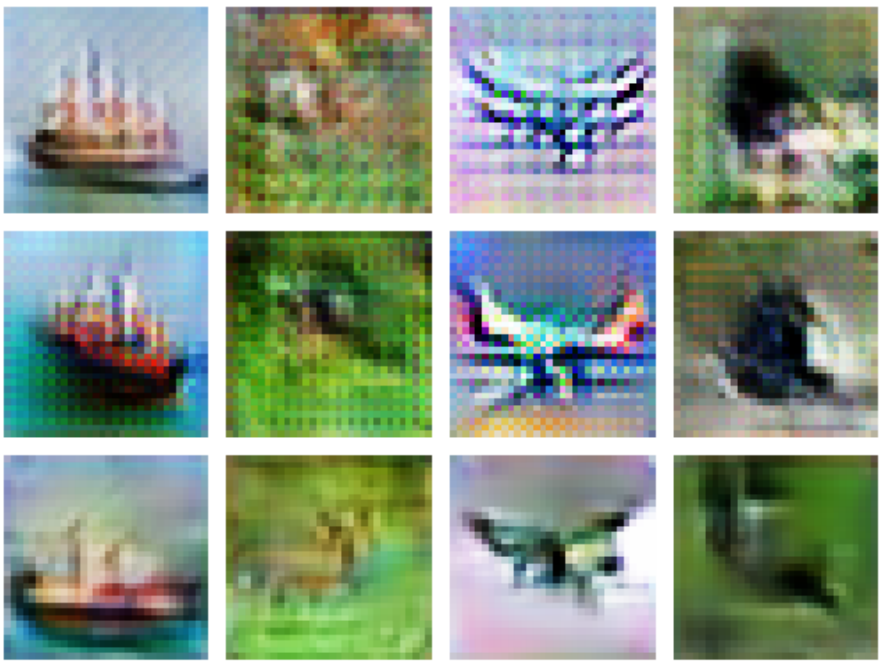


Figure 4.3: Checkerboard pattern comparison with different network architectures. Images on the first line were generated by a network with transpose convolutions in the last two layers and standard upsampling in all other layers; images on the second line were generated by a network with transpose convolutions on the last layers and standard upsampling on all other layers; images on the third line were generated by a network with only standard upsampling layers. The standard upsampling layers perform nearest-neighbour interpolation (effectively copying single pixels into squares containing the same pixels). Source: Odena et al. [24].

5 WASSERSTEIN GAN

Even though Deep Convolutional GANs proved to be very successful at the task of generating images, its architecture needed to be sometimes carefully crafted – together with choosing the proper optimizer and its hyperparameters – in order to assure some probability of near-convergence that would result in high quality visual samples. The standard GAN training algorithm itself and its loss function seemed to be a bit arbitrary and lacked a strong theoretical background. The original minimax formulation for the loss function, as previously mentioned in Chapter 3, is something similar to the Jensen-Shannon divergence [15], which can be seen in empirical results. In fact, to be more precise, the cost $J^{(D)}$ (Equation 3.3) for the discriminator can to shown to be a lower bound of $2D_{JS}(p_{data}|p_g) - 2\log 2$ [26]. Using this measure as a loss function often leads to unstable training, and the loss itself does not correlate with the sample quality – for instance, it is possible to have the same loss for the discriminator in case where the sample quality is considered good, and another where there is a mode collapse with non-sense samples [26]. The non-saturating cost function for the generator (Equation 3.8) has even less theoretical foundations, and the conclusion about it are reported to be the same as the saturating loss [26]. The Wasserstein GAN (WGAN) tries to solve these problems – its loss metric is reported to be correlated with sample quality and is a measure of the generator's convergence, while it also stabilizes the training process. A Wasserstein GAN is a GAN that implements the WGAN Algorithm, and not an architecture by itself.

5.1 WASSERSTEIN METRICS

A few key concepts are necessary in order to understand the WGAN Algorithm and its theoretical background. Many mathematical definitions in this section are simplified and have details omitted. The original paper proposing the WGAN [26] has some proofs regarding certain statements, while details for other definitions can be found in mathematical textbooks.

Definition 5.1.1 *The infimum of a subset S of a partially ordered set T, denoted* inf S*, is the greatest element $t \in T$ such that $t \leq s$, for all $s \in S$. If S is finite and real valued,* inf S *is the minimum element of S.*

Definition 5.1.2 *The supremum of a subset S of a partially ordered set T, denoted* sup S*, is the smallest element $t \in T$ such that $t \geq s$, for all $s \in S$. If S is finite and real valued,* inf S *is the maximum element of S.*

The Wasserstein distance is another distance metric (i.e. symmetric measure) defined over two probability measures. It is named after the Russian mathematician Leonid Wasserstein and forms the basis of the theory behind the WGAN Algorithm.

Definition 5.1.3 *Given two probability measures μ, ν and a metric $d(x, y)$ over a given metric space, the p^{th} Wasserstein distance $W_p(\mu, \nu)$ is defined as:*

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \mathbb{E}_{(x,y)\sim\gamma}[d(x, y)^p] \right)^{(1/p)}, \tag{5.1}$$

where $\Gamma(\mu, \nu)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are μ and ν, respectively.

If we have $p = 1$, this distance measure is called Wasserstein-1 or Earth Mover's Distance (EMD). Considering also our distributions p_{data}, p_g, and a metric $d(x, y)$, we can define the Wasserstein-1 distance which is used in the WGAN Algorithm.

Definition 5.1.4 *Given two probability measures p_{data}, p_g and a metric $d(x, y) = ||x - y||$ over a given metric space, the Wasserstein-1 or Earth Mover's distance $W(p_{data}, p_g)$ is defined as:*

$$W(P_{data}, P_g) = \inf_{\gamma \in \Gamma(p_{data}, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\, ||x - y||\,]\,, \tag{5.2}$$

where $\Gamma(p_{data}, p_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are p_{data} and p_g, respectively.

The denomination Earth Mover's Distance is a term that comes from transportation theory and the optimal transport problem. We can think of μ being a pile of earth that needs to be moved to the pile ν. A transport plan is then defined as $\gamma(x, y)$, which gives the amount of mass to be moved from x to y, and has an associated cost $c(x, y)$. If the cost is simply the distance between the two points, meaning that $c(x, y) = d(x, y) = ||x - y||$, we fall under the exact definition of the EM distance.

In the context of GANs, we can think of this distance as being the cost of the optimal plan for transporting p_{data} to p_g. Under certain assumptions, the Wasserstein-1 distance is continuous everywhere and differentiable almost everywhere [26], while D_{KL} and D_{JS} are not. This is theorized to produce better and more coherent gradients for the generator.

5.2 KANTOROVICH-RUBINSTEIN DUALITY

Calculating the Wasserstein-1 distance by the infimum definition is considered intractable. It is possible, however, to reformulate the Wasserstein-1 distance as the solution of a maximization problem over 1-Lipschitz functions.

Definition 5.2.1 (K-Lipschitz function) *Given two metric spaces (X, d_x) and (Y, d_y), a function $f : X \to Y$ is called K-Lipschitz if there is a constant $K \geq 0$ such that:*

$$\frac{d_y(f(x_1), f(x_2))}{d_x(x_1, x_2)} \leq K, \tag{5.3}$$

for all $x_1, x_2 \in X$ such that $x_1 \neq x_2$.

When taking a supremum of a set S considering the restrictions of Equation 5.3, the resulting expression is often referred to in publications, without any explanations whatsoever, as:

$$\sup_{||f||_L \leq K} S. \tag{5.4}$$

The reformulation from a minimization to a maximization problem is given by a result called the Kantorovich-Rubinstein duality. Its derivation can be found in mathematical textbooks.

Definition 5.2.2 (Kantorovich-Rubinstein duality) *Given that f is 1-Lipschitz, Equation 5.2 can be reformulated as:*

$$W(P_{data}, P_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim p_{data}}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]. \tag{5.5}$$

From the duality, it is possible to rewrite Equation 5.5 for a K-Lipschitz function f as [26]:

$$K \cdot W(P_{data}, P_g) = \sup_{||f||_L \leq K} \mathbb{E}_{x \sim p_{data}}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]. \tag{5.6}$$

This supremum is also considered intractable. Let us now consider the family of K-Lipschitz functions $\{f_w\}$ parametrized by w, which can be thought of as the weights of a neural network when approximating a function f_w through it. With this neural network setup, we could consider trying to solve the following problem:

$$\max_{w \in \Omega} \mathbb{E}_{x \sim p_{data}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]. \tag{5.7}$$

Considering a neural network implementation with weights w, where we have the **strong assumption** that w is lying in a compact space Ω (meaning it is closed and bounded), if we can imagine that our estimator would calculate $W(P_{data}, P_g)$ up to a multiplicative constant K, where K would be hidden within the network and of little concern to our objectives. Under this setup, f_w is a neural network that is called the **critic**.

If the compact space Ω assumption is held, then we would calculate the Wasserstein-1 distance by a factor K. Since f_w is a neural network, it is differentiable and we could calculate the gradient through the estimation f_w to also a factor K, which we could use to train our generator-critic setup. This assumption over every Ω is naively achieved in the original paper [26] by restricting the network's weights by a small factor, such as $w_i \in [-0.01, 0.01]$, for every $w_i \in \Omega$. Regarding the metric spaces X and Y, this method enforces that the function is K-Lipschitz for **any** metric space [27]. The paper [26] also has an extensive proof that the gradient of the Wasserstein-1 can be calculated through f_w.

Given a fixed generator, the more we train our critic f_w, the better our approximation of the Wasserstein-1 distance between p_{data} and p_g will be, giving us better gradients for the generator. This is why we usually train the critic more often than the generator, unlike what is usually done for the standard GAN Algorithm. The critic can be trained to optimality, can't saturate and converges to a linear function. When the critic is trained to optimality, a useful consequence is that mode collapse can't happen under this algorithm [26].

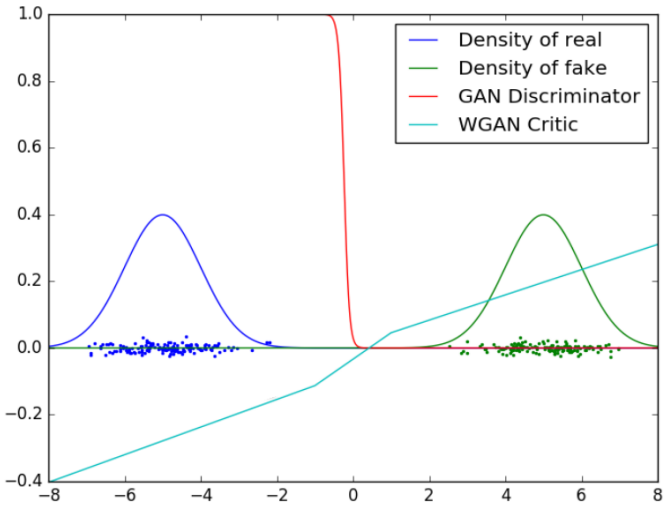


Figure 5.1: A comparison between the gradients of a minimax/saturating GAN and a WGAN when trying to learn two Gaussian distributions. The gradient of the saturating GAN reaches zero when the WGAN still provides good gradients. Source: Wasserstein GAN [26], adapted.

