Generalized Loss Functions for Generative Adversarial Networks

by

Himesh Bhatia

A thesis submitted to the

Department of Mathematics and Statistics

in conformity with the requirements for

the degree of Master of Applied Science

Queen's University

Kingston, Ontario, Canada

October 2020

Abstract

This thesis investigates the use of parameterized families of information-theoretic measures to generalize the loss functions of generative adversarial networks (GANs) under the objective of improving performance. A new generator loss function, called least kth-order GAN (LkGAN), is introduced, generalizing the least squares GANs (LSGANs) by using a kth order absolute error distortion measure with $k \geq 1$ (which recovers the LSGAN loss function when $k = 2$). It is shown that minimizing this generalized loss function under an (unconstrained) optimal discriminator is equivalent to minimizing the kth-order Pearson-Vajda divergence.

A novel loss function for the original GANs using Rényi information measures with parameter α is next presented. The GAN's generator loss function is generalized in terms of Rényi cross-entropy functionals. For any $\alpha > 0$, this generalized loss function is shown to preserve the equilibrium point satisfied by the original GAN loss based on the Jensen-Rényi divergence, a natural extension of the Jensen-Shannon divergence. It is also proved that the Rényi-centric loss function reduces to the original GANs loss function as $\alpha \to 1$.

Experimental results implemented on the MNIST and CelebA datasets under both DCGANs and StyleGANs architectures, indicate that the proposed LkGAN and RényiGAN systems confer performance benefits by virtue of the extra degrees of freedom provided by the parameters k and α, respectively. More specifically, experiments show improvements with regard to the quality of the generated images as measured by the Fréchet Inception Distance (FID) score and demonstrated by training stability and extensive simulations.

Co-authorship

I would like to acknowledge and thank my collaborators, Philippe Burlina and William Paul from Johns Hopkins University, particularly for experiments in Chapter 5, which is part of our joint paper [12].

Acknowledgments

I would like to thank Professor Fady Alajaji and Professor Bahman Gharesifard for their support and showing me the beauty of mathematical research. It was Professor Gharesifard's first year linear algebra course that piqued my interest in mathematics. I am grateful for the opportunities he gave me to conduct research under his supervision during my undergraduate degree. Professor Alajaji's course on coding theory and information theory inspired me to pursue a graduate degree researching in this field. Their passion and motivation are inspirational.

I am grateful to my family, Parveen and Babita, and my sister, Kritika, for their love and support. They motivate me to be the best person I can be. I would not be where I am today without their sacrifices and guidance. Thank you. I would also like to thank my friends for their emotional support and the fun times during the highs and lows of university life.

I would like to thank the Centre for Advanced Computing at Queen's University, especially Hartmut Schmider and Michael Hanlan, for their technical support and access to their computer cluster to run simulations that are presented in this thesis. I am thankful for the generous support of the Mathematics and Statistics Department of Queen's University and the Natural Sciences and Engineering Research Council of Canada. I would also like to acknowledge Amazon Web Services for access to high end computers, which were also used to run simulations presented in this thesis.

Finally, I would like to thank Rianna Lewis, Jennifer Read, and Johana Ng for their support and help.

Contents

Abstract **i**

Co-authorship **iii**

Acknowledgments **iv**

Contents **vi**

List of Tables **ix**

List of Figures **xi**

Chapter 1: Introduction **1**

1.1 Literature review 2

1.2 Contributions 4

1.3 Outline 7

Chapter 2: Preliminaries on information measures and neural networks **8**

2.1 Information measures 8

2.1.1 Entropies 8

2.1.2 Divergences 10

2.1.3 Mutual informations 13

2.1.4 Cross-entropies 14

2.2 Neural networks 22

2.2.1 Propagation equations 22

2.2.2 Backpropagation equations 28

2.2.3 Gradient descent algorithm: Adam optimization 30

2.2.4 Convolutional neural networks 33

Chapter 3: Generative adversarial networks **40**

3.1 Gradient penalties 45

3.2 Least squares GANs (LSGANs) 46

3.3 InfoGANs 50

3.4 Fréchet inception distance: measuring image quality 53

Chapter 4: Least kth-order GANs (LkGANs) **56**

4.1 Theoretical results 56

4.2 Experiments 59

4.2.1 Methods 59

4.2.2 Results 61

4.2.3 Discussion 80

Chapter 5: RényiGANs **84**

5.1 Theoretical results 84

5.2 Experiments 87

5.2.1 Methods 87

5.2.2 Results 91

5.2.3 Discussion 104

Chapter 6: Conclusion **107**

Bibliography **109**

Appendix A: Experiments **122**

A.1 Neural network architectures 122

A.2 Algorithms 124

A.3 MNIST results 129

List of Tables

4.1 LkGANs experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over ten trials. 63

4.1 LkGANs experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over ten trials. 64

4.2 The average and variance best FID scores for LkGAN-v2 that generated meaningful images and the average and variance of the epoch when this occurs. 64

4.3 LkStyleGANs experiments on the CelebA dataset: the best FID score over each run seen over three trials. 69

4.4 LkStyleGANs experiments on the CelebA dataset: the average and variance of the best FID score and the average and variance epoch this occurs taken over three trials. 70

5.1 RényiGANs experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over ten trials. 92

5.2 The average and variance best FID scores for RényiGANs that generated meaningful images and the average and variance of the epoch when this occurs. 93
5.3 RényiStyleGANs experiments on the CelebA dataset: the best FID over each run seen over three trials. 98
5.4 RényiStyleGANs experiments on the CelebA dataset: the average and variance of the best FID score and the average and variance this occurs taken over three trials. 99

A.1 The generator's architecture for MNIST dataset. 123
A.2 The discriminator's architecture for MNIST dataset. 123
A.3 LkGANs-v1 experiments on the MNIST dataset: the best FID over each run seen over ten trials. 129
A.4 LkGANs-v2 experiments on the MNIST dataset: the best FID over each run seen over ten trials. 130
A.5 LkGANs-v3 experiments on the MNIST dataset: the best FID over each run seen over ten trials. 131
A.6 RényiGANs experiments on the CelebA dataset: the best FID over each run seen over ten trials. 132

List of Figures

2.1 A neuron. 24

2.2 A neural network with three layers that maps $(x_1, x_2, x_3) \mapsto (y_1, y_2, y_3, y_4)$. 24

2.3 Padding a $5 \times 5 \times 1$ image with $t = 1$. The white pixels are the padding added to the image. The image is represented by the blue pixels. . . . 36

2.4 A visual representation of Example 6. The dark blue pixels are the pixels of the input image. The light blue and dark blue pixels are the pixels that the kernels are convoluted with. 38

4.1 Evolution of the average FID scores throughout training for LKGANs. 65

4.2 Plots of the FID scores versus epochs for the best performing LkGAN for each version and its LSGAN counterpart for a selection of three trials. 66

4.3 Sample generated images of the best performing LkGAN for each version and its LSGAN counterpart. 67

4.4 Evolution of the average FID scores throughout training for LkStyleGANs. 71

4.5 Plots of the FID scores versus epochs for the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trials 1, 2, and 3. 72

4.5 Plots of the FID scores versus epochs for the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trials 1, 2, and 3. 73
4.6 Sample generated images of the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trial 1. 75
4.7 Sample generated images of the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trial 2. 77
4.8 Sample generated images of the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trial 3. 79

5.1 Evolution of the average FID scores throughout training for RényiGANs. 93
5.2 Plots of the FID scores versus epochs for the best performing RényiGAN for each version and its DCGAN counterparts for a selection of three trials. 95
5.3 Sample generated images of the best performing RényiGAN in terms of FID scores for each version and its DCGAN counterpart. 97
5.4 Evolution of average FID scores throughout training for RényiStyleGANs. 99
5.5 Plots of the FID scores versus epochs for the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trials 1, 2, and 3. 100
5.6 Sample generated images of the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trial 1. 101
5.7 Sample generated images of the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trial 2. 102
5.8 Sample generated images of the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trial 3. 103

Chapter 1

Introduction

Unsupervised learning is the problem of educing information from a large unlabeled dataset and, in the context of generative models, is a relatively new area that has received much attention. Two prominent objectives in generative modeling consist of determining the underlying probability distribution function of a dataset or generating data that mimics it. Classical techniques for the former include maximum likelihood estimators, methods of moments estimators and Bayesian estimators. The main approaches for the latter include generative adversarial networks (GANs) [32], [9], [67], [18], autoencoders/variational autoencoders (VAEs) [43], generative autoregressive models [64], invertible flow based latent vector models [44], and hybrids of the above models [33].

Compared to other approaches, GANs have garnered the most interest (e.g., see surveys in [18], [83], [84]). Generative models can be incorporated into reinforcement learning and time series predictions. Generative models can also be trained in a semi-supervised manner, where labels and training examples are missing. Unlabelled data is easier to obtain and require less memory space, and semi-supervised algorithms are good at generalizing the missing data. Traditionally, machine learning algorithms are not able to train models that have multi-modal outputs from a single input. However, generative models are designed to produce several different outputs that are equally acceptable [30], [41].

Explicitly defining the probability distribution over the data to generate new data using classical methods, such as maximum likelihood estimation, may be computationally expensive [30]. In the case of VAEs, approximation methods are needed for intractable functions and furthermore the approximate probability distribution is not guaranteed to converge to the true distribution [55], [56], [9].

In contrast, GANs optimize a loss function using game theory and information theory. Furthermore, they can represent distributions that lie on low dimensional manifolds, which VAEs and estimating densities are unable to do [9]. Moreover, GANs do not rely on Markov Chains and no variational bound is needed [30]. As such, GANs are the focus of this thesis.

1.1 Literature review

The original GANs [32] consist of a generative neural network competing with a discriminatory neural network in a min-max game. Several variants of GANs have been studied and implemented. Deep convolutional GANs (DCGANs) use convolutional layers to learn higher dimensional dependencies that are inherent in complex datasets such as images [67]. Although DCGANs produced better results than other state-of-the-art generative models such as VAEs and autoregressive models, they can be difficult to train and can suffer from mode collapse [9], [84]. Mode collapse occurs when the generator produces one output, which leads to the discriminator being able to tell apart real from fake data perfectly during training. Researchers have diligently attempted to fix these mentioned issues. For example, StyleGANs [41] change the architecture of the generative neural network to produce realistic high resolution images, while Wasserstein GANs [9] reduce the problem of mode collapse by using the Wasserstein-1 distance as the loss function.

GANs have been applied to data privacy problems, where the goal is to hide certain features of user data to protect their privacy, but mask these features judiciously in order not to compromise other useful data [39]. It has also been used for computer vision problems, such as generating fake images of handwritten numbers, or landscape paintings [30]. Thus the flexibility of GAN design allows for innovation and applicability to a wide range of data.

The use of information theory to study and improve neural networks is a relatively new yet promising direction of research; e.g., see [63], [65], [2], [15], [85], [79], [5], [89] and [88] and the references therein. While many GANs loss functions are based on the Jensen-Shannon divergence, there are other divergence measures and tools in information theory that can be directly applied to the design of GANs. The family of loss functions that simplify down to f-divergences was thoroughly studied in [63], [27], and [49]. Bridging the gap between maximum likelihood learning and GANs, especially those with loss functions that simplify down to f-divergences, have also been analyzed in [89]. Using the symmetric Kullback-Leibler divergence, researchers have also shown that a variant of VAEs is connected to GANs [14]. InfoGANs use variational mutual information maximization with latency codes to achieve unsupervised representation learning with considerable success [15].

A new least squares loss function that simplifies down to the Pearson χ^2 divergence was examined in [55]. It was shown through experiments that the resulting least squares GANs (LSGANs) are more stable than DCGANs. The promising results of LSGANs and the fact that the Pearson-Vajda $|\chi|^k$ divergence of order $k \geq 1$ generalizes the Pearson χ^2 divergence is one motivation for this thesis.

The use of the Rényi divergence in the context of GANs is sparse. Rényi used the simplest set of postulates that characterize Shannon's entropy and introduced his own entropy and divergence measures (parameterized by its order α) that generalize the Shannon entropy and the KL divergence, respectively [69]. Moreover, the original Jensen-Rényi divergence [36] as well as the identically named divergence [45] used in this paper are non-f-divergence generalizations of the Jensen-Shannon divergence. Traditionally, Rényi's entropy and divergence have had applications in a wide range of problems, including lossless data compression [13], [16], [66], hypothesis testing [20], [4], error probability [11], and guessing [6], [82]. Recently, the Rényi divergence and its variants (including Sibson's mutual information) were used to bound the generalization error in learning algorithms [26], and to analyze deep neural networks (DNNs) [85], variational inference [51], Bayesian neural networks [50], and generalized learning vector quantization [58]. However, there does not exist prior work on generalizing Jensen-Shannon divergence GANs loss functions that are not in the family of f-divergences. This provides another motivation for this work.

1.2 Contributions

The novel contributions of this thesis are described in what follows. We revisit LSGANs by considering a more general parameterized classes of loss functions that include the original LSGANs' generator loss function as a special case. More specifically, we introduce *least kth-order GANs (LkGANs)* by considering the kth-order absolute error loss function for the generator ($k \geq 1$). We prove that minimizing this loss function is equivalent to minimizing the kth-order Pearson-Vajda divergence [61], which recovers the Pearson χ^2 divergence examined in [55] when $k = 2$. The LkGANs' generator loss function also preserves the theoretical minimum of LSGANs' generator loss function, which is achieved when the generator's distribution is equal to the true distribution. LkGANs are implemented and compared with LSGANs on the MNIST [47] and CelebA [53] datasets using the DCGAN and StyleGAN architectures. Experimentally, LkGANs are shown to outperform LSGANs in terms of generated image quality, as measured by the Fréchet inception distance (FID) score [37], and the rate at which they converge to meaningful results. LkGANs are also observed to reduce the problem of mode collapse during training.

We also revisit the original GAN's generator optimization problem by considering more general parameterized classes of loss functions that subsume the original function as a special case. An important objective is to identify generalized loss functions that can be analytically minimized under an (unconstrained) optimal discriminator, with the minimum theoretically achieved when the generator's distribution is the true dataset distribution. To this end, we consider a new GAN's generator loss function expressed in terms of the negative sum of two Rényi cross-entropy functionals of order α, where $\alpha > 0$ and $\alpha \neq 1$. We show that minimizing this α-parameterized loss function under an optimal discriminator results in the minimization of the Jensen-Rényi divergence [45], which is a natural extension of the Jensen-Shannon divergence as it uses the Rényi divergence instead of the Kullback-Leibler (KL) divergence in its expression. Note that this Jensen-Rényi divergence measure, which reduces to the

Jensen-Shannon divergence as α approaches 1, differs from an earlier namesake measure introduced in [36], [35] and defined using the Rényi entropy. We also prove that our generator loss function of order α converges to the original GAN loss function in [32] when $\alpha \to 1$. Previously, researchers generalized the GANs loss function using the f-divergence measure [19] [63]. However as the Jensen-Rényi divergence is not itself an f-divergence, it can be interpreted as a non-f-divergence generalization of the Jensen-Shannon divergence. We call the resulting network RényiGANs.

Finally, we implement the newly proposed RényiGAN loss function using the DCGAN and StyleGAN architectures [41]. Our experiments use the MNIST [47] and CelebA [53] datasets and provide comparisons with the baseline DCGAN and StyleGAN systems. Experiments show that the Rényi-centric GAN systems perform as well as, or better, than their baseline counterparts in terms of visual quality of the generated images (as measured by the FID score), particularly when spanning α over a range of values as it helps the avoidance of local minimums. We show that employing L_1 normalization with the Rényi generator loss function confers greater stability, quicker convergence, and better FID scores for both RényiGANs and DCGANs. Consistent stability and slightly improved FID scores are also noted when comparing RényiStyleGAN with StyleGAN. We finally compare these GAN systems with the simplified gradient penalty [57], showing that the Rényi-type systems provide substantial reductions in computational training time vis-a-vis the baselines, for similar levels of FID. [1]

[1]Our codes and results can be found in https://github.com/renyigan-lkgan?tab=repositories.

1.3 Outline

We introduce the preliminary background on information-theoretic measures and neural networks in Chapter 2. In Chapter 3, we describe the original GAN systems, including recent papers that improve the design of these networks by altering the GANs loss function, such as LSGANs and InfoGANs. In Chapter 4, we analyze LkGANs, which generalize LSGAN's generator loss function, and we provide experimental results comparing LkGANs with LSGANs. In Chapter 5, we present theoretical results of RényiGANs, which generalize GANs, and show experimental results comparing RényiGANs with DCGANs and StyleGANs. Finally, we provide conclusions in Chapter 6.

Chapter 2

Preliminaries on information measures and neural networks

We provide background material on information measures and neural networks. We use the $(\mathbb{R}, \mathcal{B}(\mathbb{R}), \mu)$ measure space, where $\mathcal{B}(\mathbb{R})$ is the Borel σ-algebra on $\mathbb{R}$ and μ is the Lebesgue measure. We also use the short-form $\int_{\mathbb{R}} f \mathrm{d}\mu := \int_{x \in \mathbb{R}} f(x) \mathrm{d}\mu(x)$, where $f : \mathbb{R} \to \mathbb{R}$ is a measurable function. Throughout, we use $\mathbf{A}$ to denote a random variable, $\mathbf{a}$ to denote a matrix, and $\boldsymbol{a}$ to denote a vector.

2.1 Information measures

We describe entropy, divergence, mutual information, and cross-entropy measures used in this work. More details on the properties of these quantities can be found in the texts [3], [17], [21], and [87].

2.1.1 Entropies

Entropy measures the uncertainty of the outcome of an experiment that is modelled by a random variable. Shannon puts forth a set of postulates that models entropy and provided a simple and elegant formula. We first present the definitions of Shannon entropy [76].

Definition 1 *The* ***Shannon entropy*** *of a discrete random variable* $\mathbf{X}$ *with probability mass function* $p_{\mathbf{X}}$ *and finite alphabet (or range)* $\mathcal{X} \subset \mathbb{Z}$ *is defined as*

$$H(p_{\mathbf{X}}) := -\sum_{x\in\mathcal{X}} p_{\mathbf{X}}(x)\log(p_{\mathbf{X}}(x)). \tag{2.1}$$

Definition 2 *The* ***differential Shannon entropy*** *of a continuous random variable* $\mathbf{Y}$ *with probability density function* $f_{\mathbf{Y}}$ *and support* $S_{\mathbf{Y}} \subset \mathbb{R}$ *is defined as*

$$h(f_{\mathbf{Y}}) := -\int_{S_{\mathbf{Y}}} f_{\mathbf{Y}}\log(f_{\mathbf{Y}})\mathrm{d}\mu. \tag{2.2}$$

Several researchers refined Shannon's original postulates. Using a new set of postulates, Rényi created a generalization of Shannon entropy [69].

Definition 3 *For* $\alpha > 0$, $\alpha \neq 1$, *the* ***Rényi entropy of order*** $\boldsymbol{\alpha}$ *of a discrete random variable* $\mathbf{X}$ *with probability mass function* $p_{\mathbf{X}}$ *and finite alphabet* $\mathcal{X} \subset \mathbb{Z}$ *is defined as*

$$H_\alpha(p_{\mathbf{X}}) := \frac{1}{1-\alpha}\log\left(\sum_{a\in\mathcal{X}} p_{\mathbf{X}}(a)^\alpha\right). \tag{2.3}$$

Definition 4 *[81] For* $\alpha > 0$, $\alpha \neq 1$, *the* ***differential Rényi entropy of order*** $\boldsymbol{\alpha}$ *of a continuous random variable* $\mathbf{Y}$ *with probability density function* $f_{\mathbf{Y}}$ *and support*

$S_{\mathbf{Y}} \subset \mathbb{R}$ *is*

$$h_\alpha(f_{\mathbf{Y}}) := \frac{1}{1-\alpha} \log \left(\int_{S_{\mathbf{Y}}} f_{\mathbf{Y}}^{\alpha} \mathrm{d}\mu \right). \tag{2.4}$$

2.1.2 Divergences

Divergence measures are used to quantify the dissimilarity between distributions. We first present the definitions of the Kullback-Leiber divergence, Jensen-Rényi divergence, and the Pearson-Vajda divergences.

Definition 5 *[46] The* ***Kullback-Leibler divergence*** *between two distributions functions p and q with the same support or finite alphabet is defined as*

$$\mathrm{KL}(p\|q) := \mathop{\mathbb{E}}_{\mathbf{A}\sim p} \left(\log \frac{p(\mathbf{A})}{q(\mathbf{A})} \right). \tag{2.5}$$

Note that when we say *distributions functions*, we mean that the definition works for probability mass functions p and q and probability density functions p and q.

The KL-divergence can be generalized via the Rényi divergence.

Definition 6 *[81] Given $\alpha > 0$, $\alpha \neq 1$, the* ***Rényi divergence of order $\boldsymbol{\alpha}$*** *between two probability mass functions p and q with the same finite alphabet $\mathcal{X} \subset \mathbb{Z}$ is defined as*

$$\mathrm{D}_\alpha(p\|q) := \frac{1}{\alpha-1} \log \left(\sum_{x\in\mathcal{X}} p^\alpha(x) q^{1-\alpha}(x) \right). \tag{2.6}$$

Note that $\mathrm{D}_\alpha(p\|q) \geq 0$ with equality if and only if $p = q$, see [81].

Definition 7 *[81] For $\alpha > 0$, $\alpha \neq 1$ and two probability density functions p and q,* *the* ***differential Rényi divergence of order $\boldsymbol{\alpha}$*** *with common support $\mathcal{R} \subset \mathbb{R}$ is defined as*

$$\mathrm{D}_\alpha(p\|q) := \frac{1}{\alpha - 1} \log\left(\int_{\mathcal{R}} p^\alpha q^{1-\alpha} \mathrm{d}\mu\right). \tag{2.7}$$

Similarly, $\mathrm{D}_\alpha(p\|q) \geq 0$ with equality if and only if $p = q$ almost everywhere (a.e.), see [81]. Furthermore, the Rényi divergence reduces to the KL-divergence as $\alpha \to 1$.

Theorem 1 *[81] For two distribution functions p and q, assume $\mathrm{D}_\gamma(p\|q) < \infty$ for some $\gamma > 1$. Then we have that*

$$\lim_{\alpha \to 1} \mathrm{D}_\alpha(p\|q) \quad = \quad \mathrm{KL}(p\|q). \tag{2.8}$$

For simplicity of analysis, we assume in what follows the finiteness of $\mathrm{D}_\gamma(\cdot\|\cdot)$ for some $\gamma > 1$ so that (2.8) holds. Being a function of an f-divergence, useful properties and bounds on the Rényi divergence can be elucidated from the study of f-divergences, see [75] and related references.

We next describe the Pearson-Vajda divergence, which is itself an f-divergence [61].

Definition 8 *The* ***Pearson-Vajda divergence of order $\boldsymbol{k}$****, denoted by $|\chi|^k(p\|q)$, between two probability density functions p and q with common support $\mathcal{R} \subset \mathbb{R}$, where $k \geq 1$, is given by*

$$|\chi|^k(p\|q) := \int_{\mathcal{R}} \frac{|q - p|^k}{p^{k-1}} \mathrm{d}\mu. \tag{2.9}$$

Note that $|\chi|^k(p\|q) \geq 0$ with equality if and only if $p = q$ (a.e.). Also when $k = 2$,

$|\chi|^k(\cdot\|\cdot)$ reduces to the Pearson χ^2 divergence measure.

Definition 9 *[61] For two probability density functions p and q with common support $\mathcal{R} \subset \mathbb{R}$, the* ***Pearson χ^2 divergence*** *between p and q is defined as*

$$\chi^2(p\|q) := \int_{\mathcal{R}} \frac{(q-p)^2}{p} \mathrm{d}\mu. \tag{2.10}$$

We now describe the definition of Jensen Shannon divergence, which is the mixture of two KL-divergences.

Definition 10 *[52] The* ***Jensen-Shannon divergence*** *between two distribution functions p and q is given by*

$$\mathrm{JSD}(p\|q) \;:=\; \frac{1}{2}\mathrm{KL}\left(p\middle\|\frac{p+q}{2}\right) + \frac{1}{2}\mathrm{KL}\left(q\middle\|\frac{p+q}{2}\right). \tag{2.11}$$

Replacing the KL-divergences with Rényi divergences gives a natural extension of the Jensen-Shannon divergence, which we call Jensen-Rényi divergence. This Jensen-Rényi divergence was recently introduced in [45] for discrete distributions and studied in the context of generalized (Rényi-type) f-divergences. It differs from the identically named divergence studied in [36] and [35], an earlier extension of the Jensen-Shannon divergence consisting of the difference between the Rényi entropy of a mixture of multiple probability distributions and the mixture of the Rényi entropies of the individual distributions. Other recent (but different) extensions of the Jensen-Shannon divergence can be found in [59] and the references therein.

Definition 11 *The **Jensen-Rényi divergence of order $\boldsymbol{\alpha}$** between two probability distributions p and q, where $\alpha > 0$, $\alpha \neq 1$, is given by*

$$\mathrm{JR}_\alpha(p\|q) \;:=\; \frac{1}{2}\mathrm{D}_\alpha\left(p\middle\|\frac{p+q}{2}\right) + \frac{1}{2}\mathrm{D}_\alpha\left(q\middle\|\frac{p+q}{2}\right). \tag{2.12}$$

By the non-negativity of the Rényi divergence, it follows that $\mathrm{JR}_\alpha(p\|q) \geq 0$ with equality if and only if $p = q$ (a.e.). Finally, since $\lim_{\alpha\to1} \mathrm{D}_\alpha(p\|q) = \mathrm{KL}(p\|q)$, we have that

$$\lim_{\alpha\to1} \mathrm{JR}_\alpha(p\|q) = \mathrm{JSD}(p\|q). \tag{2.13}$$

2.1.3 Mutual informations

Shannon mutual information between two random variables $\mathbf{X}$ and $\mathbf{Y}$ measures the amount of information that $\mathbf{Y}$ contains about $\mathbf{X}$ (or, symmetrically, that $\mathbf{X}$ contains about $\mathbf{Y}$). We present the definition of the Shannon mutual information.

Definition 12 *[76] The **Shannon mutual information** between two random variables $\mathbf{X}$ and $\mathbf{Y}$, with joint distribution function $p_{\mathbf{X},\mathbf{Y}}$ and marginal distribution functions $p_{\mathbf{X}}$ and $p_{\mathbf{Y}}$, is defined as*

$$\mathrm{I}(p_{\mathbf{X}}; p_{\mathbf{Y}}) := \mathrm{KL}(p_{\mathbf{X},\mathbf{Y}}\|p_{\mathbf{X}}p_{\mathbf{Y}}). \tag{2.14}$$

Note that $\mathbf{X}$ and $\mathbf{Y}$ are independent random variables if and only if $\mathrm{I}(p_{\mathbf{X}}; p_{\mathbf{Y}}) = 0$.

Arimoto, Csiszár, and Sibson have generalized Shannon's mutual information (e.g., see [82]). We present Arimoto's generalization below.

Definition 13 *[7] For $\alpha > 0$, $\alpha \neq 1$, the **Arimoto-Rényi mutual information of order $\boldsymbol{\alpha}$** between two random variables $\mathbf{X}$ and $\mathbf{Y}$, with joint distribution function $p_{\mathbf{X},\mathbf{Y}}$ and marginal distribution functions $p_{\mathbf{X}}$ and $p_{\mathbf{Y}}$, is defined as*

$$\mathrm{I}_{\alpha}^{a}(p_{\mathbf{X}}; p_{\mathbf{Y}}) := H_{\alpha}(p_{\mathbf{X}}) - H_{\alpha}^{a}(p_{\mathbf{X}|\mathbf{Y}}), \tag{2.15}$$

where

$$H_{\alpha}^{a}(p_{\mathbf{X}|\mathbf{Y}}) := \frac{\alpha}{1-\alpha} \log \left(\mathop{\mathbb{E}}_{\mathbf{A} \sim p_{\mathbf{Y}}} \left(\| p_{\mathbf{X}|\mathbf{Y}}(\cdot|\mathbf{A}) \|_{\alpha} \right) \right), \tag{2.16}$$

*where $\|f\|_{\alpha} = (\int_{\mathbb{R}} f^{\alpha} \mathrm{d}\mu)^{\frac{1}{\alpha}}$ is the **$\boldsymbol{\alpha}$-norm** for some measurable function f.*

2.1.4 Cross-entropies

Shannon cross-entropy is a measure of the dissimilarity between two probability mass functions or probability density functions p and q, and is closely related to $\mathrm{KL}(p\|q)$. Indeed, like the KL divergence, it measures the difference between conducting an experiment using an approximate distribution function q when the true distribution is p.

Definition 14 *The **Shannon cross-entropy** between two probability mass functions p and q with common finite alphabet $\mathcal{X} \subset \mathbb{Z}$ is defined as*

$$H(p; q) := -\sum_{x \in \mathcal{X}} p(x) \log(q(x)). \tag{2.17}$$

Note that $H(p; q) = \mathrm{KL}(p\|q) + H(p) \geq H(p)$. Thus $H(p; q) = H(p)$ if and only if $p = q$.

Definition 15 *[60] If p and q are two probability density functions with common support $\mathcal{R} \subset \mathbb{R}$, then the* ***differential Shannon cross-entropy*** *between p and q is defined as*

$$h(p;q) := -\int_{\mathcal{R}} p \log(q) \mathrm{d}\mu. \tag{2.18}$$

Similarly $h(p;q) = h(p)$ if and only if $p = q$ (a.e.).

Similar to entropy and divergences, Shannon cross-entropy can be generalized using the Rényi cross-entropy.

Definition 16 *[80] The* ***Rényi cross-entropy of order $\boldsymbol{\alpha}$*** *between two probability mass functions p and q with common finite alphabet $\mathcal{X} \subset \mathbb{Z}$, where $\alpha > 0$, $\alpha \neq 1$, is given by*

$$H_\alpha(p;q) \quad := \quad \frac{1}{1-\alpha} \log \left(\sum_{x \in \mathcal{X}} p(x) q^{\alpha-1}(x) \right). \tag{2.19}$$

We next introduce an analogous definition of Rényi cross-entropy between two probability density functions.

Definition 17 *The* ***differential Rényi cross-entropy of order $\boldsymbol{\alpha}$*** *between two probability density functions p and q with common support $\mathcal{R} \subset \mathbb{R}$, where $\alpha > 0$, $\alpha \neq 1$, is given by*

$$h_\alpha(p;q) \quad := \quad \frac{1}{1-\alpha} \log \left(\int_{\mathcal{R}} p q^{\alpha-1} \mathrm{d}\mu \right). \tag{2.20}$$

Note that $h_\alpha(p;q)$ reduces to $h_\alpha(p)$, when $p = q$ (a.e.). Under some finiteness conditions, $\lim_{\alpha \to 1} h_\alpha(p;q) = h(p;q)$, as we show in the next theorem.

Theorem 2 *Let p and q be two probability density functions with common support $\mathcal{R} \subset \mathbb{R}$. If $h(p;q) < \infty$, then*

$$\lim_{\alpha \downarrow 1} h_\alpha(p;q) = h(p;q).$$

Moreover, if $\underset{A \sim p}{\mathbb{E}} \left(\frac{1}{q(A)}\right) < \infty$, then

$$\lim_{\alpha \uparrow 1} h_\alpha(p;q) = h(p;q).$$

The proof of the theorem requires the following result, which we recall from [81].

Lemma 1 *For any $x > \frac{1}{2}$,*

$$(x-1)\left(1 + \frac{1-x}{2}\right) \leq \log(x) \leq x - 1.$$

Proof of Theorem 2 Working in the measure space $(\mathcal{R}, \mathcal{B}(\mathcal{R}), \mu)$, where $\mathcal{B}(\mathcal{R})$ is the Borel σ-algebra and μ is the Lebesgue measure on $\mathbb{R}$, we first note that by setting

$$x_\alpha = \underset{\mathbf{A} \sim p}{\mathbb{E}} (q(\mathbf{A})^{\alpha-1}) = \int_{\mathcal{R}} p q^{\alpha-1} \mathrm{d}\mu$$

we have that $\lim_{\alpha \downarrow 1} x_\alpha = 1$. Also, Lemma 1 yields that

$$\lim_{\alpha \downarrow 1} \frac{\log(x_\alpha)}{x_\alpha - 1} = 1.$$

We then can write

$$\lim_{\alpha \downarrow 1} h_\alpha(p;q) \quad = \quad \lim_{\alpha \downarrow 1} \frac{1}{1-\alpha} \log(x_\alpha)$$

$$\begin{aligned}
&= \lim_{\alpha\downarrow 1}\frac{x_\alpha - 1}{1-\alpha}\frac{\log(x_\alpha)}{x_\alpha - 1}\\
&= \lim_{\alpha\downarrow 1}\left(\frac{x_\alpha - 1}{1-\alpha}\right)\lim_{\alpha\downarrow 1}\left(\frac{\log(x_\alpha)}{x_\alpha - 1}\right)\\
&= \lim_{\alpha\downarrow 1}\frac{x_\alpha - 1}{1-\alpha}
\end{aligned}$$

if and only if $\lim_{\alpha\downarrow 1}\frac{x_\alpha-1}{1-\alpha}$ exists. We next show the existence of this limit and verify that it is indeed equal to $h(p;q)$. Consider

$$\lim_{\alpha\downarrow 1}\frac{x_\alpha - 1}{1-\alpha} = \lim_{\alpha\downarrow 1}\int_{\mathcal{R}}\frac{p\times q^{\alpha-1} - p}{1-\alpha}\mathrm{d}\mu.$$

In order to invoke the monotone convergence theorem, we prove that the integrand is non-decreasing and bounded below as $\alpha\downarrow 1$. Noting that

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{p\times q^{\alpha-1}-p}{1-\alpha} = \frac{pq^{\alpha-1}(1-(\alpha-1)\log(q))-p}{(\alpha-1)^2},$$

it is enough to show that

$$pq^{\alpha-1}(1-(\alpha-1)\log(q)) - p \le 0.$$

Indeed, we have

$$\begin{aligned}
p[q^{\alpha-1}(1+\log(q^{1-\alpha}))-1] &\le p[q^{\alpha-1}(1+q^{1-\alpha}-1)-1] \qquad (2.21)\\
&= p[1-1]\\
&= 0,
\end{aligned}$$

where (2.21) holds since $\log(x)\le x-1$, for $x>0$. We next show that the integrand is bounded from below. The lower bound can be obtained by letting $\alpha \to \infty$:

$$\lim_{\alpha \to \infty} \frac{p \times q^{\alpha-1} - p}{1-\alpha} = 0.$$

Hence, by the monotone convergence theorem, we have that

$$\lim_{\alpha \downarrow 1} \int_{\mathcal{R}} \frac{p \times q^{\alpha-1} - p}{1-\alpha} \mathrm{d}\mu = \int_{\mathcal{R}} \lim_{\alpha \downarrow 1} \frac{p \times q^{\alpha-1} - p}{1-\alpha} \mathrm{d}\mu.$$

Finally, by L'Hôpital's rule, we obtain that

$$\begin{aligned}\lim_{\alpha \downarrow 1} \frac{x_\alpha - 1}{1-\alpha} &= -\int_{\mathcal{R}} \frac{p \log(q)}{1} \mathrm{d}\mu \\ &= h(p; q).\end{aligned}$$

Next, we shall prove taking $\alpha \uparrow 1$. For $\alpha < 1$, $x_\alpha = \underset{\mathbf{A} \sim p}{\mathbb{E}} (q(\mathbf{A})^{\alpha-1}) < \infty$ by the fact that x_α is non-increasing in $\alpha > 0$ and by the assumption that $\underset{\mathbf{A} \sim p}{\mathbb{E}} \left(\frac{1}{q(\mathbf{A})} \right) < \infty$. Hence, $\lim_{\alpha \uparrow 1} x_\alpha = 1$. Using Lemma 1, we can apply the same steps as above with the alteration of the following argument. Consider

$$\lim_{\alpha \uparrow 1} \frac{x_\alpha - 1}{1-\alpha} = \lim_{\alpha \uparrow 1} \int_{\mathbb{X}} \frac{pq^{\alpha-1} - p}{1-\alpha} \mathrm{d}\mu.$$

We know that the integrand is non-increasing in $\alpha > 0$. To use the monotone convergence theorem, we need to show that the integrand is bounded above. Note that

$$\begin{aligned}\lim_{\alpha \to 0} \frac{p \times q^{\alpha-1} - p}{1-\alpha} &= \frac{p}{q} - p \\ &\leq \frac{p}{q}.\end{aligned}$$

Since we assumed $\underset{\mathbf{A}\sim p}{\mathbb{E}}\left(\frac{1}{q(\mathbf{A})}\right) < \infty$, we have that $\frac{p}{q} < \infty$ (a.e.). As a result, the integrand is bounded above (a.e.). Following the same steps as in the previous part, we conclude that

$$\lim_{\alpha\uparrow 1} h_\alpha(p;q) = h(p;q),$$

finishing the proof. ■

We next show that the differential Rényi cross-entropy is monotonically decreasing in $\alpha > 0$, $\alpha \neq 1$.

