GEOMETRIC AND SPECTRAL LIMITATIONS IN GENERATIVE ADVERSARIAL NETWORKS

by

MAHYAR KHAYATKHOEI

A dissertation submitted to the
School of Graduate Studies
Rutgers, The State University of New Jersey
In partial fulfillment of the requirements
For the degree of
Doctor of Philosophy
Graduate Program in Computer Science
Written under the direction of
Ahmed Elgammal
and approved by

New Brunswick, New Jersey
October, 2021
Generative Adversarial Networks (GANs) [42] have become one of the most successful and popular generative models in the recent years, with a wide variety of applications across the robust intelligence domains, such as image manipulation, text and audio synthesis, style transfer, and semi-supervised learning, to name a few. The main advantage of GANs over their classical counterparts stems from the use of Deep Neural Networks (DNNs), which can utilize the ongoing revolution in the availability of data and computation power to effectively discover complex patterns. Yet, with this exceptional power, comes an exceptional limitation: the black-box behavior associated with DNNs. This lack of understanding and clarity not only places the profound promise of GANs under a shadow of mistrust, but also greatly hinders any effort to increase their efficiency.

As such, studying GANs’ limitations and biases is perhaps as important, if not more important, as advancing their design and performance. The main focus
of this dissertation is to study two fundamental limitations in GANs, namely a geometric and a spectral limitation. We investigate these limitations in depth, both empirically and theoretically, unveil their causes and consequences across different applications, and finally provide solutions to these limitations.

We start by providing an introduction to density estimation and generative modeling in Chapter 1. In this chapter, we review different approaches to density estimation, highlight the advantages and disadvantages of each method, and discuss the issues that motivated the development of modern DNN-based generative models. The main goal of this chapter is to draw both a historic and a pragmatic line from classical density estimation methods to the modern GANs.

Chapters 2 and 3 elaborate and extend on the results presented in [64]. In Chapter 2, we expose and study the limitation of GANs in learning distributions with disconnected support. We first discuss why having a disconnected support is not a singular and pathological phenomenon, rather a common property of many real world data distributions. Then we theoretically and empirically illustrate the difficulties of GANs in learning such distributions, and its ramifications for the practitioner. In Chapter 3, we propose and evaluate an approach for dealing with the geometric limitation discussed in the previous chapter. This model is based on using an ensemble of generative DNNs, each of which will learn to focus on one connected component of the distribution’s support. Moreover, a prior learning approach is proposed to address the problem of how to choose the “best” ensemble for a given distribution. The final GAN model, denoted $DM$-$WGAN$, trains end-to-end, can learn distributions supported on connected and disconnected manifolds, and infers the required number of members in the ensemble automatically without any explicit supervision. We conclude this chapter by reviewing several existing variants of GANs and their relation with the introduced geometric limitation and our proposed solution.

Chapters 4 and 5 elaborate and extend on the results presented in [63]. In
Chapter 4, we uncover another fundamental limitation in GANs: a spatial frequency bias. Specifically, we empirically and theoretically show that GANs’ performance is not indifferent to the frequency of the underlying signal that carries a distribution. This chapter provides an insight into which datasets and domains are more prone to sub-optimal learning when GANs are used, and perhaps more importantly, what part of a signal is more likely to be missed by GANs. The findings are particularly crucial to the applications that use GANs to manipulate high resolution data, such as in medical and satellite imaging, or where GANs are used to augment or extrapolate data, such as in semi-supervised learning and simulation. In Chapter 5, we propose an efficient approach for matching the spatial bias of GANs to the known biases of a distribution. This approach, denoted *Frequency Shifted Generators*, utilizes the observation that the spatial frequency bias is not a fixed bias and can be efficiently translated to construct a generative DNN that is specifically targeted at a desired spatial frequency. We also show that it is possible to construct an ensemble of such shifted generators, each focusing on a specific frequency, to address the spatial frequency bias in a more general sense.

Finally, in Chapter 6, we connect our separate discussions of the two fundamental limitations in the previous chapters, and discuss the broader impact of our findings on the bigger picture of distribution learning and generative modeling. We particularly comment on the open questions and directions of future research into the limitations of GANs, and more generally, of DNNs.
Acknowledgements

There are many people without whom the development and writing of this dissertation would not have been possible, and I wish to extend my sincere gratitude to them all, though some names might elude my memory.

I’d specially like to thank my advisor Ahmed Elgammal, for providing me with the freedom and support to explore my own ideas, and the guidance to navigate the labyrinth of research and publication. I am forever grateful for his always unassuming advice, patience, and encouragement.

Maneesh Singh, my mentor during my internships at Verisk Analytics, who provided me with a great experience in an industrial research environment, numerous valuable discussions, and financial support in two years of my studies.

All my friends and past and present colleagues in the Arts and AI Lab, Shahrzad Ziaee, Mohamed Abdellatif, Diana Kim, Yizhe Zhu, Bingchen Liu, Mohamed Elhoseiny, and Babak Saleh, with whom I have had many valuable discussions, and have shared exciting moments.

Mubbasir Kapadia, who first advised me when I arrived at Rutgers, and involved me in various exciting projects and introduced me to many excellent scholars, particularly Glen Berseth, Brandon Haworth, Muhammad Usman and Petros Faloutsos, from whom I have learned a lot professionally and personally.

Finally, my father who first sparked my interest in science through chemistry, and thought me how to think scientifically, my uncle Mahmoud who introduced me to computers and programming, my mother who raised me and supported me through many hard days and to whom I owe everything I have, and my sister who has always been my role model, my support, and my best friend.
Table of Contents

Abstract ................................................................. ii
Acknowledgements ....................................................... v

1. From Density Estimation to Generative Adversarial Networks  1
   1.1. Introduction to Density Estimation .............................. 1
   1.2. Non-parametric Models .......................................... 5
   1.3. Parametric Models ................................................ 8
   1.4. Deep Neural Networks ............................................ 13
       1.4.1. Neural Density Estimation .................................. 16
       1.4.2. Variational Auto Encoders ................................. 19
       1.4.3. Generative Adversarial Networks ......................... 23

2. Difficulties of Learning Disconnected Manifolds .................. 32
   2.1. Introduction to Disconnected Manifolds ....................... 32
   2.2. The Connectedness of Support in GANs ........................ 33
   2.3. Off-Manifold Samples ............................................ 34
   2.4. Mode Dropping and Mode Collapse ............................. 35
   2.5. Local Convergence .............................................. 36

3. Disconnected Manifold Learning .................................... 38
   3.1. Ensemble of Generators ........................................ 38
   3.2. Prior Learning ................................................ 40
   3.3. Evaluation Metrics ............................................. 47
   3.4. Experiments .................................................. 49
Chapter 1
From Density Estimation to Generative Adversarial Networks

1.1 Introduction to Density Estimation

Density estimation, in the most abstract sense, can be explained as the very desire to “understand” a phenomenon through a limited number of observations. The word “understand” is placed inside quotation marks to emphasize its lack of a concrete and widely-accepted definition, nonetheless, one can loosely define it as finding a common underlying structure to one’s observations, in order to be able to interpolate and extrapolate such observations. Utilizing the rich framework of probability theory, we can restate this definition as finding the underlying probability distribution of observations.