5.3 THE WGAN ALGORITHM

Given the propositions from section 5.2, we can now introduce the WGAN Algorithm. A downside for this algorithm is the fact that one can't use high learning rates or momentum based optimizers such Adam with $\beta_1 > 0$ anymore. The proposed solution is to use RMSProp with lower learning rates instead. The original suggested values are $\alpha = 0.00005$, $c = 0.01$ and $k = 5$.

Algorithm 5 WGAN Training Algorithm.

1: **while** generator has not converged **do**
2: **for** t = 1 **to** k **do**
3: **sample** $\{x^{(i)}\}_{i=1}^{m} \sim p_{data}$
4: **sample** $\{z^{(i)}\}_{i=1}^{m} \sim p_z$
5: $\text{grad}_c \leftarrow \nabla\big[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(G(z^{(i)}))\big]$
6: $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, \text{grad}_c)$
7: $w \leftarrow \text{clip}(w, -c, c)$
8: **end for**
9: **sample** $\{z^{(i)}\}_{i=1}^{m} \sim p_z$
10: $\text{grad}_g \leftarrow -\nabla\big[\frac{1}{m}\sum_{i=1}^{m} f_w(G(z^{(i)}))\big]$
11: $w_g \leftarrow w_g - \alpha \cdot \text{RMSProp}(w_g, \text{grad}_g)$
12: **end while**

5.4 WGAN IMPLEMENTATION

Keras starts to show its limitations when implementing the WGAN Algorithm. For weight clipping, it is possible to set a `kernel_constraint` parameter that clips the weights when adding layers for the discriminator. Since we are estimating f_w with a neural network, the critic should have a linear activation instead of a sigmoid activation that restricts it to the interval $[0, 1]$. Notice that now we are not directly using binary cross-entropy with log probabilities anymore, but we are instead using the arithmetic mean of the scores f_w.

While it is possible to set a custom loss function according to Lines 5 and 10 with Keras, we will focus on the TensorFlow 2.0 implementation. The reason for this is that further extensions of the WGAN Algorithm will require not only custom loss functions, but also direct gradient manipulation. Given the modified discriminator model, the implementation is very similar to Algorithm 4, although we would instead have a loop for training the discriminator/critic with k steps. The last modification would be the custom loss function for both generator and critic. A possible definition for the loss functions is shown in Algorithm 6.

Algorithm 6 WGAN TensorFlow 2.0 loss functions.

```
1: def critic_loss(x_real, x_fake):
2:     return -(tf.reduce_mean(x_real) - tf.reduce_mean(x_fake))
3: def generator_loss(x_fake):
4:     return -tf.reduce_mean(x_fake)
```

5.5 GRADIENT PENALTY

Even though the WGAN paper introduced a new theoretical framework for new research to be based upon, the actual results from applying the standard WGAN Algorithm with weight clipping

to enforce the Lipschitz constraint can be underwhelming. For instances, the samples produced are of lower quality than the ones produced by DCGAN, while also having considerably slower training. After the publication of the original paper, many variations on the DCGAN have been proposed, such as the Wasserstein GAN - Gradient Penalty Algorithm (WGAN-GP).

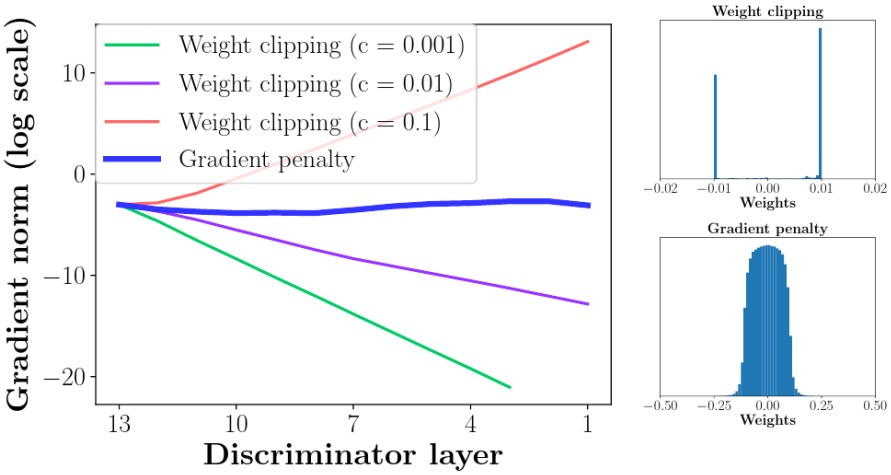


Figure 5.2: Comparison between different methods of enforcing the Lipschitz constrain for the critic on the Swiss Roll dataset. The gradients for lower numbered layers tend to explode, or sometimes vanish (with $c = 0.1$), while WGAN-GP stays normalized around 1. The figure also shows the distribution of the weights in the critic for each constraint method. Source: Improved Training of Wasserstein GANs [28].

The WGAN-GP Algorithm proposes a different way to enforce the Lipschitz constraint – given the fact that a differentiable function f_w is 1-Lipschitz if and only if it has gradients at most 1 everywhere [28], the constraint is enforced by restricting the gradient norm of the critic. In order to do that, we sample $\hat{x} \sim p_{\hat{x}}$, where $p_{\hat{x}}$ is a new distribution defined by an interpolation along straight lines of p_{data} and p_g. The justification for this is that enforcing norm at most 1 everywhere would be intractable [28], and empirically enforcing it along straight lines provides good results. Using the notation from the original paper [28], where the sign for the generator and the critic is swapped, our new loss function then becomes:

$$L = \mathbb{E}_{x \sim p_g}[f_w(x)] - \mathbb{E}_{x \sim p_{data}}[f_w(x)] + \lambda \cdot \mathbb{E}_{x \sim p_{\hat{x}}}[(||\nabla f_w(\hat{x})||_2 - 1)^2], \tag{5.8}$$

where λ is called the penalty coefficient and has a default suggested value of 10. This form of normalization, however, cannot be used together with batch normalization for the critic, since we are penalizing the loss on a per-input basis instead of the entire batch. One can either avoid using batch normalization altogether on the critic or using alternatives such as layer normalization [29]. Another advantage of the WGAN-GP method is the fact that it is possible to implement it with the Adam optimizer with momentum and a higher learning rate. Algorithm 7 shows the modified WGAN Algorithm with gradient penalty. Unlike previous architectures seen in this work, the gradient penalty algorithm is able to properly train very deep neural networks for both generator and discriminator, such as the 101-layer ResNet [23], a network architecture initially developed for image classification tasks.

Algorithm 7 WGAN-GP Algorithm with batch size m.

1: **while** generator has not converged **do**
2: **for** t = 1 **to** k **do**
3: **for** i = 1 **to** m **do**
4: **sample** $x \sim p_{data}$
5: **sample** $z \sim p_z$
6: **sample** $\epsilon \sim U(0, 1)$ from a uniform distribution
7: $x_g \leftarrow G(z)$
8: $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)x_g$
9: $L^{(i)} \leftarrow f_w(x_g) - f_w(x) + \lambda \cdot \mathbb{E}_{x \sim p_{\hat{x}}}[(||\nabla f_w(\hat{x})||_2 - 1)^2]$
10: **end for**
11: $w \leftarrow w + \alpha \cdot \text{Adam}(w, \nabla \frac{1}{m} \sum_{i=1}^{m} L^{(i)})$
12: **end for**
13: **sample** $\{z^{(i)}\}_{i=1}^{m} \sim p_z$
14: $w_g \leftarrow w_g - \alpha \cdot \text{Adam}(w_g, \nabla \left[\frac{1}{m} \sum_{i=1}^{m} f_w(G(z^{(i)}))\right])$
15: **end while**

5.6 WGAN-GP IMPLEMENTATION

The WGAN-GP Algorithm requires more sophisticated features from a deep learning library, since the loss $L^{(i)}$ is a function of a gradient for every i, which we then use to calculate another gradient, $\nabla \frac{1}{m} \sum_{i=1}^{m} L^{(i)}$. Once more, this can be done by manipulating the gradient tape in TensorFlow 2.0.

For the generator, there are no changes in the overridden function `train_step`, other than using the Adam optimizer instead of RMSProp. The critic part is also relatively straightforward – we initially calculate L_0 with the standard `critic_loss` function, setting L_0^{batch} = `critic_loss(x_real, x_fake)`, and then calculate the final loss by setting $L^{batch} = L_0^{batch} + \lambda \cdot$ `gradient_penalty(m, x_real, x_fake)`, where x_real and x_fake are the images in batch instead of the inner loop on Algorithm 7. This gradient penalty function is defined in Algorithm 8. The expression on Line 3 is equivalent to $\epsilon x + (1 - \epsilon)x_g$, but has less multiplications. The critic/"discriminator" function f_w is defined here as a call to `self.D`.

Algorithm 8 Gradient penalty function. (Python)

```
1: gradient_penalty(self, m, x_real, x_fake):
2:   e = tf.random.uniform([batch_size, 1, 1, 1], 0.0, 1.0)
3:   x_hat = x_real + e * (x_fake - x_real)
4:   with tf.GradientTape() as tape:
5:     tape.watch(x_hat)
6:     f = self.D(x_hat, training=True)
7:   grad = tape.gradient(f, [x_hat])[0]
8:   L2 = tf.sqrt(tf.reduce_sum(tf.square(grad), axis=[1, 2, 3]))
9:   return tf.reduce_mean(L2 - 1.0) ** 2
```

6 SPECTRAL NORMALIZATION GAN

Spectral Normalization is one of the most successful regularization methods proposed to help improving and stabilizing the training of Generative Adversarial Networks, and is built upon the Wasserstein GAN theoretical groundwork. Unlike the WGAN, the Spectral Normalization GAN (SNGAN) [30] does not need to calculate the extra gradient $\nabla f_w(\hat{x})$, and enables the usage of much higher learning rates. This contrasts heavily with both WGAN and WGAN-GP – the former requiring very slow training rates, and the latter being computationally heavy with the extra gradient requirement, which requires a forward pass and the actual gradient calculation. Published in 2018, its authors claimed that it was the first architecture capable of producing decent ImageNet [31] samples with only a single generator and discriminator. Extensions to the SNGAN involving the use of discriminator sampling [32] seem to remain the state of the art for smaller architectures. The general idea of such regularization methods is to enforce regularity conditions on the derivative of f_w, This particular method is based on earlier work from Yoshida and Miyato [33], and requires us to first introduce some definitions related to algebra and spectral theory.

6.1 MATRIX NORMS

The following are some definitions that will enable us to define the spectral norm.