Theorem 3 *Let p and q be two probability density functions with common support $\mathcal{R} \subset \mathbb{R}$. Then for $\alpha > 0$, $\alpha \neq 1$, $h_\alpha(p;q)$ is monotonically decreasing in α.*

Proof We first consider

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\alpha}h_\alpha(p;q) &= \frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\frac{1}{1-\alpha}\log\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right)\right)\\ &= \frac{1}{(1-\alpha)^2}\log\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right) + \frac{1}{1-\alpha}\frac{1}{\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu}\frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right).\end{aligned}$$

We next consider

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right) &= \lim_{\epsilon\downarrow 0}\int_{\mathcal{R}} p\left(\frac{q^{\alpha+\epsilon-1}-q^{\alpha-1}}{\epsilon}\right)\mathrm{d}\mu\\ &= \lim_{\epsilon\downarrow 0}\int_{\mathcal{R}} pq^{\alpha-1}\left(\frac{q^{\epsilon}-1}{\epsilon}\right)\mathrm{d}\mu.\end{aligned}$$

In order to apply the monotone convergence theorem, we prove the integrand is decreasing in $\epsilon \in (0,1)$ as $\epsilon \downarrow 0$ and is bounded above. Note that

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon}\frac{q^\epsilon - 1}{\epsilon} = \frac{1 + q^\epsilon(\epsilon\log(q) - 1)}{\epsilon^2},$$

hence, it is enough to show that

$$1 + q^\epsilon(\epsilon \log(q) - 1) \geq 0.$$

Indeed, we have

$$\begin{aligned} 1 + q^\epsilon(\log(q^\epsilon) - 1) &\geq 1 + q^\epsilon\left(1 - \frac{1}{q^\epsilon} - 1\right) \qquad (2.22) \\ &= 0, \end{aligned}$$

where (2.22) holds since $1 - \frac{1}{x} \leq \log(x)$, for all $x > 0$. We now show that the integrand is bounded above by letting $\epsilon \to 1$:

$$\lim_{\epsilon \to 1} \frac{q^\epsilon - 1}{\epsilon} = q - 1 < \infty.$$

Hence by the monotone convergence theorem we have

$$\lim_{\epsilon \downarrow 0} \int_{\mathcal{R}} pq^{\alpha-1}\left(\frac{q^\epsilon - 1}{\epsilon}\right) \mathrm{d}\mu = \int_{\mathcal{R}} \lim_{\epsilon \downarrow 0} pq^{\alpha-1}\left(\frac{q^\epsilon - 1}{\epsilon}\right) \mathrm{d}\mu.$$

By L'Hôpital's rule, we obtain that

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right) = \int_{\mathcal{R}} pq^{\alpha-1}\log(q)\mathrm{d}\mu.$$

Thus we have that

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}h_\alpha(p;q) = \frac{1}{(1-\alpha)^2}\log\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right) + \frac{1}{1-\alpha}\frac{1}{\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu}\int_{\mathcal{R}} pq^{\alpha-1}\log(q)\mathrm{d}\mu$$

$$= \frac{\int_{\mathcal{R}} pq^{\alpha-1}(h_\alpha(p;q)+\log(q))\mathrm{d}\mu}{(1-\alpha)\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right)}.$$

We next consider two cases, $\alpha < 1$ and $\alpha > 1$. For $\alpha < 1$, we consider

$$\begin{aligned} h_\alpha(p;q)+\log(q) &= \frac{1}{1-\alpha}\log(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu) + \frac{\log(q^{1-\alpha})}{1-\alpha} \\ &= \frac{\log(q^{1-\alpha}\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu)}{1-\alpha} \\ &\le \frac{q^{1-\alpha}\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu - 1}{1-\alpha}, \end{aligned} \tag{2.23}$$

where (2.23) holds since $\log(x) \le x-1$, for all $x > 0$. We thus have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\alpha}h_\alpha(p;q) &\le \frac{\int_{\mathcal{R}} pq^{\alpha-1}(q^{1-\alpha}\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu - 1)\mathrm{d}\mu}{(1-\alpha)^2\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right)} \\ &= \frac{\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu - \int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu}{(1-\alpha)^2\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right)} \\ &= 0. \end{aligned}$$

Finally, for $\alpha > 1$, we consider

$$\begin{aligned} h_\alpha(p;q)+\log(q) &= \frac{\log(q^{1-\alpha}\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu)}{1-\alpha} \\ &\ge \frac{q^{1-\alpha}\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu - 1}{1-\alpha}, \end{aligned} \tag{2.24}$$

where (2.24) holds since $-\log(x) \ge -(x-1)$, for all $x > 0$. Thus, we have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\alpha}h_\alpha(p;q) &\le \frac{\int_{\mathcal{R}} pq^{\alpha-1}(q^{1-\alpha}\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu - 1)\mathrm{d}\mu}{(1-\alpha)^2\left(\int_{\mathcal{R}} pq^{\alpha-1}\mathrm{d}\mu\right)} \\ &= 0, \end{aligned}$$

which finishes the proof. ■

The above definition of differential Rényi cross-entropy can be extended (assuming the integral exists) by only requiring q to be a non-negative function (such as a non-normalized density function); in this case we call the resulting measure as the (differential) *Rényi cross-entropy functional* and denote it by $\mathcal{H}_\alpha(p;q)$. Similarly, we henceforth denote the Shannon cross-entropy functional by $\mathcal{H}(p;q)$.

2.2 Neural networks

Modelled after the human brain, neural networks are functions that consist of superpositions of activation functions and affine linear transformations. In this section, we present the basic definitions of neural networks and a widely used gradient descent optimization technique called Adaptive Moment Estimation (Adam) [42].

2.2.1 Propagation equations

We follow the treatment provided in [31] unless otherwise stated. Let $\mathbb{X} \subset \mathbb{R}$ be the space of input signals and $\mathbb{Y} \subset \mathbb{R}$ be the space of output signals. We denote

$$\begin{aligned} C^0(\mathbb{X},\mathbb{Y}) &:= \{f : \mathbb{X} \to \mathbb{Y} : f \text{ is continuous.}\}, \\ C^n(\mathbb{X},\mathbb{Y}) &:= \{f : \mathbb{X} \to \mathbb{Y} : f \text{ is } n\text{-times continuously differentiable}\}, \end{aligned}$$

where $n \in \mathbb{N}$.

Definition 18 *[48] An **activation function** $\sigma : \mathbb{X} \to \mathbb{Y}$ is a function that is continuous (a.e.).*

Examples of common activation functions are the Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{if } x \geq 0, \end{cases}$$

the Leaky Rectified Linear Unit (LeakyReLU) with parameter $\alpha \in \mathbb{R}_{>0}$ [31],

$$\mathrm{LeakyReLU}_\alpha(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha x, & \text{if } x \leq 0, \end{cases} \tag{2.25}$$

$\sigma(x) = \tanh(x)$, and $\sigma(x) = \frac{1}{1+e^{-x}}$. Often $\sigma(x) = \tanh(x)$ and $\sigma(x) = \frac{1}{1+e^{-x}}$ are referred to as sigmoid functions, sigmoidal functions [22], or squashing functions [38]. We primarily use ReLU, LeakyReLU, $\tanh(x)$, and $\frac{1}{1+e^{-x}}$ as our activation functions in this thesis.

Definition 19 *A **neuron** is a function $f : \mathbb{X}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{Y}$ such that for an input, $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{X}^n$, weight vector $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$, and bias $b \in \mathbb{R}$,*

$$f(\boldsymbol{x}, \boldsymbol{w}, b) = \sigma\left(\sum_{k=1}^{n} x_k w_k + b\right), \tag{2.26}$$

where $\sigma : \mathbb{X} \to \mathbb{Y}$ is an activation function.

Figure 2.1 is a representation of a neuron. It can be shown that neurons can approximate any $C^1(\mathbb{X}^n, \mathbb{Y})$ function [48].

Neural networks consist of neurons arranged in layers, which correspond to superpositions of activation functions and affine linear transforms, see Figure 2.2. Let $\mathbb{X}^n$ be the input space and $\mathbb{Y}^m$ be the output space of the neural network. Suppose

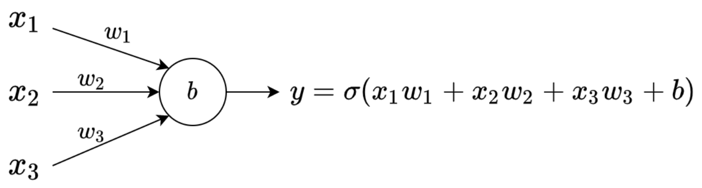


Figure 2.1: A neuron.

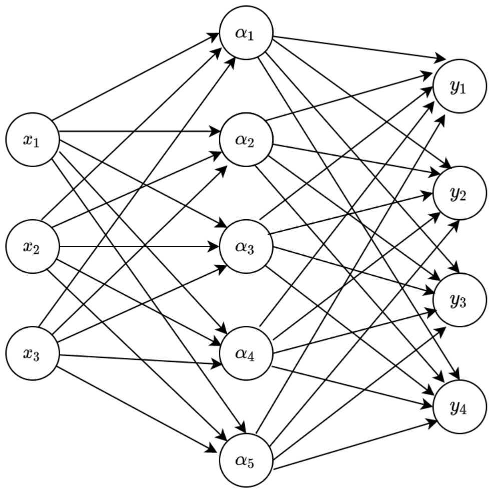


Figure 2.2: A neural network with three layers that maps $(x_1, x_2, x_3) \mapsto (y_1, y_2, y_3, y_4)$.

a neural network has k layers, where first layer is the input layer and the kth layer is the output layer. We denote $l_j \in \mathbb{N}$ as the number of neurons in layer $j \in \{1, \ldots, k\}$. Note that $l_1 = n$ and $l_k = m$.

Definition 20 *We call* $\mathbf{w}^{(j)} \in \mathbb{R}^{l_j} \times \mathbb{R}^{l_{j-1}}$ *the* ***weight matrix*** *of layer* j *and* $\boldsymbol{b}^{(j)} \in \mathbb{R}^{l_j}$ *the* ***bias vector*** *of layer* j*, where* $j \in \{2, \ldots, k\}$.

Note that we define $w_{h,i}^{(j)} \in \mathbb{R}$ as the weight of the connection from the hth neuron from the $(j-1)$th layer to the ith neuron in the jth layer. Hence $\mathbf{w}^{(j)} = \left[w_{h,i}^{(j)} \right]_{h,i}^{T}$. Similarly $b_i^{(j)} \in \mathbb{R}$ is the bias associated with the ith neuron in the jth layer. The function $\boldsymbol{\sigma}^{(j)} : \mathbb{R}^{l_j} \to \mathbb{R}^{l_j}$ applies an identical activation function, $\sigma^{(j)} : \mathbb{R} \to \mathbb{R}$, component-wise, i.e., $\boldsymbol{\sigma}^{(j)}(a_1, \ldots, a_{l_j}) = (\sigma^{(j)}(a_1), \ldots, \sigma^{(j)}(a_{l_j}))$, where $a_i \in \mathbb{R}$, for $i \in \{1, \ldots, l_j\}$.

Definition 21 *A* ***neural network with k layers*** *is a function $f_{\boldsymbol{\theta}} : \mathbb{X}^n \to \mathbb{Y}^m$ of the form*

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\sigma}^{(k)}(\mathbf{w}^{(k)}\boldsymbol{\sigma}^{(k-1)}(\mathbf{w}^{(k-1)}(\ldots\boldsymbol{\sigma}^{(2)}(\mathbf{w}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)})) + \boldsymbol{b}^{(k-1)}) + \boldsymbol{b}^{(k)}), \tag{2.27}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{X}^n$ is the input and $\boldsymbol{\theta} = (\mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(k)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(k)})$ is called the ***tuple of parameters*** *(or* ***parameters*** *for short) of the neural network.*

For a fixed $\boldsymbol{x} \in \mathbb{X}^n$, we let $\alpha_i^{(j)}$ be the **activation of the ith neuron in layer** $j \in \{1, \ldots, k\}$ defined it recursively as follows:

$$\alpha_i^{(j)} := \begin{cases} x_i, & \text{if } j = 1, \\ \sigma^{(j)}\left(\sum_{h=1}^{l_{j-1}} w_{h,i}^{(j)} \alpha_h^{(j-1)} + b_i^{(j)}\right), & \text{if } j \in \{2, \ldots, k\}, \end{cases} \tag{2.28}$$

where $i \in \{1, \ldots l_j\}$. Equivalently, in vector form,

$$\boldsymbol{\alpha}^{(j)} := (\alpha_1^{(j)}, \ldots, \alpha_{l_j}^{(j)}) = \begin{cases} \boldsymbol{x}, & \text{if } j = 1, \\ \boldsymbol{\sigma}^{(j)}(\mathbf{w}^{(j)}\boldsymbol{\alpha}^{(j-1)} + \boldsymbol{b}^{(j)}), & \text{if } j \in \{2, \ldots, k\}, \end{cases} \tag{2.29}$$

where $\boldsymbol{\alpha}^{(j)} \in \mathbb{R}^{l_j}$. Note that we denote $y_i := \alpha_i^{(k)}$ and $\boldsymbol{y} := \{y_1, \ldots, y_m\} \in \mathbb{Y}^m$.

We denote

$$z_i^{(j)} := \sum_{h=1}^{l_{j-1}} w_{h,i}^{(j)} \alpha_h^{(j-1)} + b_i^{(j)}, \tag{2.30}$$

where $i \in \{1, \ldots l_j\}$ is the neuron in layer $j \in \{2, \ldots, k\}$. Equivalently, in vector form,

$$\boldsymbol{z}^{(j)} := \mathbf{w}^{(j)}\boldsymbol{\alpha}^{(j-1)} + \boldsymbol{b}^{(j)}, \tag{2.31}$$

where $\boldsymbol{z}^{(j)} \in \mathbb{R}^{l_j}$.

When constructing a neural network, the weights and the biases are the parameters that can be fine tuned to optimize the neural network. There are several tasks that a neural network can perform, such as classification, regression, or density estimation [31]. The two main types of learning algorithms for neural networks to achieve their task are supervised and unsupervised learning.

Supervised learning algorithms learn features of a dataset that contains a desired output for a given input from the dataset. For example, the MNIST dataset contains handwritten numerical images with labels indicating the value for each image [47]. As such, we can train a neural network to classify handwritten numerical images.

Definition 22 *Let $\boldsymbol{a} \in \mathbb{Y}^m$ be the* ***desired output*** *for some given input $\boldsymbol{x} \in \mathbb{X}^n$. We call $(\boldsymbol{a}, \boldsymbol{x})$ the* ***training data****.*

To train the neural network, we need to define a loss function to measure how well a neural network is performing.

Definition 23 *The* ***loss function*** *is defined as a continuous function $V : \mathbb{Y}^m \times \mathbb{Y}^m \to \mathbb{R}$, which measures the difference between the desired output $\boldsymbol{a}$ and the* ***actual output*** *$f_{\boldsymbol{\theta}}(\boldsymbol{x})$, where $f_{\boldsymbol{\theta}} : \mathbb{X}^n \to \mathbb{Y}^m$ is a neural network. That is $(\boldsymbol{a}, f_{\boldsymbol{\theta}}(\boldsymbol{x})) \mapsto V(\boldsymbol{a}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$.*

Note that in the literature, loss functions are also known as cost, objective, or value functions. Loss functions do not need to be a metric on $\mathbb{Y}^m$. Examples of common loss functions are the square error

$$V(\boldsymbol{a}, \boldsymbol{y}) = \|\boldsymbol{a} - \boldsymbol{y}\|_2$$

or the Shannon cross-entropy functional

$$V(\boldsymbol{a}, \boldsymbol{y}) = \mathcal{H}(\boldsymbol{a}; \boldsymbol{y}) = -\left(\sum_{i=1}^{m} a_i \ln(y_i) \;\; + (1 - a_i) \ln(1 - y_i)\right),$$

if $\boldsymbol{a}$ is a probability mass function and $y_i \in (0, 1)$, $i \in \{1, \ldots, m\}$, where $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the actual output of a neural network, $f_{\boldsymbol{\theta}}$, with k layers, for some given training data $(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{Y}^m \times \mathbb{X}^n$.

Unsupervised learning algorithms learn features of an unlabelled dataset. This can be done by learning the underlying distribution of the dataset, either explicitly or implicitly. A popular unsupervised learning algorithm is generative adversarial networks (GANs). It contains a generator neural network, whose task is to generate data that mimics some unlabelled dataset, and a discriminator neural network, whose task is to distinguish real and fake data. As such the generator's loss function measures whether the generator manages to fool the discriminator into labelling real and fake data as the same value. GANs, which are the subject of this thesis, will be described in detail in the next chapter.

2.2.2 Backpropagation equations

Backpropagation equations were first introduced by Rumelhart, Hinton, and Williams in their seminal 1986 paper [72]. The aim of backpropagation equations is to find $\frac{\partial V}{\partial w_{h,i}^{(j)}}$ and $\frac{\partial V}{\partial b_i^{(j)}}$ so that we can minimize the loss function over the parameters of the neural network that we can control.

Theorem 4 *[72] [31] [62] Let $f_{\boldsymbol{\theta}} : \mathbb{X}^n \to \mathbb{Y}^m$ be a neural network with k layers, $l_j \in \mathbb{N}$ be the number of neurons in layer $j \in \{1, \dots k\}$, and $\boldsymbol{x} \in \mathbb{X}^n$. Let $\boldsymbol{y} = (y_1, \dots, y_m) := f_{\boldsymbol{\theta}}(x) \in \mathbb{Y}^m$ be the actual output. Then backpropagation equations of the neural network $f_{\boldsymbol{\theta}}$ are*

$$\frac{\partial V}{\partial b_i^{(j)}} = \frac{\partial V}{\partial z_i^{(j)}} \tag{2.32}$$

$$\frac{\partial V}{\partial w_{h,i}^{(j)}} = \frac{\partial V}{\partial z_i^{(j)}} \alpha_h^{(j-1)}, \tag{2.33}$$

where $i \in \{1, \dots, l_j\}$ is the neuron in layer $j \in \{2, \dots, k\}$, and where $h \in \{1 \dots, l_{j-1}\}$ is the neuron in layer $j - 1$. Furthermore,

$$\frac{\partial V}{\partial z_i^{(j)}} = \begin{cases} \frac{\partial V}{\partial y_i} \sigma' \left(z_i^{(j)} \right), & \text{if } j = k, \\ \sum_{p=1}^{l_{j+1}} \frac{\partial V}{\partial z_p^{(j+1)}} w_{i,p}^{(j+1)} \sigma' \left(z_i^{(j)} \right), & \text{if } j \in \{2, \dots, k-1\}, \end{cases} \tag{2.34}$$

where $\sigma' \left(z_i^{(j)} \right) := \frac{\mathrm{d}\sigma(a)}{\mathrm{d}a} |_{a = z_i^{(j)}}$.

Proof We first prove (2.32) and (2.33). For neuron $i \in \{1, \dots, l_j\}$ in layer $j \in \{2, \dots, k\}$,

$$\frac{\partial V}{\partial z_i^{(j)}} = \frac{\partial V}{\partial b_i^{(j)}} \frac{\mathrm{d} b_i^{(j)}}{\mathrm{d} z_i^{(j)}} = \frac{\partial V}{\partial b_i^{(j)}}, \tag{2.35}$$

where (2.35) is due to the chain rule, and $\frac{\mathrm{d}b_i^{(j)}}{\mathrm{d}z_i^{(j)}} = 1$ from (2.30). For neuron $h \in \{1, \ldots l_j\}$ in the jth layer and neuron $i \in \{1, \ldots l_{j-1}\}$ in the $(j-1)$th layer, we have that

$$\begin{aligned}\frac{\partial V}{\partial w_{h,i}^{(j)}} &= \sum_{p=1}^{l_j} \frac{\partial V}{\partial z_p^{(j)}} \frac{\partial z_p^{(j)}}{\partial w_{h,i}^{(j)}} \qquad (2.36)\\ &= \sum_{p=1}^{l_j} \frac{\partial V}{\partial z_p^{(j)}} \frac{\partial}{\partial w_{h,i}^{(j)}} \left(\sum_{m=1}^{l_{j-1}} w_{m,p}^{(j)} \alpha_m^{(j-1)} + b_p^{(j)} \right),\end{aligned}$$

where (2.36) is due to the chain rule and product rule. Moreover,

$$\frac{\partial}{\partial w_{h,i}^{(j)}} \left(\sum_{m=1}^{l_{j-1}} w_{m,p}^{(j)} \alpha_m^{(j-1)} + b_p^{(j)} \right) = \begin{cases} \alpha_h^{(j-1)}, \text{ if } p = i, \\ 0, \text{ otherwise.} \end{cases}$$

Thus,

$$\frac{\partial V}{\partial w_{h,i}^{(j)}} = \frac{\partial V}{\partial z_i^{(j)}} \alpha_h^{(j-1)},$$

We next relate $\frac{\partial V}{\partial z_i^{(k)}}$ to $\frac{\partial V}{\partial y_i}$, where $i \in \{1, \ldots, l_k\}$ is the neuron in the output layer. By the fact that $y_i = \sigma(z_i^{(k)})$ for neuron $i \in \{1, \ldots, l_k\}$ in the output layer, we have that

$$\frac{\partial y_p}{\partial z_i^{(k)}} = \begin{cases} \sigma' \left(z_i^{(k)} \right), & \text{if } p = i, \\ 0, & \text{otherwise,} \end{cases}$$

where $p \in \{1, \dots l_k\}$. Thus,

$$\frac{\partial V}{\partial z_i^{(k)}} = \sum_{p=1}^{l_k} \frac{\partial V}{\partial y_p} \frac{\partial y_p}{\partial z_i^{(k)}} = \frac{\partial V}{\partial y_i} \sigma' \left(z_i^{(k)} \right), \tag{2.37}$$

where (2.37) is due to the chain rule and product rule.

We next relate $\frac{\partial V}{\partial z_i^{(j)}}$ for neuron $i \in \{1, \dots, l_j\}$ in layer $j \in \{2, \dots, k-1\}$ to $\frac{\partial V}{\partial z_p^{(j+1)}}$, where $p \in \{1, \dots, l_{j+1}\}$ is the neuron in layer $j+1$. Firstly,

$$\begin{aligned} z_p^{(j+1)} &= \sum_{h=1}^{l_j} w_{h,p}^{(j+1)} \alpha_h^{(j)} + b_p^{(j+1)} \\ &= \sum_{h=1}^{l_j} w_{h,p}^{(j+1)} \sigma \left(z_h^{(j)} \right) + b_p^{(j+1)} \\ \Rightarrow \frac{\partial z_p^{(j+1)}}{\partial z_i^{(j)}} &= w_{i,p}^{(j+1)} \sigma' \left(z_i^{(j)} \right). \end{aligned}$$

Hence,

$$\begin{aligned} \frac{\partial V}{\partial z_i^{(j)}} &= \sum_{p=1}^{l_{j+1}} \frac{\partial V}{\partial z_p^{(j+1)}} \frac{\partial z_p^{(j+1)}}{\partial z_i^{(j)}} \qquad (2.38) \\ &= \sum_{p=1}^{l_{j+1}} \frac{\partial V}{\partial z_p^{(j+1)}} w_{i,p}^{(j+1)} \sigma' \left(z_i^{(j)} \right), \end{aligned}$$

where (2.38) is due to the chain rule and product rule. ■

2.2.3 Gradient descent algorithm: Adam optimization

Simply applying gradients to the weights and biases in a neural network does not guarantee that it will converge to its local optimum, let alone its global optimum. Indeed, there are no gradient descent algorithms that guarantee that a non-convex function whose parameters lie on a non-convex space, such as a neural network, will converge to a local or global optimum. Moreover, loss functions have randomness due to the fact that they are evaluated using different subsamples of data. Random noise is also commonly added in network architectures, such as in StyleGANs [41], which contribute to the randomness of loss functions. For such loss functions, stochastic gradient descent (SGD) algorithms are useful.

One such popular SGD algorithm is called Adaptive Moment Estimation (Adam). Related to RMSProp and AdaGrad [24] [31] [42], Adam uses first order gradients, initialization bias correction, and adaptive learning rates by calculating the first and second moments of the gradients. We include the algorithm below (see Algorithm 1).

Let $(X_1, \ldots, X_T) \subset \mathbb{X}$ be random samples taken independently from some probability distribution p defined on $\mathbb{X}$, where $T \in \mathbb{N}$ is the time horizon. The distribution p can be a uniform distribution if $\mathbb{X}$ is a countable finite set or a Gaussian distribution if $\mathbb{X} \subset \mathbb{R}$. Recall that an estimator $\hat{\phi} : \mathbb{X}^T \to \mathbb{R}$ for an unknown parameter ϕ is called unbiased if $\mathbb{E}(\hat{\phi}(X_1, \ldots, X_T)) = \phi$ [70].

Let $((\boldsymbol{a}_1, \boldsymbol{x}_1), \ldots, (\boldsymbol{a}_T, \boldsymbol{x}_T)) \subset \mathbb{Y}^m \times \mathbb{X}^n$ be the tuple of training data. Let $V(\boldsymbol{a}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ be the loss function, where $f_{\boldsymbol{\theta}} : \mathbb{X}^n \to \mathbb{Y}^m$ is a neural network with k layers. Note the goal is to find $\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} \mathbb{E}(V(\boldsymbol{a}, f_{\boldsymbol{\theta}}(\boldsymbol{x})))$, where the randomness arises from randomly sampling the training data, or random noise added to it. Let $V(\boldsymbol{a}_t, f_{\boldsymbol{\theta}}(\boldsymbol{x}_t))$ be the realizations of the objective function at each time step $t \in \{1, \ldots, T\}$.

Using the definitions of $\hat{\boldsymbol{m}}_t$, $\hat{\boldsymbol{v}}_t$, and $\boldsymbol{g}_t$ from the Adam algorithm, we next show that $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{v}}_t$ are unbiased estimators of the gradient's, $\boldsymbol{g}_t$, first and second moments provided some condition on $\boldsymbol{g}_t$.

Algorithm 1 Overview of Adam algorithm

```
**Initialize** $\boldsymbol{\theta}_0$ by randomly sampling a probability distribution, step-size $\alpha \in \mathbb{R}_{>0}$, first moment vector $\boldsymbol{m}_0 = \mathbf{0}$, second moment vector $\boldsymbol{v}_0 = \mathbf{0}$, exponential decay rates $\beta_1$, $\beta_2 \in [0, 1)$, and timestep $t = 0$.
**while** $t \in \{0, \ldots T\}$ **do**
  $t = t + 1$,
  $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} V(\boldsymbol{a}_t, f_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{x}_t))$,
  $\boldsymbol{m}_t = \beta_1 \cdot \boldsymbol{m}_{t-1} + (1 - \beta_1) \cdot \boldsymbol{g}_t$,
  $\boldsymbol{v}_t = \beta_2 \cdot \boldsymbol{v}_{t-1} + (1 - \beta_2) \cdot \boldsymbol{g}_t^2$,
  $\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}$, bias corrected estimate of $\mathbb{E}(\boldsymbol{g}_t)$,
  $\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}$, bias corrected estimate of $\mathbb{E}(\boldsymbol{g}_t^2)$,
  $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \cdot \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t + \boldsymbol{\epsilon}}}$, where $\boldsymbol{\epsilon}$ is chosen close to the zero vector and has positive entries. [1]
**end while**
```

Theorem 5 *If $\mathbb{E}(\boldsymbol{g}_t) = \boldsymbol{a}$ and $\mathbb{E}(\boldsymbol{g}_t^2) = \boldsymbol{b}$ for all $t \in \{1, \ldots, T\}$, then $\mathbb{E}(\hat{\boldsymbol{m}}_t) = \boldsymbol{a}$ and $\mathbb{E}(\hat{\boldsymbol{v}}_t) = \boldsymbol{b}$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are constant vectors.*

Proof It is direct to determine that $\boldsymbol{m}_t = (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} \boldsymbol{g}_i$ and $\boldsymbol{v}_t = (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} \boldsymbol{g}_i^2$ by recursion.

We prove that $\mathbb{E}(\hat{\boldsymbol{m}}_t) = a$; the other proof is identical.

$$\begin{aligned}
\mathbb{E}(\hat{\boldsymbol{m}}_t) &= \mathbb{E}\left(\frac{1-\beta_1}{1-\beta_1^t} \sum_{i=1}^t \beta_1^{t-i} \boldsymbol{g}_i\right) \\
&= \frac{1-\beta_1}{1-\beta_1^t} \sum_{i=1}^t \beta_1^{t-i} \mathbb{E}(\boldsymbol{g}_i) \\
&= \boldsymbol{a} \frac{1-\beta_1}{1-\beta_1^t} \sum_{i=1}^t \beta_1^{t-i} \\
&= \boldsymbol{a}, \qquad (2.39)
\end{aligned}$$

[1] Note that $\frac{\boldsymbol{a}}{\boldsymbol{b}}$ in the algorithm above denotes the element wise division of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of the same size.

where (2.39) is due to the finite geometric series. ■

The recommended values of the Adam algorithm are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\boldsymbol{\epsilon} = (10^{-8}, \ldots, 10^{-8})$. Although the Adam algorithm is commonly used, it does not guarantee that $\boldsymbol{\theta}_t$ will converge to a global optimal $\boldsymbol{\theta}^*$. In fact, researchers proved that there exists an online convex optimization problem where the Adam algorithm fails to converge to an optimal solution [68]; selecting β_1 and β_2 to vary with time is suggested as a remedy.

2.2.4 Convolutional neural networks

In computer vision, convolving kernels (or also known as filters, masks, templates, or windows) onto images can give information about the spacial dependencies between pixels [29]. The concept of filtering descends directly from Fourier transforms for signal processing [29]. Kernels can sharpen images, increase their contrast, reduce noise, and detect edges of objects and features in the image [29]. The same theory can be applied to neural networks that take images as their inputs. Instead of flattening out the images, we can apply kernels and train the neural network to pick out spacial features and patterns, which in practice improves the accuracy of these networks [31].

We provide the definitions of images, kernels, and convolutions below, which can be found in Chapter 3 of [29] and Chapter 9 of [31].

Definition 24 *An* ***image*** $\mathbf{x}$ *of size* $m \times n \times p$ *is* p *matrices of size* $m \times n$*, where* $p \in \mathbb{N}$ *is the number of* ***channels****, where* ***pixel at*** $\boldsymbol{(i, j)}$ ***in channel*** $\boldsymbol{k}$ *refers to the location* (i, j, k) *in the image* $\mathbf{x}$*, and where* $\mathbf{x}(i, j, k)$ *is the* ***pixel value at*** $\boldsymbol{(i, j)}$ ***in***

channel k. *That is*

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}(0,0,1) & \dots & \mathbf{x}(0,n-1,1) \\ \mathbf{x}(1,0,1) & \dots & \mathbf{x}(1,n-1,1) \\ \vdots & \ddots & \vdots \\ \mathbf{x}(m-1,0,1) & \dots & \mathbf{x}(m-1,n-1,1) \end{pmatrix} \cdots \begin{pmatrix} \mathbf{x}(0,0,p) & \dots & \mathbf{x}(0,n-1,p) \\ & & \mathbf{x}(1,n-1,p) \\ & & \vdots \\ & & \mathbf{x}(m-1,n-1,p) \end{pmatrix} \tag{2.40}$$

Note that we use the shorthand *pixel* (i, j, k) to refer to the pixel (i, j) in channel k. Often pixels values are $\mathbf{x}(i, j, k) \in \{0, \dots, 255\}$, where pixel value $\mathbf{x}(i, j, k) = 0$ is the darkest pixel value, and $\mathbf{x}(i, j, k) = 255$ is the brightest. Note henceforth, we assume that $\mathbf{x}(i, j, k) \in \mathbb{R}$. A black and white image has only one channel, $p = 1$. A coloured image usually has three channels, $p = 3$, which are red, blue, and green (RBG) [29] [31].

Note that a **kernel k of size $\boldsymbol{r \times t \times w}$** is defined similarly to images, however, pixels of the kernel are called **coefficients**, and the coefficients value can be real numbers [29]. Furthermore, we shall assume that $w = p$, that is the kernel has the same number of channels as the image it is convolved with.

We next define the linear convolutional operation.