Before stating the above definition formally, we have to deal with the most pertinent question of all: why do we care about finding the underlying structure of observations? To motivate this, let us review a few exemplar situations.

- **Filtering.** When recording a phenomenon, often capturing all the details are hard or impractical (e.g. photographing an exotic bird obscured by the surrounding trees, or a blackhole many light years away); however, if we knew the underlying structure of the particular phenomenon (how a bird or a blackhole might look like in general), just by recording some partial information, we could infer the plausible missing parts, filtering out the implausible reconstructions. Image inpainting and reconstruction [101, 139, 55, 10], denoising [15, 126], and super-resolution [75, 102] are some examples
of such applications.

- **Compression.** Efficient data compression is primarily a matter of discovering representational redundancies in data, that is, discovering the underlying structure of a phenomenon embedded in some potentially redundant space (e.g., portrait images in pixel space). Knowing this structure allows coding algorithms to achieve almost perfect compression [2, 83].

- **Measurement and Prediction.** In order to measure or predict any properties of a phenomenon, the first and foremost requirement is to be able to define and isolate that phenomenon. For example, predicting whether an image is describing a dog relies on the ability to discover the underlying structure of dog images in the pixel space. Similarly, predicting whether a physical system’s behavior is abnormal (e.g., a cancerous cell, or a malfunctioning gearbox) relies heavily on the ability to identify the structure of the typical behavior of that system (e.g., how a normal cell or a normal gearbox looks or acts like). Semi-supervised learning [18, 113, 134] and anomaly detection [116, 98] are some examples of such applications.

- **Simulation.** Evaluating physical systems and models is often prohibitively time consuming and energy inefficient, such as running very costly particle simulations to evaluate cosmological theories, fluid simulations to evaluate the resilience of certain machinery, and human crowd simulation to evaluate the bottlenecks of an architecture for emergency evacuation scenarios. Knowing the structure of each of these phenomena enables their quick and efficient sampling without having to run a costly physical simulation repeatedly [90, 97, 65].

Now let us formally state the definition of “finding the underlying structure of observations” through probability theory. Given a set of observations \( \{x_i\}_{i=1}^n \) embedded in a \( d \)-dimensional vector space, that is \( x_i \in \mathbb{R}^d \), we assume that
there exists an underlying true probability distribution, described by a density function $p_{\text{data}}(x) : \mathbb{R}^d \rightarrow \mathbb{R}^+$ with support $\mathcal{S} \subset \mathbb{R}^d$ such that $\int_{\mathcal{S}} p_{\text{data}}(x) dx = 1$, from which the observations were drawn independently, hence independent and identically distributed (i.i.d.) samples. Then, the probability of a new observation coming from any measurable subset $U \subset \mathcal{S}$ is given by $\int_U p_{\text{data}}(x) dx$, where the integral is taken with respect to the Lebesgue measure on $\mathcal{S}$. Crucially, note that what we have stated so far is true by assumption, that is, we are restricting our discussion to applications where a density function on some support (possibly low-dimensional) exists and the samples are i.i.d. The reason we are restricting ourselves with these particular assumptions is that they enable the use of the powerful tools and theories of probability theory.

The task of density estimation is to recover the unknown true distribution $p_{\text{data}}(x)$ described above. However, in general, given that all we know about $p_{\text{data}}(x)$ is a limited number of samples, a myriad of possible distributions could explain the observations equally well. Consequently, we have no choice but to make further assumptions about the density function, such that the recovered distribution has certain desirable properties. Such assumptions are incorporated by choosing a family of density functions $p_{\text{model}}(x) \in \mathcal{P}$ that have the desirable properties (either explicitly through model design or implicitly through regularization). For example, some level of smoothness in the model distribution is usually desirable as one expects close regions to have similar probabilities; some form of symmetry always exists in physical phenomena, and hence should similarly exist in the model distribution (such as translation symmetry in images and amplitude symmetry in audio); certain dependencies might exist in a process (such as close-by pixels in images having similar values, or a stream of audio having certain periodic characteristics); and many more application dependent assumptions.

Having the freedom to choose a suitable family of model distributions for any given task is therefore crucial to the success of density estimation. However, what
particularly complicates this choice is the need to be able to effectively search a chosen family to recover the best match for the unknown true distribution. In general, the more expressive a family of model distributions, the harder its evaluation and optimization. As such, much of the research in density estimation, and indeed much of the advancement, is concerned with figuring out ingenious ways of designing more expressive models that can be optimized efficiently. Note that in this context, being more expressive means containing less a priori assumptions: the ideal model would contain only those assumptions explicitly enforced by the practitioner for a specific application, and nothing more.

In this chapter, we will review the major approaches for choosing, implicitly or explicitly, the family of model distributions $\mathcal{P}$, and discuss the advantages and disadvantages of each approach. Our goal is to illustrate, first, why classical density estimation methods were limited to a small family of model distributions; second, how the efforts for enabling the use of more expressive model distributions for density estimation lead to the renaissance of deep learning; third, the different ways in which Deep Neural Networks (DNNs), trained with back-propagation, were adopted for density estimation; fourth, why Generative Adversarial Networks (GANs) are at present the most expressive among the modern DNN-based models; and finally, reaching at the motivating question of this dissertation, are GANs fully expressive, or do they still contain certain limitations and restrictions? Note that the different categories of approaches that will be presented in the following sections are not mutually exclusive, that is, some approaches may fit into more than one of these categorizes; nonetheless, this particular partitioning is a convenient abstraction that facilitates the discussion of primarily limitations that became the motivation for GANs.
1.2 Non-parametric Models

The core idea in non-parametric density estimation is to define the density function \( p_{\text{model}}(x) \) explicitly in terms of the observations (hence non-parametric). Informally, the “closer” a location \( x \in \mathbb{R}^d \) to the observations, the larger its density value ought to be. This type of modeling is specially useful because it can, in theory, and under certain assumptions, represent any distribution \( p_{\text{data}}(x) \); however, we will see the issues that will make its use limited to a very smooth family of distributions, particularly in case of high-dimensional data.