Definition 6.1.1 *Given a linear transformation represented by the $n \times n$ matrix A, if there is a vector $v \in \mathbb{R}^n$ and a scalar λ such that $Av = \lambda v$, then λ is called the eigenvalue of A for a vector v, which is called an eigenvector.*

Definition 6.1.2 *Given a square matrix A, the singular values $\sigma_i(A)$ of A are given by the square roots of the non-negative eigenvalues of A^*A, where A^* denotes the conjugate transpose of A.*

We can now define the spectral norm, which is a matrix norm induced by the L2 norm.

Definition 6.1.3 *Given a matrix A, the spectral norm of A, denoted $||A||_2$ or $\sigma_{max}(A)$, is the maximum singular value of A.*

$$||A||_2 = \max_i \sigma_i(A) = \sqrt{\lambda_{max}(A^*A)} \tag{6.1}$$

$$= \max_{v:v\neq 0} \frac{||Av||_2}{||v||_2}. \tag{6.2}$$

$||A||_2$ is **alternatively denoted** as $\sigma(A)$. Equation 6.2 can be directly derived from Definition 6.1.1 by taking the L2 norm.

6.2 SPECTRAL NORMALIZATION AND LIPSCHITZ NORM

The spectral normalization method controls the Lipschitz constant by constraining the spectral norm for each layer [30], in a manner that is considerably more computationally efficient than the gradient penalty method. We can now define the Lipschitz norm.

Definition 6.2.1 (Lipschitz norm) *Consider a weight matrix W and a layer with linear activation function $g : \boldsymbol{h_{in}} \to \boldsymbol{h_{out}}$, such that $g(\boldsymbol{h}) = W\boldsymbol{h}$. The Lipschitz norm $||g||_{Lip}$ is defined as:*

$$||g||_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\boldsymbol{\nabla} g(\boldsymbol{h}))\,, \tag{6.3}$$

where the gradient with respect to the input can be rewritten as $\boldsymbol{\nabla} g(\boldsymbol{h}) = W$.

By rewriting the gradient, we have that $||g||_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\boldsymbol{\nabla} g(\boldsymbol{h})) = \sup_{\boldsymbol{h}} \sigma(W)$. Since it is not a function of $\boldsymbol{h}$, the last expression can also be rewritten, and we have that $||g||_{\text{Lip}} = \sigma(W)$.

If the Lipschitz norm for every activation function a_l for every layer l of the critic is equal to 1 (which is the case for ReLU and Leaky ReLU [30]), then by doing some algebra [30] with the layer notation for neural networks we can find an upper bound for the Lipschitz norm of the critic function f, resulting in the following bound:

$$||f||_{\text{Lip}} \leq \sigma(W^1)\, \sigma(W^2) \,...\, \sigma(W^L). \tag{6.4}$$

We can now define the spectral normalization operation. The weight matrix W is normalized by:

$$W_{\text{SN}}(W) = \frac{W}{\sigma(W)}. \tag{6.5}$$

By doing some algebra on Equations 6.4 and 6.5, it is possible to conclude that $\sigma(W_{\text{SN}}(W)) = 1$, which give us an upper bound for our critic: $||f||_{\text{Lip}} \leq 1$. In practice, this form of normalization is computationally lightweight and performs better than both weight clipping and gradient penalty.

6.3 IMPLEMENTATION

Since we are simply normalizing the weight matrix of the discriminator, spectral normalization only requires manipulating a weight matrix, which in practice can be done by setting a custom layer that manipulates those weights. Calculating $\sigma(W)$ can be computationally expensive when using naïve methods, but it can be approximated by algorithms such as the power iteration method [33]. This is a numerical method and is out of the scope of this work, but the authors provide an implementation of the power iteration method and the Python definition of a spectral normalization layer which applies such method. More recently, this method has been included in popular deep learning libraries, such as the `torch.nn.utils.spectral_norm` operation on PyTorch and the `tfa.layers.SpectralNormalization` layer on TensorFlow.

6.4 HINGE LOSS

An alternative loss function, called hinge loss, was reported by Miyato et al. [30] to perform well with the SNGAN – in fact, it outperformed all other losses under the settings of the paper. Its concept is borrowed from the maximum margin classifier literature and the Support Vector Machine (SVM), where a classifier is trained to maximize the geometric margin between classes. For Generative Adversarial Networks, it is also possible train the discriminator to maximize the margin between (implicit) classes by using the hinge loss function. This is an oversimplification and is described in this way only for intuitive purposes, but theoretical details can be found in the Geometric GAN paper by Lim et al. [34]. The hinge loss for the generator remains the non-saturating loss (Equation 3.8), and the discriminator hinge loss for a discriminator with **linear output** is defined as:

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\,\mathbb{E}_{x\sim p_{data}}\,[\min(0, -1 + D(\boldsymbol{x}))] - \mathbb{E}_{z\sim p_z}\,[\min(0, -1 - D(G(\boldsymbol{z})))]. \tag{6.6}$$

7 QUANTITATIVE ANALYSIS

Unlike other learning tasks such as training an image classifier, where there are available correct labels and the loss function is calculated by some measure of the deviation of the output and the expected labels, generative models such as GANs and VAEs do not inherently have a loss function that is correlated with the output's subjective quality. As we have seen in Chapter 5, the Wasserstein GAN partly solves this problem, but its loss function is exclusive to this kind of architecture and therefore cannot be used to compare the WGAN to other types of generative models. Since this area of research is focused on improving the quality of the samples, a way of measuring how different generative models compare to each other is vital to the advancement of the field. Many different measures have been proposed in the last few years, such as the Inception Score (IS) [25], the Fréchet Inception Distance (FID) [35], the Geometry Score [36], the Sliced Wasserstein Distance (SWD) [37], the Fréchet Joint Distance, Multi-scale structural similarity (MS-SSIM) [38], precision and recall methods and many others. Of these, only the IS and FID seem to have stood the test of time, at least in the context of image generation. As of this writing, research papers still usually only include both IS and FID measures when reporting results, and are for this reason the subject of this chapter.

Other than subjective quality, generative models should ideally have a very varied outputs. Poorly trained GANs, for instance, may suffer from a problem called **mode collapse**, where the generator outputs images very similar to each other that are then classified as real by the discriminator, thus making the generator "win" the minimax game in a manner that produces poorly performing generators. An optimal generator measure should therefore combine both sample quality and sample variety.

7.1 INCEPTION SCORE

The Inception Score [25] is calculated using the outputs of the Inception-v3 network [22], which is an image classifier trained over the ImageNet [31] dataset, outputting probabilities for 1000 classes. Its concept is relatively simple – for a given input x and class label y:

1. We would like the outputs $p(y|x)$ of the Inception-v3 network to have low entropy (i.e. high class probability, meaning that the network is sure that there is an object from a certain class in the input).
2. We would like the marginal $p(y) = \int p(y|x = G(z))\, dz$ to have high entropy for every y (i.e. low class probability for every y, meaning that our generator won't be biased to output images from a certain class with high probability).

Both of these requirements can be satisfied by the Inception Distance metric. The authors exponentiate the result to make the results easier to compare, which is the standard practice when calculating the IS in recent research papers.

Definition 8.1.1 (Inception Score) *For an Inception-v3 network trained over the ImageNet dataset with 1000 classes and outputs following a distribution p, the Inception Score IS for a generator G with distribution p_g is defined as:*

$$\mathrm{IS}(G) = \exp\Big(\mathbb{E}_{x\sim p_g} D_{\mathrm{KL}}(p(y|x) \mid p(y)) \Big). \tag{7.1}$$

Higher IS measures therefore mean that the distance between $p(y|x)$ and the marginal $p(y)$ is high, which coherently satisfies requirements 1 and 2.

Implementing the Inception Score is very straightforward – deep learning libraries such as Keras already have the Inception-v3 network built-in, which can be used to process the inputs. The actual score can be then calculated by using Equation 2.7. A modified version of this equation is commonly used in practice to avoid numerical instability when calculating the logarithm of probabilities close to zero. Although we have $p(y|x)$ directly from the output of the Inception-v3 network, we must find a way to calculate $p(y)$. This is usually done by creating an estimator $\hat{p}(y)$ over N samples [39]. We can define the estimator as:

$$\hat{p}(y) = \frac{1}{N} \sum_{i=1}^{N} p(y|x^{(i)}) \,, \tag{7.2}$$

and the actual expression for the Inception Score which is used in practice is defined as:

$$\mathrm{IS}(G) = \exp\left(\frac{1}{N} \sum_{i=1}^{N} D_{\mathrm{KL}}(\, p(y|x^{(i)}) \mid \hat{p}(y) \,) \right). \tag{7.3}$$

The Inception Score has received much criticism since its initial publication. The main criticism concerns using the classification score of an ImageNet-trained network to images of a generator which was trained on other datasets. Regardless of that, this score seems reliable enough to still be used. Papers related to deep generative modeling often still publish IS measures even when the generator is trained using other datasets. Another important fact to mention is that the IS was found to be correlated with actual scores from human annotators [25]. From its definition, we can also conclude that a **higher IS score represents a better score**.

7.2 FRÉCHET INCEPTION DISTANCE

The Fréchet Inception Distance is arguably the current research community standard generative modeling metric. It also uses the Inception classifier for the metric calculation, but with a relatively more sophisticated approach. Unlike the IS, the FID uses the statistics of the real dataset which was used to train the generative model, although the Inception network itself is usually pre-trained on ImageNet. The introduction of artifacts and noise to input images can heavily alter the resulting FID, something which often does not happen with the IS.

The Fréchet Inception Distance has a definition based on a measure called the Fréchet or Wasserstein-2 distance. More specifically, we use the Fréchet distance definition for two multivariate Gaussian distributions, and we assume that the pooling activation results of the last layer prior to the class probability outputs of the Inception-v3 network follow a multivariate Gaussian distribution. The reasons for this assumption and more details about this method can be found in the original paper [35]. This last pooling layers has dimension 2048 and can be interpreted as extracted features from the input.

Definition 8.2.1 (Fréchet Inception Distance for two multivariate Gaussians) *Given two multivariate Gaussian distributions $\mathcal{N}(\mu_{data}, \Sigma_{data})$ and $\mathcal{N}(\mu_g, \Sigma_g)$, the Fréchet Inception Distance is defined as:*

$$d^2 = ||\mu_{data} - \mu_g||_2^2 + \mathrm{tr}(\Sigma_{data} + \Sigma_g - 2 \cdot \sqrt{\Sigma_{data} \times \Sigma_g} \,). \tag{7.4}$$

To make this definition more clear, since we are considering multivariate distributions: Σ is a covariance or auto-covariance matrix, which is a square matrix that is calculated in a pair-wise manner for a given input vector; the trace tr of a square matrix is defined as the sum of the elements in the main diagonal of this matrix; and the notation $\sqrt{\Sigma_{data} \times \Sigma_g}$ refers to the matrix square root operation, which can be calculated by many different methods which are implemented in linear algebra packages – for instance, on SciPy this can be accessed by calling `scipy.linalg.sqrtm`.