Definition 25 *Let* $\mathbf{x}$ *be an input image that is of size* $m \times n \times p$ *and let* $\mathbf{k}$ *be a kernel of size* $r \times t \times p$. *A* ***linear convolutional operation in a single channel k***, $\mathbf{k} * \mathbf{x}$, *on some pixel* (i, j, k), *where* $k \in \{1, \dots, p\}$, *is defined as*

$$\mathbf{k} * \mathbf{x}(i, j, k) := \sum_{u \in [r]} \sum_{v \in [t]} \mathbf{x}(i - u, j - v, k)\mathbf{k}(u, v, k), \tag{2.41}$$

*where $[a] := \{0, \ldots, a-1\}$ for some $a \in \mathbb{N}$. The **linear convolutional operation over all channels** is defined as*

$$\mathbf{k} * \mathbf{x}(i,j) := \sum_{k=1}^{p} \mathbf{k} * \mathbf{x}(i,j,k). \tag{2.42}$$

Note that $\mathbf{k} * \mathbf{x}(i,j) \in \mathbb{R}$. For the sake of brevity, linear convolutional operations will be referred to as convolutions. Henceforth, we will also be using square images of size $n \times n \times p$, and square kernels of size $r \times r \times p$.

To apply convolutions on the pixels near the edge of the input image, the image is padded with zeros (also called **zero-padding**). If the image is not zero-padded, then the network shrinks quickly or we have to use small kernels, which limits the network's expressive power [31]. However, zero-padding lowers the values of pixels on the edge, which in practical terms makes the border pixels darker than they are [29]. One way to combat this is to use cyclic convolutional operation, which is defined as

$$\mathbf{k} * \mathbf{x}(i,j) := \sum_{k=1}^{p} \sum_{u \in [r]} \sum_{v \in [r]} \mathbf{x}(i-u \pmod r, j-v \pmod r, k)\mathbf{k}(u,v,k).$$

This is equivalent to padding the image with pixels from the opposite edge of the image [40] [29]. Another padding technique is to interpolate the pixel values beyond the image size [29]. Let $t \in \mathbb{Z}_{\geq 0}$ be the number of pixels padding the image in each direction of the image. We then write the convolution between kernel $\mathbf{k}$ and image $\mathbf{x}$ with padding $t \in \mathbb{Z}_{\geq 0}$ as $\mathbf{k} *_t \mathbf{x}(i,j)$. For example, if $t = 1$, the kernel is size $3 \times 3 \times 1$, and the image is size $5 \times 5 \times 1$, then the image is padded with one pixel in each direction to ensure that convoluting on the edge pixels, such as pixel $(0,0,1)$, is possible, as shown in Figure 2.3 below.

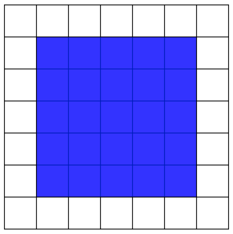


Figure 2.3: Padding a $5 \times 5 \times 1$ image with $t = 1$. The white pixels are the padding added to the image. The image is represented by the blue pixels.

Convolutions can be applied to certain pixels rather than every pixel in the image. This is called downsampling [31], which is a common practice to reduce computing time. If downsampling is periodic, it is called strides. We provide the definition of a convoluted image and the equations of strides below.

Definition 26 *The **convoluted image** $\mathbf{y}$ resulting from convoluting an input image $\mathbf{x}$ of size $n \times n \times p$ with kernel $\mathbf{k}$ of size $r \times r \times p$, padding t, and **stride** $s \in \mathbb{N}$ is defined as*

$$\mathbf{y} \;=\; [\mathbf{k} *_t \mathbf{x}(i,j)]_{i\times s, j\times s}\,. \tag{2.43}$$

We rewrite the right-hand expression in (2.43) as

$$\mathbf{k} *_{s,t} \mathbf{x} \;:=\; [\mathbf{k} *_t \mathbf{x}(i,j)]_{i\times s, j\times s}\,. \tag{2.44}$$

For an $n \times n \times p$ input image $\mathbf{x}$, kernel $\mathbf{k}$ of size $r \times r \times p$, convoluted with $\mathbf{x}$ with stride $s \in \mathbb{N}$, and padding of size $t \in \mathbb{N}$, the output image $\mathbf{y}$ is size $m \times m \times 1$ [25], where

$$m = \left\lfloor \frac{n + 2t - r}{s} \right\rfloor + 1. \tag{2.45}$$

Note that if the input image $\mathbf{x}$ is convoluted with k kernels $(\mathbf{k}_1, \ldots, \mathbf{k}_k)$, then the convoluted image $\mathbf{y}$ is size $m \times m \times k$. By abuse of notation, we write $\mathbf{y} = \mathbf{k} *_{s,t} \mathbf{x}$, where $\mathbf{k} = (\mathbf{k}_1, \ldots, \mathbf{k}_k)$ is the tuple of kernels. The value of the padding t, strides s, and kernels' dimensions r can increase the size of the output image, as shown below.

Example 6 *Assume that an input of a convolutional layer is an image of size* $5 \times 5 \times 1$*. Let us assume that the padding is* $t = 2$*, there are* 3 *kernels of size* $3 \times 3 \times 1$*, and the stride is* $s = 1$*. Then according to* (2.45)*, we have*

$$\begin{aligned} m &= \left\lfloor \frac{n + 2t - r}{s} \right\rfloor + 1 \\ &= 5 + 2 \cdot 2 - 3 + 1 \\ &= 7. \end{aligned}$$

Hence, the resultant image is $7 \times 7 \times 3$*; see Figure 2.4.*

We next define a convolutional neural network with k layers. Let $\mathbb{X} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^p$ be the input space of images of size $n \times n \times p$. Let l_j be the number of kernels in layer j, s_j be the stride in layer j, and t_j be the padding in layer j, where $j \in \{2, \ldots, k\}$.

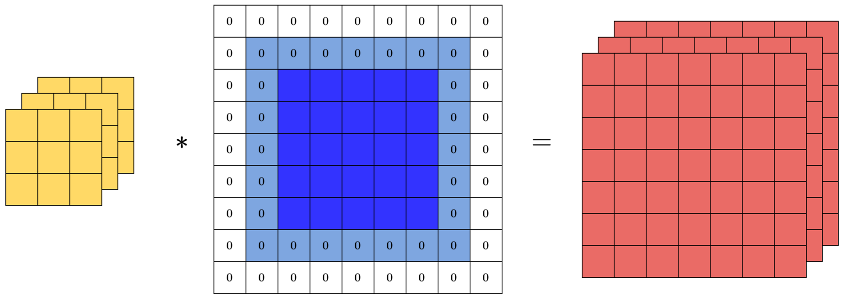


Figure 2.4: A visual representation of Example 6. The dark blue pixels are the pixels of the input image. The light blue and dark blue pixels are the pixels that the kernels are convoluted with.

We recursively define the **activated image** in layer $j \in \{1, \ldots, k\}$ as

$$\mathbf{a}^{(j)} := \begin{cases} \mathbf{x}, & \text{if } j = 1 \\ \boldsymbol{\sigma}^{(j)}(\mathbf{k}^{(j)} *_{s_j,t_j} \mathbf{a}^{(j-1)} + \mathbf{b}^{(j)}), & \text{if } j \in \{2, \ldots, k\}, \end{cases} \tag{2.46}$$

where $\mathbf{x} \in \mathbb{X}$ is the input image, and $\mathbf{k}^{(j)} = (\mathbf{k}_1^{(j)}, \ldots, \mathbf{k}_{l_j}^{(j)})$ is the tuple of kernels of size $r_j \times r_j \times l_{j-1}$ in layer j, where $l_1 = p$. The bias, $\mathbf{b}^{(j)}$, in layer j is of size $m_j \times m_j \times l_j$, where m_j is predicted by (2.45). After convoluting with $\mathbf{k}^{(j)}$ and adding the bias to each pixel, $\boldsymbol{\sigma}^{(j)}$ applies an identical activation function on each pixel.

Note that convolutional neural networks have an extra stage called the **pooling stage**, which summarises the statistics of the outputs in some neighbourhood of pixels (such as max pooling) [31]. Since this thesis focuses on deep convolutional GANs, pooling will not be used [67].

Definition 27 *Let $\mathbb{X} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^p$ be the space of $n \times n \times p$ images and $\mathbb{Y} \subset \mathbb{R}^{m_k} \times \mathbb{R}^{m_k} \times \mathbb{R}^{l_k}$ be the output space. A **convolutional neural network with k***

layers *is a function* $f_{\boldsymbol{\theta}} : \mathbb{X} \to \mathbb{Y}$ *of the form*

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\sigma}^{(k)}(\mathbf{k}^{(k)} *_{s_k,t_k} \boldsymbol{\sigma}^{(k-1)} \dots \boldsymbol{\sigma}^{(2)}(\mathbf{k}^{(2)} *_{s_2,t_2} \mathbf{x} + \mathbf{b}^{(2)}) + \mathbf{b}^{(k)}), \tag{2.47}$$

where $\mathbf{x} \in \mathbb{X}$ *is the input image,* $\mathbf{k}^j = (\mathbf{k}_1^{(j)}, \dots, \mathbf{k}_{l_j}^{(j)})$ *is the tuple of kernels in layer* j*,* $\mathbf{b}^{(j)}$ *is the bias in layer* j*,* $\boldsymbol{\theta} = (\mathbf{k}^{(2)}, \dots, \mathbf{k}^{(k)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(k)})$ *is tuple of kernels and biases, called the* ***parameters of the convolutional neural network****, and* $\boldsymbol{\sigma}^{(j)}$ *is the activation function that is applied to every pixel in the resulting convolved image.*

Note that convolutions are equivalent to multiplying the input by a Toeplitz or cyclic matrix [40] [31]. Since convolutions can be rewritten as multiplying the input with a Toeplitz or cyclic matrix, the backpropagation equations of convolutional neural networks are identical to neural networks. Convolutional neural networks are superior to regular neural networks because of parameter sharing [31]. This reduces memory overhead and improves the efficiency of training. Moreover, it allows the neural network to learn the spacial dependencies between pixels of images.

Chapter 3

Generative adversarial networks

Generative adversarial networks (GANs) were introduced in [32]. GANs use two neural networks, one discriminator and one generator neural network in a min-max zero-sum game. The generator neural network creates fake data (also sometimes referred to as generated data) while the discriminator distinguishes fake from real data. The aim of the generator is to mimic the real data as close as possible and to fool the discriminator. Hence implicitly, the generator neural network tries to learn the underlying distribution of the real data.

We provide the definitions of generator and discriminator neural networks below. We will be working in the context of image generation. Let $\mathbb{X} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^3$ be the set of $n \times n \times 3$ coloured images, where $(\mathbb{X}, \mathcal{B}(\mathbb{X}), \mu)$ is a measure space over the space of images, $\mathcal{B}(\mathbb{X})$ is the Borel σ-algebra, and μ is the Lebesgue measure. Let $\mathbb{A} \subset \mathbb{X}$ be the set of images that we want to mimic. This can be the set of landscapes images, handwritten numbers images, or images of celebrity faces. The elements of $\mathbb{A}$ are referred to as **real images**.

Definition 28 *Let $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}), \mu)$ be a measure space, where $\mathcal{Z} \subset \mathbb{R}^{3n^2}$. A **generator***

neural network *is a neural network*

$$g_{\boldsymbol{\theta}}(\mathcal{Z}, \mathcal{B}(\mathcal{Z}), \mu) \to (\mathbb{X}, \mathcal{B}(\mathbb{X}), \mu),$$

whose task is to generate images $\mathbf{x} \in \mathbb{X}$ *such that the discriminator (defined below) declares that* $\mathbf{x} \in \mathbb{A}$*, where* $\mathbb{A}$ *is the set of real images and* $\boldsymbol{\theta}$ *is the parameters of the generator neural network.*

Note that the images that the generator creates are called **fake images**. Fake images are also sometimes referred to as **generated images**.

Definition 29 *A **discriminator neural network** is a neural network*

$$D_{\tilde{\boldsymbol{\theta}}} : (\mathbb{X}, \mathcal{M}, \mu) \to ([0,1], \mathbb{B}([0,1]), \mu),$$

whose task is to classify real and fake images, where $\tilde{\boldsymbol{\theta}}$ *is the parameters of the discriminator neural network.*

We will write the shorthand $g := g_{\boldsymbol{\theta}}$ and $D := D_{\tilde{\boldsymbol{\theta}}}$.

We next provide details of the GANs loss functions. Let $p_{\mathbf{X}} : \mathbb{X} \to [0,1]$ be the probability density function of real images. Let $p_{\mathbf{Z}} : \mathcal{Z} \to [0,1]$ be the density function from which the generative neural network draws samples. This is usually a multivariate Gaussian random variable. Let $p_g : \mathbb{X} \to [0,1]$ be the probability density function of images that the generator neural network produces.

Definition 30 *The **GANs loss function** is defined as*

$$V(g, D) = -\mathcal{H}(p_{\mathbf{X}}; D) - \mathcal{H}(p_{\mathbf{Z}}; 1 - D \circ g), \tag{3.1}$$

where $D \circ g$ denotes the functional composition.

The discriminator and generator neural networks play a min-max game

$$\min_g \max_D V(g, D) = \min_g \max_D \left(\mathop{\mathbb{E}}_{\mathbf{A} \sim p_{\mathbf{X}}} [\log(D(\mathbf{A}))] + \mathop{\mathbb{E}}_{\mathbf{B} \sim p_{\mathbf{Z}}} [\log(1 - D(g(\mathbf{B})))] \right). \quad (3.2)$$

Intuitively, the discriminator tries to label real images as 1 and fake images as 0, whereas the generator tries to induce the discriminator to label fake images as 1.

We next provide a proof of the solution to the min-max loss function (3.2). Note this result is derived in [32], but we provide a detailed proof here.

Proposition 7 *Consider the optimization problem* (3.2). *Then*

$$D^*(\mathbf{x}) = \frac{p_{\mathbf{X}}(\mathbf{x})}{p_{\mathbf{X}}(\mathbf{x}) + p_g(\mathbf{x})} \text{ (a.e.)}, \quad (3.3)$$

where $D^* := \mathrm{argmax}_D V(g, D)$. Furthermore, if $D = D^*$, then

$$\begin{aligned} V(D^*, g) &= 2\mathrm{JSD}(p_{\mathbf{X}} \| p_g) - 2\log(2) \\ &\geq -2\log(2), \end{aligned}$$

with equality if and only if $p_g = p_{\mathbf{X}}$ (a.e.).

Proof We have that

$$\begin{aligned} V(g, D) &= \mathop{\mathbb{E}}_{\mathbf{A} \sim p_{\mathbf{X}}} [\log(D(\mathbf{A}))] + \mathop{\mathbb{E}}_{\mathbf{B} \sim p_{\mathbf{Z}}} [\log(1 - D(g(\mathbf{B})))] \\ &= \int_{\mathbb{X}} p_{\mathbf{X}} \log[D] \mathrm{d}\mu + \int_{\mathcal{Z}} p_{\mathbf{Z}} \log[1 - D(g)] \mathrm{d}\mu \end{aligned}$$

Note, we have

$$\begin{aligned}\int p_{\mathbf{Z}}(\boldsymbol{c})\log[1-D(g(\boldsymbol{c}))]\mathrm{d}\boldsymbol{c} &= \int p_g(\mathbf{x})\log[1-D(\mathbf{x})]\mathrm{d}\mathbf{x}\\ \Rightarrow V(g,D) &= \int_{\mathbb{X}} p_{\mathbf{X}}\log[D]+p_g\log[1-D]\mathrm{d}\mu.\end{aligned}$$

As a result,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}D}V(D,g) &= \lim_{\epsilon\downarrow 0}\int_{\mathbb{X}} p_{\mathbf{X}}\left(\frac{\log\left(\frac{D+\epsilon}{D}\right)}{\epsilon}\right)+p_g\left(\frac{\log\left(\frac{1-D+\epsilon}{1-D}\right)}{\epsilon}\right)\mathrm{d}\mu\\ &= \lim_{\epsilon\downarrow 0}\int_{\mathbb{X}} p_{\mathbf{X}}\left(\frac{\log\left(1+\frac{\epsilon}{D}\right)}{\left(1+\frac{\epsilon}{D}\right)-1}\right)\left(\frac{\left(1+\frac{\epsilon}{D}\right)-1}{\epsilon}\right)\\ &\quad +p_g\left(\frac{\log\left(1+\frac{\epsilon}{1-D}\right)}{\left(1+\frac{\epsilon}{1-D}\right)-1}\right)\left(\frac{\left(1+\frac{\epsilon}{1-D}\right)-1}{\epsilon}\right)\mathrm{d}\mu\\ &= \lim_{\epsilon\downarrow 0}\int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D}\frac{\log\left(1+\frac{\epsilon}{D}\right)}{\left(1+\frac{\epsilon}{D}\right)-1}+\frac{p_g}{1-D}\frac{\log\left(1+\frac{\epsilon}{1-D}\right)}{\left(1+\frac{\epsilon}{1-D}\right)-1}\mathrm{d}\mu. \qquad (3.4)\end{aligned}$$

Next we show (3.4) can be upper and lower bounded by $\int_{\mathbb{X}}\frac{p_{\mathbf{X}}}{D}+\frac{p_g}{1-D}\mathrm{d}\mu$. Indeed, we have that

$$\lim_{\epsilon\downarrow 0}\int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D}\frac{\log\left(1+\frac{\epsilon}{D}\right)}{\left(1+\frac{\epsilon}{D}\right)-1}+\frac{p_g}{1-D}\frac{\log\left(1+\frac{\epsilon}{1-D}\right)}{\left(1+\frac{\epsilon}{1-D}\right)-1}\mathrm{d}\mu \;\leq\; \int_{\mathbb{X}}\frac{p_{\mathbf{X}}}{D}+\frac{p_g}{1-D}\mathrm{d}\mu,$$

as $\log(y)\leq y-1$ for all $y>0$.

Also,

$$\begin{aligned}&\lim_{\epsilon\downarrow 0}\int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D}\frac{\log\left(1+\frac{\epsilon}{D}\right)}{\left(1+\frac{\epsilon}{D}\right)-1}+\frac{p_g}{1-D}\frac{\log\left(1+\frac{\epsilon}{1-D}\right)}{\left(1+\frac{\epsilon}{1-D}\right)-1}\mathrm{d}\mu\\ \geq\; &\lim_{\epsilon\downarrow 0}\int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D}\left(\frac{D}{\epsilon}\right)\left(1-\frac{D}{D+\epsilon}\right)+\frac{p_g}{1-D}\left(\frac{1-D}{\epsilon}\right)\left(1-\frac{1-D}{1-D+\epsilon}\right)\mathrm{d}\mu \qquad (3.5)\end{aligned}$$

$$\begin{aligned} &= \lim_{\epsilon \downarrow 0} \int_{\mathbb{X}} p_{\mathbf{X}} \left(\frac{1}{D+\epsilon} \right) + p_g \left(\frac{1}{1-D+\epsilon} \right) \mathrm{d}\mu \qquad (3.6) \\ &= \int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D} + \frac{p_g}{1-D} \mathrm{d}\mu, \end{aligned}$$

where (3.5) is because $\log(y) \geq 1 - \frac{1}{y}$ for all $y > 0$. We can exchange the limit with the integral in (3.6) because of the monotone convergence theorem. This is because $\frac{1}{D+\epsilon}$ and $\frac{1}{1-D+\epsilon}$ are monotonically increasing as $\epsilon \downarrow 0$ and are bounded below by 0. Hence,

$$\frac{\mathrm{d}}{\mathrm{d}D} V(D,g) = \int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D} + \frac{p_g}{1-D} \mathrm{d}\mu \qquad (3.7)$$

If $\frac{p_{\mathbf{X}}}{D^*} + \frac{p_g}{1-D^*} = 0$ (a.e.), then $\frac{\mathrm{d}}{\mathrm{d}D} V(D,g)|_{D=D^*} = 0$. Solving for D^*, we get $D^* = \frac{p_{\mathbf{X}}}{p_{\mathbf{X}}+p_g}$ (a.e.). Furthermore D^* is the maximum as

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}D} \int_{\mathbb{X}} \frac{p_{\mathbf{X}}}{D} + \frac{p_g}{1-D} \mathrm{d}\mu &= \lim_{\epsilon \to 0} \int_{\mathbb{X}} \frac{1}{\epsilon} \left(\frac{p_{\mathbf{X}}}{D+\epsilon} - \frac{p_{\mathbf{X}}}{D} + \frac{p_g}{1-D+\epsilon} - \frac{p_g}{1-D} \right) \mathrm{d}\mu \\ &= \lim_{\epsilon \to 0} \int_{\mathbb{X}} p_{\mathbf{X}} \frac{-1}{(D+\epsilon)(D)} + p_g \frac{-1}{(1-D+\epsilon)(1-D)} \mathrm{d}\mu \qquad (3.8) \\ &= \int_{\mathbb{X}} -\frac{p_{\mathbf{X}}}{D^2} - \frac{p_g}{(1-D)^2} \mathrm{d}\mu \\ &< 0. \end{aligned}$$

Using the same argument as for (3.6) and the monotone convergence theorem, we can exchange the limit with the integral in (3.8).

Next, note that $\min_g \max_D V(g,D) = \min_g V(g, D^*)$.

$$\begin{aligned} V(g, D^*) &= \mathop{\mathbb{E}}_{\mathbf{A} \sim p_{\mathbf{X}}} [\log(D^*(\mathbf{A}))] + \mathop{\mathbb{E}}_{\mathbf{B} \sim p_{\mathbf{Z}}} [\log(1 - D^*(g(\mathbf{B})))] \\ &= \mathop{\mathbb{E}}_{\mathbf{A} \sim p_{\mathbf{X}}} [\log(D^*(\mathbf{A}))] + \mathop{\mathbb{E}}_{\mathbf{C} \sim p_g} [\log(1 - D^*(\mathbf{C}))] \end{aligned}$$

$$\begin{aligned}
&= \underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}\left[\log\left(\frac{p_{\mathbf{X}}(\mathbf{A})}{p_{\mathbf{X}}(\mathbf{A})+p_g(\mathbf{A})}\right)\right] + \underset{\mathbf{C}\sim p_g}{\mathbb{E}}\left[\log\left(1-\frac{p_{\mathbf{X}}(\mathbf{C})}{p_{\mathbf{X}}(\mathbf{C})+p_g(\mathbf{C})}\right)\right] \\
&= \int_{\mathbb{X}}\left(p_{\mathbf{X}}\log\left(\frac{p_{\mathbf{X}}}{p_{\mathbf{X}}+p_g}\right)\right)\mathrm{d}\mu + \int_{\mathbb{X}}\left(p_g\log\left(1-\frac{p_{\mathbf{X}}}{p_{\mathbf{X}}+p_g}\right)\right)\mathrm{d}\mu \\
&= -\log(4) + \int_{\mathbb{X}}\left(p_{\mathbf{X}}\log\left(\frac{2*p_{\mathbf{X}}}{p_{\mathbf{X}}+p_g}\right)\right) + \left(p_g\log\left(\frac{2*p_g}{p_{\mathbf{X}}+p_g}\right)\right)\mathrm{d}\mu \\
&= -\log(4) + 2(\mathrm{JSD}(p_{\mathbf{X}}||p_g)),
\end{aligned}$$

where $\mathrm{JSD}(p_{\mathbf{X}}||p_g)$ is the Jensen-Shannon Divergence, which is minimized when $p_g = p_{\mathbf{X}}$ (a.e.). ■

GANs are notoriously hard to train; e.g., see [34] [83] [84] [18] [8]. They suffer from a myriad of problems such as mode collapse, which is the generator producing only one image and hence the discriminator is perfectly able to tell apart real from fake images [8] [9]. Several modifications have been made to GANs to improve their training stability and the quality of their generated images. For example, DCGANs use convolutional neural networks for the generator and discriminator to improve the quality of generated images [67], or LSGANs try to improve training stability [55]. StyleGANs alter the generator's network architecture to create highly realistic images by mapping the input into an intermediate latent space from which different features of the images can be controlled, such as eye colour or age for images of faces [41].

3.1 Gradient penalties

A commonly used method of improving the training stability of GANs is gradient penalties [34] [71]. Inspired by Wasserstein GANs [9], researchers proposed altering the Wasserstein GANs' loss function to encourage the discriminator to be Lipschitz [34]. Recall that the discriminator neural network $D : \mathbb{X} \to [0, 1]$, where $\mathbb{X} \subset \mathbb{R}$, is called Lipschitz if for all $\mathbf{x}_1$, $\mathbf{x}_2 \in \mathbb{X}$, $|D(\mathbf{x}_1) - D(\mathbf{x}_2)| \leq c\|\mathbf{x}_1 - \mathbf{x}_2\|_F$, where $c \geq 0$ and $\|\mathbf{a}\|_F = \sqrt{\sum_{i,j,k} \mathbf{a}(i,j,k)^2}$ is the Frobenius norm of some image $\mathbf{a} \in \mathbb{X}$ [31]. Experimentally, adding a gradient penalty to the discriminator's loss function for Wasserstein GANs confers faster convergence and better quality image generation [34]. The use of a gradient penalty was further extended to a larger class of GANs loss functions and researchers proved that incorporating a gradient penalty is equivalent to adding noise to the inputs of the discriminator during its training [71]. The gradient penalty proposed by [71] was further simplified by [57].

Definition 31 *[57] The* ***simplified gradient penalty*** *is defined as*

$$\tilde{\gamma} \mathop{\mathbb{E}}_{\mathbf{A} \sim p_{\mathbf{X}}} \left(\left\| \nabla_{\mathbf{x}} D(\mathbf{x}) \big|_{\mathbf{x}=\mathbf{A}} \right\|_2^2 \right), \tag{3.9}$$

where $\tilde{\gamma} > 0$.

Note that in our simulations, we use $\tilde{\gamma} = 5$ as recommended by [57]. Adding the simplified gradient penalty to the discriminator penalizes the discriminator for creating gradients for real data [57]. It improves the training of GANs; however it leads to a considerable increase in training time [57]. Applying the logit functional [31], given by $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$ for $p \in (0,1)$, to the discriminator's output is preferred as its gradients are easier to implement and are more robust [71].

3.2 Least squares GANs (LSGANs)

To improve the training issues of GANs and the generated image quality, a novel loss function was proposed which simplifies down to the Pearson χ^2 divergence rather than the original Jensen-Shannon divergence [55], [56]. The work was motivated by the fact that for the original GANs' loss function, the gradient updates of the generator vanish if the discriminator labels the generated images as real images, even if the generated images do not mimic the real images well. As training continues and the discriminator is perfectly able to tell apart fake images from real images, then the generator cannot improve the quality of fake images as its gradients vanish. Hence the discriminator wins the zero-sum game. Instead, by using least squares loss function, the gradients of the generator do not vanish for samples that lie far from the true distribution [55], [56]. As such, LSGANs tend to provide better training stability and generated image quality than GANs. A rigorous proof of this fact is not provided in the original LSGANs paper, however, experiments show that LSGANs have better training stability than GANs [55], [56].

Definition 32 *The **LSGANs loss functions** are defined as*

$$V_D(D,g) = \frac{1}{2}\underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}[(D(\mathbf{A})-\beta)^2] + \frac{1}{2}\underset{\mathbf{B}\sim p_{\mathbf{Z}}}{\mathbb{E}}[(D(g(\mathbf{B}))-\alpha)^2] \tag{3.10}$$

$$V_g(D,g) = \underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}[(D(\mathbf{A})-\gamma)^2] + \underset{\mathbf{B}\sim p_{\mathbf{Z}}}{\mathbb{E}}[(D(g(\mathbf{B}))-\gamma)^2], \tag{3.11}$$

where D is the discriminator neural network, g is the generator neural network, α, β, $\gamma \in \mathbb{R}$ are constants, $V_D(D,g)$ is the discriminator's loss function, and $V_g(D,g)$ is the generator's loss function.

The LSGANs optimization problems are

$$\min_D V_D(D,g) = \min_D \frac{1}{2}\underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}[(D(\mathbf{A})-\beta)^2] + \frac{1}{2}\underset{\mathbf{B}\sim p_{\mathbf{Z}}}{\mathbb{E}}[(D(g(\mathbf{B}))-\alpha)^2] \tag{3.12}$$

$$\min_g V_g(D,g) = \min_g \underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}[(D(\mathbf{A})-\gamma)^2] + \underset{\mathbf{B}\sim p_{\mathbf{Z}}}{\mathbb{E}}[(D(g(\mathbf{B}))-\gamma)^2]. \tag{3.13}$$

We provide solutions to the LSGANs optimization problems. Note that this result is presented in [55] and [56].

Theorem 8 *Let $(\mathbb{X}, \mathcal{B}(\mathbb{X}), \mu)$ be the measure space of $n\times n\times 3$ images and $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}), \mu)$ the measure space where $\mathcal{Z} \subset \mathbb{R}^{3n^2}$. Consider the optimization problem* (3.12) *for training the discriminator neural network $D : \mathbb{X} \to [0, 1]$ and* (3.13) *for the generator neural network. Then*

$$D^* := \frac{\alpha p_g + \beta p_{\mathbf{X}}}{p_g + p_{\mathbf{X}}} \text{ (a.e.)}, \tag{3.14}$$

where $D^* = \operatorname{argmin}_D V_D(D, g)$. If $D = D^*$ and $\alpha - \beta = 2(\gamma - \beta)$, then

$$\begin{aligned} V_g(D^*, g) &= (\gamma-\beta)^2 \chi^2(p_{\mathbf{X}} + p_g \| 2p_g) \\ &\geq 0 \end{aligned}$$

with equality if and only if $p_g = p_{\mathbf{X}}$ (a.e.) or $\gamma = \beta$.

Proof Note that

$$\frac{\mathrm{d}}{\mathrm{d}D} V_D(D, g) = \frac{\mathrm{d}}{\mathrm{d}D} \frac{1}{2} \int_{\mathbb{X}} p_{\mathbf{X}} (D-\beta)^2 \mathrm{d}\mu + \frac{1}{2} \int_{\mathcal{Z}} p_{\mathbf{Z}} (D(g) - \alpha)^2 \mathrm{d}\mu.$$

We have that $\int_{\mathcal{Z}} p_{\mathbf{Z}}(D(g) - \alpha)^2 \mathrm{d}\mu = \int_{\mathbb{X}} p_g (D-\alpha)^2 \mathrm{d}\mu$. Hence,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}D} V_D(D, g) &= \frac{\mathrm{d}}{\mathrm{d}D} \frac{1}{2} \int_{\mathbb{X}} p_{\mathbf{X}} (D-\beta)^2 \mathrm{d}\mu + \int_{\mathbb{X}} p_g (D-\alpha)^2 \mathrm{d}\mu \\ &= \frac{\mathrm{d}}{\mathrm{d}D} \frac{1}{2} \int_{\mathbb{X}} p_{\mathbf{X}} (D-\beta)^2 + p_g (D-\alpha)^2 \mathrm{d}\mu \\ &= \lim_{\epsilon \to 0} \frac{1}{2} \int_{\mathbb{X}} \frac{p_{\mathbf{X}} (D+\epsilon-\beta)^2 - p_{\mathbf{X}} (D-\beta)^2}{\epsilon} \end{aligned}$$

$$\begin{aligned}
&+\frac{p_g(D+\epsilon-\alpha)^2-p_g(D-\alpha)^2}{\epsilon}\mathrm{d}\mu \\
=\ & \lim_{\epsilon\to 0}\frac{1}{2}\int_{\mathbb{X}} p_{\mathbf{X}}\frac{2D\epsilon+\epsilon^2-2\beta\epsilon}{\epsilon}+p_g\frac{2D\epsilon+\epsilon^2-2\alpha\epsilon}{\epsilon}\mathrm{d}\mu \\
=\ & \lim_{\epsilon\to 0}\left(\int_{\mathbb{X}} p_{\mathbf{X}}(D-\beta)+p_g(D-\alpha)\mathrm{d}\mu+\frac{\epsilon}{2}\int_{\mathbb{X}} p_{\mathbf{X}}+p_g\mathrm{d}\mu\right) \\
=\ & \int_{\mathbb{X}} p_{\mathbf{X}}(D-\beta)+p_g(D-\alpha)\mathrm{d}\mu.
\end{aligned}$$

Note that if $p_{\mathbf{X}}(D^*-\beta)+p_g(D^*-\alpha)=0$ (a.e.), then $\frac{\mathrm{d}}{\mathrm{d}D}V_D(D^*,g)=0$, for some D^*. Solving $p_{\mathbf{X}}(D^*-\beta)+p_g(D^*-\alpha)=0$ for D^*, we get

$$D^*=\frac{\alpha p_g+\beta p_{\mathbf{X}}}{p_g+p_{\mathbf{X}}}\text{ (a.e.).}$$

Furthermore, D^* minimizes $V_D(D,g)$, as

$$\begin{aligned}
\frac{\mathrm{d}^2}{\mathrm{d}D^2}V_D(D^*,g) &= \frac{\mathrm{d}}{\mathrm{d}D}\int_{\mathbb{X}} p_{\mathbf{X}}(D-\beta)+p_g(D-\alpha)\mathrm{d}\mu \\
&= \lim_{\epsilon\to 0}\int_{\mathbb{X}} p_{\mathbf{X}}\frac{(D+\epsilon-\beta)-(D-\beta)}{\epsilon}+p_g\frac{(D+\epsilon-\alpha)-(D-\alpha)}{\epsilon}\mathrm{d}\mu \\
&= \int_{\mathbb{X}} p_{\mathbf{X}}+p_g\mathrm{d}\mu \\
&> 0.
\end{aligned}$$

Now consider $V_g(D^*,g)$:

$$\begin{aligned}
V_g(D^*,g) &= \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}[(D^*(\mathbf{A})-\gamma)^2]+\mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}}[(D^*(g(\mathbf{B}))-\gamma)^2] \\
&= \int_{\mathbb{X}} p_{\mathbf{X}}(D^*-\gamma)^2\mathrm{d}\mu+\int_{\mathbb{X}} p_{\mathbf{Z}}(D^*(g)-\gamma)^2\mathrm{d}\mu
\end{aligned}$$

Note that $\int_{\mathbb{X}} p_g(D^* - \gamma)^2 \mathrm{d}\mu = \int_{\mathbb{X}} p_{\mathbf{Z}}(D^*(g) - \gamma)^2 \mathrm{d}\mu$. Hence

$$\begin{aligned}
V_g(D^*, g) &= \int_{\mathbb{X}} p_{\mathbf{X}}(D^* - \gamma)^2 \mathrm{d}\mu + \int_{\mathbb{X}} p_g(D^* - \gamma)^2 \mathrm{d}\mu \\
&= \int_{\mathbb{X}} p_{\mathbf{X}} \left(\frac{\alpha p_g + \beta p_{\mathbf{X}}}{p_g + p_{\mathbf{X}}} - \gamma \right)^2 + \int_{\mathbb{X}} p_g \left(\frac{\alpha p_g + \beta p_{\mathbf{X}}}{p_g + p_{\mathbf{X}}} - \gamma \right)^2 \mathrm{d}\mu \\
&= \int_{\mathbb{X}} \frac{((\alpha - \gamma)p_g + (\beta - \gamma)p_{\mathbf{X}})^2}{p_{\mathbf{X}} + p_g} \mathrm{d}\mu \\
&= \int_{\mathbb{X}} \frac{((\beta - \gamma)(p_{\mathbf{X}} + p_g) + (\alpha - \beta)p_g)^2}{p_{\mathbf{X}} + p_g} \mathrm{d}\mu.
\end{aligned}$$

Since $\alpha - \beta = 2(\gamma - \beta)$, then

$$\begin{aligned}
V_g(D^*, g) &= (\gamma - \beta)^2 \int_{\mathbb{X}} \frac{(2p_g - (p_{\mathbf{X}} + p_g))^2}{p_{\mathbf{X}} + p_g} \mathrm{d}\mu \\
&= (\gamma - \beta)^2 \chi^2(p_{\mathbf{X}} + p_g \| 2p_g),
\end{aligned}$$

which is minimized when $p_{\mathbf{X}} + p_g = 2p_g$, equivalently $p_g = p_{\mathbf{X}}$ (a.e.), or when $\gamma = \beta$. ■

3.3 InfoGANs

Researchers started to experiment using tools from information theory to try to control features of generated GAN images, the result of which was a 2016 paper introducing InfoGANs [15]. The main idea of InfoGANs is to deconstruct the input noise random variable of the generator into latent codes and Gaussian incompressible noise. These latent codes represent the important features of the data distribution, for example, for handwritten numbers from the MNIST dataset [47], these codes represent the thickness of the numbers, the angle at which they were written at, and the generated number.