The most popular non-parametric approach to density estimation is kernel density estimation (KDE). Formally, KDE models the closeness of any location \( x \in \mathbb{R}^d \) to the observations \( \{x_i\}_{i=1}^n \) using a kernel function \( K(.) \) that satisfies:

\[
K(x) \geq 0 \quad (1.1)
\]

\[
K(x) \text{ is symmetric.} \quad (1.2)
\]

\[
\int_{\mathbb{R}^d} K(x) dx = 1 \quad (1.3)
\]

where the second requirement ensures that closeness retains its commutative property, and the first and second requirements make sure that the following construction for the model density \( p_{\text{model}}(x) \) is in fact a valid probability density:

\[
p_{\text{model}}(x) = \frac{1}{n} \frac{1}{h^d} \sum_{i}^{n} K \left( \frac{x - x_i}{h} \right) \quad (1.4)
\]

The parameter \( h > 0 \) is known as the smoothing bandwidth, and controls how localized the concept of closeness becomes: \( h \to \infty \) makes all observations similarly close to any location \( x \), resulting in a constant density function everywhere; \( h \to 0 \) makes all observations similarly far from any \( x \not\in \{x_i\}_{i=1}^n \), resulting in a collapsed density function (a sum of \( n \) Dirac delta functions placed at the observations). Consequently, \( h \) controls a bias variance trade-off for the model density \( p_{\text{model}}(x) \):
larger $h$ results in a smoother density function estimate, with a large bias but small model variance, conversely, smaller $h$ results in a more jagged density function, with a large model variance but small bias. In practice, there are rules of thumb for setting this smoothing bandwidth [117], but in general, we need larger $h$ the higher the dimension of data (we will see why shortly), and smaller $h$ the larger the number of observations in order to allow for more flexibility in $p_{\text{model}}(x)$.

While the choice of bandwidth greatly affects KDE, the choice of which kernel to use has little effect, theoretically, on KDE's performance. Nevertheless, in practice, certain kernels might provide marginally better results. The most common ones are Gaussian and Epanechnikov, defined as follows:

$$\text{Gaussian: } K(u) = \frac{1}{(2\pi)^{d/2}} \exp \left( -\frac{1}{2} ||u||^2 \right)$$

$$\text{Epanechnikov: } K(u) = \left( \frac{3}{4} \right)^d \prod_{k=1}^{d} (1 - u_k^2)$$

The Gaussian kernel is simple to implement and provides certain practical advantages, while the Epanechnikov kernel provides the lowest asymptotic mean squared error among all other kernels. Note that here we are considering the multiplicative version of the respective kernels with the same bandwidth in all dimensions for ease of discussion. See [117, 118] for in-depth discussions and comparisons.

KDE has desirable asymptotic properties. It is unbiased when $h \to 0$, that is, it becomes equal to the true density $p_{\text{data}}(x)$ in expectation. Moreover, it is consistent, that is, it converges in probability to $p_{\text{data}}(x)$ asymptotically (as the number of observations approaches infinity), for a suitably small choice of bandwidth $h$.

The flexibility of non-parametric density estimation in general, and KDE in
particular, together with its strong asymptotic properties (unbiased and consistent) make it a popular and strong choice in low-dimensional density estimation. However, it suffers from two important flaws. First, it will fail when \( p_{\text{data}}(x) \) is supported on a low-dimensional manifold, that is, the support of the true distribution \( S \) has Lebesgue measure zero, because then \( \int_{\mathbb{R}^d} p_{\text{data}}(x) \, dx = 0 \) and therefore \( p_{\text{data}}(x) \) is clearly not a member of the family of density functions defined by \( p_{\text{model}}(x) \) in Eq. 1.4. In this scenario, KDE can at best learn a cover of the true distribution’s support.

The second, and arguably more problematic, flaw is encountered when dealing with high-dimensional data. As the dimensionality of space increase, the volume of small regions vanish (consider how the volume of a cube with sides of length less than 1 vanishes as the number of dimensions increases). This in turn means that KDE’s model \( p_{\text{model}}(x) \), if using a small bandwidth, will collapse towards observations. This can be equivalently viewed as the space expanding exponentially, making almost everywhere equally far from the observations, as measured by the kernel. Consequently, KDE must either use a much larger bandwidth \( h \), which will limit its family of model distributions \( \mathcal{P} \) to only very smooth densities, greatly sacrificing its flexibility, or must gain access to exponentially more observations, an impractical amount of data requirement in most applications. This flaw can be more formally explained through the rate of convergence of KDE [117], which is no faster than \( O(n^{-\frac{4}{4+d}}) \), showing that as the number of dimensions \( d \) increases, KDE convergence towards the true density \( p_{\text{data}}(x) \) becomes prohibitively slow. This phenomenon is known as the curse of dimensionality, and was one of the main limitations that inspired the research into parametric density estimation methods, which we will discuss in the next section.
1.3 Parametric Models

The most natural approach for learning an unknown true probability distribution, described by a density function $p_{data}(x)$, is to simply model it with some “expressive” well-behaved parametric function, and then “tune” the corresponding parameters to achieve some measure of closeness to the true distribution. This idea seems quite easy and straight-forward, but as we will see, the words placed inside quotation marks do not always go hand in hand, that is, the need to be able to tune a model can greatly restrict how expressive that model can be, and it is exactly the pursuit of possible ways to remove such restrictions that brought about Generative Adversarial Networks (GANs).

Parametric models, as the name implies, model the true distribution $p_{data}(x)$ by first choosing a family of parameterized continuous functions $F(x; \theta) : \mathbb{R}^d \rightarrow [0, \infty]$, where $\theta$ denotes the parameters, and then constructing the family of model distributions as follows:

$$p_{model}(x; \theta) = \frac{1}{Z(\theta)} F(x; \theta)$$

where the normalization factor $Z(\theta)$, commonly referred to as the partition function, is necessary to make sure $p_{model}(x)$ is a valid probability density function, i.e. $\int_{\mathbb{R}^d} p_{model}(x; \theta) dx = 1$.

Next, in order to evaluate a model distribution with the true distribution, that is, measuring how different they are, we can employ the Kullback-Leibler
divergence [72]:

\[
KL(p_{\text{data}}(x), p_{\text{model}}(x; \theta)) = \int_S p_{\text{data}}(x) \log \left( \frac{p_{\text{data}}(x)}{p_{\text{model}}(x; \theta)} \right) dx
\]

\[
= \mathbb{E}_{p_{\text{data}}(x)} [\log p_{\text{data}}(x)] - \mathbb{E}_{p_{\text{data}}(x)} [\log p_{\text{model}}(x; \theta)]
\]

\[
= H(p_{\text{data}}(x)) - \mathbb{E}_{p_{\text{data}}(x)} [\log p_{\text{model}}(x; \theta)]
\]

(1.8)

Notice that the first term above is the entropy of the true distribution – a constant value – and therefore we can drop it as we are only interested in comparing different model distributions and choosing the one closest to the true distribution. Thus, we can simply consider the second term as a measure of difference, referred to as the log-likelihood loss, which is the negative of the generalized log-likelihood defined below:

\[
l(\theta) = \mathbb{E}_{p_{\text{data}}(x)} [\log p_{\text{model}}(x; \theta)]
\]

\[
= \mathbb{E}_{p_{\text{data}}(x)} [\log F(x; \theta)] - \log Z(\theta)
\]

(1.9)