In practice, when implementing the FID a small value may be added to the diagonal of the covariance matrices when calculating the matrix square root, in order to avoid numerical problems. Calculating the square root matrix also may produce imaginary components, which are then discarded and only the real part is kept. The FID was also found to correlate with human judgment [35]. Unlike the IS, the FID is not a score but a distance metric. The original paper [35] transforms the IS into a distance in order to compare them, but the important to have in mind is that a **lower FID represents a better result**.

7.3 LIMITATIONS OF INCEPTION METRICS

The IS and FID measures that are published in papers practically always use an Inception v3 network pre-trained on the ImageNet dataset – more specifically, this a dataset that contain many real-world photos which are divided in 1000 classes. For generative models, a popular dataset for benchmarking is the CIFAR10 dataset, which contain smaller (32×32) real-world photos which are subdivided in only 10 classes (such as cat and ship). Since both datasets are related to real-world objects, there is a considerable class overlap, but training a generative model on CIFAR10 and then calculating the IS/FID through an ImageNet-based Inception network may produce measures that are not ideal. A detailed discussion about this topic (for the Inception Score) can be found on a paper by Barratt et al. [39].

Calculating the Inception Score based on samples from a generator that has been trained on a dataset that is radically different from ImageNet (such as CelebA, which contains *only* portraits of celebrities) tends to produce lower scores when compared to CIFAR10 scores. This most likely happens due to the fact that only one class has a high marginal probability. Regardless of that, the IS has been often used in research papers with datasets such as CelebA and the LSUN Bedroom dataset (containing *only* photos of bedrooms). Chapter 9 includes a practical analysis of the IS calculated from CelebA-trained generators.

The Fréchet Inception Distance has considerably less problems when used with generators trained on datasets other than ImageNet. Since the statistics of an internal layer of the Inception network are used (instead of the output class probabilities in the IS), together with the fact that in this case we use both generated and real samples to calculate the measure, the Fréchet distance is arguably a better metric for comparing generative models. Even if the inputs belong to a distribution that radically differs from ImageNet, one can assume that the Inception network learns useful internal representations that are able to extract useful features which can then by used to calculate the metric.

8 LARGE-SCALE GENERATIVE MODELING

The research mentioned in the previous chapters of this work is concerned with the problem of optimizing the generator to produce higher quality samples, while usually having the assumption that the architecture consists of a single generator and discriminator. Even if GANs are considered to have unstable and slow training, it is possible to train relatively recent architectures on consumer-grade GPUs in a few hours when using low-resolution datasets, such as CelebA downsampled to 64×64. However, in recent years there has been increased interest in the possibility of developing larger architectures which can better use computational resources, which often involve a higher amount of VRAM and parallelism between multiple GPUs. While larger scale architectures have been tremendously successful and are still the state-of-the-art in terms of output quality, this should be considered as a sub-field of deep generative modeling research. As we will see, larger scale architectures still receive the benefits of more solid research involving the study of simpler models. Since it is practically impossible to train such architectures with consumer-grade hardware, these architectures are only mentioned in this chapter for completion, and are omitted from experimental runs in Chapter 9.

8.1 PROGRESSIVE GROWING GAN

The Progressive Growing GAN (sometimes referred to as ProGAN), published in late 2017 by a team at NVIDIA [40], is the first architecture that was able to produce shockingly realistic 1024×1024 images. Since all previous available datasets had a relatively low resolution, this paper also introduced the new CelebA-HQ dataset. The architecture consists of a mirrored generator and discriminator. The network starts with low-resolution images, with higher-resolution layers being smoothly added during training. By doing this, training times are considerably reduced, which is of extreme importance when considering high resolution datasets. Figure 8.1 shows an overview of the training process and a few generated samples.

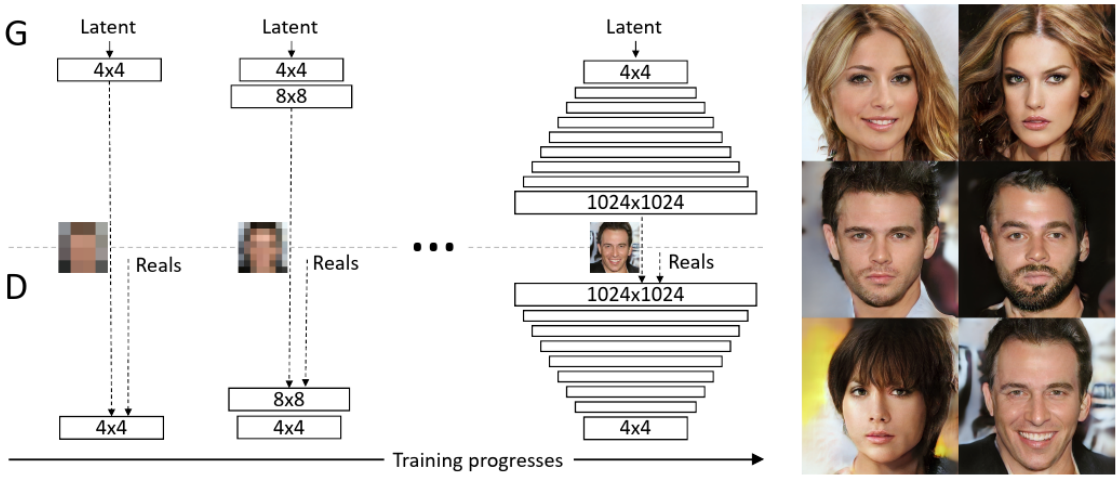


Figure 8.1: Progressive GAN architecture overview – layers are added to both generator and discriminator during training. Generated samples on the right are downsampled from the 1024×1024 originals. Source: NVIDIA [40].

Instead of suddenly being added during training, new layers are added by a linearly growing factor α, increasing from 0 to 1, while at the same time the previous set of layers receive a factor of $1 - \alpha$.

The paper shows that this method can be applied to different GAN losses, and presents results with both WGAN-GP and the Least-Squares GAN (LSGAN), the latter not being previously mentioned in this work due to its underwhelming results, while also being less stable when used in the progressive growing framework [40].

Applying the Progressive Growing GAN architecture to lower-resolution datasets also provides us with interesting results. Figure 8.2 shows a comparison between generated samples coming from networks trained on the CelebA-HQ dataset (first row) and samples from networks trained on the CelebA (second row). The samples in Figure 8.2 (d) come from a converged ProGAN setup, which still contain many artifacts and contrasts heavily with the clean and high-resolution samples generated by the same architecture as shown in Fig. 8.2 (a), when trained using CelebA-HQ. The authors state that the CelebA dataset itself contains many artifacts (such as aliasing, compression and blur) [40], which trained generators seem to reproduce. This result is particularly interesting because it clearly shows that even successful large-scale architectures are unable to provide good quality samples from popular low-resolution datasets that were used by the research community for many years.

Figure 8.2: Comparison between generated samples from different datasets and architectures – (a) ProGAN trained on the CelebA-HQ dataset (1024×1024 images) and samples from (b) WGAN-GP trained on CelebA and (c) (d) ProGAN trained on CelebA (downsampled to 128×128 from 178×218). Source: Karras et al. [40], adapted.

8.2 BIG GAN

The BigGAN, published in late 2018 by a team at DeepMind [41], improves previous state-of-the-art results by using a larger architecture with larger batch sizes. Its architecture in based

on the Self Attention GAN (SAGAN) [42], and also uses Spectral Normalization (Chapter 6). It was trained on a Google TPU v3 Pod, which has considerably more computing power than the previously used 8 × Tesla V100 for the ProGAN, and is an empirical study focused on experimenting with extremely large architectures in practice. By the time of its publication, it had improved both of the previous state-of-the-art IS and FID (held by ProGAN). One interesting finding by the researchers was the fact that increasing the batch size by a factor of 8 increased the previous state-of-the-art IS by 46% (note that, as previously mentioned in Chapter 3, this comparison can be considered rather unfair since ProGAN had an unsupervised architecture). Using progressive growing in the BigGAN was deemed by its authors as unnecessary. The paper provides training details and challenges, but seems to be more focused on scaling problems rather than providing new paradigms. Each experiment for the BigGAN is reported to consume up to 4915 kWh. More recently, a new and improved implementation called BigGAN-PyTorch was released, which allegedly can be run on 4 to 8 GPUs.

8.3 STYLE GAN

The StyleGAN, initially published in late 2018, is an improvement on the ProGAN architecture developed by the same team at NVIDIA that had initially introduced the progressive growing framework. The first version of the StyleGAN had initially delivered state-of-the-art results. StyleGAN v2, release in late 2019, has improved these results even further, currently having the best published IS and FID scores for unsupervised learning.

StyleGAN uses the ProGAN as the baseline, and borrows concepts from the style transferring literature in order to improve it. Instead of having the latent codes $\boldsymbol{z} \in \mathcal{Z}$ directly as inputs to the generator, the StyleGAN uses a non-linear mapping $f : \mathcal{Z} \rightarrow \mathcal{W}$, where the outputs $\boldsymbol{w} \in \mathcal{W}$ are produced by running $\boldsymbol{z}$ through a 8-layer MLP. The architecture is still progressive growing, but we now have have two convolutions for every resolution level – with the exception of the first level, which has a learned constant as input instead of a convolution. Before the second convolution is applied, the input goes through Adaptive Instance Normalization (AdaIN) [43]. This enables the addition of styles on arbitrary levels of detail – copying styles from lower resolutions can lead to the change of higher level details (such hair style and eyeglasses), while copying styles from higher resolutions can lead to the change of lower level details (mostly the color scheme and minor lightning details). As Figure 8.3 shows, the inputs $\boldsymbol{w}$ first go through an affine mapping A, which can be implemented as a fully connected network, and outputs values y_s and y_b (scaling and bias factors, respectively). The AdaIN operation first normalizes the input $\boldsymbol{x}$, with the following operation being performed:

$$\text{AdaIN}(\boldsymbol{x}, y_s, y_b) = y_s \frac{\boldsymbol{x} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})} + y_b. \tag{8.1}$$

Unlike batch normalization, $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$ are calculated for each channel and sample. Also shown in Figure 8.3 is the addition of random noise, which is fed through another learned affine mapping B. This allows the generation of images with slightly different minor variations, such as positioning of the exact placement of hairs and freckles, a feature which can be better understood by watching the videos released by NVIDIA.