Let $p_{\mathbf{Z}} : \mathbb{R}^{3n^2-p} \to [0,1]$ be a probability density function corresponding to a multivariate Gaussian random variable $\mathbf{Z}$. Let $\mathcal{W} \subset \mathbb{R}^p$ be the space of latent codes. It is also referred to as the latent space. Let $\mathbf{W}$ be a random variable of latent codes, which is independent from $\mathbf{Z}$, with probability density function $p_{\mathbf{W}}$. Let $g : \mathbb{R}^{3n^2} \to \mathbb{X}$ be the InfoGANs generator neural network. By an abuse of notation, we let $g = g(\mathbf{Z}, \mathbf{W})$.

Definition 33 *The **InfoGANs loss function** is defined as*

$$V(g,D) := \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}[\log(D(\mathbf{A}))] + \mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z},\mathbf{W}}}[\log(1 - D(g(\mathbf{B})))] - \lambda \mathrm{I}(p_{\mathbf{W}}; p_g) \tag{3.15}$$

where $\lambda > 0$, and $\mathrm{I}(p_{\mathbf{W}}; p_g)$ is the Shannon mutual information as defined in Definition 12.

Similar to GANs, InfoGANs' generator and discriminator play a min-max game

$$\begin{aligned} & \min_g \max_D V(g,D) \\ = \; & \min_g \max_D \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}[\log(D(\mathbf{A}))] + \mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z},\mathbf{W}}}[\log(1 - D(g(\mathbf{B})))] - \lambda \mathrm{I}(p_{\mathbf{W}}; p_g) \end{aligned} \tag{3.16}$$

Note that the mutual information does not depend on the discriminator. Thus we have that $D^* = \frac{p_{\mathbf{X}}}{p_{\mathbf{X}}+p_g}$ (a.e.) and the InfoGANs optimization problem simplifies to $\min_g 2\mathrm{JSD}(p_{\mathbf{X}} \| p_g) - \log(4) - \lambda \mathrm{I}(p_{\mathbf{W}}; p_g)$. This optimization problem is maximizing the mutual information, $\mathrm{I}(p_{\mathbf{W}}; p_g)$. Hence InfoGANs try to ensure that the information in their latent codes are not lost in the generation process.

Since, it is hard to determine $p_{\mathbf{W}|g}$, which arises from $\mathrm{I}(p_{\mathbf{W}}; p_g)$, it is simpler to use an auxiliary distribution, $q_{\mathbf{W}|g}$, which can be implemented using a neural network [15].

That is the $q_{\mathbf{W}|g}$ neural network is tasked to predict the latent code $\mathbf{W}$ based on the generated image $g(\mathbf{Z}, \mathbf{W})$.

Definition 34 *The* ***revised InfoGANs loss function*** *is defined as*

$$V(g, D) := \underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}[\log(D(\mathbf{A}))] + \underset{\mathbf{B}\sim p_{\mathbf{Z},\mathbf{W}}}{\mathbb{E}}[\log(1 - D(g(\mathbf{B})))] - \lambda L_I(g, q) \tag{3.17}$$

where $\lambda > 0$, $L_I(g, q) := \underset{\mathbf{B}\sim p_{\mathbf{W}}}{\mathbb{E}}\left(\underset{\mathbf{A}\sim p_{g|\mathbf{W}}(\cdot|\mathbf{B})}{\mathbb{E}}[\log(q_{\mathbf{W}|g}(\mathbf{B}|\mathbf{A}))]\right) + \mathrm{H}(p_{\mathbf{W}})$, *and* $q_{\mathbf{W}|g}$ *is an auxiliary probability density function that approximates* $p_{\mathbf{W}|g}$.

The optimization problem for InfoGANs thus becomes

$$\begin{aligned}\min_q \min_g \max_D V(g, D) &= \min_q \min_g \max_D \underset{\mathbf{A}\sim p_{\mathbf{X}}}{\mathbb{E}}[\log(D(\mathbf{A}))] \\ &\quad + \underset{\mathbf{B}\sim p_{\mathbf{Z},\mathbf{W}}}{\mathbb{E}}[\log(1 - D(g(\mathbf{B})))] - \lambda L_I(g, q) \end{aligned}\tag{3.18}$$

Note that $L_I(g, q)$ is a lower bound of the mutual information $\mathrm{I}(p_{\mathbf{W}}; p_g)$.

Theorem 9 *[15] For a generator neural network* $g(\mathbf{Z}, \mathbf{W})$, *where* $\mathbf{Z}$ *is a Gaussian random variable and* $\mathbf{W}$ *is the latent codes random variable, where* $\mathbf{Z}$ *and* $\mathbf{W}$ *are independent, we have that*

$$L_I(g, q) \leq \mathrm{I}(p_{\mathbf{W}}; p_g). \tag{3.19}$$

As $q_{\mathbf{W}|g}$ converges to $p_{\mathbf{W}|g}$, the lower bound becomes tight [15]. Since $L_I(g, q)$ does not depend on D, we have that $D^* = \frac{p_{\mathbf{X}}}{p_{\mathbf{X}}+p_g}$ (a.e.), and hence the revised InfoGANs optimization problem becomes $\min_q \min_g 2\mathrm{JSD}(p_{\mathbf{X}}\|p_g) - \log(4) - \lambda L_I(g, q)$. Thus the revised InfoGANs optimization problem tries to maximize $L_I(g, q)$, which is easier to implement than the mutual information $\mathrm{I}(p_{\mathbf{W}}; p_g)$.

3.4 Fréchet inception distance: measuring image quality

Several metrics are used to measure generated image quality. These include likelihood estimation, kernel density estimates (KDEs), the inception score [74], and the Fréchet inception distance (FID) score [37]. Other metrics can be found in [86]. KDEs (also known as Parzen window estimates) were commonly used to evaluate image quality; however, it was discovered that they may favour generative models that produce low quality images and low log-likelihood results and as such KDEs should be avoided [78]. Moreover, high likelihood does not necessarily correspond to better quality images [78]. Hence, these metrics are not an accurate measure of generated images visual fidelity to the real images. Currently, in the GAN literature, the most common metrics are the FID score and the inception score [86].

The inception score uses a pretrained neural network called the inception network (Inception Network version 3) [77] that classifies images from the ImageNet dataset [23] [73]. The inception score correlates well with human judgement. We provide the definition of the inception score below.

Definition 35 *[74] Let $\mathcal{J} : \mathbb{R}^{299} \times \mathbb{R}^{299} \times \mathbb{R}^{3} \to \mathbb{R}^{1000}$ be the Inception Network that classifies images of size $299 \times 299 \times 3$ into 1000 different categories. For a GAN with generator g with fake probability density function p_g, the* ***inception score*** *is*

$$\mathrm{IS}(p_g) := e^{\mathbb{E}_{\mathbf{A}\sim p_g}[\mathrm{KL}(p_{\mathcal{J}}(y|\mathbf{A})\|p_{\mathcal{J}}(y))]}, \tag{3.20}$$

where $p_{\mathcal{J}}(y|\mathbf{a})$ is the probability that a given image $\mathbf{a}$ belongs to a category $y \in \{1, \ldots, 1000\}$ as predicted by the inception network $\mathcal{J}$, and $p_{\mathcal{J}}(y) = \int_{\mathbb{X}} p_g p_{\mathcal{J}}(y|\cdot)\mathrm{d}\mu$.

A large inception score means that $p_{\mathcal{J}}(y|\mathbf{a})$ has low entropy and the average cross-entropy between $p_{\mathcal{J}}(y|\mathbf{a})$ and $p_{\mathcal{J}}(y)$ across all generated images $\mathbf{a}$ is high. In other words, a large inception score implies more realistic generated images and a greater variety of images. To accurately calculate the inception score, at least $50,000$ generated images are recommended [74].

The inception score does not compare p_g with the true distribution $p_{\mathbf{X}}$. Hence it could be the case that the true distribution $p_{\mathbf{X}}$ has a lower inception score than the generated images [10]. In other words, it is possible that the generator is producing a large variety of images that look realistic to the inception network but the images of $p_{\mathbf{X}}$ are not of a large variety. For example, $p_{\mathbf{X}}$ is the distribution of moving van images, which is only one category of the Inception network. As such, a high inception score is not desirable. Moreover, if we use a dataset that is not classified by the inception network, such as the MNIST dataset, then the inception score does not provide a proper measurement of the quality of images it produces [10].

Instead, it is common practice to use Fréchet inception distance (FID) score [37]. We provide the definition below.

Definition 36 *[37] Let $\mathcal{J} : \mathbb{R}^{299} \times \mathbb{R}^{299} \times \mathbb{R}^3 \to \mathbb{R}^{1000}$ be the Inception Network that classifies images of size $299 \times 299 \times 3$ into 1000 different categories. For a GAN with generator g with fake probability density function p_g and true distribution $p_{\mathbf{X}}$, the* ***Fréchet inception distance score*** *is*

$$\mathrm{FID}(p_g, p_{\mathbf{X}}) := \|\mu_g - \mu_{\mathbf{X}}\|_2^2 + \mathrm{Tr}\left(\Sigma_g + \Sigma_{\mathbf{X}} - 2(\Sigma_g \Sigma_{\mathbf{X}})^{\frac{1}{2}}\right), \tag{3.21}$$

where $\mu_g = \underset{\mathbf{A}\sim p_g}{\mathbb{E}}(\mathcal{J}(\mathbf{A}))$ *is the mean vector of length* 1000*, and*

$$\Sigma_g = \left[\underset{\mathbf{A}\sim p_g}{\mathbb{E}}\left(\left[p_{\mathcal{J}}(k|\mathbf{A}) - \underset{\mathbf{B}\sim p_g}{\mathbb{E}}(p_{\mathcal{J}}(k|\mathbf{B}))\right]\left[p_{\mathcal{J}}(l|\mathbf{A}) - \underset{\mathbf{B}\sim p_g}{\mathbb{E}}(p_{\mathcal{J}}(l|\mathbf{B}))\right]\right)\right]_{k,l}$$

is the covariance matrix of size 1000×1000*, where* $p_{\mathcal{J}}(y|\mathbf{a})$ *is the probability that a given image* $\mathbf{a}$ *belongs to a category* $y \in \{1, \dots, 1000\}$ *as predicted by the network* $\mathcal{J}$*. Similarly, we define* $(\mu_{\mathbf{X}}, \Sigma_{\mathbf{X}})$ *for* $p_{\mathbf{X}}$*.*

At least $10,000$ samples of real and fake images are needed to estimate (μ_g, Σ_g) and $(\mu_{\mathbf{X}}, \Sigma_{\mathbf{X}})$ [37]. Similar to the inception score, the FID score categorizes real and fake images into 1000 categorizes using the inception network. Note that the distribution of the inception network's output for real and fake images is assumed to be a multivariate Gaussian distribution. The FID score is simply the Wasserstein-2 distance between two multivariate Gaussian distributions [37]. In comparison, the KL-divergence between the two multivariate Gaussian distributions is given by [28]

$$\begin{aligned}\mathrm{KL}(p_{\mathbf{X}}\|p_g) &= \frac{1}{2}\left(\log\left(\frac{\det\Sigma_g}{\det\Sigma_{\mathbf{X}}}\right) + \mathrm{Tr}(\Sigma_g^{-1}\Sigma_{\mathbf{X}})\right) \\ &\quad + \frac{1}{2}\left[(\mu_{\mathbf{X}} - \mu_g)^T\Sigma_g^{-1}(\mu_{\mathbf{X}} - \mu_g) - 1000\right].\end{aligned}$$

However, the KL-divergence is more difficult to calculate than the Wassertein-2 distance since it requires finding the inverse of the covariance matrix Σ_g.

The FID score is better than the inception score because it directly compares the real and fake images. It is shown to be more consistent than the inception score in detecting Gaussian blur, Gaussian noise, implanted black rectangles, and swirled images [37]. Moreover, the FID score is shown to be sensitive to mode collapse [54].

Chapter 4

Least kth-order GANs (LkGANs)

4.1 Theoretical results

Motivated by generalizing LSGANs, we employ the Pearson-Vajda divergence of order k, $|\chi|^k$, which generalizes the Pearson χ^2 divergence. We provide the least kth-order GANs (LkGANs) loss functions below.

Definition 37 *The **Least** $\boldsymbol{k}$**th-order GANs loss functions**, $k \geq 1$, are defined as*

$$V_D(D, g) = \frac{1}{2}\mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left[(D(\mathbf{A}) - \beta)^2\right] + \frac{1}{2}\mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}}\left[(D(g(\mathbf{B})) - \alpha)^2\right] \tag{4.1}$$

$$V_{k,g}(D, g) = \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left(|D(\mathbf{A}) - \gamma|^k\right) + \mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}}\left(|D(g(\mathbf{B})) - \gamma|^k\right), \tag{4.2}$$

where D is the discriminator neural network, g is the generator neural network, α, β, $\gamma \in [0, 1]$ are constants, $V_D(D, g)$ is the discriminator's loss function, and $V_{k,g}(D, g)$ is the generator's loss function.

Note that when $k = 2$, we have that the LkGANs' generator's loss function is the LSGANs generator's loss function, i.e., $V_{2,g}(D, g) = V_g(D, g)$. The L$k$GANs optimization problems are

$$\min_D V_D(D, g) = \min_D \frac{1}{2} \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}} \left[(D(\mathbf{A}) - \beta)^2\right] + \frac{1}{2} \mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}} \left[(D(g(\mathbf{B})) - \alpha)^2\right] \quad (4.3)$$

$$\min_g V_{k,g}(D, g) = \min_g \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}} \left(|D(\mathbf{A}) - \gamma|^k\right) + \mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}} \left(|D(g(\mathbf{B})) - \gamma|^k\right). \quad (4.4)$$

We next provide solutions to the optimization problems above.

Theorem 10 *Let $k \geq 1$, $(\mathbb{X}, \mathcal{B}(\mathbb{X}), \mu)$ be the measure space of $n \times n \times 3$ images, and $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}), \mu)$ the measure space where $\mathcal{Z} \subset \mathbb{R}^{3n^2}$. Consider the optimization problem (4.3) for training the discriminator neural network $D : \mathbb{X} \to [0, 1]$ and (4.4) for the generator neural network. Then*

$$D^* := \frac{\alpha p_g + \beta p_{\mathbf{X}}}{p_g + p_{\mathbf{X}}} \text{ (a.e.)}, \quad (4.5)$$

where $D^* = \operatorname{argmin}_D V_D(D, g)$. If $D = D^*$ and $\alpha - \beta = 2(\gamma - \beta)$, then

$$\begin{aligned} V_{k,g}(D^*, g) &= |\gamma - \beta|^k |\chi|^k (p_{\mathbf{X}} + p_g \| 2p_g) \\ &\geq 0, \end{aligned}$$

with equality if and only if $p_g = p_{\mathbf{X}}$ (a.e.) or $\gamma = \beta$.

Proof The proof that the solution of (4.3) is $D^* = \frac{\alpha p_g + \beta p_{\mathbf{X}}}{p_g + p_{\mathbf{X}}}$ (a.e.) is analogous to the one in Theorem 8. Note, we have that

$$\mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}} \left(|D^*(g(\mathbf{B})) - \gamma|^k\right) = \mathop{\mathbb{E}}_{\mathbf{B}\sim p_g} \left(|D^*(\mathbf{B}) - \gamma|^k\right).$$

Hence,

$$\begin{aligned}
V_{k,g}(D^*,g) &= \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left(|D^*(\mathbf{A})-\gamma|^k\right)+\mathop{\mathbb{E}}_{\mathbf{B}\sim p_g}\left(|D^*(\mathbf{B})-\gamma|^k\right)\\
&= \mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left(\left|\frac{\alpha p_g(\mathbf{A})+\beta p_{\mathbf{X}}(\mathbf{A})}{p_g(\mathbf{A})+p_{\mathbf{X}}(\mathbf{A})}-\gamma\right|^k\right)+\mathop{\mathbb{E}}_{\mathbf{B}\sim p_g}\left(\left|\frac{\alpha p_g(\mathbf{B})+\beta p_{\mathbf{X}}(\mathbf{B})}{p_g(\mathbf{B})+p_{\mathbf{X}}(\mathbf{B})}-\gamma\right|^k\right)\\
&= \int_{\mathbb{X}} p_{\mathbf{X}}\left|\frac{(\alpha-\gamma)p_g+(\beta-\gamma)p_{\mathbf{X}}}{p_g+p_{\mathbf{X}}}\right|^k \mathrm{d}\mu+\int_{\mathbb{X}} p_g\left|\frac{(\alpha-\gamma)p_g+(\beta-\gamma)p_{\mathbf{X}}}{p_g+p_{\mathbf{X}}}\right|^k \mathrm{d}\mu\\
&= \int_{\mathbb{X}}\frac{|(\alpha-\gamma)p_g+(\beta-\gamma)p_{\mathbf{X}}|^k}{(p_g+p_{\mathbf{X}})^{k-1}}\mathrm{d}\mu\\
&= \int_{\mathbb{X}}\frac{|(\alpha+\beta-\beta-\gamma)p_g+(\beta-\gamma)p_{\mathbf{X}}|^k}{(p_g+p_{\mathbf{X}})^{k-1}}\mathrm{d}\mu\\
&= \int_{\mathbb{X}}\frac{|(\alpha-\beta)p_g+(\beta-\gamma)(p_{\mathbf{X}}+p_g)|^k}{(p_g+p_{\mathbf{X}})^{k-1}}\mathrm{d}\mu.
\end{aligned}$$

Since $\alpha-\beta=2(\gamma-\beta)$, we have that

$$\begin{aligned}
V_{k,g}(D^*,g) &= |\gamma-\beta|^k\int_{\mathbb{X}}\frac{|2p_g-(p_{\mathbf{X}}+p_g)|^k}{(p_g+p_{\mathbf{X}})^{k-1}}\mathrm{d}\mu\\
&= |\gamma-\beta|^k|\chi|^k(p_{\mathbf{X}}+p_g\|2p_g), \qquad (4.6)
\end{aligned}$$

which is minimized when $p_{\mathbf{X}}+p_g=2p_g$, equivalently $p_{\mathbf{X}}=p_g$ (a.e.), or when $\gamma=\beta$. ■

LkGANs confer an extra degree of freedom provide by virtue of the parameter $k\geq 1$. Moreover the underlying global equilibrium point of LkGANs is the same as LSGANs, that is the minimum is theoretically achieved when the generator's distribution is the true distribution.

4.2 Experiments

4.2.1 Methods

For experiments, the $28 \times 28 \times 1$ MNIST [47] and the $64 \times 64 \times 3$ CelebA [53] datasets were used to test the new LkGANs loss functions. For comparison, LSGANs were also implemented. The architectures of the generator and discriminator neural networks were kept constant while testing on each dataset; see Section A.1 in the Appendix for details. The FID score was used to evaluate the quality of the generated images and to compare the rate at which the new networks converged to their optimal FID scores.

For the MNIST dataset, several versions of LkGANs were implemented with different α, β, and γ parameters. Version 1 has $\alpha = 0.6$, $\beta = 0.4$, and $\gamma = 0.5$. Version 2 has $\alpha = 1$, $\beta = 0$, and $\gamma = 0.5$. Version 3 has $\alpha = 0$, $\beta = 1$, and $\gamma = 1$, which are the same parameters used in [55]. We also tested $\alpha = 0$, $\beta = 1$, and $\gamma = 0.5$, which are the parameters used in [56], however, we observed that it performed similar to Version 2. As such, we did not include the results in the thesis. We label such LkGANs as LkGAN-v1-k, LkGAN-v2-k, and LkGAN-v3-k. See Algorithm 2 in Section A.2 for more details. The range of tested k values were $k \in \{1, 1.2, 1.4, \ldots, 3\}$.

A variant of LkGANs was also tested where the k parameter alters every epoch between 1 and 3 with increments of 0.1. See Algorithm 3 in A.2 for more details. Switching the k parameter every epoch changes the shape of the generator's loss function and does not affect the global minimum, which occurs when $p_{g^*} = p_x$ for all $k \geq 1$. If there exists a generator such that $p_g \neq p_x$ for some $k \geq 1$, the generator is a local minimum of $V_{k,g}(D, g)$, and it is not a local minimum for some $k' \geq 1$, then altering k every epoch creates gradients at previous local minimums. We label this

LkGAN as LkGAN$-[1,3]$.

For the MNIST dataset, seeds 123, 5005, 1600, 199621, 60677, 20435, 15859, 33764, 79878, 36123 were used for trials 1 to 10, respectively. For the SGD algorithm, the Adam optimizer with a learning rate of $\alpha_{Adam} = 2 \times 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-7}$ was used for the networks as recommended by [67]. The batch size was chosen to be 100 for the $60,000$ MNIST images. The networks were trained on the MNIST dataset for a total number of 100 epochs, or 6 million images.

For CelebA, seeds 1000, 2000, and 3000 were chosen for trials 1, 2, and 3 respectively. The publicly available StyleGAN code from [41] was modified to test the LkGANs loss functions. We refer to this as LkStyleGANs. LkStyleGANs were implemented for $k \in \{1, 2, 3\}$. We refer to LkStyleGANs when $k = 2$ as LSStyleGANs.

Three variants of LkStyleGANs with differing α, β, and γ parameters were also tested. Version 1 has $\alpha = 0.6$, $\beta = 0.4$, and $\gamma = 0.5$. We denote this by LkStyleGAN-v1. Version 2 has $\alpha = 1$, $\beta = 0$, and $\gamma = 0.5$. We denote this by LkStyleGAN-v2. Version 3 has $\alpha = 0$, $\beta = 1$, and $\gamma = 1$, which are the parameters tested in [55]. We denote this by LkStyleGAN-v3. LkStyleGANs with and without simplified gradient penalties were also implemented. We denote LkStyleGANs with simplified gradient penalties as LkStyleGAN-GP.

The original StyleGANs architectural defaults were left in place for the LkStyleGANs. The batch size was chosen to be 128. Unlike the original StyleGANs, which changes the resolution of its generated images during training, the resolution of the generated images for LkStyleGANs was fixed at $64 \times 64 \times 3$ throughout training. As recommended by the original StyleGANs paper, the Adam optimizer with a learning rate of $\alpha_{Adam} = 0.001$, $\beta_1 = 0.0, \beta_2 = 0.99$ and $\epsilon = 10^{-8}$ was used as the SGD algorithm [41].

The LkStyleGANs were trained for 25 million images or roughly 120 epochs.

One NVIDIA GP100 GPU and two Intel Xeon 2.6 GHz E7 − 8867 v3 CPUs were used for training LkGANs and LSGANs on the MNIST and the CelebA datasets.

Note we say that a network has *training stability* if it converges to meaningful results (i.e., the networks do not suffer from mode collapse).

4.2.2 Results

MNIST

A total of ten trials were run while controlling the random seeds in each trial. For each trial and each epoch, the FID scores were calculated. The Inception network was not used to categorize the images and to calculate the FID scores because the Inception network is not trained to recognize MNIST images. Instead the raw generated and real images were used to calculate the FID scores with the assumption that the distribution of raw generated images and the distribution of raw real images can be approximated using a multivariate Gaussian distribution.

The lowest FID score over 100 epochs in a single trial is called the best FID score. We present the best FID scores for all ten trials for Version 1 in Table A.3, Version 2 in Table A.4, and Version 3 in Table A.5 in the Appendix. We present the average and variance of the best FID scores over the ten trials and the epochs when they occur in Table 4.1. We highlight the best performing LkGAN based on the average best FID scores it achieves during training for Versions 1, 2, and 3. Since LkGAN-v2 generated meaningful results for certain trials, we also present the average and variance of the best FID scores when this occurs in Table 4.2

We present the plots of the average FID score of the best performing LkGAN versus epochs for each version in Figure 4.1 and its LSGAN counterpart. The average was calculated over ten trials. We also present the plots of FID scores over epochs of the best performing LkGAN for all versions for three select trials in Figure 4.2 and its LSGAN counterpart. Figure 4.3 shows sample images for each trial generated by the best performing LkGAN on average over all ten trials for all three versions and its LSGAN counterpart.

Table 4.1: LkGANs experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over ten trials.

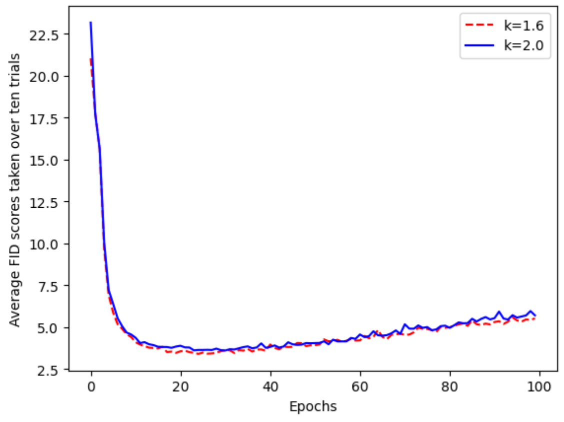
	Average best FID score	Best FID scores variance	Average epoch	Epoch variance
LkGAN-v1-1.0	3.17	1.91×10^{-2}	**21.50**	**13.65**
LkGAN-v1-1.2	3.15	$\mathbf{8.26 \times 10^{-3}}$	26.40	90.04
LkGAN-v1-1.4	3.28	4.08×10^{-2}	24.70	54.61
LkGAN-v1-1.6	**3.13**	2.03×10^{-2}	28.40	52.64
LkGAN-v1-1.8	3.23	2.62×10^{-2}	22.50	19.05
LkGAN-v1-2.2	3.51	2.91×10^{-2}	27.90	138.69
LkGAN-v1-2.4	3.59	7.17×10^{-2}	23.30	43.61
LkGAN-v1-2.6	3.76	7.57×10^{-2}	28.50	118.05
LkGAN-v1-2.8	3.99	8.60×10^{-2}	26.50	65.39
LkGAN-v1-3.0	4.13	4.15×10^{-2}	23.40	54.24
LkGAN-v1-$[1, 3]$	3.47	5.75×10^{-2}	22.70	16.61
LSGAN-v1	3.34	1.42×10^{-2}	26.90	57.69
LkGAN-v2-1.0	**30.77**	771.14	**56.80**	1310.16
LkGAN-v2-1.2	36.38	743.57	73.40	1049.24
LkGAN-v2-1.4	58.6	1.78×10^{-2}	73.4	1575.24
LkGAN-v2-1.6	41.99	647.38	67.80	1504.16
LkGAN-v2-1.8	58.65	3.98×10^{-25}	99.00	**0.00**
LkGAN-v2-2.2	58.65	9.74×10^{-26}	99.00	**0.00**
LkGAN-v2-2.4	58.62	5.74×10^{-3}	91.00	560.20
LkGAN-v2-2.6	58.65	4.32×10^{-26}	98.80	0.16
LkGAN-v2-2.8	58.65	1.20×10^{-26}	98.40	0.44
LkGAN-v2-3.0	58.65	$\mathbf{7.48 \times 10^{-27}}$	97.90	1.49
LkGAN-v2-$[1, 3]$	36.49	736.76	72.2	1093.96
LSGAN-v2	58.65	5.20×10^{-17}	91.40	519.84

Table 4.1: LkGANs experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over ten trials.

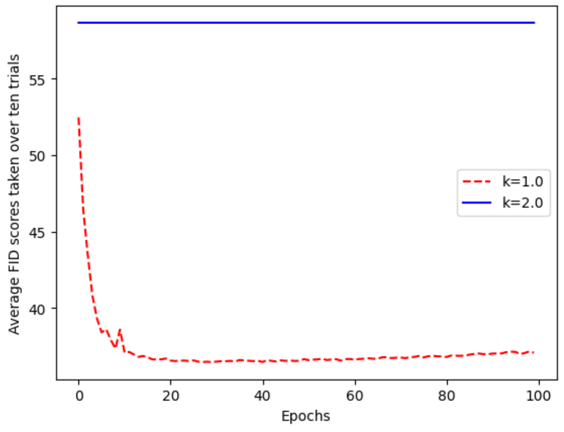
	Average best FID score	Best FID scores variance	Average epoch	Epoch variance
LkGAN-v3-1.0	58.65	3.09×10^{-22}	99.0	0.0
LkGAN-v3-1.2	58.65	1.49×10^{-22}	99.0	0.0
LkGAN-v3-1.4	58.65	7.92×10^{-23}	99.0	0.0
LkGAN-v3-1.6	58.65	4.58×10^{-23}	99.0	0.0
LkGAN-v3-1.8	58.65	2.63×10^{-23}	99.0	0.0
LkGAN-v3-2.2	58.65	9.29×10^{-24}	99.0	0.0
LkGAN-v3-2.4	58.65	6.02×10^{-24}	99.0	0.0
LkGAN-v3-2.6	58.65	3.85×10^{-24}	99.0	0.0
LkGAN-v3-2.8	58.65	2.76×10^{-24}	99.0	0.0
LkGAN-v3-3.0	58.65	$\mathbf{1.69 \times 10^{-24}}$	99.0	0.0
LkGAN-v3-$[1,3]$	58.65	$\mathbf{1.69 \times 10^{-24}}$	99.0	0.0
LSGAN-v3	58.65	1.61×10^{-23}	99.0	0.0

Table 4.2: The average and variance best FID scores for LkGAN-v2 that generated meaningful images and the average and variance of the epoch when this occurs.

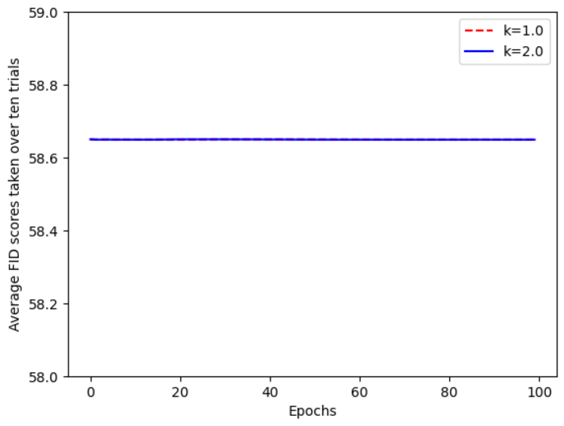
	Average best FID score	Best FID scores variance	Average epoch	Epoch variance
LkGAN-v2-1.0	3.00	1.47×10^{-2}	35.40	55.44
LkGAN-v2-1.2	**2.99**	8.60×10^{-3}	35.00	165.50
LkGAN-v2-1.6	3.13	$\mathbf{5.18 \times 10^{-3}}$	**27.00**	**2.67**
LkGAN-v2-$[1,3]$	3.24	4.80×10^{-2}	32.00	41.50



(a) Average FID scores versus epochs for LkGAN-v1-1.6 and LSGAN-v1.

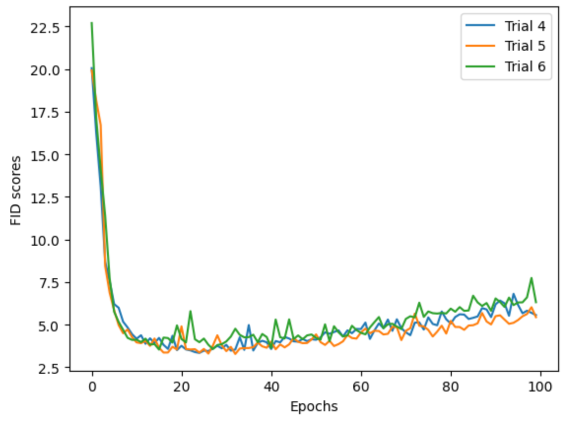


(b) Average FID scores versus epochs for LkGAN-v2-1.0 and LSGAN-v2.

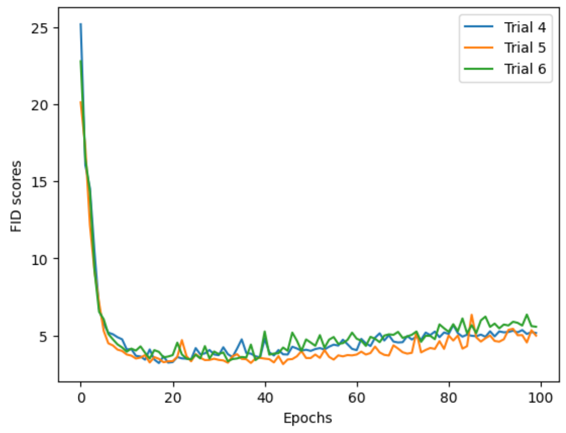


(c) Average FID scores versus epochs for LkGAN-v3-1.0 and LSGAN-v3.