Finally, in order to find the best model distribution such that \( p_{\text{model}}(x; \theta^*) = p_{\text{data}}(x) \), we can simply minimize the log-likelihood loss – equivalent to minimizing the KL divergence – by approximating the expectation over the true distribution with the sample average using the i.i.d. samples \( \{x_i\}_{i=1}^n \) from \( p_{\text{data}}(x) \):

\[
\theta^* = \arg\min_{\theta} -l(\theta)
\]

\[
= \arg\max_{\theta} \mathbb{E}_{p_{\text{data}}(x)} [\log F(x; \theta)] - \log Z(\theta)
\]

(1.10)

\[
\approx \arg\max_{\theta} \sum_{i=1}^n \log F(x_i; \theta) - \log Z(\theta)
\]

Assuming the chosen family of model distributions is expressive enough, the above optimization problem asymptotically (in the number of samples) becomes equal to minimizing the KL divergence, and since the minimum of KL divergence
is attained only when $p_{model}(x; \theta) = p_{data}(x)$, we can asymptotically recover the unknown true distribution by solving this optimization. The statistical properties of the above optimization can be understood through the central limit theorem and related tail bounds, governing the quality of the approximation made in the last line of Eq. 1.10. Notice that the above optimization is equivalent to the well-studied maximum likelihood estimation and therefore undergoes the same statistical bounds and properties (such as consistency and efficiency). For a detailed statistical analysis of this optimization and relevant proofs, see [76] for large-sample analysis and [119] for none-asymptotic analysis.

Now, the question is how to actually find the solution to Eq. 1.10. To do so, we need to be able to compute the gradient of both $F(x; \theta)$ and $Z(\theta)$ with respect to $\theta$ (in order to be able to either solve analytically, or use efficient gradient-based optimization methods). We have the freedom to choose $F(x; \theta)$ from a family of differentiable functions, for example neural networks, a large and expressive enough class of functions. However, what complicates this freedom is that computing $Z(\theta)$ and its derivative may become intractable, as the partition function is integrating $F(x; \theta)$ over the entire sampling space. Only for a narrow family of functions, $Z(\theta)$ has analytical solution or is constant (such as the family of Gaussian kernels), and for most practical applications the true distribution is not part of such families. In other words, we have to greatly sacrifice the expressiveness of our family of model distributions so that we can actually optimize Eq. 1.10 and find the best matching model.

However, note that while the computation of $Z(\theta)$ might be intractable, computing the gradient of $\log Z(\theta)$, which is ultimately all we need in order to use gradient-based optimization methods, can be derived as follows:
\[
\frac{\partial \log Z(\theta)}{\partial \theta} = \frac{1}{Z} \int_{\mathbb{R}^d} \frac{\partial F(x; \theta)}{\partial \theta} dx \\
= \frac{1}{Z} \int_{\mathbb{R}^d} F(x; \theta) \frac{\partial \log F(x; \theta)}{\partial \theta} dx \\
= \int_{\mathbb{R}^d} p_{\text{model}}(x; \theta) \frac{\partial \log F(x; \theta)}{\partial \theta} dx \\
= \mathbb{E}_{p_{\text{model}}(x; \theta)} \left[ \frac{\partial \log F(x; \theta)}{\partial \theta} \right]
\] (1.11)

Notice, therefore, that we can approximate the gradient of the log-likelihood of Eq. 1.9, and optimize Eq. 1.10, by only having access to unbiased samples from the model distribution \(p_{\text{model}}(x; \theta)\) and the true distribution \(p_{\text{data}}(x)\) to compute the respective sample averages:

\[
\frac{\partial l(\theta)}{\partial \theta} = \mathbb{E}_{p_{\text{data}}(x)} \left[ \frac{\partial \log F(x; \theta)}{\partial \theta} \right] - \mathbb{E}_{p_{\text{model}}(x; \theta)} \left[ \frac{\partial \log F(x; \theta)}{\partial \theta} \right]
\] (1.12)

This derivation paves the way for the use of larger and more expressive families of parametric functions in parametric density estimation. In particular, we can now optimize model distributions in which the partition function is intractable, so long as we are able to generate samples from the model, and computing the derivative of its unnormalized density \(F(x; \theta)\) is tractable. An important instance of such models are graphical models where the density function is written as the product of factors defined by the edges of a graph. Most notably, Restricted Boltzmann Machines (RBMs), Deep Belief Networks (DBNs), and Deep Boltzmann Machines (DBMs) all benefit from variations of the above idea to effectively optimize density functions defined as factorization over different graphical structures (see [41] for a comprehensive review of these models). The advent of these models, particularly DBNs [48], and their demonstrated success in training deeper architectures, contributed greatly to the revitalization of deep learning. However, two fundamental issues still hinder the performance of these models: sampling
from the model distribution, and modeling low-dimensional distributions in high-dimensional embedding space.

The first issue, sampling from the model distribution, is challenging because we do not directly have access to the actual model probabilities, since computing $Z(\theta)$ is intractable. In order to bypass the computation of $Z(\theta)$, we have to sample the model $p_{\text{model}}(x; \theta)$ in a way that only depends on conditional distributions, that is, sampling some parts of the model while fixing some other parts, in which case the partition function $Z(\theta)$ would cancel out. As such, the sampling procedure is done through different variants of Markov Chain Monte Carlo (MCMC) methods, most commonly through Gibbs sampling (for a review and comparison of different sampling methods see [70]). However, such sampling methods typically suffer from a mixing issue – they need to take many iterative samples from the conditional distributions before their samples start to converge to a close enough approximate of the model distribution – and therefore are either very slow, requiring many sampling steps before every gradient update (if we let the Markov chain converge), or very high variance, making the gradient update very noisy (if we run the Markov chain only a few iterations). Consequently, our choice for the family of model distributions becomes restricted to those models which not only can be sampled (i.e. have tractable inference), but efficiently so, therefore excluding large models, having many latent layers and operating on high-dimensional spaces, where the optimization would become prohibitively slow or unstable.

The second issue comes into light when we consider the case of true distributions that are supported on a low-dimensional manifold embedded in $\mathbb{R}^d$, that is, $\mathcal{S}$ has Lebesgue measure zero. For example, consider the distribution of $128 \times 128$ RGB pictures of human faces: while the images are embedded in the 49152 dimensional pixel space, the support of the images lies on a measure zero manifold in this space – consider the probability of finding any face by selecting a random point in $\mathbb{R}^{49152}$. Such distributions do not have density in $\mathbb{R}^d$, and
therefore cannot be modeled well by \( p_{\text{model}}(x; \theta) \), as is apparent when considering the fact that \( \int_{\mathbb{R}^d} p_{\text{data}}(x) \, dx = 0 \) when the true distribution \( p_{\text{data}}(x) \) has measure zero support. Therefore, \( p_{\text{model}}(x; \theta) \) can at best estimate a cover for the support of such true distributions.

The aforementioned two issues motivate a new perspective into the task of density estimation, namely that in order to achieve \( p_{\text{model}}(x; \theta) = p_{\text{data}}(x) \), it appears that we do not really need to have access to the explicit distribution \( p_{\text{model}}(x; \theta) \), all that is required is a sampling method and an evaluation function. As such, we might be able to use much more flexible models if we decouple these two requirements: optimizing one function for generating the samples, and a separate function for recognizing the true distribution. This idea is the essence of the most successful adoptions of neural networks for density estimation to date, which we will review next.