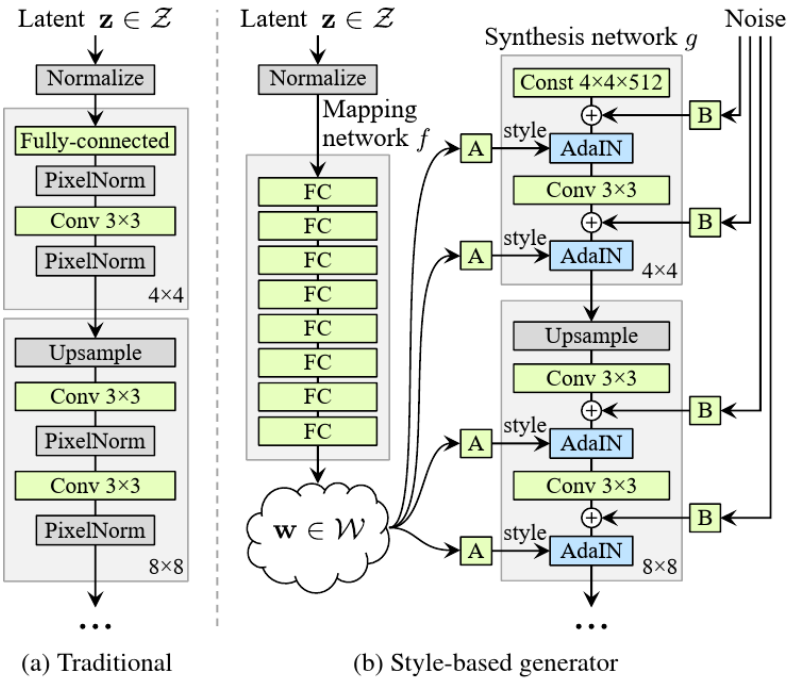


Figure 8.3: The traditional generator (a) refers to the standard Progressive Growing GAN generator, while (b) shows the novel StyleGAN generator. The affine maps A and B are implemented as neural networks, and the $4 \times 4 \times 512$ constant, which takes the place of $z \in \mathcal{Z}$ in the original ProGAN, can be learned through backpropagation. Source: NVIDIA [44].

The architecture was tested with both WGAN-GP and nonsaturating losses, the latter found to be better for longer training times when used together with R1 regularization [45]. Considering that this architecture does not use z directly as input, the researchers also found a trade-off between image variety and quality called the truncation trick. Areas with low density in the training data can be difficult to learn, and by truncating values in $\mathcal{W}$ it is possible to ignore such rarer data points and produce images that are higher in visual quality. Figure 8.5 shows an example of a generated sample containing artifacts related to areas with low probability in the training dataset.

StyleGAN v2, which is not discussed in detail here, further improves the previous architecture by removing normalization artifacts which could be seen in generated samples from its previous iteration. The NVIDIA team also compared and proposed a new StyleGAN architecture without progressive growing, which outperformed the previous formulations and currently hold the best FID reported by the research community. Figure 8.4 shows a few generated samples from this architecture.

Figure 8.4: Samples generated by StyleGAN v2. The network is able to produce samples that are very varied (they show different higher level details such as gender, hair length, skin color, presence or absence of earrings and sunglasses), while still being able to produce minor details such as age marks on the skin. Source: random samples from `https://thispersondoesnotexist.com/`, downsampled from the 1024×1024 originals.

Figure 8.5: Left image (generated by StyleGAN v2) – an example of visual artifacts related to points of low probability in the training dataset, which can be avoided by the truncation trick; right image (a real photo) – sample from the training data used to train this network. A possible explanation for this image is the fact that the training dataset, called Flickr-Faces-HQ (FFHQ), has only a few images containing people with uncommon makeup applications, which is not enough for the generator to learn coherent representations. Source: the generated image is a random sample from `https://thispersondoesnotexist.com/`.

9 EXPERIMENTAL STUDIES

This chapter introduces the results achieved by running experiments with multiple GAN architectures, which are then quantified by the metrics introduced in Chapter 7. Before discussing the results from the generative networks, some details about the implementation and variability of the Inception Score (IS) and Fréchet Inception Distance (FID) are discussed. The architectures analyzed in this section are the DCGAN, a Wasserstein GAN with Gradient Penalty, and a ResNet-based Spectral Normalization GAN. The original weight clipping version of the WGAN was left out of the experimental studies due to its very slow training process and underwhelming results, but the theoretical guarantees and advantages are still present in the gradient penalty version. In this chapter we will also discuss the correlation between network losses and our chosen quantitative metrics. All experiments were ran on a NVIDIA GTX 1060 GPU with 6GB VRAM, and the training process took between 20 and 30 hours for each model.

9.1 INCEPTION METRICS IN PRACTICE

Other than the issues that are considered limitations by design of both IS and FID measures (see Chapter 7, Section 7.3), many other issues are found in practice. For instance, different weights of the pre-trained Inception v3 network (used for both IS and FID) may result in different scores. This is particularly problematic when using implementations made with different deep learning libraries. The original IS implementation, made by OpenAI, requires the outdated TensorFlow 1.x, which is dependent on many older libraries. For perhaps this reason, some researchers use a PyTorch IS implementation, but this has been reported to produce considerably different IS scores when compared to the original. For this work, both TensorFlow 1.x and PyTorch implementations were compared, and considerably different scores were indeed observed, although this is only a minor issue for the FID measure. A huge variability in this score was also found when calculating the IS with different numbers of samples. Table 9.1 shows IS reference results for both CelebA and CIFAR10 with different numbers of samples.

	Inception Score				
	CelebA			CIFAR10	
Library	N = 100	1,000	10,000	N = 10,000	50,000
TensorFlow	2.87 ± 0.55	4.22 ± 0.30	4.48 ± 0.11	10.99 ± 0.32	11.23 ± 0.09
PyTorch	2.86 ± 0.58	3.93 ± 0.23	4.04 ± 0.21	9.67 ± 0.14	9.67 ± 0.14

Table 9.1: Inception Score measures for subsets of the CelebA dataset containing N = {100, 1000, 10000} images and the CIFAR10 training dataset with N = {10000, 50000} images. All experiments were done with $n_{\text{splits}} = 10$ and shuffled inputs. The TensorFlow scores were obtained with the original OpenAI implementation with a custom wrapper, while the PyTorch scores were obtained with a custom implementation.

The Fréchet Inception Distance was found to be less variable across implementations with different libraries (Table 9.2), but it was noted that there is still a huge variability in final scores which is also related to the chosen number of samples for the calculation of the measure.

	Fréchet Inception Distance			
	DCGAN (epoch 25) ⟷ CelebA			
Library	N = 100	1,000	5,000	50,000
TensorFlow	203.96	138.81	115.23	110.82
PyTorch	189.94	138.79	115.60	111.11

Table 9.2: Fréchet Inception Distance measures between a trained DCGAN at epoch 25 and the reference training dataset for subsets of N = {100, 1000, 5000, 50000} images. All experiments were done with $n_{\text{splits}} = 10$. The TensorFlow measures were obtained with the original implementation by Heusel et al. [35], while the PyTorch measures were obtained with an implementation by M. Seitzer [46].

An additional practical problem related to computing Inception-based metrics is worth mentioning – before feeding the images in the Inception v3 network, we must consider that it expects inputs of dimension 299×299. The TensorFlow pre-trained Inception v3 network has a resize layer, by default, before the input layer, which applies bilinear filtering. For PyTorch implementations, a custom resize function is also applied with bilinear filtering. This is not an issue for smaller architectures, but it is interesting to notice that samples produced by large-scale architectures have some information discarded when calculating the FID.

For further experiments in this section, the OpenAI TensorFlow implementation is used for calculating the IS, and the PyTorch implementation is used for calculating the FID due to its considerably lower running times. All measurements in the next sections are made with $n_{\text{splits}} = 10$ and $N = 1000$. This number of samples was chosen due to its low required computing times while being somewhat close to convergence, as can be seen in Tables 9.1 and 9.2. Calculating the IS and FID measures for each epoch (1000 generated samples) took approximately 30 seconds for each method - the PyTorch FID implementation being considerably more efficient than the TensorFlow IS implementation, even thought it requires processing twice the number of samples plus additional matrix operations. The running time for the FID calculation increases considerably with a higher number of samples.

9.2 DCGAN

For the first experiment, a DCGAN with architecture similar to Figure 4.1 is trained on the CelebA dataset, containing 202,599 portraits of celebrities. It is important to note that there is no standard DCGAN, and the original paper [13] proposes many different configurations regarding batch normalization and activation functions for both inner layers and the generator's output. For this experiment, LeakyReLU was used in both discriminator and generator with a slope of 0.2. The generator uses transpose convolutional layers instead of standard upsampling, and the generator's last convolution has a tanh activation with images normalized to $[-1, 1]$. The chosen optimizer was Adam with learning rate $\alpha = 0.0002$ and exponential decay rates $\beta_1 = 0.5$, $\beta_2 = 0.999$, with a batch size of 32. For this experiment, the generator outputs 64×64 images and the discriminator has 64×64 images as input, with the real ones being resized to this size from the original 178×218 by applying bilinear filtering.

Figure 9.2 shows the FID progression throughout 26 epochs for the DCGAN. The lowest measured FID occurs at epoch 13, with a measure of 111.92. The IS (Figure 9.1) also was calculated for each epoch, but it doesn't seem to correlate with the subjective image quality when used with a CelebA generator, while the FID is clearly correlated to it by the author's subjective judgement. In fact, the calculated correlation coefficient between IS and FID is only $r = -0.0877$ (Pearson), indicating almost no correlation between the two quantitative methods. For the DCGAN, the discriminator loss (defined here and in further experiments as the average

loss since epoch 0) also showed only a very weak correlation ($r = 0.2278$) between discriminator loss and the FID measure (which, in this case, is the only reliable image quality measure).

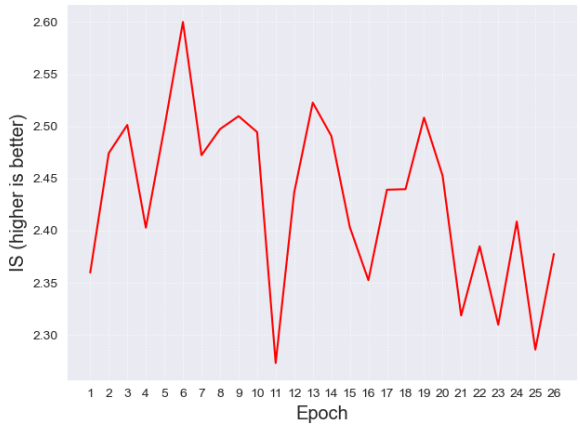


Figure 9.1: IS measure progression for a DCGAN trained on CelebA for 26 epochs.

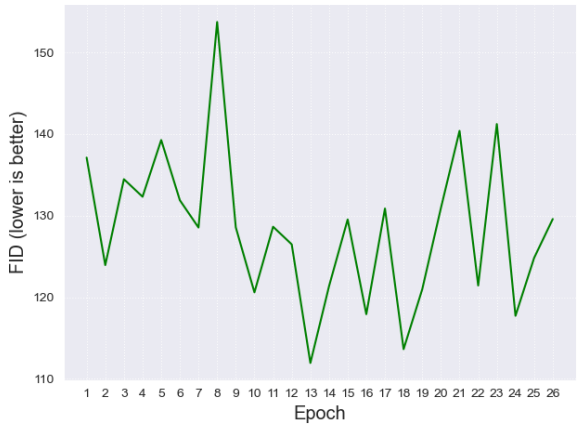


Figure 9.2: FID measure progression for a DCGAN trained on CelebA for 26 epochs.