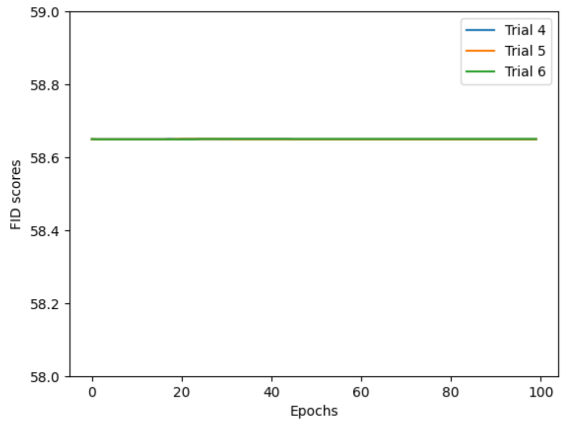
Figure 4.1: Evolution of the average FID scores throughout training for LKGANs.



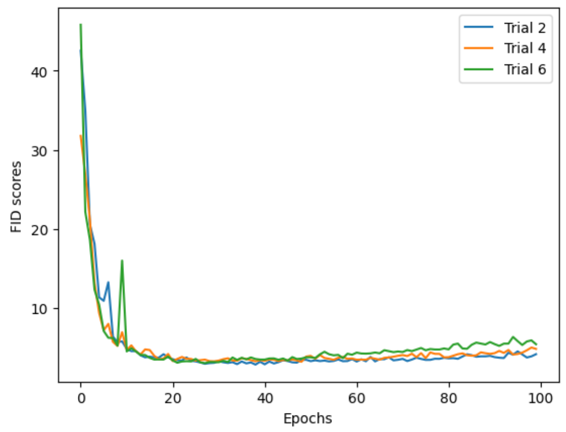
(a) LSGAN-v1.



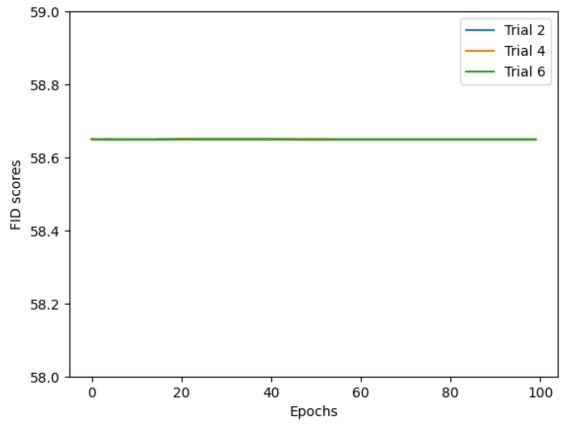
(b) LkGAN-v1-1.6.



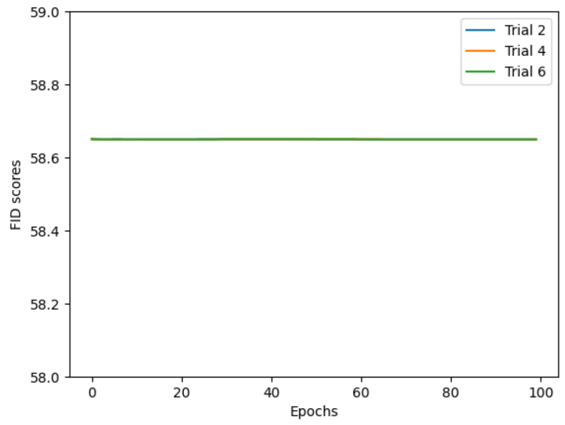
(c) LSGAN-v2.



(d) LkGAN-v2-1.0.

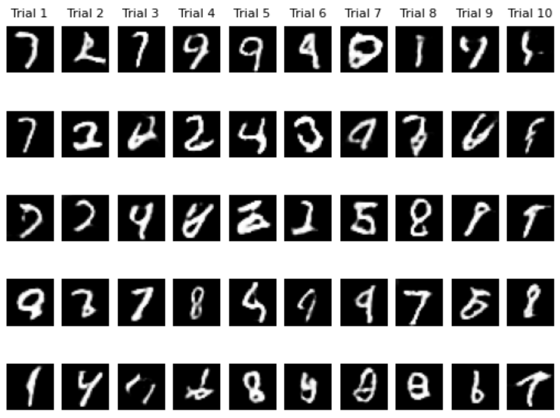


(e) LSGAN-v3.

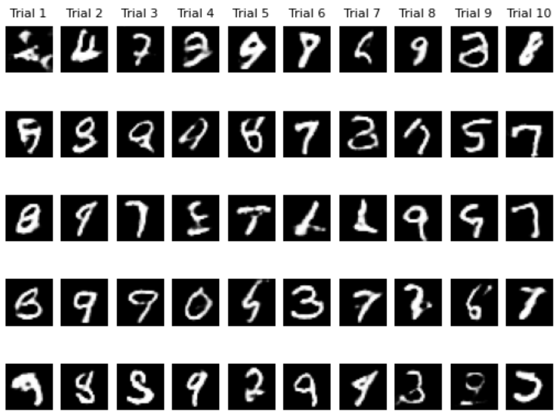


(f) LkGAN-v3-1.0.

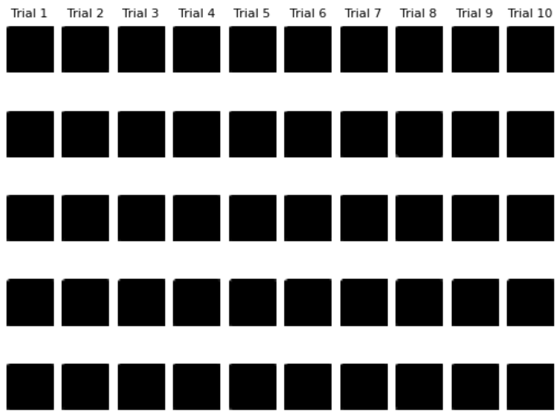
Figure 4.2: Plots of the FID scores versus epochs for the best performing LkGAN for each version and its LSGAN counterpart for a selection of three trials.



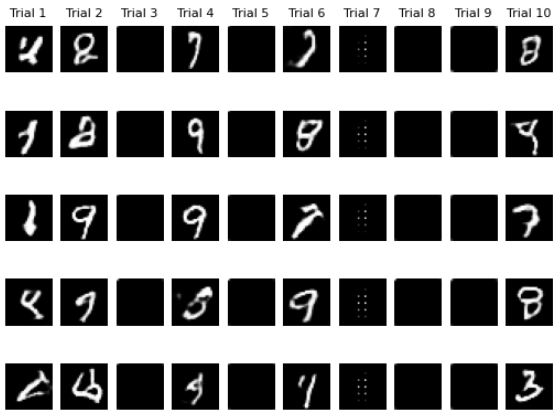
(a) LSGAN-v1 sample images.



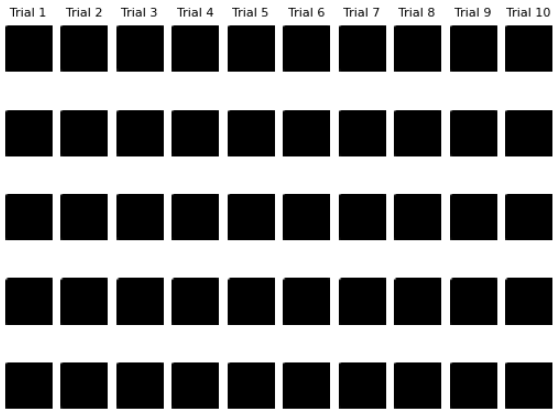
(b) LkGAN-v1-1.6 sample images.



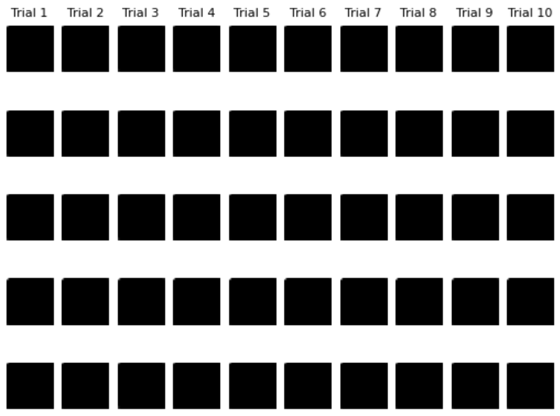
(c) LSGAN-v2 sample images.



(d) LkGAN-v2-1.0 sample images.



(e) LSGAN-v3 sample images.



(f) LkGAN-v3-1.0 sample images.

Figure 4.3: Sample generated images of the best performing LkGAN for each version and its LSGAN counterpart.

CelebA

Due to a significant increase in the computing time for LkStyleGANs, only a limited number of trials were tested. LkStyleGANs with and without the simplified gradient penalty were tested over three trials while controlling the seed in each trial. The addition of the simplified gradient penalty significantly increased training time. Hence only Version 2 of LkStyleGAN-GP was tested for $k = \{1, 2, 3\}$. The FID scores were calculated every $80,000$ images. The lowest FID score over the 25 million images is referred to as the "best FID scores" and is presented for each trial in Table 4.3. The average and variance of best FID scores taken over the three trails are presented in Table 4.4.

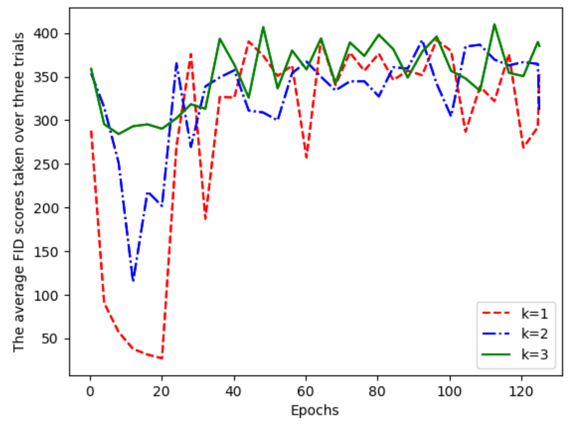
We present the plots of the average FID scores taken over the three trials versus epochs in Figure 4.4. We also present the plots of the best performing LkStyleGAN for all three versions with and without gradient penalty for all three trials in Figure 4.5. We present sample generated images of the best performing LkStyleGANs and LSStyleGANs for each trial in Figures 4.6, 4.7, and 4.8.

Table 4.3: LkStyleGANs experiments on the CelebA dataset: the best FID score over each run seen over three trials.

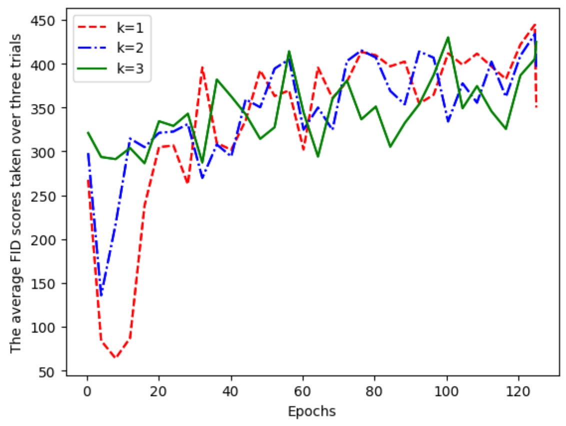
	Trial 1	Trial 2	Trial 3
LkStyleGAN-v1-1.0	26.56	24.89	29.38
LkStyleGAN-v1-3.0	188.11	260.69	224.68
LSStyleGAN-v1	109.35	121.69	92.91
LkStyleGAN-v2-1.0	40.57	96.56	39.02
LkStyleGAN-v2-3.0	266.61	230.39	219.48
LSStyleGAN-v2	128.96	136.18	88.26
LkStyleGAN-v3-1.0	17.11	12.32	27.07
LkStyleGAN-v3-3.0	82.43	34.73	71.25
LSStyleGAN-v3	11.10	17.48	32.80
LkStyleGAN-v2-1.0-GP	4.33	4.21	4.39
LkStyleGAN-v2-3.0-GP	4.76	4.69	4.33
LSStyleGAN-v2-GP	4.07	4.52	4.66

Table 4.4: L*k*StyleGANs experiments on the CelebA dataset: the average and variance of the best FID score and the average and variance epoch this occurs taken over three trials.

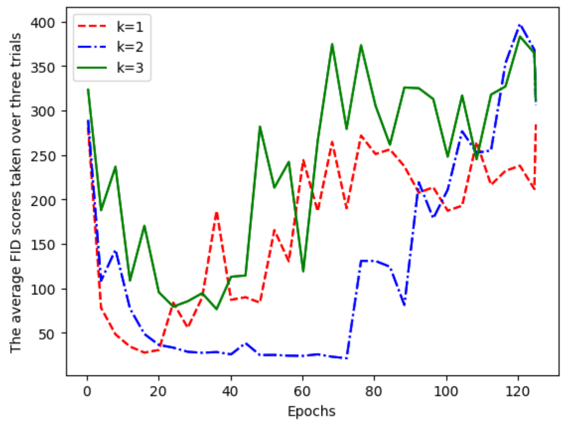
	Average best FID score	Best FID score variance	Average epoch	Epoch variance
L*k*StyleGAN-v1-1.0	**26.94**	**3.43**	20.10	**0.00**
L*k*StyleGAN-v1-3.0	224.49	877.99	26.79	154.36
LSStyleGAN-v1	107.98	138.98	**14.73**	14.36
L*k*StyleGAN-v2-1.0	**58.72**	716.46	**10.71**	**3.59**
L*k*StyleGAN-v2-3.0	238.83	**405.79**	56.27	904.61
LSStyleGAN-v2	117.80	444.99	14.73	154.36
L*k*StyleGAN-v3-1.0	**18.83**	**37.75**	56.26	2358.60
L*k*StyleGAN-v3-3.0	62.80	415.01	**28.13**	172.32
LSStyleGAN-v3	20.46	82.89	79.04	**25.12**
L*k*StyleGAN-v2-1.0-GP	**4.31**	$\mathbf{5.60 \times 10^{-3}}$	124.73	$\mathbf{3.64 \times 10^{-2}}$
L*k*StyleGAN-v2-3.0-GP	4.59	3.54×10^{-2}	**123.26**	3.59
LSStyleGAN-v2-GP	4.42	6.34×10^{-2}	124.87	$\mathbf{3.64 \times 10^{-2}}$



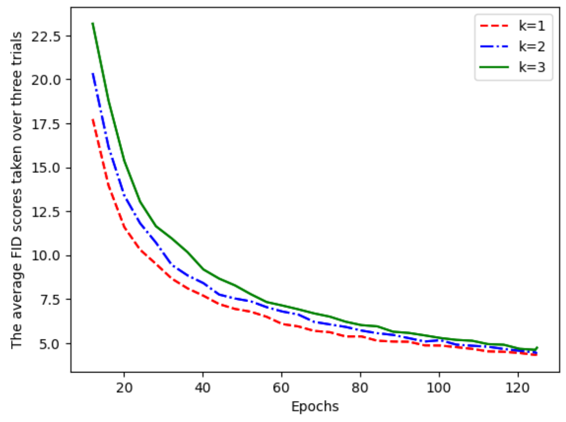
(a) Average FID scores versus epochs for LkStyleGAN-v1.



(b) Average FID scores versus epochs for LkStyleGAN-v2.

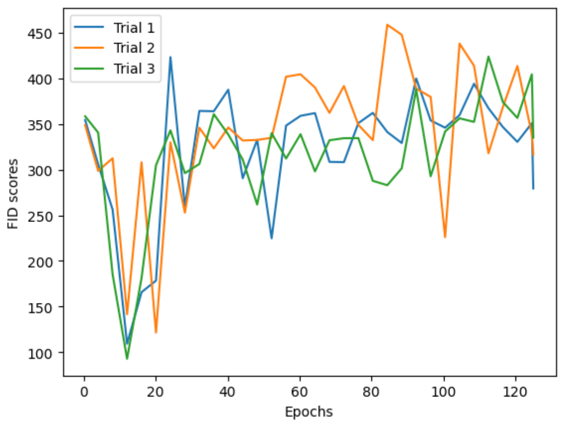


(c) Average FID scores versus epochs for LkStyleGAN-v3.

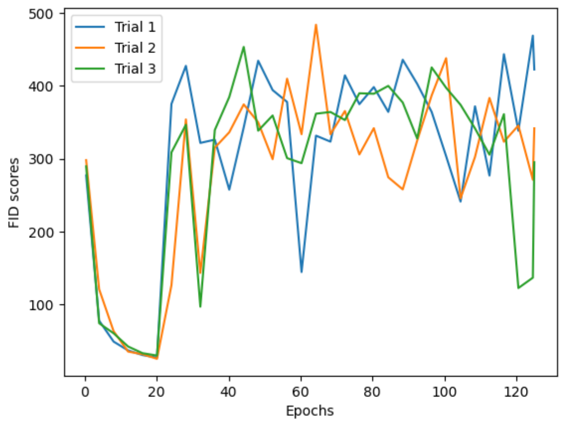


(d) Average FID scores versus epochs for LkStyleGAN-v2-GP.

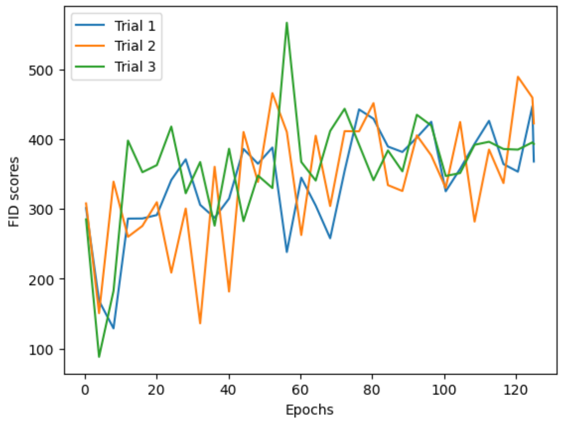
Figure 4.4: Evolution of the average FID scores throughout training for LkStyleGANs.



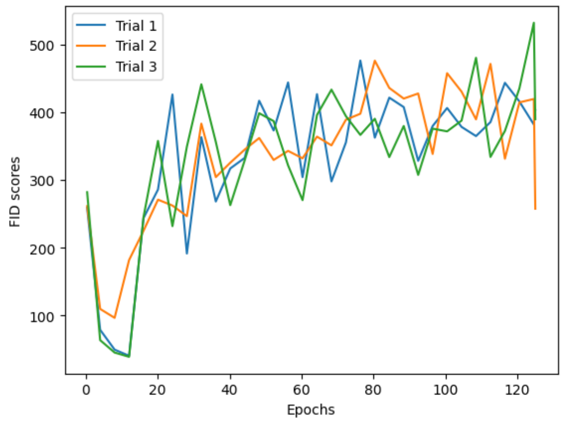
(a) LSStyleGAN-v1.



(b) LkStyleGAN-v1-1.0.

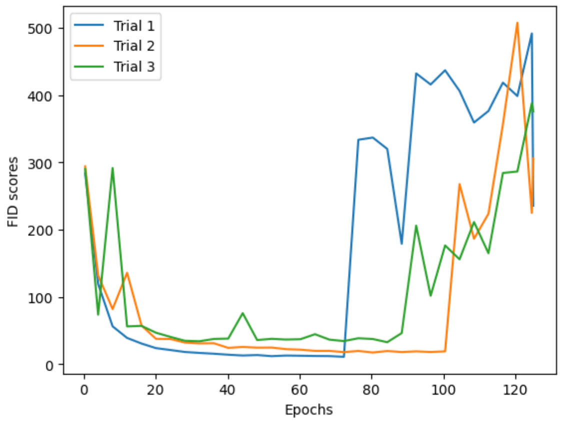


(c) LSStyleGAN-v2.

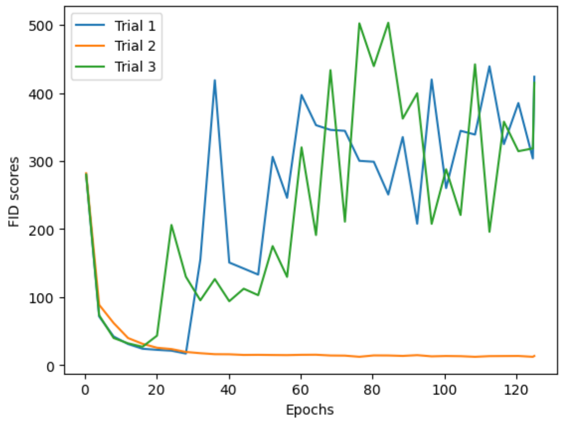


(d) LkStyleGAN-v2-1.0.

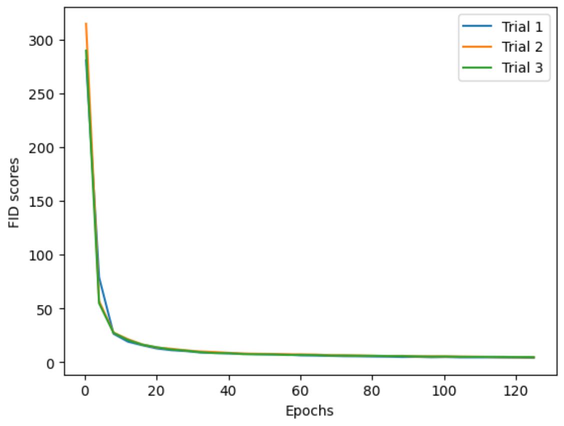
Figure 4.5: Plots of the FID scores versus epochs for the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trials 1, 2, and 3.



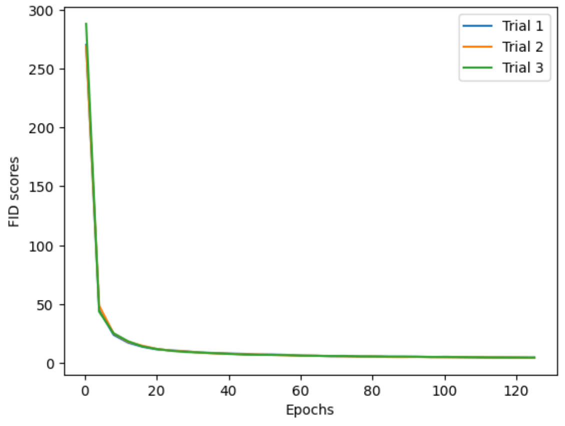
(e) LSStyleGAN-v3.



(f) LkStyleGAN-v3-1.0.

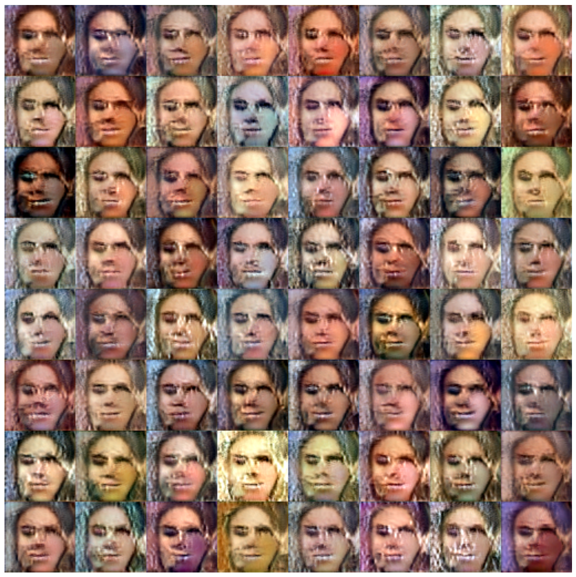


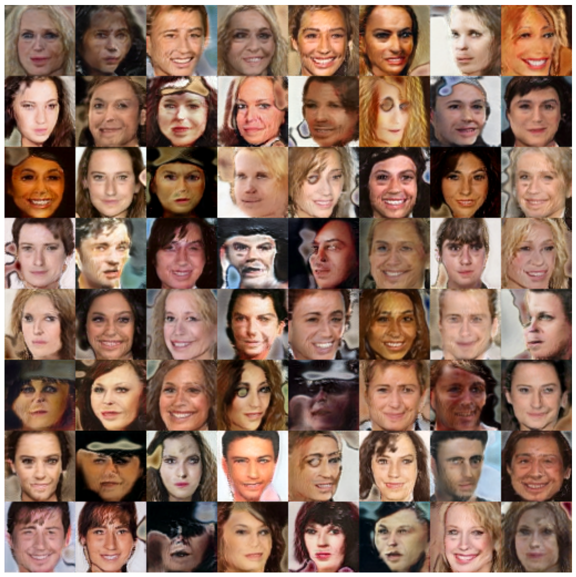
(g) LSStyleGAN-v2-GP.

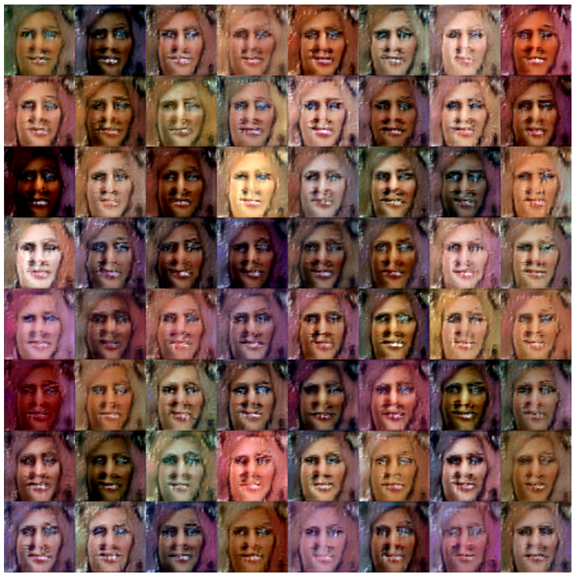


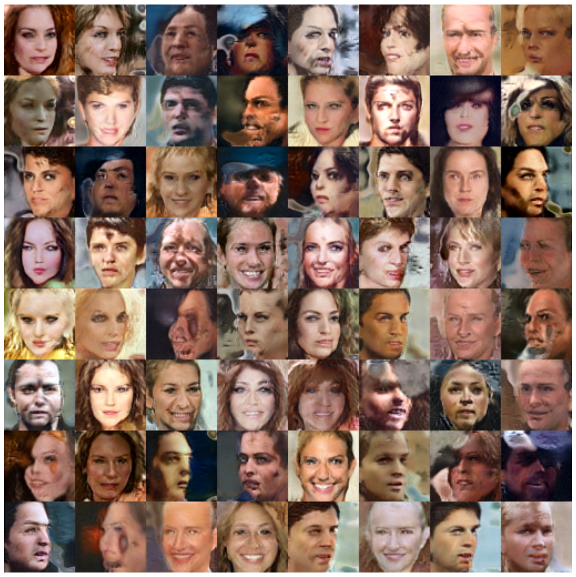
(h) LkStyleGAN-v2-1.0-GP.

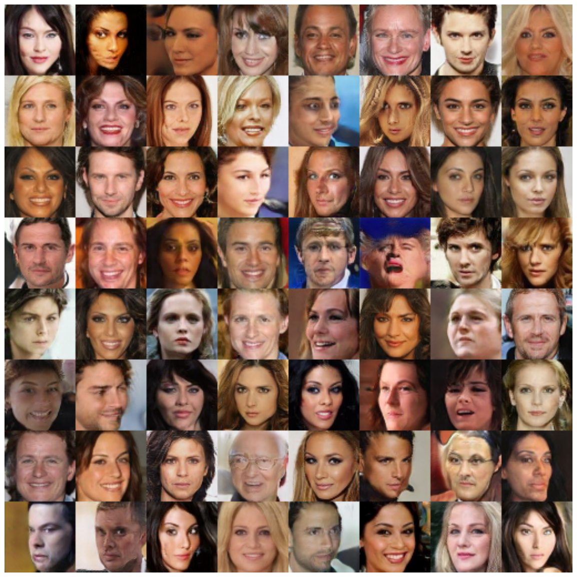
Figure 4.5: Plots of the FID scores versus epochs for the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trials 1, 2, and 3.

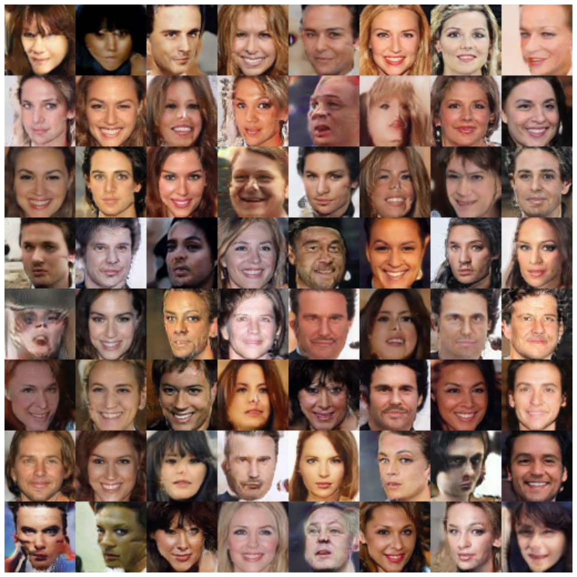

(a) LSStyleGAN-v1: FID score 109.35.

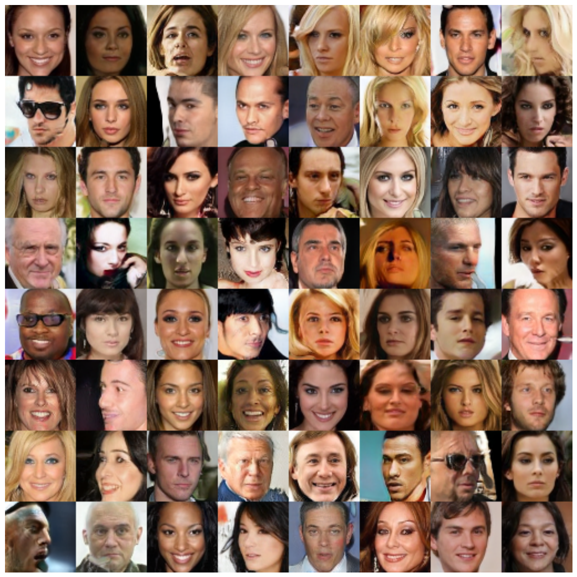

(b) LkStyleGAN-v1-1.0: FID score 26.56.


(c) LSStyleGAN-v2: FID score 128.96.


(d) LkStyleGAN-v2-1.0: FID score 40.57.


(e) LSStyleGAN-v3: FID score 11.10.


(f) LkStyleGAN-v3-1.0: FID score 17.11.


(g) LSStyleGAN-v2-GP: FID score 4.07.

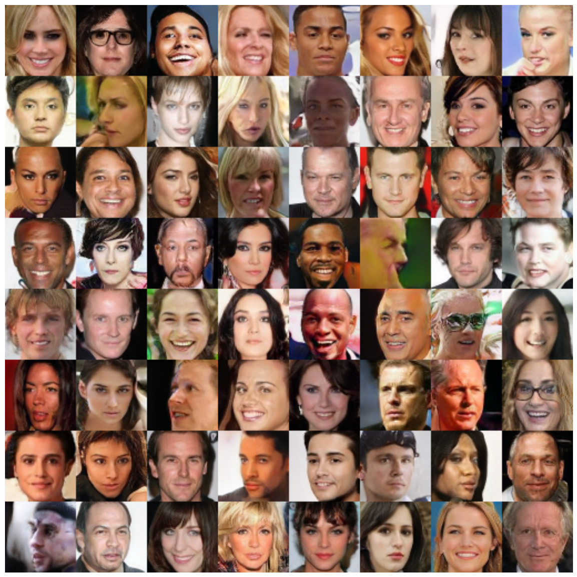
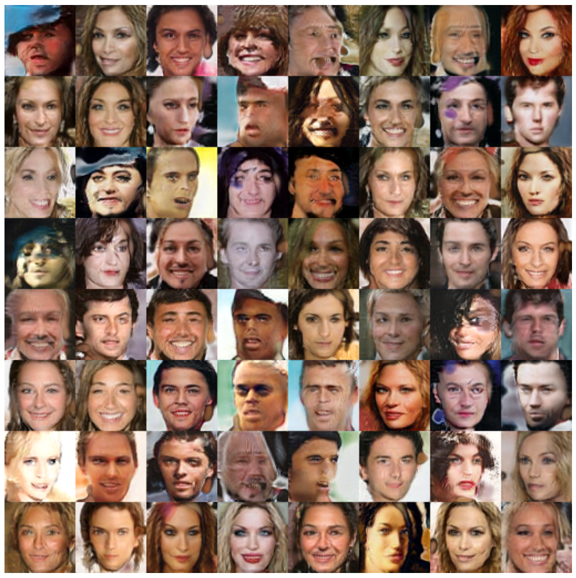

(h) LkStyleGAN-v2-1.0-GP: FID score 4.33.

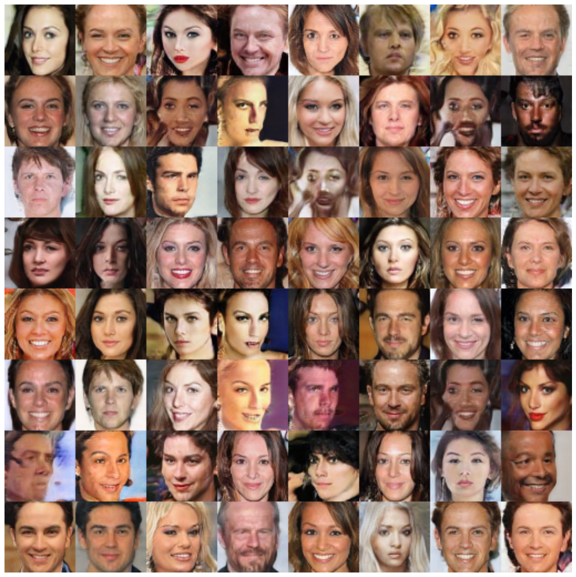
Figure 4.6: Sample generated images of the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trial 1.

(a) LSStyleGAN-v1: FID score 121.69.

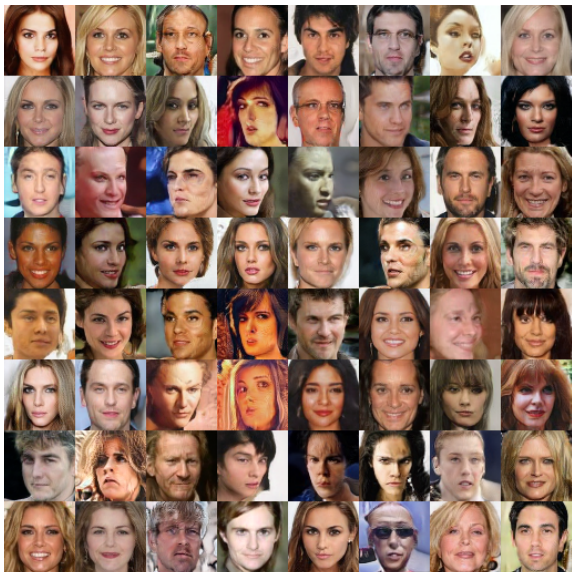

(b) LkStyleGAN-v1-1.0: FID score 24.89.