1.4 Deep Neural Networks

Deep Neural Networks (DNNs) are a family of almost everywhere smooth functions constructed by a repeated application of a transformation \( A^l(x) \), transforming a tensor input to a tensor output, followed by a scalar point-wise activation function \( \sigma^l(x) \):

\[
F(x; \theta) = \sigma^L \circ A^L \circ \sigma^{L-1} \circ A^{L-1} \circ \ldots \sigma^2 \circ A^2 \circ \sigma^1 \circ A^1(x)
\]  

where \( l \in \{L, L-1, \ldots, 1\} \) indices different layers, \( A^l \) and \( \sigma^l \) can each be parameterized – potentially with other DNNs – and \( \theta \) refers to the collection of all trainable parameters involved in the model. One way to interpret DNNs are as a hierarchical directed graphical model, where the outputs of each middle (latent) layer, \( h^l = \sigma^l(A^l(h^{l-1})) \), forms a particular feature representation for the inputs, that is, a particular distribution capturing certain properties of the inputs, with
the outermost layer forming a desirable function, for example a density function, based on the features represented by the penultimate layer. The outer layers are commonly considered as modeling more and more complex interactions (patterns) between inputs, but this is not necessarily the case in theory, since a two-layer neural network can be as expressive as a hundred-layer neural network given a large enough number of hidden units; nonetheless, deeper models appear to have a better generalization capability compared to shallow models in practice, and allow for incorporation of certain desirable representational characteristics such as part-whole and hierarchical relations.

Various architectures can be incorporated into DNNs, most notable are Multi Layer Perceptron (MLP) where an affine transformation \( A^l(x) = W^l x + b^l \) is used in each layer, Recurrent Neural Networks (RNNs) where the transformation is shared between layers, that is \( A^l = A^{l-1} \), Convolutional Neural Networks (CNNs) where the transformation is represented as a spatial convolution between a set of kernels and the input, encoding translational symmetries into DNNs at multiple scales, and Residual Neural Networks (ResNets) where the transformation includes a summation with the outputs of one or more previous layers (a skip connection), such as \( A^l(h^{l-1}) = B^l(h^{l-1}) + h^{l-1} \), hence modeling the residual between two layers. There are also many possible choices for the activation function \( \sigma^l(x) \): the sigmoid function \( \frac{1}{1+e^{-x}} \), tanh function \( \frac{e^x - e^{-x}}{e^x + e^{-x}} \), Rectified Linear Units (ReLUs) \( \max(0, x) \) and its leaky variants such as \( \max(-\alpha x, x) \) for \( \alpha \in (0,1) \), to name a few. Many other architectures can be composed as a combination of the aforementioned architectures and activation functions, so long as the DNN remains almost everywhere smooth.

The smoothness requirement is due to the fact that the training of DNNs are carried out using automatic differentiation (backpropagation [111]) to compute the gradients of a task specific loss surface – defining the objective of a particular task where DNNs are modeling an underlying process – with respect to the
parameters of the DNN. These gradients are then used to update the parameters such that the loss is minimized. The update is carried out by gradient-based optimization methods, specifically the stochastic mini-batch gradient methods that can efficiently deal with large datasets, and provide certain regularization benefits due to noisy parameter updates [62]. Most notable among such methods are Momentum [103], AdaDelta [140], Adam [66], and AMSGrad [107]. See [41] for an in-depth discussion of different variants of DNNs and their training.

The primary interest in DNNs arises from the fact that they can, in theory, represent any continuous smooth function so long as they have at least one latent layer \( (L > 1) \) with a suitable non-linear activation function, a result that follows from the Universal Approximation Theorem [51] and its various extensions [71]. This means that when using DNNs to model a task, we are not restricting the task to any a priori assumption, that is, DNNs are as expressive a model as one would ever want between two spaces. However, note that these theorems argue for the existence of a desirable DNN, whether such a DNN is tractable (does not require an infinite number of parameters and is optimizable), and whether we can retrieve it (hence achieve generalization) is not guaranteed.

Although single layer and two layer neural networks (Perceptron and MLP) have been around since the 1960s [109, 110], deeper neural networks were considered too hard and unstable to learn until recently, due to a myriad of optimization issues, such as vanishing or exploding gradients. The introduction of RBMs, and their success in training stacks of latent variables in DBNs [48] in 2006, made them a viable choice for pre-training DNNs as well, reviving interest in the use of DNNs. Later on, in early 2010s, the introduction of efficient auto-differentiation software (\textit{e.g.} Torch [19], Caffe [57], and Tensorflow [1]) and powerful parallel processing hardware (GPUs), and several implicit and explicit regularization and normalization techniques, most notably Dropout [122], use of Rectified Linear
Units (ReLUs) [92], Batch Normalization [56], and adaptive stochastic gradient-based optimizations (e.g. Adam), made the training of DNNs much more efficient and stable, quickly propelling DNNs into the forefront of machine learning. Their unparalleled success in discriminative applications, beating then state-of-the-art regression and classification techniques – symbolic methods, kernel methods, and probabilistic graphical models – and more importantly, their scalability and expressivity, allowing for an effective utilization of the arriving – and still ongoing – explosion in the availability of data, made them a suitable, and perhaps inevitable, candidate for the density estimation task. Among the many ways that they were adopted for density estimation, we will review the most successful ones in the following sections, and discuss their advantages and limitations.

1.4.1 Neural Density Estimation

Neural Density Estimation attempts to model the density function $p_{\text{model}}(x; \theta)$ directly using a neural network, that is, it constructs $p_{\text{model}}(x; \theta)$ using neural networks such that it is a smooth function integrating to 1 over $\mathbb{R}^d$, consequently optimizable using gradient-based methods per Eq. 1.10. There are two main approaches to achieve such a construction: Autoregressive Density Estimation, and Normalizing Flows.

Autoregressive Density Estimation [130] builds on the fact that any joint density $p_{\text{model}}(x; \theta)$, defined over $\mathbb{R}^d$, can be written as a factorization of $d$ conditional distributions, due to the product rule of probability:

$$p_{\text{model}}(x; \theta) = \prod_{t=1}^{d} p_{\text{model}}(x^t|x^{1:t-1})$$

(1.14)

where each conditional density function $p_{\text{model}}(x^t|x^{1:t-1})$ is then modeled as a one-dimensional analytical parametric density $g(x^t; \phi)$, commonly a fixed-variance Gaussian or a mixture of Gaussians, whose parameters are modeled using neural
networks satisfying the conditional property, \( i.e. \phi \equiv F(x^{1:t-1}; \theta) \). One way to implement this is by using Recurrent Neural Networks (RNNs), that is, letting the parameters \( \phi \) be a function of a latent state \( h^t \), which is itself a function of the previous latent state \( h^{t-1} \) and the \( t \)th input variable \( x^t \) – repeated applications of the recursive equation \( h^t = F(x^t, h^{t-1}; \theta) \) – a typical choice for which are Long-Short-Term Memory recurrent neural networks (LSTMs) [125].