This architecture was trained for 26 epochs, with the best results being achieved at epoch 13. The training was stopped after noticing that was diverging from the previous best results, although it is possible that the network could eventually improve with further training.

Figure 9.3 shows samples generated after the first epoch (row 1), the worst epoch (row 2, epoch 8) and the best epoch (rows 3 and 4). The DCGAN already starts to produce discernible results after the first epoch. The worst epoch contains the highest amount of artifacts, which indicates that the FID can correctly assign bad scores for images containing them. The first two images (left to right) from the last row shows an indication of minor mode collapse, where the generator learns to produce similar images to fool the discriminator, which may be related to the very small chosen batch size.

Figure 9.3: Samples generated by the DCGAN – First row: epoch 1 (FID = 137.11); Second row: epoch 8 (FID = 153.74); Third and fourth rows: epoch 13 (FID = 111.92).

9.3 WGAN-GP

For the second experiment a WGAN using gradient penalty was trained, with an architecture similar to the DCGAN from Section 9.2. Changes include adding batch normalization to and standard ReLU for the generator, and using layer normalization on the discriminator/critic. The generator's output is still activated by a tanh convolution. The chosen optimizer is still Adam, this time with a more aggressive learning rate of $\alpha = 0.0004$, and exponential decay rates $\beta_1 = 0$, $\beta_2 = 0.9$, the last two terms being suggested in the original paper [28]. The chosen batch size was 128, and the used gradient penalty coefficient was $\lambda = 10$ (see Algorithm 7). Another change is that we now update the discriminator/critic for $n_{\text{critic}} = 5$ times for each update to the generator, since for Wasserstein GANs there are benefits in training the critic more frequently (see Chapter 5 for the motivation behind this reasoning).

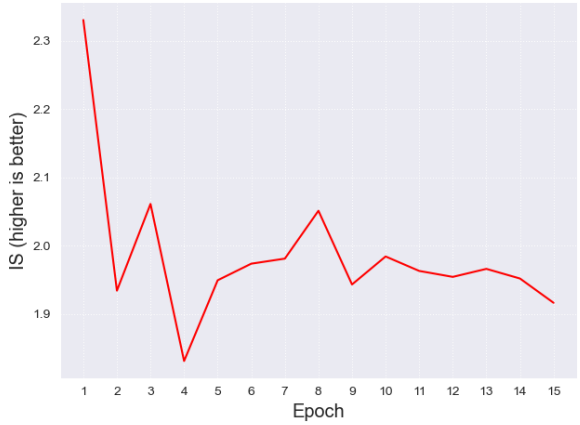


Figure 9.4: IS measure progression for a WGAN with gradient penalty trained on CelebA for 15 epochs.

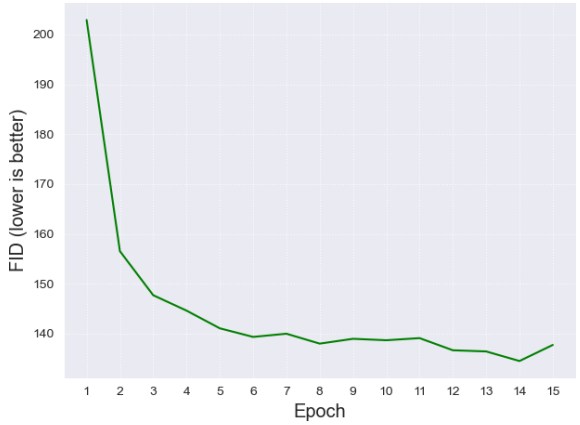


Figure 9.5: FID measure progression for a WGAN with gradient penalty trained on CelebA for 15 epochs.

The measured Inception Scores (Figure 9.4) for this setup are once again not correlated to subjective image quality (in fact, there is a moderate inverse correlation), while the FID still remains an accurate measure of subjective quality. It is quite clear from the results shown in Figure 9.5 that Wasserstein GANs have much more stable training. However, this comes at a cost, with the training of the WGAN under these parameters taking twice as much time for each epoch as the DCGAN presented in the Section 9.2, despite having a higher learning rate parameter.

Although the WGAN-GP was trained only for 15 epochs, it is likely that the image quality would further improve with more training. Another interesting result is the correlation of $r = -0.9112$ between the FID measure and the discriminator/critic's loss. Figures 9.6 and 9.7 show the critic loss progression and its high correlation with FID measures, respectively.

Figure 9.8 shows the samples generated after the first epoch (row 1, which is also the worst epoch), epoch 2 (row 2, which is the second worst epoch) and the best epoch (rows 3 and 4, epoch 14).

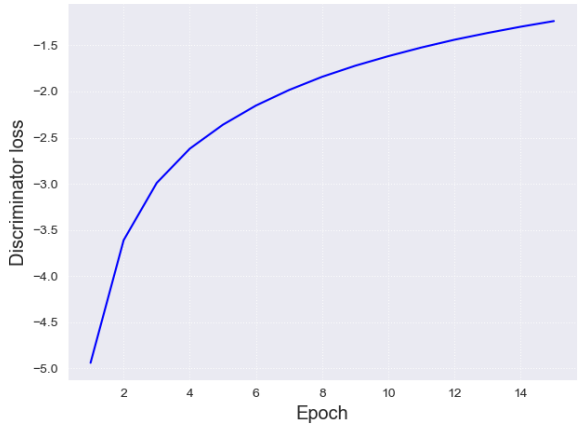


Figure 9.6: WGAN-GP discriminator/critic loss during training. The critic loss slowly starts converging into a linear function, as seen in Figure 5.1 from Chapter 5, which was reported in the original paper.

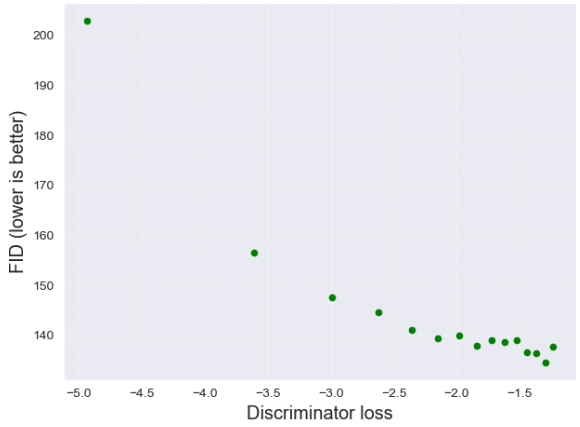


Figure 9.7: Scatter plot showing the correlation between increasing discriminator/critic loss and improving (lowering) FID measures ($r = -0.9112$).

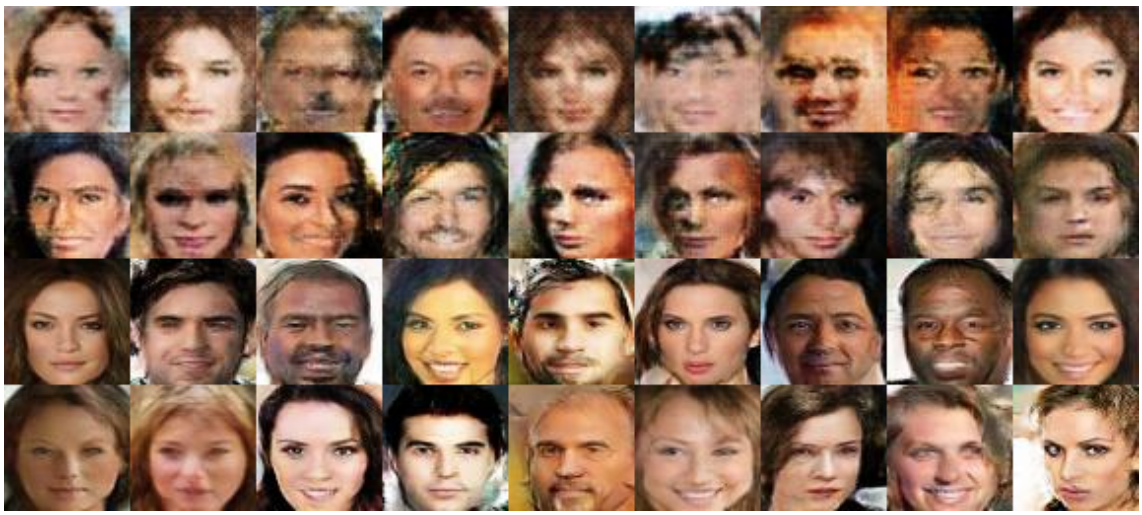


Figure 9.8: Samples generated by the WGAN-GP – First row: epoch 1 (FID = 202.91); Second row: epoch 2 (FID = 156.53); Third and fourth rows: epoch 14 (FID = 134.50). *Note: this experiment was the first experiment in this work by order of execution, and contained a left-over crop transform before applying bilinear filtering to resize the image to* 64×64. *This slightly alters the dataset and provides different looking pictures, but this should not lead to changes in the conclusions coming from the experiments contained in this section. This might have worsened the low correlation between IS and FID measures in this section but, as we can see the previous section, the DCGAN already shows a situation with low correlation even without cropping.*

9.4 SNGAN

For the last three experiments, an SNGAN was trained with a ResNet-based architecture for both generator and discriminator. For more details about this architecture and residual blocks, see He et al. [23]. The architecture used here is, however, much shallower than the ResNet used as image classifiers, although it was reported that it is possible to train 101-layer ResNet architectures with the generative adversarial training framework, the first one achieving this being the WGAN-GP [28]. The chosen optimizer was again Adam, this time with $\alpha = 0.0002$ and exponential decay rates $\beta_1 = 0.5$, $\beta_2 = 0.9$, which is a configuration slightly different than the one reported in the original paper [30], and a batch size of 64. The hinge loss was used, as described in Chapter 6

(Equation 6.6). The definition of the ResNet architecture used for this experiment can be found in the appendix of the original SNGAN paper [30], where it was used to train a generator on the CIFAR10 dataset, although a few modifications were done for it to support inputs of dimensions larger than 32×32.

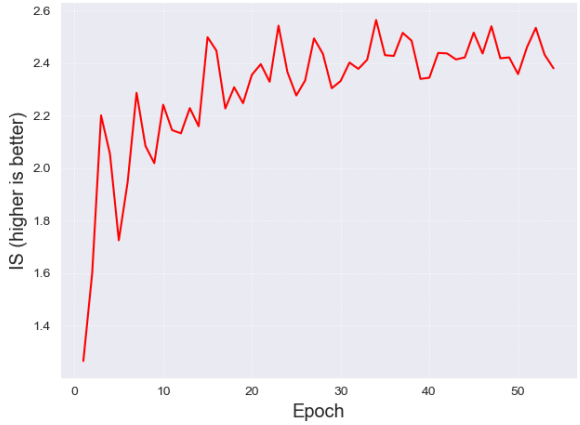


Figure 9.9: IS measure progression for a ResNet-based SNGAN trained on CelebA for 54 epochs.