(c) LSStyleGAN-v2: FID score 136.18.

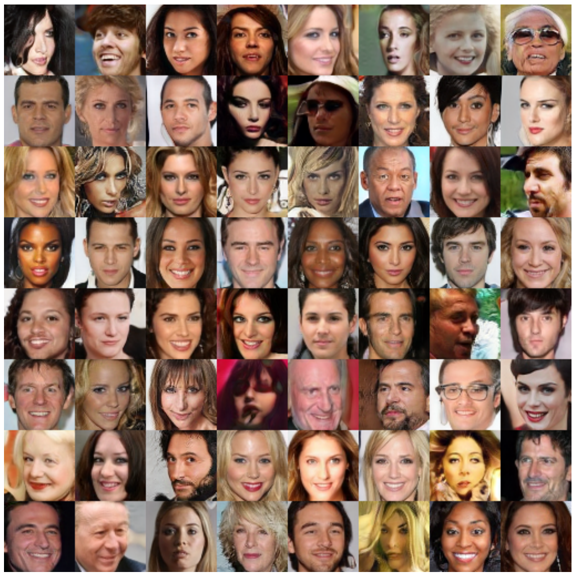
(d) LkStyleGAN-v2-1.0: FID score 96.56.



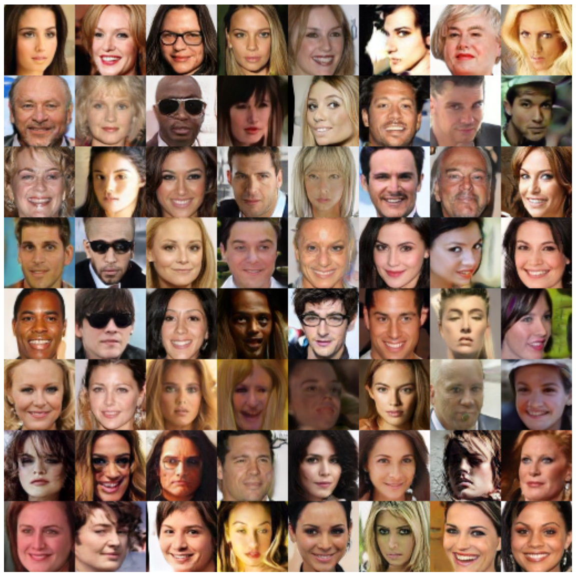
(e) LSStyleGAN-v3: FID score 17.48.



(f) LkStyleGAN-v3-1.0: FID score 12.32.

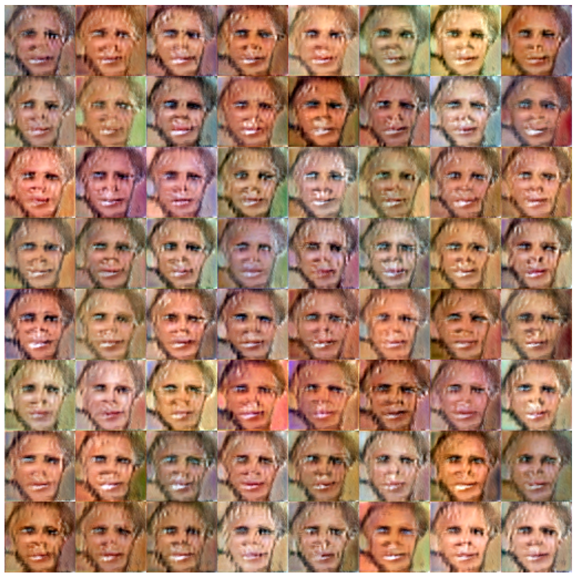


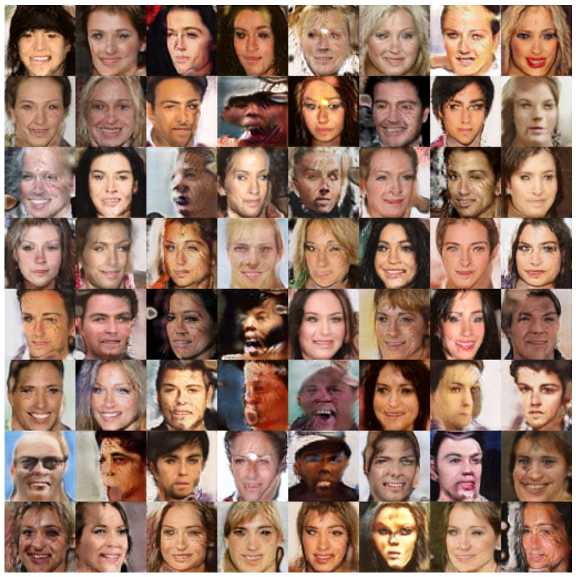
(g) LSStyleGAN-v2-GP: FID score 4.52.

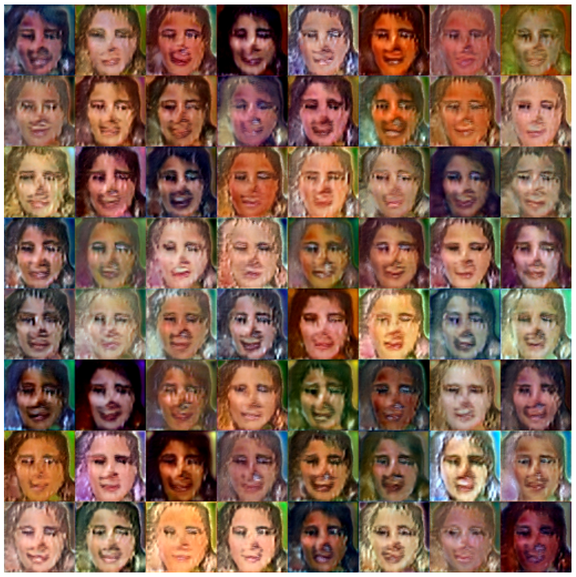


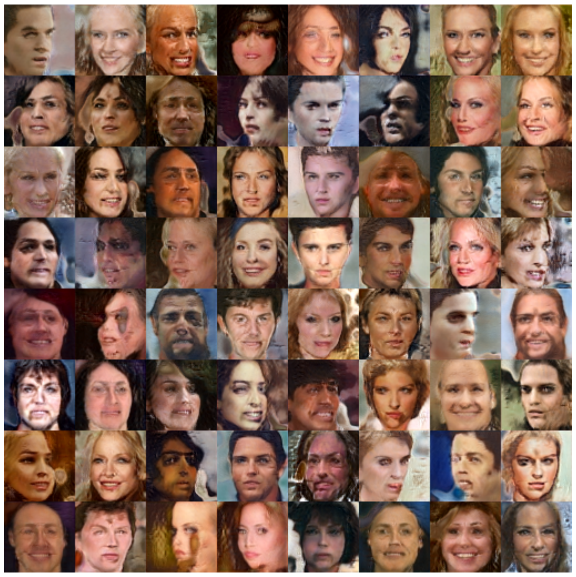
(h) LkStyleGAN-v2-1.0-GP: FID score 4.21.

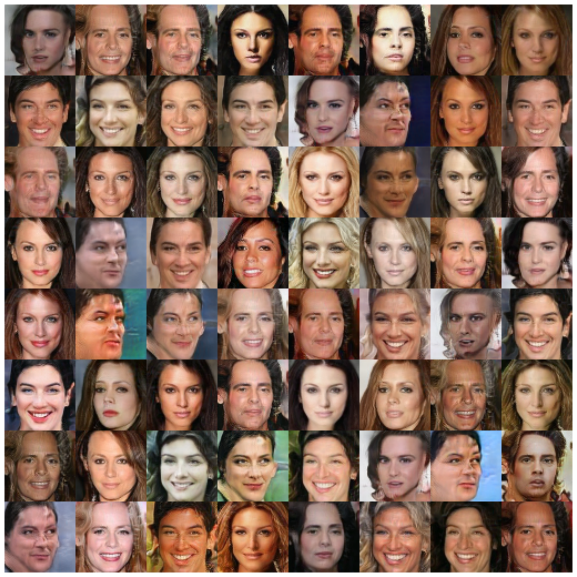
Figure 4.7: Sample generated images of the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trial 2.

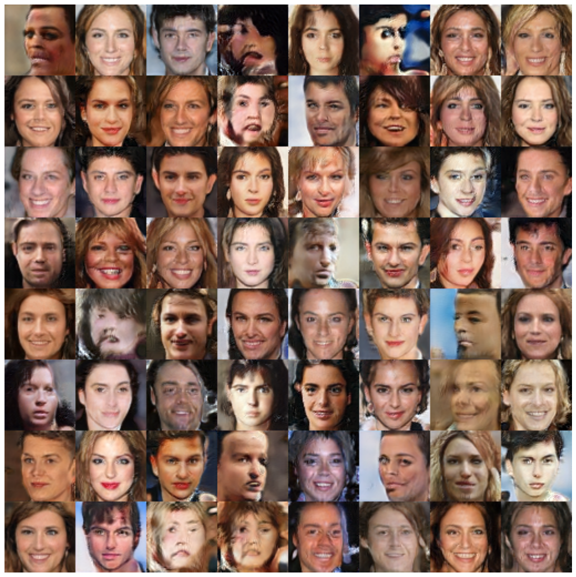

(a) LSStyleGAN-v1: FID score 92.91.

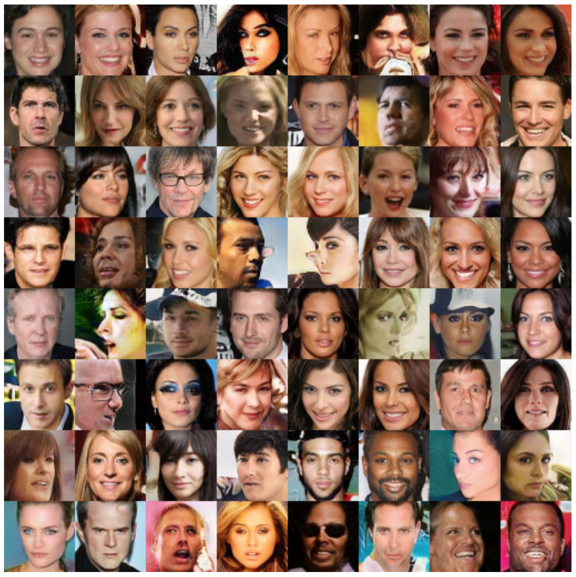

(b) LkStyleGAN-v1-1.0: FID score 29.38.


(c) LSStyleGAN-v2: FID score 88.26.


(d) LkStyleGAN-v2-1.0: FID score 39.02.


(e) LSStyleGAN-v3: FID score 32.80.


(f) LkStyleGAN-v3-1.0: FID score 27.07.


(g) LSStyleGAN-v2-GP: FID score 4.66.

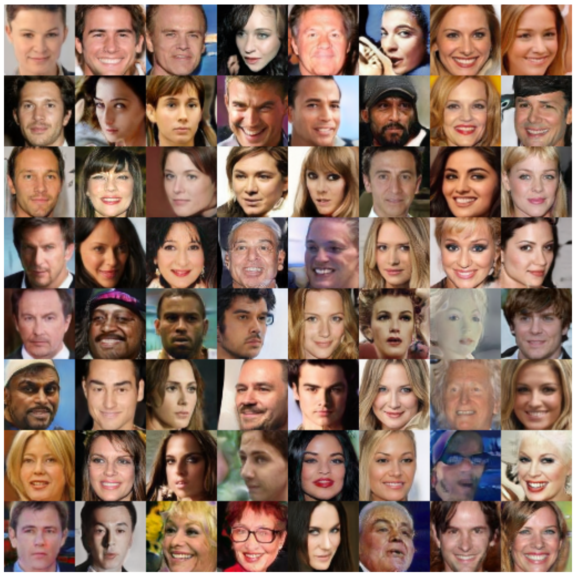

(h) LkStyleGAN-v2-1.0-GP: FID score 4.39.

Figure 4.8: Sample generated images of the best performing LkStyleGANs for each version and their LSStyleGANs counterparts for trial 3.

4.2.3 Discussion

MNIST

When $k < 2$, LkGANs-v1 outperformed LSGANs-v1 in terms of the generated image quality and the rate at which they converged to their best FID scores. On average, most LkGANs-v1 converged to their best FID scores in fewer epochs than LSGAN-v1. However, as training continued, both LSGAN-v1 and LkGAN-v1 FID scores worsened over time when they did converge to meaningful results. This could be because the LkGANs loss functions may produce gradients for generated images that are close to real images for a poorly behaving discriminator, which incorrectly creates gradients for fake images that are close to real images. We hypothesize that this problem can be ameliorated by the use of the simplified gradient penalty, which penalizes the discriminator if it creates non-zero gradient updates for real images. Increasing $k > 2$ has an adverse effect on the generated images' quality. Changing k every epoch has no appreciable effects on the generated image quality. For a single trial, LkGANs took 11.16 minutes to train on one GPU.

For Version 2, LSGAN was unstable during training and suffered from mode collapse in all ten trials. In contrast, LkGAN-v2-1.0 converged five out of ten trials, LkGAN-v2-1.2 and LkGAN-v2-[1, 3] converged four out of ten trials, and LkGAN-v2-1.6 converged three out of ten trials. Table 4.2 shows that LkGAN-v2-1.2 performed the best when it did converge.

The choice of α, β, and γ parameters has a great effect on the quality of the generated images and training stability. Comparing the average best FID scores of LkGANs in Version 1 and Version 2 in Tables 4.1 and 4.2 reveals the fact that

Version 1 improved training stability at the expense of image quality. Indeed, LkGAN-v1 exhibited training stability for all ten trials for each LkGAN tested. In contrast, the best performing LkGAN-v2 was LkGAN-v2-1.0, which exhibited training stability five out of the ten trials. We recall from Equation (4.6) that $V_{k,g}(D^*, g) = |\gamma - \beta|^k |\chi|^k (p_{\mathbf{X}} + p_g \| 2p_g)$. We note that $|\gamma - \beta|^k$ scales the gradients of the generator's updates and hence we hypothesize that the closer this term is to 0, the less the effect the generator's loss function has on the updates of the generator's parameters and hence it dampens extreme fluctuations in the gradient updates.

However, when setting $|\gamma - \beta|^k = 0$, which corresponds to Version 3, we observed that LkGANs and LSGAN were not stable during training. This could be due to the fact that the parameters, α, β, and γ, do not satisfy Theorem 10. Another reason is if the discriminator converges to its global optimum, then the generator is unable to improve the quality of its images when $|\gamma - \beta|^k = 0$. Hence the discriminator is able to perfectly tell apart real data from fake data. It could also imply that the network architecture parameters, such as the discriminator's learning rate, must be fine tuned for LkGANs-v3 and LSGAN-v3 to converge to meaningful results.

Further mathematical analysis and experiments are needed. It would be useful to experiment with different network architectures, different learning rates for the generator's and discriminator's Adam optimizer, and a different optimizer, such as RMSProp. Determining the best choice of parameters α, β, and γ, which allows LkGANs to converge and outperform LSGANs for these different network architecture parameters, is necessary. It also would be useful to analyze and implement LkGANs where $|\gamma - \beta|^k$ changes during training.

CelebA

For the CelebA dataset, the choice of $k = 1$ produced the best performing LkStyleGAN. In contrast, LSStyleGANs and LkStyleGANs-3.0 suffered from mode collapse in all three trials for both Verions 1 and 2; see Figures 4.6a, 4.7a, 4.8a, 4.6c, 4.7c, and 4.8c for the evidence of mode collapse for LSStyleGAN-v1 and LSStyleGAN-v2. For Version 3, LSStyleGAN converged to meaningful results, however, as training continued, it suffered from mode collapse.

Similarly, LkStyleGAN-v1-1.0 and LkStyleGAN-v3-1.0 converged to meaningful results in most trials and suffered from mode collapse as training continued. The exception to this is trial 2 of LkStyleGAN-v3-1.0, which exhibited training stability; see Figure 4.5f. Note that, LkStyleGAN-v2-1.0 only converged in the two out of three trials. It supports the fact that the choice of α, β, and γ parameters have a great effect on the networks' convergent behaviour. However, contrary to the MNIST experiments, LkStyleGAN-v1-1.0 generated image quality was superior to that of the generated images of convergent LkStyleGAN-v2-1.0.

Furthermore, LkStyleGAN-v3-1.0 and LSStyleGAN-v3 converged to meaningful results and outperformed the other versions. This implies that the conditions provided by Theorem 10 do not need to be satisfied for certain network architectures. It supports the hypothesis that the best choice of the α, β, and γ parameters differs for different network architecture parameters.

Without the simplified gradient penalty, these networks took 87.87 hours to train for one trial on one GPU. The addition of the simplified gradient penalty increased training time to 106.35 hours, an increase of 21%. However, the simplified gradient penalty improved the quality of generated images throughout training, as seen in

Figures 4.5h and 4.5g. This confirms our hypothesis that the discriminator creates gradients despite the fact that the generated images are close to real images. The average FID scores increased as k increased, consistent with the previous experiments; see Figure 4.4d.

In summary, the choice of the α, β, and γ parameters has a significant effect on training stability and quality of generated images, and the best choice of these parameters differs for different network architectures. However, when LkGANs, $k = 1$, do converge, they consistently improved the quality of generated images in terms of FID scores, converged to their optimal FID score quicker, and improved training stability compared to their counterpart, LSGANs. Furthermore, they give rise to interesting theoretical problems and experiments for future research.

Chapter 5

RényiGANs

5.1 Theoretical results

Motivated by generalizing the original GANs, we employ the (differential) Rényi cross-entropy loss functional and Jensen-Rényi divergence of parameter $\alpha > 0$, $\alpha \neq 1$, which generalizes the Shannon cross-entropy functional and Jensen-Shannon divergence. We present the RényiGANs loss functions below.

Definition 38 *The* ***RényiGANs loss functions of parameter*** $\boldsymbol{\alpha}$, *where* $\alpha > 0$, $\alpha \neq 1$, *are defined as*

$$\begin{aligned} V_D(D,g) &= -\mathcal{H}(p_{\mathbf{X}}; D) - \mathcal{H}(p_{\mathbf{Z}}; 1 - D \circ g) \qquad (5.1) \\ V_{\alpha,g}(D,g) &= -\mathcal{H}_\alpha(p_{\mathbf{X}}; D) - \mathcal{H}_\alpha(p_{\mathbf{Z}}; 1 - D \circ g), \qquad (5.2) \end{aligned}$$

where $\circ$ *denotes functional composition.*

Note that for RényiGANs, $V_D(D,g) = V(D,g)$, where $V(D,g)$ is the original GANs loss function as defined in (3.1).

The RényiGANs optimization problems are

$$\max_D V_D(D,g) = \max_D \left(-\mathcal{H}(p_{\mathbf{X}};D) - \mathcal{H}(p_{\mathbf{Z}};1-D\circ g)\right) \tag{5.3}$$

$$\min_g V_{\alpha,g}(D,g) = \min_g \left(-\mathcal{H}_\alpha(p_{\mathbf{X}};D) - \mathcal{H}_\alpha(p_{\mathbf{Z}};1-D\circ g)\right), \tag{5.4}$$

The RényiGANs' generator tries to induce the discriminator to classify the fake images as 1 by minimizing the negative sum of the two *Rényi cross-entropy functionals* $\mathcal{H}_\alpha(p_{\mathbf{X}},D)$ and $\mathcal{H}_\alpha(p_{\mathbf{Z}},1-D\circ g)$, hence generalizing the original GANs loss function by employing a richer α-parameterized class of information functionals. We next present a result that shows that as $\alpha \to 1$, we recover the original GANs loss function, $V(D,g)$. The proof is an immediate consequence of Theorem 2.

Theorem 11 *If $V(D,g) < \infty$, then*

$$\lim_{\alpha\downarrow 1} V_{\alpha,g}(D,g) = V(D,g). \tag{5.5}$$

Moreover, if $\mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left(\frac{1}{D(\mathbf{A})}\right) < \infty$ *and* $\mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}}\left(\frac{1}{1-D(g(\mathbf{B}))}\right) < \infty$ *then*

$$\lim_{\alpha\uparrow 1} V_{\alpha,g}(D,g) = V(D,g). \tag{5.6}$$

If the discriminator converges to the optimal discriminator, we show analytically that for any $\alpha > 0$, $\alpha \neq 1$, the optimal generator induces a probability distribution that perfectly mimics the true dataset distribution, as in GANs. This result is formalized as follows.

Theorem 12 *Let $\alpha > 0$, $\alpha \neq 1$, $(\mathbb{X},\mathcal{B}(\mathbb{X}),\mu)$ be the measure space of $n\times n\times 3$ images, and $(\mathcal{Z},\mathcal{B}(\mathcal{Z}),\mu)$ the measure space where $\mathcal{Z}\subset\mathbb{R}^{3n^2}$. Consider the optimization* *problem* (5.3) *for training the discriminator neural network* $D : \mathbb{X} \to [0,1]$ *and* (5.4) *for the generator neural network. Then*

$$D^* \;:=\; \frac{p_{\mathbf{X}}}{p_g + p_{\mathbf{X}}} \text{ (a.e.)}, \tag{5.7}$$

where $D^* = \operatorname{argmax}_D V_D(D, g)$*. If* $D = D^*$*, then*

$$\begin{aligned} V_{\alpha,g}(D^*, g) &= 2\mathrm{JR}_\alpha\left(p_{\mathbf{X}} \| p_g\right) - 2\log(2) \\ &\geq -2\log(2), \end{aligned}$$

with equality if and only if $p_g = p_{\mathbf{X}}$ *(a.e.).*

Proof The proof that the solution to (5.3) is given by $D^* = p_{\mathbf{X}}/(p_{\mathbf{X}}+p_g)$ is analogous to the one in Theorem 7. We have that

$$\mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{Z}}}\left[(1 - D^*(g(\mathbf{B})))^{\alpha-1}\right] = \mathop{\mathbb{E}}_{\mathbf{B}\sim p_{\mathbf{g}}}\left[(1 - D^*(\mathbf{B})))^{\alpha-1}\right].$$

Hence,

$$\begin{aligned} V_{\alpha,g}(D^*, g) &= \frac{1}{\alpha-1}\log\left(\mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left[(D^*(\mathbf{A}))^{\alpha-1}\right]\right) + \frac{1}{\alpha-1}\log\left(\mathop{\mathbb{E}}_{\mathbf{B}\sim p_g}\left[(1 - D^*(\mathbf{B}))^{\alpha-1}\right]\right) \\ &= \frac{1}{\alpha-1}\log\left(\mathop{\mathbb{E}}_{\mathbf{A}\sim p_{\mathbf{X}}}\left[\left(\frac{2p_{\mathbf{X}}(\mathbf{A})}{p_{\mathbf{X}}(\mathbf{A}) + p_g(\mathbf{A})}\right)^{\alpha-1}\right]\right) \\ &\quad + \frac{1}{\alpha-1}\log\left(\mathop{\mathbb{E}}_{\mathbf{B}\sim p_g}\left[\left(\frac{2p_g(\mathbf{B})}{p_{\mathbf{X}}(\mathbf{B}) + p_g(\mathbf{B})}\right)^{\alpha-1}\right]\right) - 2\log(2) \\ &= 2\left[\frac{1}{2}\mathrm{D}_\alpha\left(p_{\mathbf{X}}\middle\|\frac{p_{\mathbf{X}} + p_g}{2}\right) + \frac{1}{2}\mathrm{D}_\alpha\left(p_g\middle\|\frac{p_{\mathbf{X}} + p_g}{2}\right)\right] - 2\log(2) \\ &= 2\mathrm{JR}_\alpha\left(p_{\mathbf{X}} \| p_g\right) - 2\log(2), \end{aligned}$$

which is minimized when $p_g = p_{\mathbf{X}}$ (a.e.). ■

This theorem implies that the introduction of the new loss function does not alter the underlying global equilibrium point of RényiGANs when compared to the classical GANs (which use a Shannon-centric loss function), namely that the minimum is theoretically achieved when the generator's distribution is the true dataset distribution. Using the above Rényi-centric loss function allows control of the shape of the generator's loss function via the α parameter.

5.2 Experiments

5.2.1 Methods

The $28 \times 28 \times 1$ MNIST [47] and $64 \times 64 \times 1$ CelebA [53] datasets were used to test the RényiGANs loss functions. The FID scores were used to evaluate the quality of the generated images and to compare the rate at which the new networks converge to their optimal scores. The structure of the generator and discriminator neural networks were kept constant when testing on each dataset; see A.1 for further details. For comparison, the classical GANs loss functions were also tested on the MNIST dataset.

We denote RényiGAN-α as RényiGANs that use a fixed value value of α during training; see Algorithm 4. For the MNIST dataset, in addition to implementing RényiGAN-α, RényiGANs were tested while altering α for every epoch of the simulation. This changes the shape of the loss function of the generator. However, changing α does not affect the global minimum as for all $\alpha > 0$, the global minimum is realized when $p_{\mathbf{X}} = p_g$. Assuming that a generator $p_g \neq p_{\mathbf{X}}$ is realized such that it is not a local minimum of $V_\alpha(D,g)$ for all $\alpha > 0$, then changing α every epoch creates non-zero gradients at previous local minimums, hence helping the algorithm overcome the problem of getting stuck in local minimums. We denote this as RényiGAN-$[\beta_1, \beta_2]$, with the α value starting at $\alpha = \beta_1$ and ranging over the interval $[\beta_1, \beta_2]$; see Algorithm 5.

One goal was to examine whether the new generalized loss functions have appreciable benefits over the classical GANs loss function and whether it provides better training stability. It is known that deep convolutional GANs (DCGANs) exhibit stability issues which motivate us to investigate modifications to the loss functions involving the addition of the L_1 norm. Specifically, those stability issues arise when the GANs generator tries to minimize its cost function to $-\infty$ by labelling $D(g(z)) = 1$ for all fake images $g(z)$. In the early stages of the simulations, if the discriminator does not successfully converge to its optimal value and the generator is able to induce the discriminator to label poorly generated images as 1, then in later epochs, once the discriminator converges to its optimal value and is able to tell apart real and fake images perfectly, the generator's loss function produces no gradients. In other words, the optimal discriminator does not allow the generator to improve the quality of fake images which leads to the discriminator winning problem. A similar argument was noted in [9]. Thus to remedy the stability problem, a modified RényiGANs' generator's loss function was tested by taking the L_1 norm of its deviation from $-2\log(2)$, its theoretically minimal value predicted by Theorem 12; this yields the following minimization problem for the generator network:

$$\min_g \Big| V_{\alpha,g}(D,g) - (-2\log 2) \Big| \tag{5.8}$$

Using the L_1 norm ensures that the generator's loss function does not try to label its images as 1, but rather tries to label them as 1/2. Hence in the early training stages, if the generator converges to images that are labelled 1/2 by the discriminator, then in the later stages, if the discriminator converges to its theoretical optimal value (given in Theorem 12), the generator's loss function has non-zero gradient updates and is only able to label fake images as 1/2 when $p_g = p_{\mathbf{X}}$. Note that the altered loss function in (5.8) translates into composing the LkGAN error function using $k = 1$ and $\gamma = -\log(2)$ (see Chapter 4) with the RényiGAN loss function. Indeed, the improved stability property of LkGANs (particularly when $k = 1$) is the main motivation for using this L_1 normalization. We denote the resulting scheme under (5.8) by RényiGAN-L_1.

The RényiGANs and classical GANs loss functions were also tested with and without the addition of simplified gradient penalties. We denote this as RényiGAN-GP and DCGAN-GP.

In summary, four version evaluations were considered with six different loss functions within each version. Version 1 has RényiGAN-0.5, RényiGAN-3.0, RényiGAN-$[0, 0.9]$, RényiGAN-$[0, 3.0]$, RényiGAN-$[1.1, 4]$, and DCGAN. Version 2 has the six original loss functions with L_1 normalization, Version 3 has gradient penalty, and Version 4 has gradient penalty and L_1 normalization incorporated in the loss functions.

For the MNIST dataset, seeds 123, 5005, 1600, 199621, 60677, 20435, 15859, 33764, 79878, 36123 were used for trials 1 to 10, respectively. Tables A.1 and A.2 in the Appendix detail the generator's and discriminator's architecture used for testing the loss functions on MNIST. For the SGD algorithm, the Adam optimizer with a learning rate of $\alpha_{Adam} = 2 \times 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-7}$ was used for the networks as recommended by [67]. The batch size was chosen to be 100 for the $60,000$ MNIST images. The networks were trained on the MNIST dataset for a total number of 250 epochs, or 15 million images.

For CelebA, seeds 1000, 2000, and 3000 were chosen for trials 1, 2, and 3 respectively. For comparison, the original StyleGAN with the classical GANs loss function was implemented. The publicly available StyleGAN code from [41] was modified to test the RényiGANs loss functions. We refer to this as RényiStyleGANs. RényiStyleGANs were implemented for $\alpha \in \{3.0, 9.0\}$. RényiStyleGANs and StyleGANs with and without the simplified gradient penalty was also tested. We denote this as RényiStyleGAN-GP and StyleGAN-GP.

The original StyleGAN architectural defaults were left in place for RényiStyleGANs. The batch size was chosen to be 128. Unlike the original StyleGAN, which changes the resolution of its generated images during training, the resolution of the generated images for RényiStyleGANs and was fixed at $64 \times 64 \times 3$ throughout training. Similarly, the StyleGANs that were used to compare RényiStyleGANs have a fixed generated images resolution of $64 \times 64 \times 3$. As recommended by the original StyleGANs paper, the Adam optimizer with a learning rate of $\alpha_{Adam} = 0.001$, $\beta_1 = 0.0, \beta_2 = 0.99$ and $\epsilon = 10^{-8}$ was used as the SGD algorithm. The RényiStyleGANs were trained for 25 million images or roughly 120 epochs.

One NVIDIA GP100 GPU and two Intel Xeon 2.6 GHz $E7 - 8867$ v3 CPUs were used for training RényiGANs and DCGANs, and four NVIDIA V 100 GPUs were used for training RényiStyleGANs and StyleGANs.

5.2.2 Results

MNIST

As in Chapter 4, a total of ten trials were run while controlling the random seed in each trial. We did not use the Inception network to calculate the FID scores because it is not trained on classifying handwritten MNIST images. Instead, the FID scores were calculated using the raw real and fake images. The distribution of raw real and fake images was assumed to be a multivariate Gaussian distribution.

The best FID score is the lowest FID score over 250 epochs. The average and variance of all ten best FID scores, and the average and variance of the epoch when the best FID score is achieved are presented the results in Table 5.1. The table of best FID scores for all trials (Table A.6) is in the Appendix. A few variants of RényiGANs converged to meaningful results. These networks' results are presented in Table 5.2.

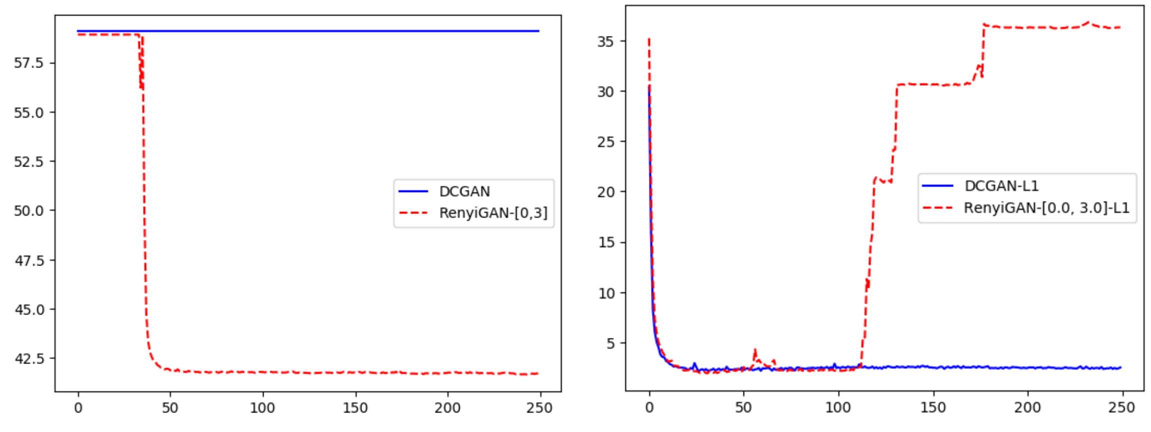
The plots of average FID score of the best performing RényiGAN versus epochs for each version are presented in in Figure 5.1 and its DCGAN counterpart. The average was calculated over ten trials. Also presented are the plots of FID scores over epochs of each version's DCGANs and the best performing RényiGAN for three selected trials in Figure 5.2. We show sample images for each trial generated by the best performing RényiGAN according to Table 5.1 in Figure 5.3. We also present the generated images produced by DCGANs in Figure 5.3.

Table 5.1: RényiGANs experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over ten trials.