However, this recurrent formulation has two drawbacks: first, it takes \( d \) sequential steps to compute \( p_{\text{model}}(x; \theta) \), making it very slow on high-dimensional data; second, it is sensitive to the order in which the input is presented to it, since we are assuming a fixed order of input variables in Eq. 1.14, and it is not tractable to learn separate models for each of the \( d! \) many orders and pick the best one for a given task. One way to address the former drawback is to use a fully-connected neural network, with \( d \) inputs and \( d \) outputs, and drop out connections to ensure that output \( t \) will only be connected to inputs 1 to \( t-1 \); then, output \( t \) can be interpreted as computing the parameters of the \( t \)th conditional density \( g(x^t; \phi) \). The resulting model will satisfy the conditional property by construction, and at the same time will be able to calculate \( p_{\text{model}}(x; \theta) \) efficiently. One example of such a model is the Masked Autoencoder for Distribution Estimation (MADE) [35] (masked convolutions [133, 114] and causal convolutions [132] are other ways of dropping connections). The latter problem is addressed by training an ensemble of models under different random orders for the input, and combining their densities (e.g. using a simple average) [131]. However, these models still are limited in their expressivity: they cannot model distributions supported on low-dimensional manifolds (since \( g(x^t; \theta) \) is required to be a density function on \( \mathbb{R} \)), and they can only model conditional densities for certain orderings of inputs.

A Normalizing Flow models the family of model distributions \( p_{\text{model}}(x; \theta) \) as an invertible differentiable transformation \( F(z; \theta) : \mathbb{R}^d \to \mathbb{R}^d \) applied to a variable
$z \in \mathbb{R}^d$ with a well-defined probability density $p(z)$, for example a standard Gaussian. Note that the dimensions of the support of $z$ and the support of $x$ must agree in order for $F$ to be invertible (hence the assumption of $\mathbb{R}^d$). Given the invertibility and differentiability assumptions, this model then has a well-defined probability distribution described by the following density function:

$$p_{\text{model}}(x; \theta) = p(F^{-1}(x; \theta)) \left| \det \left( \frac{\partial F^{-1}}{\partial x} \right) \right|$$  \hspace{1cm} (1.15)

In order for Eq. 1.15 to be tractable, the transformation $F$ must be constructed such that it can be inverted efficiently, and the determinant of its Jacobian can be computed efficiently. Importantly, note that if two transformations have the above properties, then their composition also has these properties, which reveals a simple receipt for building deeper models that still remain a valid normalizing flow. There have been various approaches in developing normalizing flows using neural networks, that is, modeling the transformation $F$ with a neural network. A common way to achieve this is by modeling $F$ as a sequence of autoregressive layers:

$$x^t = G(z^t; \theta)$$  \hspace{1cm} (1.16)

where $\theta$ is a function of $x^{1:t-1}$ or $z^{1:t-1}$ – modeled by a neural network – that parameterize the transformation $G$. If $G$ is smooth and invertible, we have that the resulting model $F$ is an autoregressive flow by construction such that:

$$\left| \det \left( \frac{\partial F^{-1}}{\partial x} \right) \right| = \prod_{t=1}^{d} \left| \frac{\partial G}{\partial z^t} \right|^{-1} \hspace{1cm} (1.17)$$

since the Jacobian has an upper-triangular form by construction.

Several methods, most notably Inverse Autoregressive Flows (IAF) [68] and
Masked Autoregressive Flows (MAF) [99], consider an affine $G$:

$$x^t = \alpha_i z^t + \beta_i$$  \hspace{1cm} (1.18)

where $\alpha_i$ and $\beta_i$ are scalar functions of $x^{1:t-1}$ or $z^{1:t}$, modeled using neural networks. It is also possible to make the model more expressive by using a non-affine transformation for $G$, such as monotonically increasing neural networks [52]. Convolutional Neural Networks (CNNs) has also been adopted for modeling $G$ [67, 50].

A common issue in normalizing flows is that in order to satisfy the requirements of efficient invertibility and differentiability, the architecture of the model must be restricted in some way or another, thus limiting its expressivity. Moreover, there is a general trade-off between sampling from the model and evaluating the density function, since one of these direction will be subject to a sequential evaluation depending on whether the parameters of $G$ are a function of input $x$ or latent $z$.

The advantage of Neural Density Estimators, in general, is forming a tractable density function, which can be efficiently evaluated and optimized. This make them a good choice for modeling priors in different applications, such as inferring the parameters of a physical system. The drawback of these models are their lower expressivity (in particular inability to faithfully learn distributions supported on low-dimensional manifolds) and slow sample generation (unless their main advantage of efficient density evaluation is sacrificed). In the following sections, we will see that if one is willing to sacrifice having a tractable density evaluation altogether, much more expressive neural network models can be derived.

## 1.4.2 Variational Auto Encoders

Instead of using the restricted family of functions that have a tractable density to directly construct the model distribution $p_{\text{model}}(x; \theta)$, we can introduce a latent variable $z \in \mathbb{R}^h$, and then use the restricted tractable densities to construct a
prior distribution \( p(z; \theta) \) and a likelihood distribution \( p(x|z; \theta) \), and form the following model distribution:

\[
p_{\text{model}}(x; \theta) = \int_{\mathbb{R}^h} p(x|z; \theta)p(z; \theta)dz
\] (1.19)

The motivation for introducing the latent variable is that this allows us to model the prior and likelihood distributions with simple tractable density functions, such as Gaussians parameterized by neural networks, while the marginal remains complex and expressive. To see why, note that Eq. 1.19 is equivalent to an infinite sum of experts, where even though each individual expert might be weak, the combination can become arbitrarily strong. Here we are considering a continuous latent variable \( z \), but in general it can be a discrete variable as well.

Computing the high-dimensional integral of this latent-variable model is in general intractable, so a direct optimization of \( p_{\text{model}}(x; \theta) \) remains infeasible (a restricted tractable construction leads to neural density estimation discussed in Section 1.4.1). One possible way to deal with this problem is to construct the likelihood \( p(x|z; \theta) \) and the prior \( p(z; \theta) \) such that the posterior \( p(z|x; \theta) \) becomes tractable, and then use MCMC sampling methods to optimize the model according to Eq. 1.12. However, this solution restricts the choice of model distributions to those where evaluating and sampling \( p(z|x; \theta) \) and \( p(x|z; \theta) \) can be carried out efficiently, that is RBFs and their variants, preventing the use of more expressive functions, such as unrestricted neural networks, where computing the posterior becomes intractable.