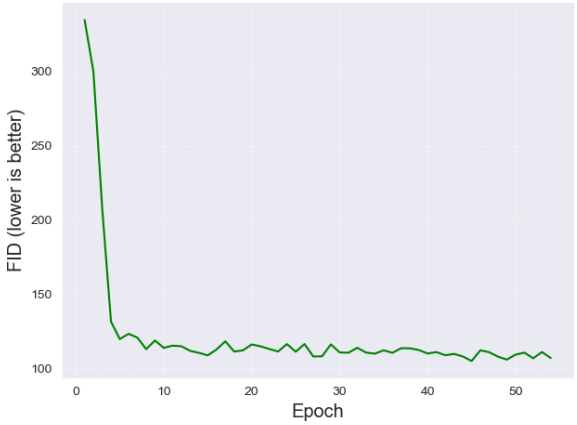


Figure 9.10: FID measure progression for a ResNet-based SNGAN trained on CelebA for 54 epochs.

Surprisingly, the Inception Score is positively correlated with image quality for CelebA samples generated by the SNGAN. This might be explained by the fact that the SNGAN is able to produce samples realistic enough for the Inception v3 network to output classifications with high probabilities, while also producing images that are varied and clearly distinguishable from each other, even early in training. Regardless of that, the previous experiments already show that the IS is an unreliable measure for datasets that do not have a high overlap with ImageNet (see Chapter 7, Section 7.3). Even more surprising is the fact that, for this setup, the FID measure starts to show its limitations. The FID for the SNGAN reaches 119.59 already at epoch 5, and remains around this value up until the training process was stopped, at epoch 54 (Figure 9.10). Meanwhile, the IS keeps increasing throughout the training process (Figure 9.9), and correlates with the author's perceived subjective quality. The IS measures, however, are less reliable during later stages in training as a way to quantify image quality, and are better used as a measure of training progress – contrasting with the FID, which is less reliable earlier in training.

For this experiment, the IS is a better predictor of image quality, despite being applied to a dataset (CelebA) which contains only portraits, which would correlate to only a couple of the possible 1000 output classes of the ImageNet-based Inception v3 network used to calculate the final score. Other than that, there is also a somewhat strong correlation ($r = -0.8872$) between the generator's loss and the IS.

Figure 9.11 shows the image quality progression during training. After the first epoch (first row), the ResNet generator is only able to produce noise. This slowly improves after the second epoch (row 2), and epoch 5 (row 3) we already have samples of relative decent quality – this is where the FID stops making sense as a measure of image quality and training progress. As mentioned above, the IS keeps increasing after epoch 5 (IS = 1.72520, FID = 119.59), while the FID keeps gravitating around 119.59. Row 4 shows samples from the best epoch as reported by the Inception Score (IS = 2.56406, epoch 34), and rows 5 and 6 show samples from the best epoch as reported by the Fréchet Inception Distance (FID = 104.84, epoch 45).

Despite achieving an impressive level of photorealism, the ResNet-based SNGAN still suffers from artifacts which are related to points of low probability in the training dataset. A few

samples generated with items such as sunglasses and hats are blurry and lack visual coherence. Some other issues related to symmetry and spatial coherence also appear for a minority of generated images, as can be seen in Figure 9.11.

Figure 9.11: CelebA samples generated by the ResNet-based SNGAN with hinge loss – First row: epoch 1; Second row: epoch 2; Third row: epoch 5; Fourth row: epoch 34; Fifth and sixth rows: epoch 45.

9.5 SNGAN ON FFHQ

As described in Chapter 8, the CelebA dataset is somewhat limited, containing issues such as compression artifacts and blur, while the images also consist of exclusively young or middle-aged adults. The Flickr-Faces-HQ (FFHQ), initially made for training the StyleGAN v2, contains 69000 images with a resolution of 1024×1024 of a more varied array of people, including children and older adults with more frequency. Some of the datapoints, however, may present challenges to training, such as those containing uncommon makeup patterns and/or costumes. There seems to be a higher variability in the person's angle in the portrait, and some photos are imperfectly cropped versions of photos involving more than one people, which may present additional challenges, particularly when considering CelebA's total of 202599 photos against 69000 photos in FFHQ. This section provides an analysis of the ResNet-based SNGAN on a smaller 128×128 version of the dataset. The architecture and parameters are exactly the same as in the SNGAN from Section 9.4, outputting 64×64 images, while the FID statistics are calculated against the 128×128 dataset. To the best of the author's knowledge, this analysis remains unpublished.

Figures 9.12 and 9.13 show the progression during training for the IS and FID measures, respectively, during a total of 103 epochs. FFHQ has less images per epoch and thus lower convergence speed, and for this reason was trained for about twice the total numbers of epochs when compared to the results from the previous section.

For the FFHQ dataset, however, the FID was found to be a more stable indicator of progress and also outperforms the IS in terms of rating the subjective quality of samples and the amount of variety found in those samples. The generator loss is still somewhat strongly correlated with the IS at $r = -0.875922$.

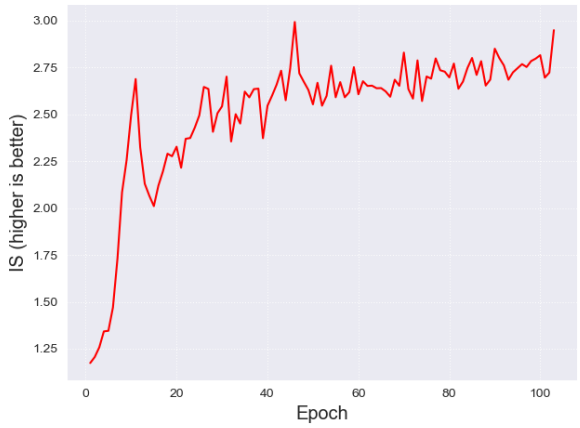


Figure 9.12: IS measure progression for a ResNet-based SNGAN trained on FFHQ for 103 epochs.

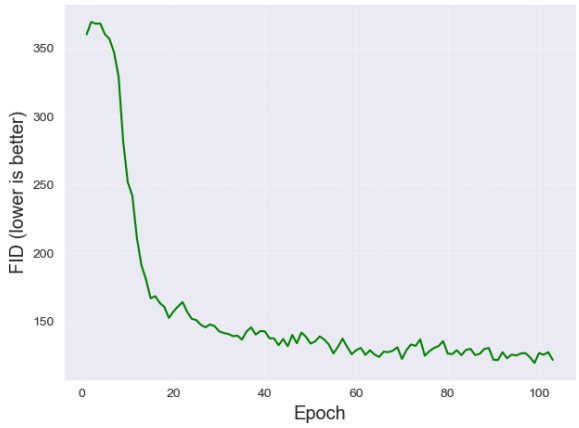


Figure 9.13: FID measure progression for a ResNet-based SNGAN trained on FFHQ for 103 epochs.

Figure 9.14 shows FFHQ generated samples in a progressive order, where each row of images comes from a different epoch. The best reported IS is reported at epoch 46, while the lowest FID is reported at epoch 99. The second-best IS happens at epoch 103, the last epoch for this experiment, where also the third-best FID is reported, indicating that training was stopped too early and that the ResNet-based SNGAN could still improve with further training.

Figure 9.14: Samples generated by the ResNet-based SNGAN with hinge loss trained on FFHQ – First row: epoch 11; Second row: epoch 46 (best IS); Third row: epoch 70; Fourth row: epoch 91; Fifth row: epoch 99 (best FID); Sixth row: epoch 103 (last epoch, third-best FID and second-best IS).

9.6 SNGAN ON CIFAR10

The last experiment consists of an analysis of the SNGAN on the CIFAR10 dataset, which still seems to be the de facto standard for measuring GAN performance and consists of 60000 images with resolution 32×32, divided in 10 classes. Since this work is focused on generative modeling with unsupervised learning, the class information is discarded. While it is interesting to see how advanced models such as the ResNet-based SNGAN are able to learn simpler datasets, this experiment will enable us to see how IS and FID measures based on greatly different datasets compare to each other. As mentioned before, there is significant overlap between CIFAR10's classes and ImageNet classes, the latter being used in the Inception v3 network, which is used to calculate both IS and FID. For this reason, both scores are expected to be very accurate measures of image quality and diversity under most circumstances.

The architecture used in this experiment is a slightly modified version of the ResNet-based SNGAN used in previous sections, made to support inputs of resolution 32×32. The network was trained for 353 epochs, and the best FID was measured at epoch 232 at FID = 49.5358, while the best IS measure was achieved at epoch 270 at IS = 7.3134. This test achieved peak correlation between IS and FID at $r = 0.933$, with both of them also being correlated with perceived subjective quality. For this experiment, the generator loss was not correlated to the IS or FID. Figures 9.15 and 9.16 show the progression during training.

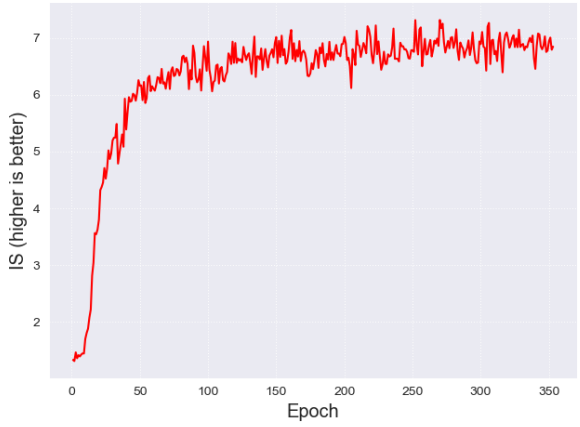


Figure 9.15: IS measure progression for a ResNet-based SNGAN trained on CIFAR10 for 353 epochs.

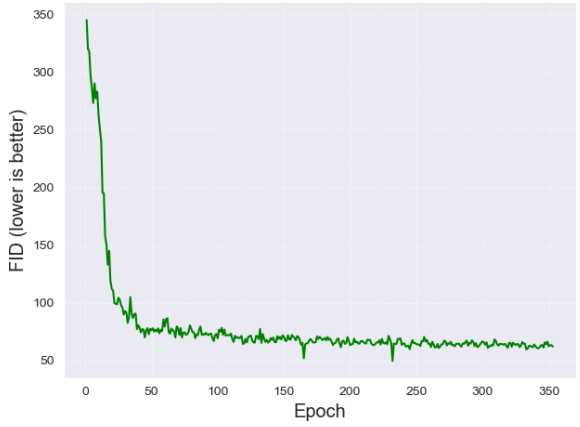


Figure 9.16: FID measure progression for a ResNet-based SNGAN trained on CIFAR10 for 353 epochs.

Figures 9.18 and 9.19 show generated samples with the best reported IS and FID measures, respectively. Although the images are low in resolution, this dataset can be considered very challenging since the pictures have very varied scenarios, which becomes an even bigger issue when training the network in an unsupervised manner.