	Average best FID score	Best FID scores variance	Average epoch	Epoch variance
RényiGAN-0.5	58.70	1.16×10^{-6}	27.80	81.56
RényiGAN-3.0	52.99	292.71	32.70	92.21
RényiGAN-$[0, 0.9]$	58.60	8.21×10^{-3}	25.80	149.56
RényiGAN-$[\mathbf{0}, \mathbf{3}]$	**41.59**	693.61	95.40	9096.64
RényiGAN-$[1.1, 4]$	58.83	8.18×10^{-3}	41.00	48.67
DCGAN	59.061	$\mathbf{2.01 \times 10^{-28}}$	**13.00**	**0.80**
RényiGAN-0.5-L_1	2.21	7.57×10^{-3}	37.40	160.04
RényiGAN-3.0-L_1	1.80	2.95×10^{-3}	86.80	4611.16
RényiGAN-$[0, 0.9]$-L_1	2.16	6.36×10^{-3}	38.10	242.89
RényiGAN-$[\mathbf{0}, \mathbf{3}]$-$\mathbf{L_1}$	**1.77**	4.90×10^{-3}	**36.10**	**36.80**
RényiGAN-$[1.1, 4]$-L_1	1.81	$\mathbf{3.18 \times 10^{-3}}$	141.80	7326.96
DCGAN-L_1	1.93	3.83×10^{-3}	52.30	2605.61
RényiGAN-GP-0.5	1.37	4.36×10^{-3}	215.40	606.64
RényiGAN-GP-3.0	**1.36**	2.51×10^{-3}	209.10	751.09
RényiGAN-GP-$[0, 0.9]$	**1.36**	2.74×10^{-3}	222.80	423.96
RényiGAN-GP-$[0, 3]$	1.41	3.09×10^{-3}	201.90	1209.69
RényiGAN-GP-$[1.1, 4]$	**1.36**	4.35×10^{-3}	**201.70**	1144.81
DCGAN-GP	**1.36**	$\mathbf{1.45 \times 10^{-3}}$	225.20	**342.56**
RényiGAN-GP-0.5-L_1	1.18	$\mathbf{2.95 \times 10^{-3}}$	212.50	624.45
RényiGAN-GP-3.0-$\mathbf{L_1}$	**1.17**	3.62×10^{-3}	221.70	**609.61**
RényiGAN-GP-$[0, 0.9]$-L_1	1.19	3.61×10^{-3}	212.20	1425.36
RényiGAN-GP-$[0, 3]$-L_1	1.22	6.33×10^{-3}	224.10	1075.09
RényiGAN-GP-$[1.1, 4]$-L_1	1.20	3.46×10^{-3}	209.20	1326.36
DCGAN-GP-L_1	1.18	1.58×10^{-3}	**200.50**	1263.05

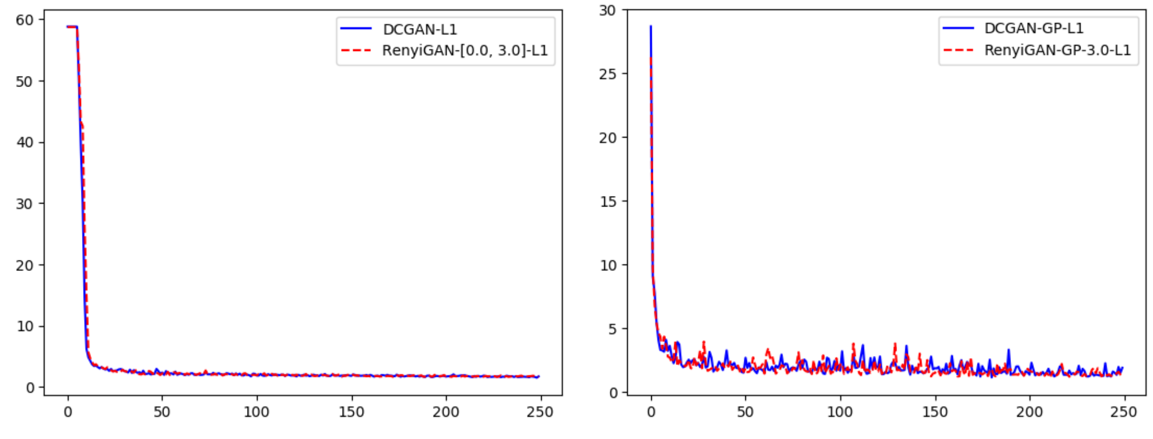
Table 5.2: The average and variance best FID scores for RényiGANs that generated meaningful images and the average and variance of the epoch when this occurs.

	Average best FID score	Best FID scores variance	Average epoch	Epoch variance
RényiGAN-3.0	1.66	**0.00**	**60.00**	**0.00**
RényiGAN-[0,3]	**1.36**	$3.81\ \times 10^{-3}$	240.00	6.00



(a) Average FID scores versus epochs for RényiGAN-$[0,3]$ and DCGAN.

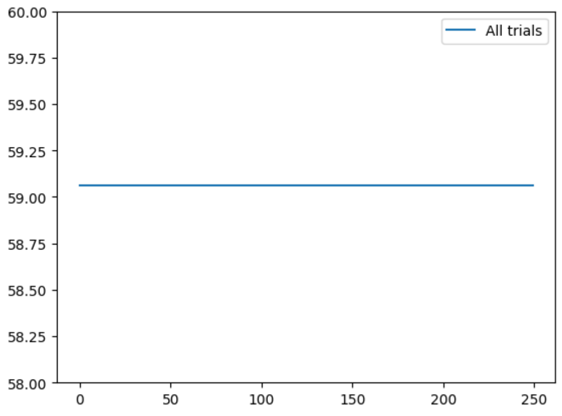
(b) Average FID scores versus epochs for RényiGAN-$[0,3]$-L_1 and DCGAN-L_1.



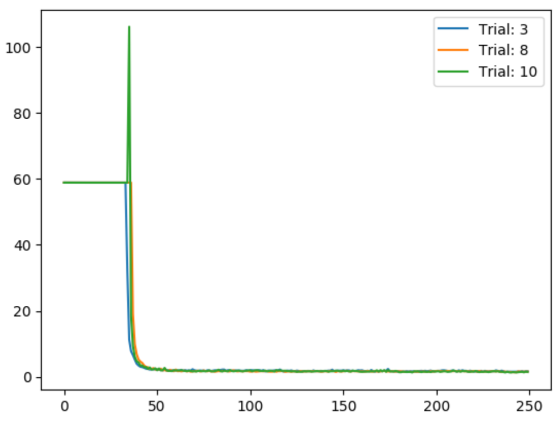
(c) Average FID scores versus epochs for RényiGAN-GP-3.0 and DCGAN-GP.

(d) Average FID scores versus epochs for RényiGAN-GP-3.0-L_1 and DCGAN-GP-L_1.

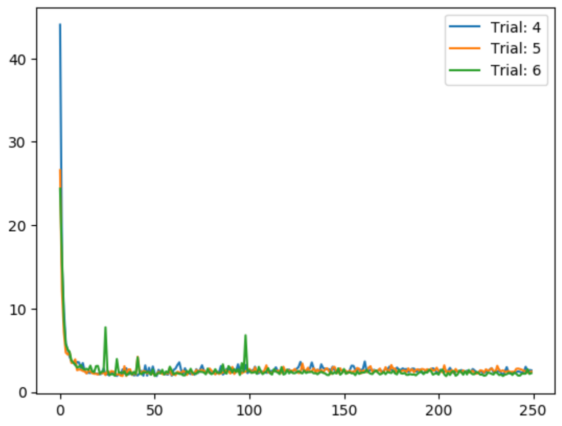
Figure 5.1: Evolution of the average FID scores throughout training for RényiGANs.



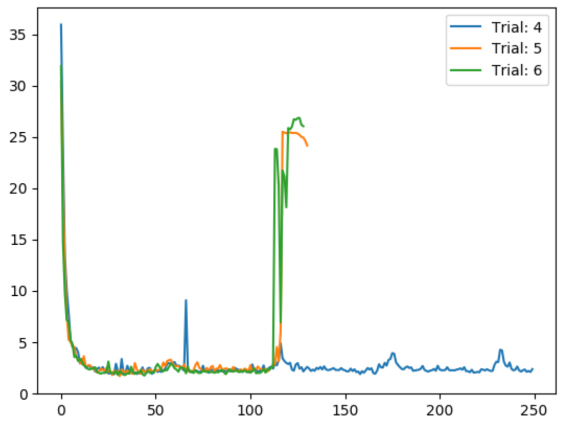
(a) DCGAN.



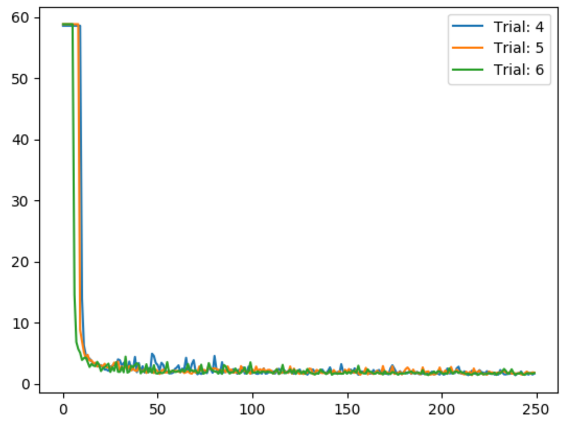
(b) RényiGAN-$[0, 3]$.



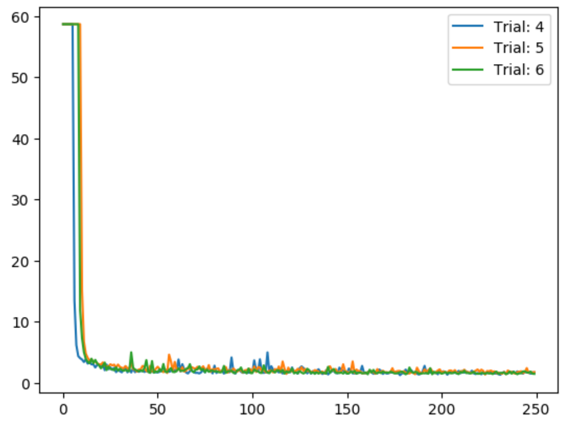
(c) DCGAN-L_1.



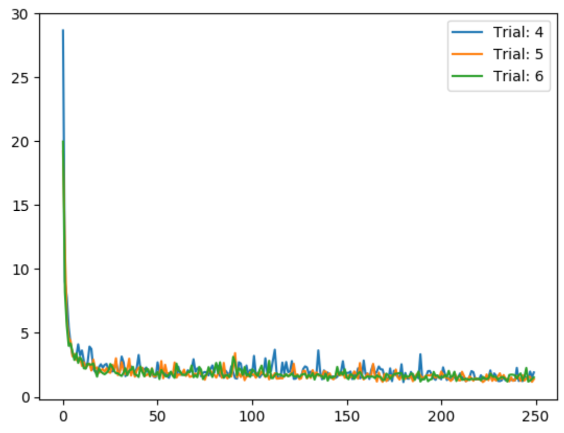
(d) RényiGAN-$[0, 3]$-L_1.



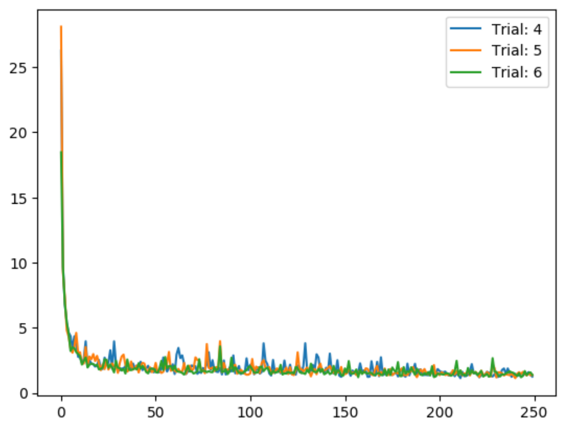
(e) DCGAN-GP.



(f) RényiGAN-GP-3.0.

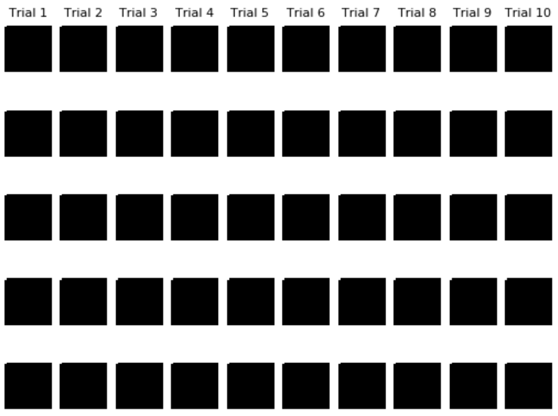


(g) DCGAN-GP-L_1.

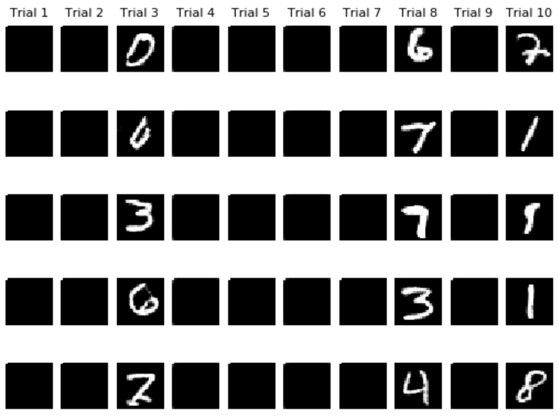


(h) RényiGAN-GP-3.0-L_1.

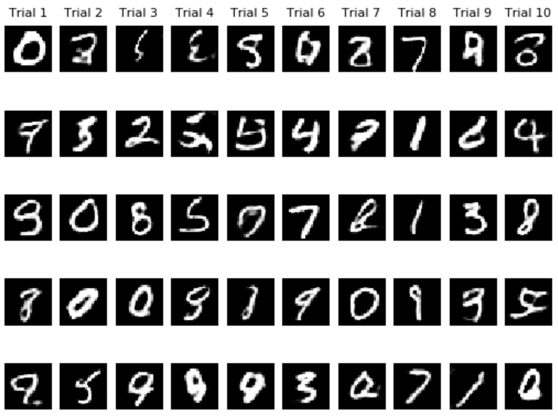
Figure 5.2: Plots of the FID scores versus epochs for the best performing RényiGAN for each version and its DCGAN counterparts for a selection of three trials.



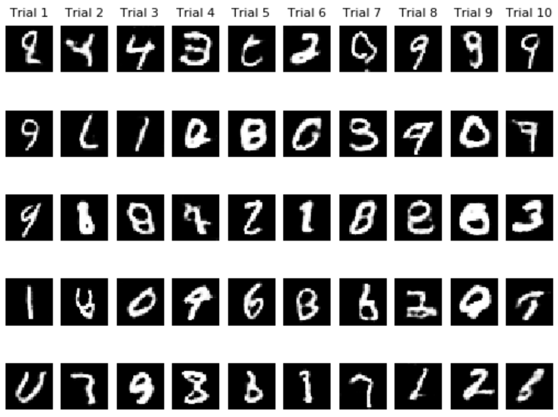
(a) DCGAN sample images.



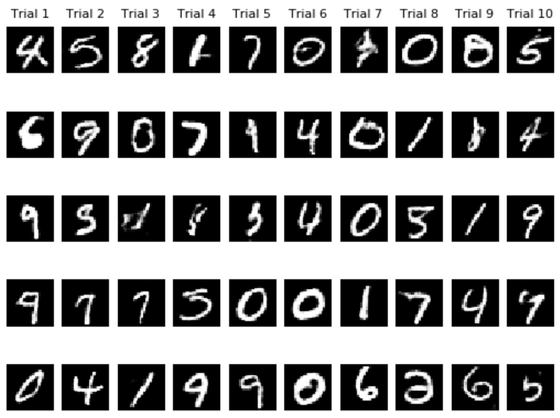
(b) RényiGAN-$[0, 3]$ sample images.



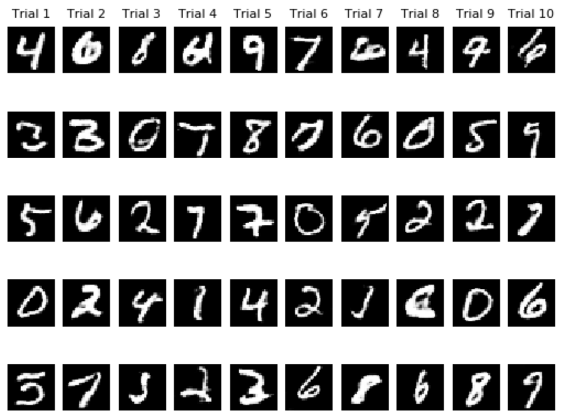
(c) DCGAN-L_1 sample images.



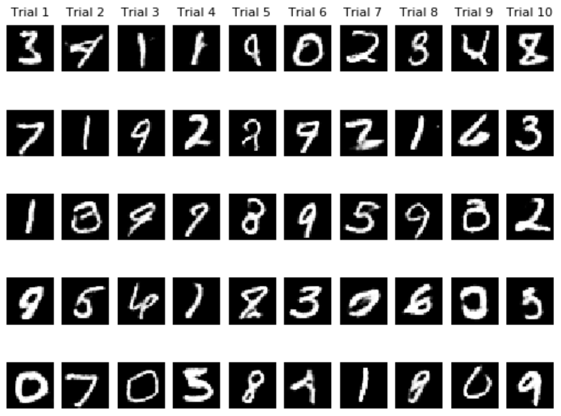
(d) RényiGAN-$[0, 3]$-L_1 sample images.



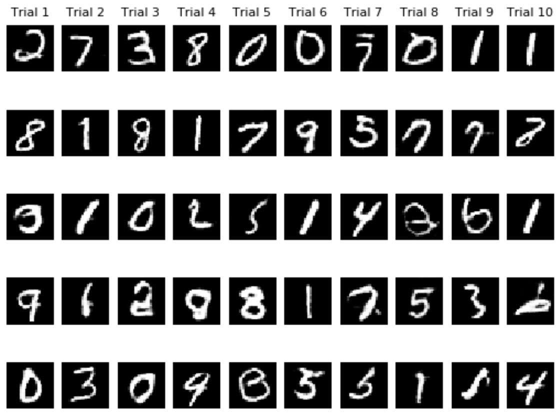
(e) DCGAN-GP sample images.



(f) RényiGAN-GP-3.0 sample images.



(g) DCGAN-GP-L_1 sample images.



(h) RényiGAN-GP-3.0-L_1 sample images.

Figure 5.3: Sample generated images of the best performing RényiGAN in terms of FID scores for each version and its DCGAN counterpart.

CelebA

Only three trials were run for a few select α values for RényiStyleGANs due to the significant increase in computing time. RényiStyleGANs with and without the simplified gradient penalty were also tested over three trials while controlling the seed in each trial. Only RényiStyleGAN-3.0-GP was tested because the addition of the simplified gradient penalties increased computing time. The FID scores were calculated every $80,000$ images. The best FID scores refers to the lowest FID score during the networks' training. The best FID scores for each trial are presented in Table 5.3. The average and variance of best FID scores taken over the three trails are presented in Table 5.4.

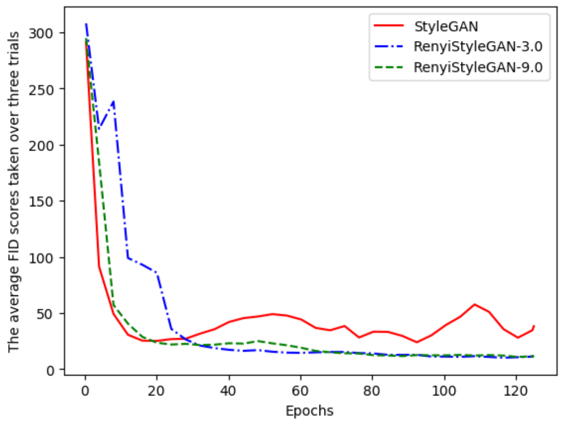
We present the plots of the average FID scores taken over the three trials versus epochs in Figure 5.4. We also present the plots of the best performing RényiStyleGAN with and without gradient penalty for all three trials in Figure 5.5. We present sample generated images of the best performing RényiStyleGAN and StyleGAN in Figures 5.6, 5.7, and 5.8.

Table 5.3: RényiStyleGANs experiments on the CelebA dataset: the best FID over each run seen over three trials.

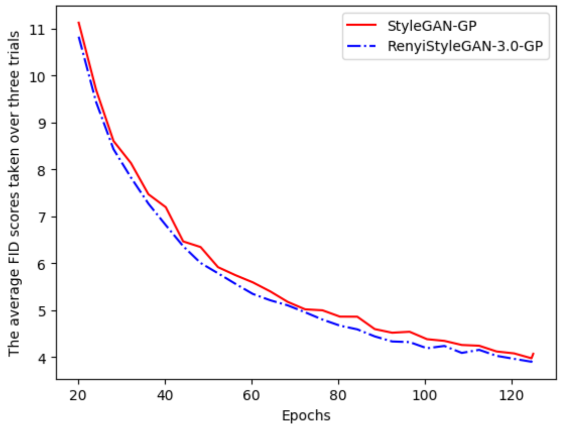
	Trial 1	Trial 2	Trial 3
RényiStyleGAN-3.0	9.67	9.60	10.14
RényiStyleGAN-9.0	11.22	8.33	11.59
StyleGAN	16.20	9.70	17.90
RényiStyleGAN-3.0-GP	3.91	3.92	3.82
StyleGAN-GP	4.06	3.90	3.85

Table 5.4: RényiStyleGANs experiments on the CelebA dataset: the average and variance of the best FID score and the average and variance this occurs taken over three trials.

	Average best FID score	Best FID score variance	Average epoch	Epoch variance
RényiStyleGAN-3.0	**9.80**	$\mathbf{5.93 \times 10^{-2}}$	112.54	**75.38**
RényiStyleGAN-9.0	10.38	2.12	115.22	111.28
StyleGAN	14.60	12.48	**93.78**	477.53
RényiStyleGAN-3.0-GP	**3.88**	$\mathbf{1.84 \times 10^{-3}}$	122.05	**4.35**
StyleGAN-GP	3.92	9.06×10^{-3}	**119.37**	58.94

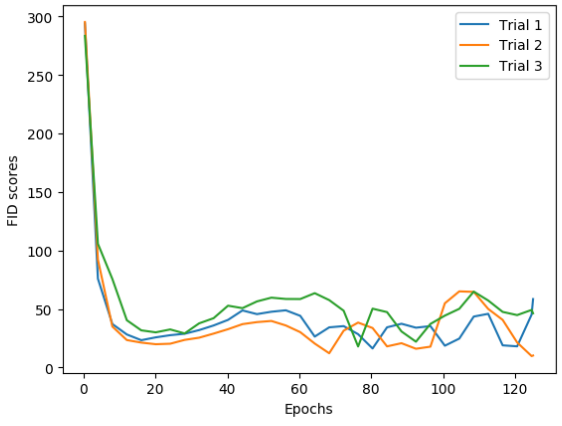


(a) Average FID scores versus epochs for RényiStyleGANs and StyleGANs.

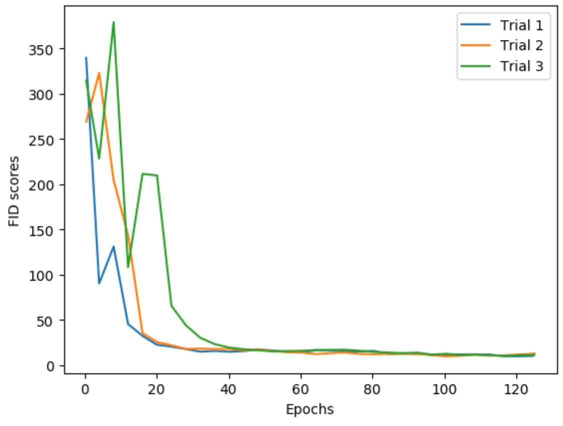


(b) Average FID scores versus epochs for RényiStyleGAN-GPs and StyleGAN-GPs.

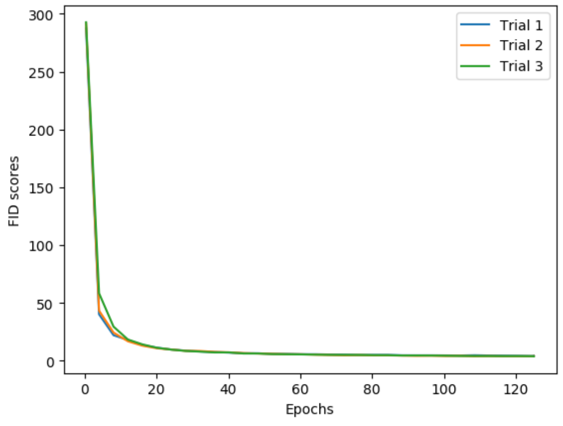
Figure 5.4: Evolution of average FID scores throughout training for RényiStyleGANs.



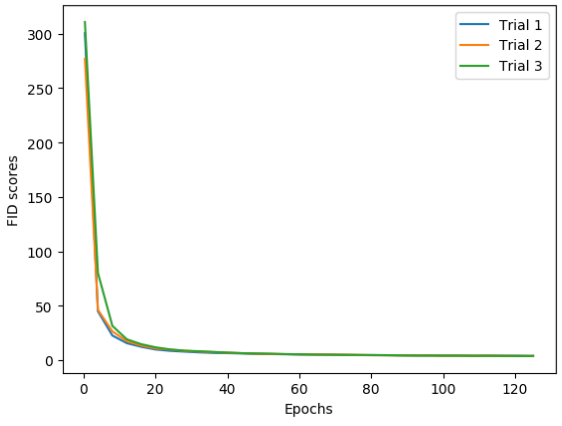
(a) StyleGAN.



(b) RényiStyleGAN-3.0.

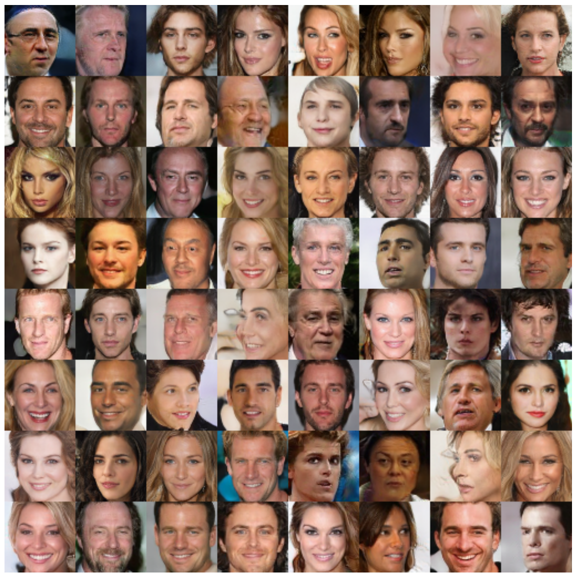


(c) StyleGAN-GP.

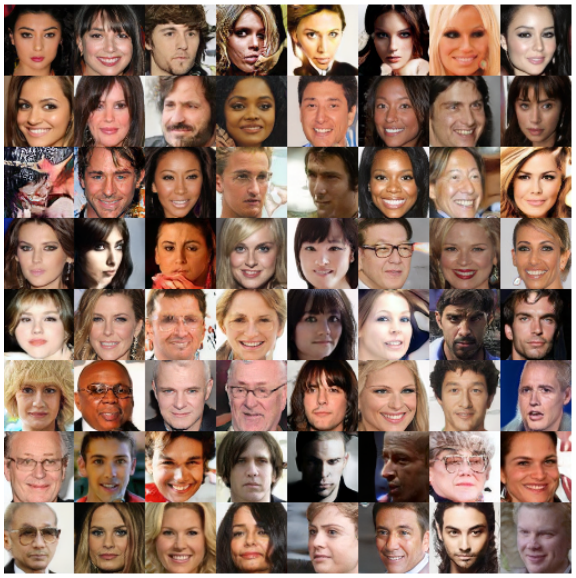


(d) RényiStyleGAN-GP-3.0.

Figure 5.5: Plots of the FID scores versus epochs for the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trials 1, 2, and 3.


(a) StyleGAN: FID score 16.20.

(b) RényiStyleGAN-3.0: FID score 9.67.


(c) StyleGAN-GP: FID score 4.06.

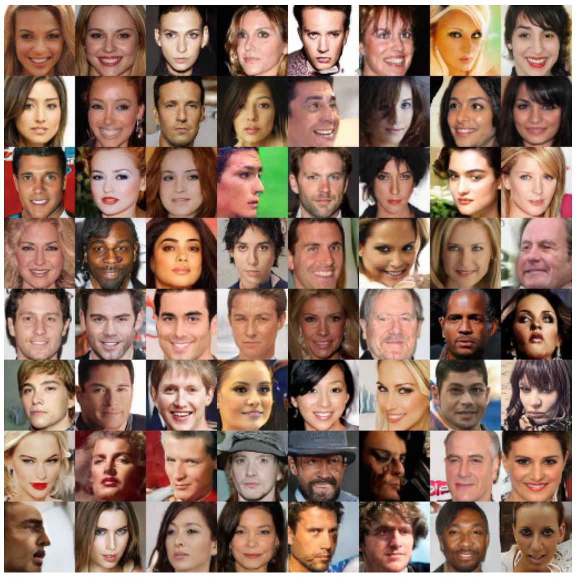
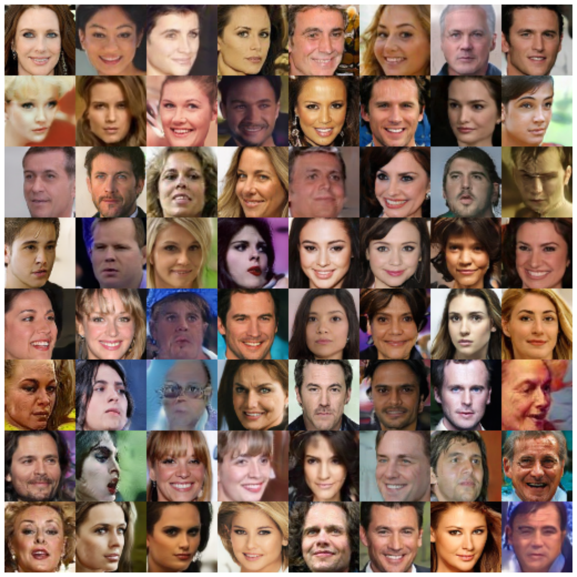
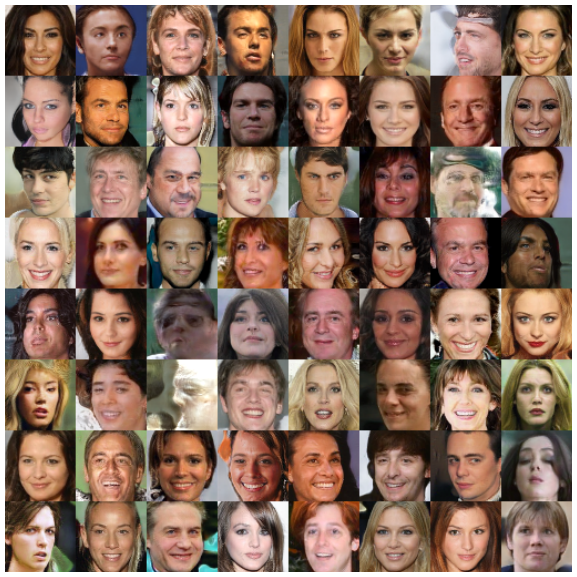
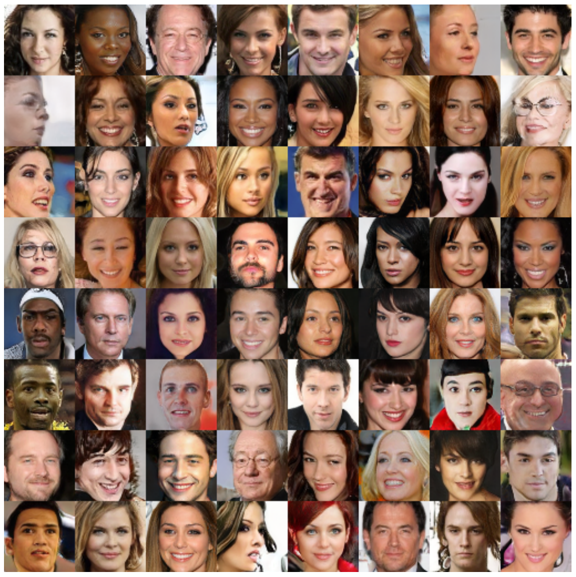

(d) RényiStyleGAN-3.0-GP: FID score 3.91.

Figure 5.6: Sample generated images of the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trial 1.


(a) StyleGAN: FID score 9.70.


(b) RényiStyleGAN-3.0: FID score 9.60.


(c) StyleGAN-GP: FID score 3.90.

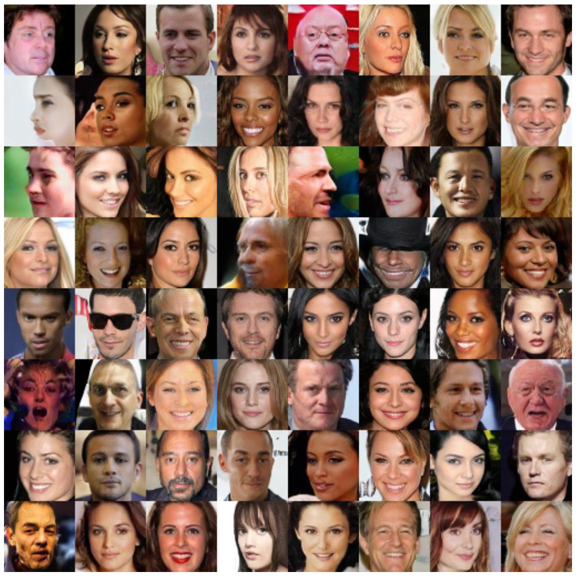
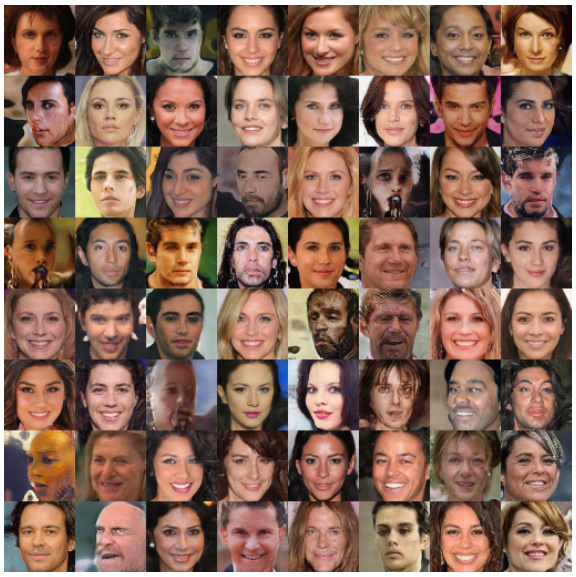
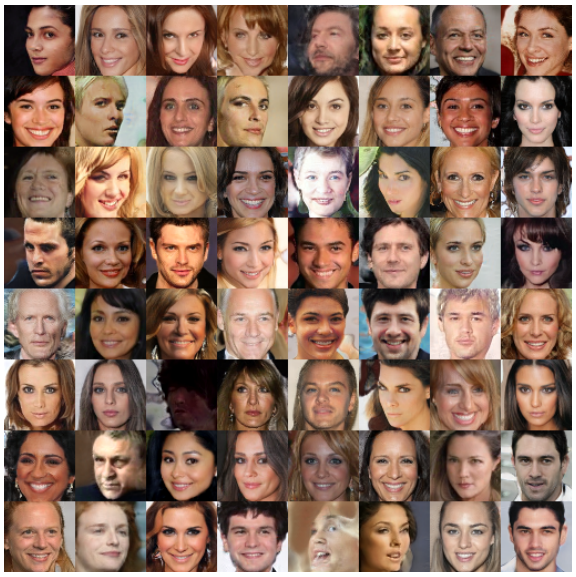
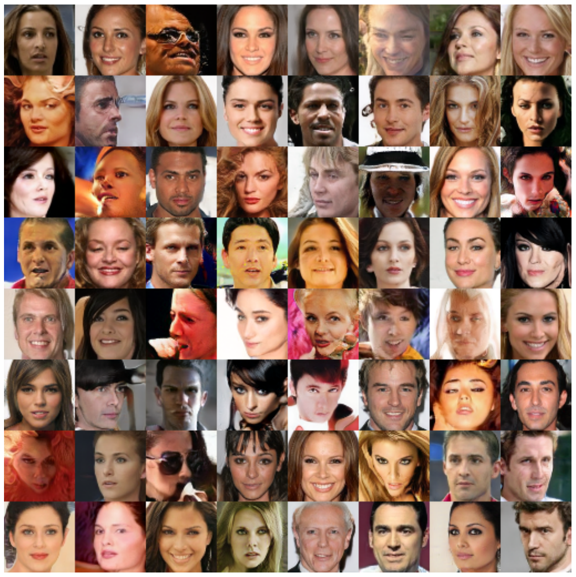

(d) RényiStyleGAN-3.0-GP: FID score 3.92.