Another way to deal with the intractability of the high-dimensional integral is to introduce an auxiliary distribution \( q(z|x; w) \) and derive a lower bound for
the marginal likelihood as follows:

\[
\log p_{\text{model}}(x; \theta) = \log \int_{\mathbb{R}^d} p(x | z ; \theta) p(z; \theta) dz \\
= \log \int_{\mathbb{R}^d} q(z | x; w) p(x | z; \theta) \frac{p(z; \theta)}{q(z | x; w)} dz \\
= \log \mathbb{E}_{q(z | x; w)} \left[ p(x | z ; \theta) \frac{p(z; \theta)}{q(z | x; w)} \right] \\
\geq \mathbb{E}_{q(z | x; w)} \left[ \log p(x | z ; \theta) - \log \frac{q(z | x; w)}{p(z; \theta)} \right] \\
= \mathbb{E}_{q(z | x; w)} \left[ p(x | z ; \theta) \right] -KL(q(z | x; w)||p(z; \theta)) \\
= \text{ELBO}(\theta, w)
\]  

where the inequality is due to Jensen’s inequality, and the lower bound is often referred to as the evidence lower bound (ELBO). Note that \(q(z)\) could be any arbitrarily distribution, and the reason a conditional form on \(x\) is often chosen is that ELBO attains its maximum when \(q(z)\) matches the correct posterior \(p(z | x; \theta)\), which is evident by observing that \(\text{ELBO}(\theta, w) = \log p_{\text{model}}(x; \theta)\) when:

\[
q(z | x; w) = p(x | z; \theta) = \frac{p(x | z; \theta)p(z; \theta)}{p_{\text{model}}(x; \theta)} \quad (1.21)
\]

Computing the gradient of ELBO is in general infeasible, due to the intractable high-dimensional integrals involved in the expectation calculation. This can be mediated by placing assumptions on the form of \(q(z | x; w)\), most notably an independence assumption on the dimensions of \(z\) which leads to mean-field Variational Bayes approximation. However, such assumptions are often too restrictive. Instead, we can only require a choice of \(q(z | x; \theta)\) that can be sampled efficiently (so that a Monte Carlo sample average can be used to approximate the expectation) and results in a tractable KL divergence (or possibly a tight bound on it). A list of possible choices of parametric distributions for \(p(x; \theta)\) and \(q(z | x; w)\) that satisfy these requirements can be found in [69] (the list is not exclusive).
A common choice is to assume a standard Gaussian prior \( p(z) = \mathcal{N}(0,I) \), and construct the two conditionals as Gaussian distributions parameterized by two neural networks \( E(x;w) \) and \( G(z;\theta) \), so that \( p(z|x;w) = \mathcal{N}(z;E(x;w)) \) and \( p(x|z;\theta) = \mathcal{N}(x;G(z;\theta)) \), in which case, efficient sampling can be achieved using the reparameterization trick, and KL divergence becomes analytical:

\[
\begin{align*}
ELBO(\theta, w) &= \\
&= \mathbb{E}_{q(z|x;w)}[p(x|z;\theta)] - KL(q(z|x;w)||p(z;\theta)) \\
&= \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[\mathcal{N}(x;G(E_\mu(x) + \epsilon E_\sigma(x)))] - KL(\mathcal{N}(z;E(x))||\mathcal{N}(0,I))
\end{align*}
\]

Therefore, we can optimize \( \mathbb{E}_{p_{data}(x)}[p_{model}(x;\theta)] \) by optimizing the sample average of ELBO over the observations \( \{x_i\}_{i=1}^n \), using gradient-based optimization methods. Another way of interpreting this task is by considering \( E(x;w) \) and \( G(z;\theta) \) as an encoder-decoder combination, where the former tries to encode observations such that the latter can best reconstruct them (the first term in ELBO), and simultaneously to make the encoding distribution as similar as possible to the prior distribution, such that after training \( p_{model}(x;\theta) \) can be sampled efficiently, hence the name Variational Autoencoders (VAE) [69, 108].

VAEs provide two important advantages. First, they enable the use of unrestricted neural networks and consequently the design of more expressive and flexible models. Second, they can both generate samples and infer latent variables efficiently. This latter quality can be very useful for effective representation learning. VAEs can be constructed using various structures on their latent variable, encoding a variety of prior knowledge about how a phenomenon behaves and must be represented, and then they will be able to learn how to best encode the observations into these structures. For example, limiting the dimensionality of the latent variable to be much smaller than that of the data (\( i.e. \ h << d \)) forces the model to discover main modes of variations in the data (\( e.g. \ main \).
textures and shapes in images), or using a combination of discrete and continuous latent variables forces the model to decouple continuous modes of variation from discrete ones (e.g. object type from lighting and color in images). There are many other ways to structure the latent variable: hierarchically-structured latent variables that can learn to represent datasets [29], groups of latent variables that can learn to represent objects of a scene [32], latent variables with a Markovian structure that can learn to represent environment states [12, 43], and quantized latent variables for high-resolution image generation [106], to name a few.

The main limitation of VAEs is the restriction caused by the requirement of $q(z|x)$ having a tractable density on the latent space $\mathbb{R}^h$ (required for efficient sampling and computation of KL divergence), and $p(x|z)$ having a tractable density on the data space $\mathbb{R}^d$ (required for having a tractable gradient). The former limits how tight the evidence lower bound can become, while the latter limits the expressivity of the model. In particular, note that VAEs, similar to all the other density estimation methods discussed so far, cannot model distributions supported on low-dimensional manifolds well – since $p(x|z; \theta)$ must be a valid density in $\mathbb{R}^d$ whereas $\int_{\mathbb{R}^d} p_{\text{data}}(x) dx = 0$ for a true distribution $p_{\text{data}}(x)$ that has measure zero support – and they can at best learn a cover of this support.

1.4.3 Generative Adversarial Networks

All the limitations we have observed thus far in various density estimation approaches stem, one way or another, from the common challenge of modeling a valid probability density function $p_{\text{model}}(x; \theta)$ and then devising smart ways to optimize this possibly intractable density towards finding the density of the unknown true distribution $p_{\text{data}}(x)$. Given such a singular point of failure, a natural question follows: what if we completely drop the attempt to model the density function, relying only on the assumption that some $p_{\text{data}}(x)$ exists on some support somewhere in the embedding space from which the observations are drawn,
and then entirely concern ourselves with learning how to generate correct samples from this unknown true distribution? In other words, what if we model \( p_{\text{model}}(x) \) implicitly rather than explicitly parameterizing it, that is, represent a model distribution entirely based on its sampling process? Generative Adversarial Networks (GANs) [42] are an answer to this question. We will see that this new perspective can indeed free us from all the major limitations we observed in the preceding density estimation models, in particular we would be able to learn distributions supported on low-dimensional manifolds in \( \mathbb{R}^d \) – even though these distributions do not have a valid density in \( \mathbb{R}^d \), they still have a well-defined generative process in \( \mathbb{R}^d \) which can be modeled quite well using neural networks – at the expense of the ability to actually evaluate the underlying density function. However, note that this is a favorable trade-off for most applications of density estimation, since most useful statistics of a phenomenon can be retrieved with arbitrarily accuracy given enough samples from it.