As mentioned before, the mentioned scores were calculated for $N = 1,000$ samples. One can immediately notice that the scores based on datasets with considerable ImageNet overlap are far different than the score achieved on datasets consisting of only portraits. This shows that both quantitative measures are only useful when comparing different architectures over the same dataset. Some published papers also fail to explicitly mention N, and a very high variability in calculated measures was noticed – for instance, the achieved FID in the last epoch was 65.51 for $N = 1,000$ samples, 30.48 for $N = 5,000$ samples and 20.51 for $N = 50,000$ samples. The original published result, for reference, was 21.7 for $N = 5,000$ samples. This leads us to conclude that both IS and FID must be used with relative care in regards to the generator's training dataset, number of samples and the current stage in the training process. Overall, the

Fréchet Inception Distance is a reliable metric for purposes of monitoring the training progress during research, but should be used with care in publications.

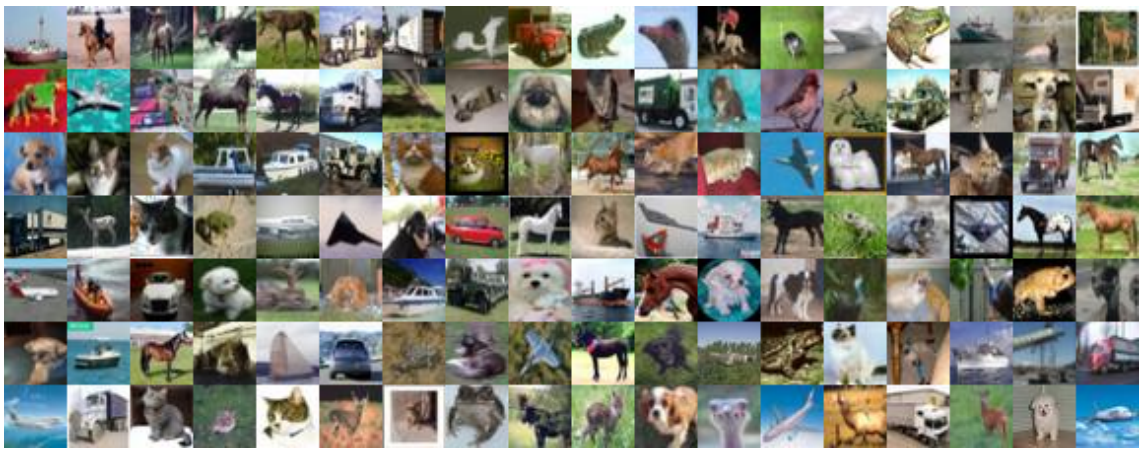


Figure 9.17: Real images from the original CIFAR10 training dataset.

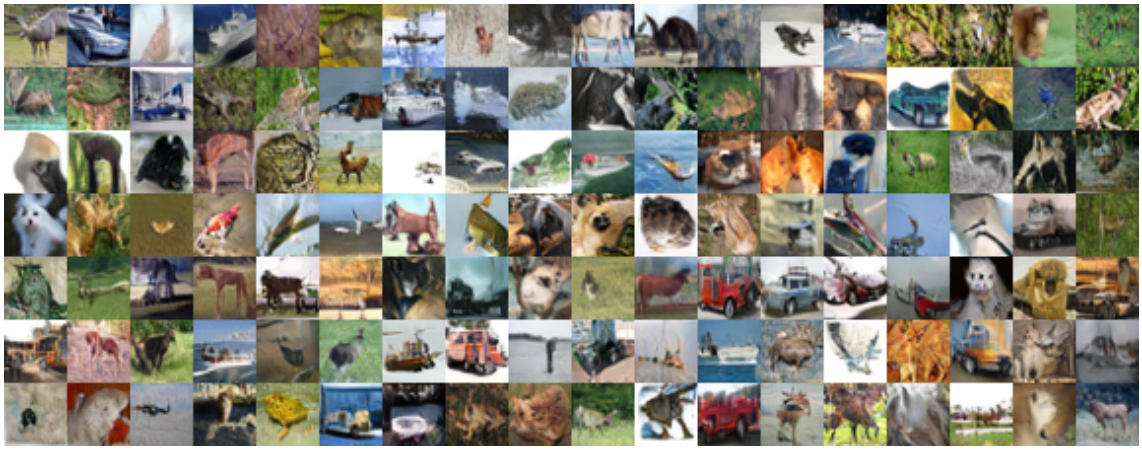


Figure 9.18: Samples generated by the ResNet-based SNGAN with hinge loss trained on CIFAR10. This is the best achieved score according to the IS measure (epoch 270, IS = 7.3134).

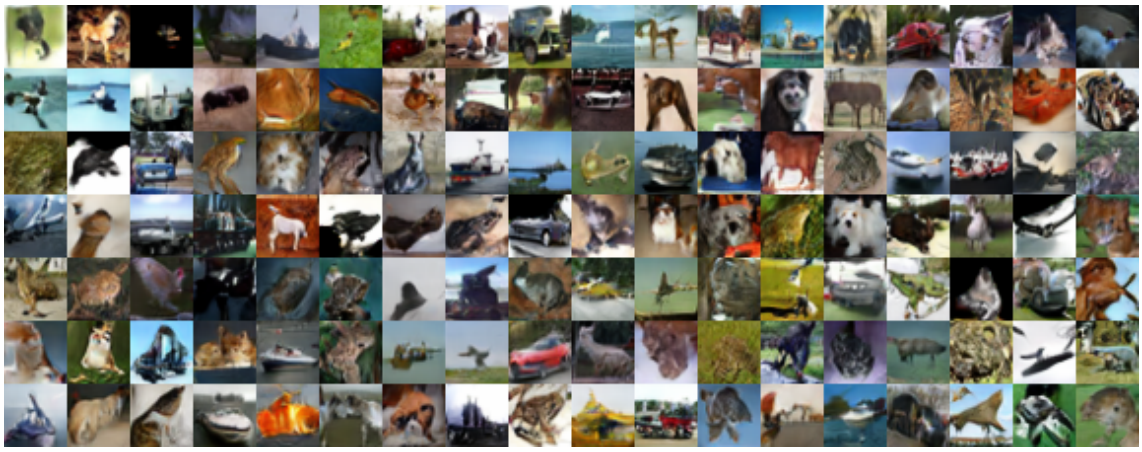


Figure 9.19: Samples generated by the ResNet-based SNGAN with hinge loss trained on CIFAR10. This is the best achieved score according to the FID measure (epoch 232, FID = 49.5358).

10 CONCLUSION

The introduction of deep neural networks to the field of generative modeling has significantly improved it. In just a few years, Generative Adversarial Networks progressed from being able to generate black-and-white digits to being able to produce high resolution portraits that look real enough to fool human beings. The level of achieved realism is so high that it makes many people wonder about what could be achieved in a few more decades, especially in regards to the generation of videos and its use for purposes of identity theft. Although known useful practical applications are still limited, research in the field of deep generative models show that, given enough computing power, it is possible for computers to learn probability distributions of extremely high complexity. More than that, they are surprisingly able to output coherent and previously unseen data instead of simply memorizing a dataset, which can be seen as a component of machine creativity and a step towards general artificial intelligence.

Since its initial publication in 2014, GANs have received numerous improvements, such as different loss functions and many ways of normalizing layers, both of which are components that can stabilize and improve the training process. The publication of the Progressive Growing GAN in late 2017 was a major breakthrough for the field. Its relatively simple concept of slowly increasing the image's resolution was highly successful in practice, and started a trend towards large scale architectures. Even though an absurd amount of electrical energy was consumed to train these large networks, it was proven for the first time that learning such a complex dataset was possible.

Although spectral normalization was highly successful when applied to small architectures, being computationally lightweight and enabling the networks to use more aggressive optimizer parameters while at the same time producing better converged results, there is still almost no published articles concerning the use of spectral normalization in large scale architectures for unsupervised learning. A further improvement on the StyleGAN v2 for learning with an order of magnitude less samples was recently published by NVIDIA [47] and contains the use of newer regularization methods (such as R_1 regularization), but it is unclear as to why the researchers avoided the spectral norm, meaning that further research could be published in this area.

In the last few years, many metrics have been proposed for quantifying the subjective quality perceived in images created by generative models, of which the two most popular ones – the Inception Score (IS) and the Fréchet Inception Distance (FID) – were analyzed in practice. The IS proved to be a less reliable measure, especially when used to compare samples generated from networks which were trained on datasets that do not have a high overlap with the ImageNet dataset, such as CelebA. Despite this, an edge case was found in which the IS was highly correlated with image quality on CelebA, particularly during early training, while at the same time the FID was unable to discern between blurry samples from early training and high-quality samples from later epochs. One can conclude that the FID is less dependent on the dataset used for the generator, and is a better measure for image quality and image diversity in most situations. It is, however, an imperfect measure, as demonstrated in Chapter 9, and more research is clearly required regarding quantitative metrics that are architecture-agnostic.

REFERENCES

[1] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[2] J. Chen, J. Chen, H. Chao, and M. Yang. Image blind denoising with generative adversarial network based noise modeling. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3155–3164, 2018.

[3] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative image inpainting with contextual attention, 2018.

[4] Veit Sandfort, Ke Yan, Perry J. Pickhardt, and Ronald M. Summers. Data augmentation using generative adversarial networks (cyclegan) to improve generalizability in ct segmentation tasks. *Scientific Reports*, 9(1):16884, Nov 2019.

[5] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis, 2016.

[6] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans, 2018.

[7] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders, 2016.

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

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Brendan J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambridge, MA, USA, 1998.

[11] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2017.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[14] Ian J. Goodfellow. On distinguishability criteria for estimating generative models, 2015.

[15] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization, 2016.

[16] Chris Donahue, Zachary C. Lipton, Akshay Balsubramani, and Julian McAuley. Semantically decomposing the latent spaces of generative adversarial networks, 2018.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[18] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[20] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Shun-ichi Amari and Michael A. Arbib, editors, *Competition and Cooperation in Neural Nets*, pages 267–285, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[21] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[22] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[24] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[25] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[26] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[27] Jonas Adler and Sebastian Lunz. Banach wasserstein gan, 2019.

[28] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[29] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[30] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

[31] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[32] Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your gan is secretly an energy-based model and you should use discriminator driven latent sampling, 2020.

[33] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning, 2017.

[34] Jae Hyun Lim and Jong Chul Ye. Geometric gan, 2017.

[35] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[36] Valentin Khrulkov and Ivan Oseledets. Geometry score: A method for comparing generative adversarial networks, 2018.

[37] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In Alfred M. Bruckstein, Bart M. ter Haar Romeny, Alexander M. Bronstein, and Michael M. Bronstein, editors, *Scale Space and Variational Methods in Computer Vision*, pages 435–446, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[38] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2017.

[39] Shane Barratt and Rishi Sharma. A note on the inception score, 2018.

[40] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[41] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

[42] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2019.

[43] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017.

[44] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[45] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge?, 2018.

[46] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.1.1.

[47] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data, 2020.