Figure 5.7: Sample generated images of the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trial 2.


(a) StyleGAN: FID score 17.90.


(b) RényiStyleGAN-3.0: FID score 10.14.


(c) StyleGAN-GP: FID score 3.85.

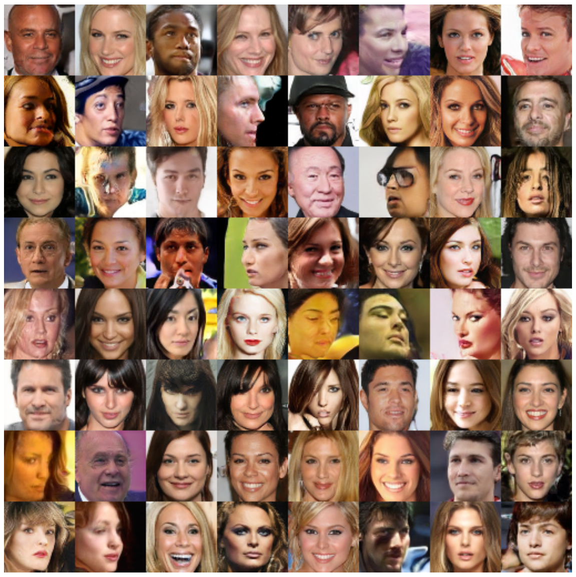

(d) RényiStyleGAN-3.0-GP: FID score 3.82.

Figure 5.8: Sample generated images of the best performing RényiStyleGANs for each version and their StyleGANs counterparts for trial 3.

5.2.3 Discussion

MNIST

The DCGAN baseline exhibits unstable training as was expected and the addition of the Rényi loss is able to ameliorate convergence but has similar instabilities. More specifically, RényiGAN-$[0, 3]$ converged in three out of ten trials, achieving an average best FID scores of 1.36, while DCGAN experienced mode collapse in all ten trials. This FID score is comparable to that of applying simplified gradient penalty to the network. Note that these networks took an average time of 42.33 minutes to train for one trial.

Applying the L_1 normalization drastically improved the convergence of all networks with no computational overhead. In fact, on average over 250 epochs and 10 trials, adding L_1 normalization on average decreased the training time for one trial to 41.79 minutes, which is a decrease of 1.27%. This is expected as the L_1 normalization is similar to the LkGANs generator loss function when $k = 1$ and $\gamma = -\log(2)$, which, as we have shown through experiments, improves training stability. Using L_1 normalization also has the added benefit of networks converging to an optimal FID value in fewer epochs than any other convergent networks across all versions. We note that RényiGAN-$[0, 3]$-L_1 outperforms all other loss functions in Version 2 and it is sufficient to train it within 50 epochs. The development of a rigorous mathematical theory that describes this phenomenon is an interesting future direction to better understand the dynamics of GANs.

In Version 3, RényiGAN-GP-$[1.1, 4]$ performs among the best compared to other RényiGAN-GP variants, with an identical performance to DCGAN-GP. Moreover, on average it converges to its best FID score in fewer epochs than DCGAN-GP. Note, however, that the use of gradient penalty increases the average computation time to 47.54 minutes for a single trial, an increase of 12.30% compared to Version 1. The best performing network in terms of FID score is RényiGAN-GP-3.0-L_1, seen in the Version 4 results of Table 5.1. Note that RényiGAN-GP-0.5-L_1, RényiGAN-GP-3.0-L_1, RényiGAN-GP-$[0, 0.9]$-L_1, and DCGAN-GP-L_1 exhibit quite similar FID scores as the difference of 0.02 FID score has no perceivable qualitative effect on the generated images. However, on average, DCGAN-GP-L_1 converges to its optimal FID score in fewer epochs than its counterparts. On average, these networks with gradient penalty and L_1 normalization took 47.17 minutes to train for a single trial, which is a slight decrease in computational time compared to applying simplified gradient penalties only.

In summary, the extra degree of freedom provided by the α parameter yields a variety of new loss functions and algorithmic designs that gives equivalent or better FID scores in fewer epochs when using either L_1 normalization or the simplified gradient penalty. Note that a difference of 0.20 FID score has no noticeable qualitative difference in MNIST generated images. In the first version, DCGAN-L_1 and RényiGAN-$[0, 3]$-L_1's generated images are qualitatively similar. Correspondingly, there is no discernible difference between Versions 2 and 3. The perceivable disparity in quality is between the first and the second versions, which has a difference in FID score of 0.41; see Figure 5.3. Moreover, the meaning of FID scores diminishes after a certain threshold when the generated images are realistic. It is useful to conduct experiments to determine these thresholds for commonly used datasets. Hence, the greatest advantage of RényiGANs when applied to MNIST is its ability to converge to realistic and diverse generated images quicker than DCGANs in most versions.

CelebA

For CelebA, we observed that RényiStyleGANs (with $\alpha > 1$) outperform StyleGANs in terms of FID scores, with setting $\alpha = 3.0$ achieving the best average FID score. However, further investigation on the best range of values of α for RényiStyleGANs is necessary.

Comparing the performance of RényiStyleGAN in Figure 5.5b to StyleGAN in Figure 5.5a reveals that RényiStyleGAN performs consistently and does not display the erratic unstable behaviour of regular StyleGANs. This is also observed in Figure 5.4a. One explanation for the difference in performance dynamics is that the Rényi loss dampens the loss of each individual sample in the batch, reducing the effect of samples that may be given spurious gradient directions. Combined with our use of the Adam optimizer to keep track of the gradient variance, the overall effect that dampening has on the entire objective function is normalized out, while still maintaining the benefit of dampening individual samples from the generator. Additionally, as we close the gap between StyleGANs with and without gradient penalty, one benefit of not needing gradient penalty is the significant reduction in computation time: RényiStyleGAN-3.0 takes 25.63 hours without gradient penalty and 31.8 hours with gradient penalty, yielding a 24% increase in computation time when using gradient penalty. Lastly, both RényiStyleGAN-GP and StyleGAN-GP perform identically; see also Figures 5.4b, 5.5c, and 5.5d.

Chapter 6

Conclusion

In this thesis, we introduced two new generator loss functions. Motivated by generalizing LSGANs, we introduced LkGANs, which were analyzed and implemented. We showed that the theoretical minimum when solving the generator optimization problem for LkGANs is achieved when the generator's distribution is equal to the true distribution of data. Using experiments on MNIST and CelebA datasets, the new LkGANs loss functions conferred greater training stability and better generated image quality than LSGANs. Experiments also revealed new theoretical and experimental research directions, such as analyzing the effects of the α, β, and γ parameters on LkGANs performance. It would also be useful to understand the interplay between the optimal choice of network architectures and parameters and different choices of k, α, β, and γ parameters.

A GANs generator loss function based on Rényi cross-entropy measures of order α ($\alpha > 0$ and $\alpha \neq 1$) was next proposed, analyzed and implemented. It was shown that the classical GANs analytical minimax result expressed in terms of minimizing the Jensen-Shannon divergence between the generator and the unconstrained discriminator distributions can be generalized for any α in terms of the broader Jensen-Rényi divergence, with the original GANs loss function provably recovered in the limit of α approaching 1. We demonstrated via experiments on MNIST and CelebA datasets that the proposed loss function yields performance improvements over the original GANs loss function in terms of the quality of the generated images and training stability. In particular, RényiGANs used with L_1 normalization does not need gradient penalty to reduce the GAN mode collapse problem. Furthermore, RényiStyleGANs provide more robust convergence dynamics than StyleGANs and can dispose of using gradient penalty without affecting image fidelity while requiring considerably less training time. Finally, we note that the Rényi-centric approach studied in this work can be judiciously adopted to other deep learning neural network architectures. For future research, it would be useful to examine the effects of the α parameter on the optimal choice of network architectures and parameters.

Future directions of applications include applying these networks on different datasets, such as high-dimensional genomic data. For future work, InfoGANs can also be generalized by using the Arimoto-Rényi mutual information of order α, ($\alpha > 0$ and $\alpha \neq 1$) and deriving a Rényi-centric lower bound, which recovers the $L_I(g, q)$ lower bound as a special case. Another direction for research is to generalize Wasserstein GANs.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alessandro Achille and Stefano Soatto. Where is the information in a deep neural network? *ArXiv:1905.12213*, 2019.

[3] Fady Alajaji and Po-Ning Chen. *An Introduction to Single-User Information Theory*. Springer, 2018.

[4] Fady Alajaji, Po-Ning Chen, and Ziad Rached. Csiszár's cutoff rates for the general hypothesis testing problem. *IEEE Transactions on Information Theory*, 50(4):663–678, 2004.

[5] Alexander A. Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. In *Proceedings of the 5th International Conference on Learning Representations*, pages 1–19, 2017.

[6] Erdal Arikan. An inequality on guessing and its applications to sequential decoding. *IEEE Transactions on Information Theory*, 42(1):99–105, 1996.

[7] Suguru Arimoto. Information measures and capacity of order α for discrete memoryless channels. In *Topics in Information Theory, Proc. Coll. Math. Soc. János Bolyai*, page 41–52, 1975.

[8] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *Proceedings of the 5th International Conference on Learning Representations*, pages 1–17, 2017.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 214–223, 2017.

[10] Shane Barratt and Rishi Sharma. A note on the inception score. In *Proceedings of the International Conference on Machine Learning 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models*, pages 1–9, 2018.

[11] Moshe Ben-Bassat and Joseph Raviv. Rényi's entropy and the probability of error. *IEEE Transactions on Information Theory*, 24(3):324–331, 2006.

[12] Himesh Bhatia, William Paul, Fady Alajaji, Bahman Gharesifard, and Philippe Burlina. Rényi Generative Adversarial Networks. *ArXiv:2006.02479*, 2020.

[13] Lorne L. Campbell. A coding theorem and Rényi's entropy. *Information and Control*, 9:423–429, 1965.

[14] Liqun Chen, Shuyang Dai, Yunchen Pu, Erjin Zhou, Chunyuan Li, Qinliang Su, Changyou Chen, and Lawrence Carin. Symmetric variational autoencoder and connections to adversarial learning. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 661–669, 2018.

[15] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[16] Thomas A. Courtade and Sergio Verdú. Cumulant generating function of codeword lengths in optimal lossless compression. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 2494–2498, 2014.

[17] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 2nd edition, 2006.

[18] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[19] Imre Csiszár. Information-type measures of difference of probability distributions and indirect observations. *Studia Sci. Math. Hungarica*, 2:299–318, 1967.

[20] Imre Csiszár. Generalized cutoff rates and Rényi's information measures. *IEEE Transactions on Information Theory*, 41(1):26–34, 1995.

[21] Imre Csiszár and János Körner. *Information theory: Coding theorems for discrete memoryless systems.* Cambridge University Press, 2nd edition, 2011.

[22] George Cybenko. Approximation by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 248–255, 2009.

[24] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[25] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv:1603.07285*, 2016.

[26] Amedeo R. Esposito, Michael Gastpar, and Ibrahim Issa. Robust generalization via α-mutual information. In *Proceedings of the International Zurich Seminar on Information and Communication*, pages 96–100, 2020.

[27] Farzan Farnia and David Tse. A convex duality framework for GANs. In *Advances in Neural Information Processing Systems 31*, pages 5248–5258, 2018.

[28] Manuel Gil, Fady Alajaji, and Támas Linder. Rényi Divergence Measures for Commonly Used Univariate Continuous Distributions. *Information Sciences*, 249:124–131, 2013.

[29] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Prentice Hall, 2nd edition, 2002.

[30] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. *ArXiv:1701.00160*, 2016.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680, 2014.

[33] Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-GAN: Combining maximum likelihood and adversarial learning in generative models. In *Proceedings of the 32nd Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, pages 3069–3076, 2018.

[34] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5769–5779, 2017.

[35] Abdessamad Ben Hamza and Hamid Krim. Jensen-Rényi divergence measure: Theoretical and computational perspectives. In *Proceedings of the IEEE International Symposium on Information Theory*, page 257, 2003.

[36] Yun He, Abdessamad Ben Hamza, and Hamid Krim. A generalized divergence measure for robust image registration. *IEEE Transactions on Signal Processing*, pages 1211 – 1220, 2003.

[37] 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. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[38] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[39] Chong Huang, Peter Kairouz, Xiao Chen, Lalitha Sankar, and Ram Rajagopal. Generative Adversarial Privacy. *ArXiv:1807.05306*, 2018.

[40] Caleb Ju and Edgar Solomonik. Derivation and analysis of fast bilinear algorithms for convolution. *SIAM review (to appear), ArXiv:1910.13367*, 2020.

[41] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, pages 1–15, 2015.

[43] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*, pages 1–14, 2014.

[44] Durk P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[45] Pawel A. Kluza. On Jensen-Rényi and Jeffreys-Rényi type f-divergences induced by convex functions. *Physica A: Statistical Mechanics and its Applications*, 2019.

[46] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[47] Yann LeCun and Corinna Cortes. MNIST handwritten digit database, 1998. Available at http://yann.lecun.com/exdb/mnist/.

[48] Moshe Leshno, Vladimir Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

[49] Chunyuan Li, Ke Bai, Jianqiao Li, Guoyin Wang, Changyou Chen, and Lawrence Carin. Adversarial learning of a sampler based on an unnormalized distribution. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019*, volume 89 of *Proceedings of Machine Learning Research*, pages 3302–3311, 2019.

[50] Yingzhen Li and Yarin Gal. Dropout inference in Bayesian neural networks with alpha-divergences. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2052–2061, 2017.

[51] Yingzhen Li and Richard E. Turner. Rényi divergence variational inference. In *Advances in Neural Information Processing Systems*, volume 29, pages 1073–1081, 2016.

[52] Jianhua Lin. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory*, 31:145–151, 1991.

[53] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision*, pages 1–11, 2015.

[54] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs created equal? A large-scale study. In *Advances in Neural Information Processing Systems*, volume 31, pages 700–709. Curran Associates, Inc., 2018.

[55] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–16, 2017.

[56] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. On the effectiveness of least squares generative adversarial networks. *ArXiv:1712.06391*, 2017.

[57] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3481–3490, 2018.

[58] Ernest Mwebaze, Petra Schneider, Frank-Michael Schleif, Sven Haase, Thomas Villmann, and Michael Biehl. Divergence based learning vector quantization. In *Proceedings of the 18th European Symposium on Artificial Neural Networks (ESANN 2010)*, pages 247–252, 2010.

[59] Frank Nielsen. On a generalization of the Jensen-Shannon divergence. *ArXiv:1912.00610*, 2019.

[60] Frank Nielsen and Richard Nock. Entropies and cross-entropies of exponential families. In *Proceedings of the 2010 IEEE International Conference on Image Processing*, pages 3621–3624, 2010.

[61] Frank Nielsen and Richard Nock. On the Chi square and higher-order Chi distances for approximating f-divergences. *IEEE Signal Processing Letters*, pages 10–13, 2013.

[62] Michael Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[63] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, volume 29, pages 271–279. Curran Associates, Inc., 2016.

[64] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *ArXiv:1609.03499*, 2016.

[65] Jose C. Principe. *Information Theoretic Learning: Rényi's Entropy and Kernel Perspectives.* Springer Science and Business Media, 2010.

[66] Ziad Rached, Fady Alajaji, and Lorne L. Campbell. Rényi entropy rate for discrete Markov sources. In *Proceedings of the 33rd Conference on Information Sciences and Systems*, pages 613–618, 1999.

[67] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the 9th International Conference on Image and Graphics*, pages 97–108, 2017.

[68] Sashank Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. In *Proceedings of the 6th International Conference on Learning Representations*, pages 1–23, 2018.

[69] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 547–561, 1961.

[70] John A. Rice. *Mathematical Statistics and Data Analysis.* Duxbury Advanced Series. Duxbury Press, USA, 3rd edition, 2007.

[71] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In

Proceedings of the 31st Conference on Neural Information Processing Systems, 2017.

[72] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, page 533 to 536, 1986.

[73] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[74] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[75] Igal Sason. On f-divergences: Integral representations, local behavior, and inequalities. *Entropy*, 20:1–32, 2018.

[76] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 379–423, 623–656, 1948.

[77] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[78] Lucas Theis, Aaron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *Proceedings of the 4th International Conference on Learning Representations*, pages 1–10, 2016.

[79] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *Proceedings of the 2015 IEEE Information Theory Workshop*, pages 1–5, 2015.

[80] Francisco J. Valverde-Albacete and Carmen Peláez-Moreno. The case for shifting the Rényi entropy. *Entropy*, 21:1–21, 2019.

[81] Tim van Erwen and Peter Harremos. Rényi divergence and Kullback-Leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797 – 3820, 2014.

[82] Sergio Verdú. α-mutual information. In *Proceedings of the 2015 IEEE Information Theory and Applications Workshop*, pages 1–6, 2015.

[83] Zhengwei Wang, Qi She, and Tomas E. Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *ArXiv:1906.01529v3*, 2020.

[84] Maciej Wiatrak, Stefano V. Albrecht, and Andrew Nystrom. Stabilizing generative adversarial network training: A survey. *ArXiv:1910.00927v2*, 2020.

[85] Kristoffer Wickstrom, Sigurd Lokse, Michael Kampffmeyer, Shujian Yu, Jose Principe, and Robert Jenssen. Information plane analysis of deep neural networks via matrix-based Rényi's entropy and tensor kernels. *ArXiv:1909.11396*, 2019.

[86] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *ArXiv:1806.07755*, 2018.

[87] Raymond W. Yeung. *Information Theory and Network Coding*. Springer, 2008.

[88] Abdellatif Zaidi, Iñaki Estella-Aguerri, and Shlomo Shamai (Shitz). On the information bottleneck problems: Models, connections, applications and information theoretic views. *Entropy*, 22:1–36, 2020.

[89] Miaoyun Zhao, Yulai Cong, Shuyang Dai, and Lawrence Carin. Bridging maximum likelihood and adversarial learning via alpha-divergence. In *Proceedings of the 34th Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, volume 34, pages 6901–6908, 2020.

Appendix A

Experiments

A.1 Neural network architectures

The StyleGAN architecture were taken directly from [41]. The architectures for the MNIST dataset are detailed below in Tables A.1 and A.2. We used the architecture guidelines provided by [67] and the GANs tutorial in [1]. We shorten some of the common terms used to describe the layers of the networks. A fully connected layer in a neural network is denoted by FC, while we have used upconv. to denote a convolution layer that is specifically padded to increase the dimensions of its input image. The bias in each upconv. layer was not used in an effort to reduce the amount of parameters and the computational training time. The parameters of the neural networks were initialized by sampling a Gaussian random variable with mean 0 and standard deviation 0.01.

Table A.1: The generator's architecture for MNIST dataset.

Generator
Input multivariate Gaussian noise vector of size 784 with mean $\mathbf{0}$ and a covariance matrix that is the identity matrix of size 784×784.
FC to $12,544$ neurons.
Reshape into $7 \times 7 \times 256$ image.
5×5 upconv. 128 LeakyRELU, batchnorm.
5×5 upconv. 64 LeakyRELU, stride 2, batchnorm.
5×5 upconv. 1 channel, tanh activation.

Table A.2: The discriminator's architecture for MNIST dataset.

Discriminator
Input $28 \times 28 \times 1$ grey image.
5×5 conv. 64 LeakyRELU, stride 2, batchnorm, dropout 0.3.
5×5 conv. 128 LeakyRELU, stride 2, batchnorm, dropout 0.3.
FC to one output, $\frac{1}{1+e^{-x}}$ activation.

A.2 Algorithms

We present the algorithms for LkGANs below. For the MNIST dataset, the constants of the algorithms are $n = 250$ epochs or 15 million images and batch size $m = 100$. For the CelebA dataset, the constants of the algorithms are $n = 125$ epochs or 25 million images and the batch size $m = 126$.

Algorithm 2 Overview of LkGAN-v1-k and LkGAN-v2-k algorithms with and without the simplified gradient penalty.

```
Initialize neural networks.
Fix number of epochs $n$.
for $i = 0$ to $n - 1$ do
    Sample batch size of $m$ noise samples $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\}$ from noise prior $p_{\mathbf{Z}}$
    Sample batch size of $m$ examples $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ from the true distribution $p_{\mathbf{X}}$
    Update the discriminator by descending its gradient without the simplified gradient penalty:
        $\nabla_{\tilde{\boldsymbol{\theta}}} \left( \frac{1}{m} \sum_{i=1}^{m} \left[ \frac{1}{2}(D(\mathbf{x}_i) - \beta)^2 + \frac{1}{2}(D(g(\boldsymbol{z}_i)) - \alpha)^2 \right] \right),$
    or with the simplified gradient penalty:
        $\nabla_{\tilde{\boldsymbol{\theta}}} \Bigg( \frac{1}{m} \sum_{i=1}^{m} \left[ \frac{1}{2}(D(\mathbf{x}_i) - \beta)^2 + \frac{1}{2}(D(g(\boldsymbol{z}_i)) - \alpha)^2 \right] + 5 \left( \frac{1}{m} \sum_{i=1}^{m} \left\| \nabla_{\mathbf{x}} \log \left( \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \right) \Big|_{\mathbf{x}=\mathbf{x}_i} \right\|_2^2 \right) \Bigg).$
    Update the generator by descending its gradient:
        $\nabla_{\boldsymbol{\theta}} \left( \frac{1}{m} \sum_{i=1}^{m} |D(g(\boldsymbol{z}_i)) - \gamma|^k \right),$
end for
```

Algorithm 3 Overview of LkGAN-v1-$[1, 3]$ and LkGAN-v2-$[1, 3]$ algorithms

```
Initialize neural networks.
Fix generator's loss function shape, $k_0 = 1$, flag $= True$ and number of epochs $n$.
for $i = 0$ to $n - 1$ do
  if flag then
    $k_i = k_i + 0.1$
    if $k_i = 3$ then
      flag $= False$
    end if
  else
    $k_i = k_i - 0.1$
    if $k_i = 1$ then
      flag $= True$
    end if
  end if
  Sample batch size of $m$ noise samples $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ from noise prior $p_{\mathbf{Z}}$
  Sample batch size of $m$ examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ from the true distribution $p_{\mathbf{X}}$
  Update the discriminator by descending its gradient:
    $$\nabla_{\tilde{\boldsymbol{\theta}}} \left( \frac{1}{m} \sum_{i=1}^{m} \left[ \frac{1}{2}(D(\mathbf{x}_i) - \beta)^2 + \frac{1}{2}(D(g(\boldsymbol{z}_i)) - \alpha)^2 \right] \right).$$
  Update the generator by descending its gradient:
    $$\nabla_{\boldsymbol{\theta}} \left( \frac{1}{m} \sum_{i=1}^{m} |D(g(\boldsymbol{z}_i)) - \gamma|^{k_i} \right),$$
end for
```

We next present the algorithms for RényiGANs. For the MNIST dataset, the constants of the algorithms are $n = 100$ epochs or 6 million images and batch size $m = 100$. For the CelebA dataset, the constants of the algorithms are $n = 125$ epochs or 25 million images and the batch size $m = 126$.

Algorithm 4 Overview of RényiGAN-α, RényiGAN-α-L_1, and RényiGAN-GP-α-L_1 algorithms

```
**Initialize** neural networks.
**Fix** number of epochs $n$.
**for** $i = 0$ **to** $n-1$ **do**
    **Sample** batch size of $m$ noise samples $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\}$ from noise prior $p_{\mathbf{Z}}$
    **Sample** batch size of $m$ examples $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ from the true distribution $p_{\mathbf{X}}$
    **Update** the discriminator by descending its stochastic gradient without the simplified gradient penalty:
```

$$\nabla_{\tilde{\boldsymbol{\theta}}}\left(-\frac{1}{m}\sum_{i=1}^{m}\left[\log D(\mathbf{x}_i) + \log(1 - D(g(\boldsymbol{z}_i)))\right]\right),$$

or with the simplified gradient penalty:

$$\nabla_{\tilde{\boldsymbol{\theta}}}\Bigg(-\frac{1}{m}\sum_{i=1}^{m}\left[\log D(\mathbf{x}_i) + \log(1 - D(g(\boldsymbol{z}_i)))\right] \\ +5\left(\frac{1}{m}\sum_{i=1}^{m}\left\|\nabla_{\mathbf{x}}\log\left(\frac{D(\mathbf{x})}{1-D(\mathbf{x})}\right)\Bigg|_{\mathbf{x}=\mathbf{x}_i}\right\|_2^2\right)\Bigg),$$

and update the generator by descending its stochastic gradient without L_1 normalization:

$$\nabla_{\boldsymbol{\theta}}\frac{1}{\alpha-1}\log\left[\left(\frac{1}{m}\sum_{i=1}^{m}[1-D(g(\boldsymbol{z}_i))]^{\alpha-1}\right)\right],$$

or with L_1 normalization:

$$\nabla_{\boldsymbol{\theta}}\left|\frac{1}{\alpha-1}\log\left[\left(\frac{1}{m}\sum_{i=1}^{m}[1-D(g(\boldsymbol{z}_i))]^{\alpha-1}\right)\right] + \log(2)\right|,$$

```
**end for**
```

Algorithm 5 Overview of RényiGAN-$[\beta_1, \beta_2]$, RényiGAN-$[\beta_1, \beta_2]$-L_1, and RényiGAN-GP-$[\beta_1, \beta_2]$-L_1 algorithms

```
Initialize neural networks.
Fix generator's loss function shape, $\alpha_0 = \mathbf{x}$, flag $= True$ and number of epochs $n$.
for $i = 0$ to $n - 1$ do
  if flag then
    $\alpha_i = \alpha_i + 0.1$
    if $\alpha_i = \beta_2$ then
      flag $= False$
    end if
    if $\alpha_i = 1.0$ then
      $\alpha_i = 1.1$
    end if
  else
    $\alpha_i = \alpha_i - 0.1$
    if $\alpha_i = \beta_1$ then
      flag $= True$
    end if
    if $\alpha_i = 1.0$ then
      $\alpha_i = 0.9$
    end if
  end if
```

```
    **Sample** batch size of $m$ noise samples $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ from noise prior $p_{\mathbf{Z}}$
    **Sample** batch size of $m$ examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ from the true distribution $p_{\mathbf{X}}$
    **Update** the discriminator by descending its stochastic gradient without the simplified gradient penalty:
        $$\nabla_{\tilde{\boldsymbol{\theta}}}\Bigg( -\frac{1}{m}\sum_{i=1}^{m}\big[\log D(\mathbf{x}_i) + \log(1 - D(g(\boldsymbol{z}_i)))\big]\Bigg),$$
    or with the simplified gradient penalty:
        $$\nabla_{\tilde{\boldsymbol{\theta}}}\Bigg( -\frac{1}{m}\sum_{i=1}^{m}\big[\log D(\mathbf{x}_i) + \log(1 - D(g(\boldsymbol{z}_i)))\big] + 5\left(\frac{1}{m}\sum_{i=1}^{m}\left\|\nabla_{\mathbf{x}}\log\left(\frac{D(\mathbf{x})}{1-D(\mathbf{x})}\right)\Big|_{\mathbf{x}=\mathbf{x}_i}\right\|_2^2\right)\Bigg),$$
    without $L_1$ normalization:
        $$\nabla_{\boldsymbol{\theta}}\frac{1}{\alpha-1}\log\left[\left(\frac{1}{m}\sum_{i=1}^{m}[1 - D(g(\boldsymbol{z}_i))]^{\alpha-1}\right)\right],$$
    or with $L_1$ normalization:
        $$\nabla_{\boldsymbol{\theta}}\left|\frac{1}{\alpha-1}\log\left[\left(\frac{1}{m}\sum_{i=1}^{m}[1 - D(g(\boldsymbol{z}_i))]^{\alpha-1}\right)\right] + \log(2)\right|,$$
**end for**
```

A.3 MNIST results

Table A.3: LkGANs-v1 experiments on the MNIST dataset: the best FID over each run seen over ten trials.

Trial number	1	2	3	4	5	6	7	8	9	10
LkGAN-v1-1.0	3.17	3.24	2.96	3.38	3.12	3.12	3.19	3.17	3.38	2.95
LkGAN-v1-1.2	3.22	3.11	2.97	3.1	3.08	3.18	3.18	3.34	3.12	3.16
LkGAN-v1-1.4	3.31	3.37	3.22	3.19	3.13	3.01	3.2	3.58	3.08	3.67
LkGAN-v1-1.6	2.98	3.17	3.27	3.21	3.14	3.33	3.13	3.26	2.89	2.93
LkGAN-v1-1.8	3.15	3.02	3.02	3.41	3.25	3.28	3.02	3.44	3.3	3.42
LkGAN-v1-2.2	3.48	3.36	3.62	3.54	3.48	3.76	3.45	3.80	3.41	3.20
LkGAN-v1-2.4	3.45	3.84	4.02	3.48	3.30	3.64	3.06	3.81	3.57	3.72
LkGAN-v1-2.6	3.70	4.35	3.78	3.72	3.30	3.92	3.60	4.03	3.65	3.53
LkGAN-v1-2.8	4.05	4.16	3.7	4.53	3.55	3.87	4.23	3.69	3.82	4.25
LkGAN-v1-3.0	4.54	4.04	3.92	4.14	4.31	3.97	3.92	4.37	3.95	4.13
LkGAN-v1-$[1, 3]$	3.41	3.16	3.89	3.44	3.06	3.71	3.56	3.34	3.68	3.43
LSGAN-v1	3.58	3.24	3.21	3.34	3.28	3.54	3.35	3.3	3.34	3.21

Table A.4: LkGANs-v2 experiments on the MNIST dataset: the best FID over each run seen over ten trials.

Trial number	1	2	3	4	5	6	7	8	9	10
LkGAN-v2-1.0	2.94	2.82	58.65	3.17	58.65	3.0	58.09	58.65	58.65	3.07
LkGAN-v2-1.2	2.85	58.65	58.65	3.11	3.02	2.98	58.65	58.65	58.65	58.65
LkGAN-v2-1.4	58.2	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-1.6	58.65	58.65	58.65	58.65	58.65	3.05	3.22	58.65	3.11	58.65
LkGAN-v2-1.8	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-2.2	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-2.4	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-2.6	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-2.8	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-3.0	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v2-[1.0, 3.0]	58.65	58.65	58.65	3.57	58.65	3.31	3.01	58.65	3.08	58.65
LSGAN-v2	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65

Table A.5: LkGANs-v3 experiments on the MNIST dataset: the best FID over each run seen over ten trials.

Trial number	1	2	3	4	5	6	7	8	9	10
LkGAN-v3-1.0	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-1.2	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-1.4	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-1.6	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-1.8	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-2.2	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-2.4	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-2.6	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-2.8	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-3.0	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LkGAN-v3-$[1.0, 3.0]$	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65
LSGAN-v3	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65

Table A.6: RényiGANs experiments on the CelebA dataset: the best FID over each run seen over ten trials.

Trial number	1	2	3	4	5	6	7	8	9	10
RényiGAN-0.5	58.69	58.69	58.70	58.69	58.70	58.70	58.70	58.69	58.69	58.69
RényiGAN-3.0	58.69	58.69	58.69	58.69	1.66	58.69	58.69	58.61	58.69	58.69
RényiGAN-$[0, 0.9]$	58.63	58.33	58.63	58.63	58.63	58.63	58.63	58.63	58.63	58.63
RényiGAN-$[0, 3]$	58.90	58.42	1.44	58.90	58.90	58.90	58.90	1.36	58.90	1.29
RényiGAN-$[1.1, 4]$	58.66	58.89	58.91	58.64	58.87	58.87	58.87	58.87	58.87	58.87
DCGAN	59.06	59.06	59.06	59.06	59.06	59.06	59.06	59.06	59.06	59.06
RényiGAN-0.5-L_1	2.20	2.32	2.18	2.36	2.13	2.15	2.31	2.10	2.13	2.17
RényiGAN-3.0-L_1	1.84	1.83	1.81	1.81	1.87	1.69	1.83	1.72	1.81	1.76
RényiGAN-$[0, 0.9]$-L_1	1.99	2.24	2.22	2.07	2.14	2.16	2.25	2.10	2.21	2.19
RényiGAN-$[0, 3]$-L_1	1.88	1.81	1.75	1.74	1.66	1.71	1.85	1.73	1.84	1.70
RényiGAN-$[1.1, 4]$-L_1	1.76	1.77	1.75	1.84	1.86	1.92	1.83	1.80	1.73	1.80
DCGAN-L_1	2.02	1.89	1.92	1.91	1.91	1.90	2.02	1.88	1.85	2.03
RényiGAN-GP-0.5	1.48	1.41	1.45	1.35	1.32	1.37	1.26	1.37	1.38	1.28
RényiGAN-GP-3.0	1.34	1.43	1.32	1.25	1.39	1.39	1.42	1.38	1.36	1.36
RényiGAN-GP-$[0, 0.9]$	1.46	1.28	1.37	1.37	1.37	1.40	1.41	1.34	1.35	1.28
RényiGAN-GP-$[0, 3]$	1.43	1.42	1.52	1.37	1.46	1.34	1.36	1.36	1.45	1.36
RényiGAN-GP-$[1.1, 4]$	1.33	1.39	1.54	1.37	1.35	1.36	1.29	1.36	1.33	1.32
DCGAN-GP	1.38	1.29	1.34	1.39	1.39	1.41	1.37	1.32	1.32	1.34
RényiGAN-GP-0.5-L_1	1.14	1.24	1.16	1.16	1.18	1.18	1.08	1.27	1.14	1.23
RényiGAN-GP-3.0-L_1	1.21	1.14	1.17	1.14	1.14	1.20	1.06	1.30	1.13	1.20
RényiGAN-GP-$[0, 0.9]$-L_1	1.26	1.15	1.28	1.16	1.10	1.17	1.10	1.21	1.21	1.24
RényiGAN-GP-$[0, 3]$-L_1	1.29	1.28	1.32	1.19	1.14	1.14	1.12	1.19	1.35	1.18
RényiGAN-GP-$[1.1, 4]$-L_1	1.29	1.14	1.31	1.19	1.19	1.14	1.20	1.18	1.25	1.14
DCGAN-GP-L_1	1.21	1.26	1.20	1.15	1.15	1.19	1.13	1.19	1.13	1.16