GANs model the problem of finding the unknown true distribution of data \( p_{\text{data}}(x) \) as a two player game where one player, called the discriminator, tries to perfectly separate true data from the data generated by a second player, called the generator, while the second player tries to generate data that can perfectly fool the first player. The intuition is that if both of these players are “infinitely smart”, the discriminator will be able to pick up on any arbitrarily fine difference between generated samples and true samples, and consequently the generator will gradually improve its samples to look more and more like true images in order to be indistinguishable for the discriminator, until it learns to generate exactly the samples from \( p_{\text{data}}(x) \), at which point the game will reach an equilibrium and stops, and the generator has implicitly learned a model distribution \( p_{\text{model}}(x) = p_{\text{data}}(x) \) by learning its sampling process.

More formally, consider a generative model defined by a smooth function \( G(z; \theta) : \mathcal{Z} \rightarrow \mathbb{R}^d \) where \( z \) is sampled from some prior distribution \( p(z) \) on \( \mathcal{Z} \)
(commonly a Uniform or standard Gaussian distribution), and a discriminative
model define by a smooth function \( D(x; w) : \mathbb{R}^d \to [0, 1] \), both of which are
constructed using DNNs. Furthermore, denote by \( p_{\text{model}}(x) \) the model distribution
induced by the sampling process \( \{ G(z; \theta) : z \sim p(z) \} \). Then GANs solve the
following min-max optimization:

\[
V(\theta, w) = \mathbb{E}_{p_{\text{data}}(x)} \left[ \log D(x; w) \right] + \mathbb{E}_{p_{\text{model}}(x)} \left[ \log (1 - D(x; w)) \right]
\]

\[ w^* = \arg \max_w V(w, \theta) \]  

\[ \theta^* = \arg \min_{\theta} V(w^*, \theta) \]  

From a probabilistic perspective, the GAN optimization is minimizing the
Jensen-Shannon Divergence (JSD) between the probability distributions of true
data and generated data, where the discriminator is defined as a probability
density and the objective, correspondingly, is defined in terms of the log likelihood
assigned by the discriminator to true and generated samples:

\[
D(x; w^*) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}
\]

\[
V(\theta, w^*) = \mathbb{E}_{p_{\text{data}}(x)} \left[ \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)} \right] + \mathbb{E}_{p_{\text{model}}(x)} \left[ \log \frac{p_{\text{model}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)} \right] 
\]

\[ = KL(p_{\text{data}} || p_{\text{data}} + p_{\text{model}}) + KL(p_{\text{model}} || p_{\text{data}} + p_{\text{model}}) \]

\[ = -2 \log(2) + 2 \text{JSD}(p_{\text{data}} || p_{\text{model}}) \]

Since JSD is only minimized when the two distributions are equal, we can conclude
that \( \theta^* = \arg \min_{\theta} V(\theta, w^*) \) results in \( p_{\text{model}}(x) = p_{\text{data}}(x) \). Importantly, the
above can only be achieved under certain assumptions, most notably that \( G \) and
\( D \) can represent any function, and that \( G^* = G(x; \theta^*) \) and \( D^* = D(x; w^*) \) can be
found. Goodfellow et al. [42] specifically showed that while these can be achieved
when optimizing the objectives in the function space of $G$ and $D$, there is no guarantee in the parameter space of $G$ and $D$ where the objective $V(\theta, w)$ becomes non-convex. In practice, the above optimization is often carried out by alternating between updating the discriminator $D(x; w)$ and the generator $G(z; \theta)$, using backpropagation and stochastic gradient-based optimization methods typical for DNNs (commonly Adam [66]).

One important pitfall of early GANs was the issue of vanishing gradient. One reason for this phenomenon, and pertaining to the above GAN objective, is that for the common formulation of $D(x; w) = \frac{1}{1+e^{-C(x; w)}}$ where $C(x; w) : \mathbb{R}^d \to \mathbb{R}$ is the internal neural network, the gradient of $V(\theta, w^*)$ with respect to $\theta$ vanishes, because the following partial derivative goes to zero as $C(x; w)$ increases on generated samples (i.e. as the discriminator is trained to convergence, decreasing $D(x; w)$ to zero):

$$\frac{\partial \log(1 - D(x; w))}{\partial C(x; w)} = \frac{1}{1 + e^{C(x; w)}}$$

(1.30)

This prevents any learning signal from reaching the generator $G(z; \theta)$. To address this issue, the original GAN paper suggests using a modification to the objective function of $G(z; \theta)$ such that the above gradient no longer vanishes as the discriminator increases $C(x; w)$ on generated samples:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{p_{model}(x)} [\log D(x; w^*)]$$
$$= \arg\min_{\theta} \mathbb{E}_{p(z)} [\log D(G(z; \theta); w^*)]$$

(1.31)

Despite this modification, the vanishing gradient issue could still happen. This is because if initially, or at some point during training, the two distributions $p_{model}(x)$ and $p_{data}(x)$ end up having a measure zero intersection in $S$ the support of $p_{data}(x)$ – a likely situation in the common case that $S$ is measure zero itself – then the discriminator can “flatly” separate generated and true data [3], that is,
\[
\frac{\partial \log C(x;w)}{\partial x}
\]
would go to zero, effectively vanishing any gradient into the generator. This issue is addressed in practice by either not training the discriminator to convergence (only a few discriminator updates per each generator update), and adding noise to both generated and true data such that their support always has some overlap.

Following the seminal work of Goodfellow et al. [42], several models were introduced which tried to modify the aforementioned objective in order to minimize other notions of divergence between the probability distributions, in order to provide certain favorable properties in matching the two distributions [94, 84, 77, 4, 89, 23, 25, 24]. Most notably, Nowozin et al. [94] proposed a framework for minimizing f-divergence which summarizes several GAN models under the same objective (including the original GAN); Li et al. [77] proposed a GAN model based on minimizing maximum mean discrepancy (MMD) between the two distributions, where the discriminator acts as a learnable kernel for MMD, enjoying the wealth of research on MMD; and Arjovsky et al. [4] proposed an alternative objective targeted at minimizing the Wasserstein-1 distance between the two distributions, a distance sensitive to the relative position of the distributions in the embedding space in addition to the differences in probability assignments.

The Wasserstein formulation is interesting in particular since it does not suffer from the vanishing gradient issues of the original GAN objective, and has a better theoretical foundation, that is, unlike JSD that will max-out between two distributions whose support have measure zero intersection, Wasserstein distance can provide informative gradients to direct the distributions towards one another. The Wasserstein GAN (WGAN [4]) uses the dual formulation of the Wasserstein-1 distance:

\[
\text{Wass}(p_{\text{data}}, p_{\text{model}}) = \max_{F(x) : \text{1-Lipschitz}} \mathbb{E}_{p_{\text{data}}(x)} [F(x)] - \mathbb{E}_{p_{\text{model}}(x)} [F(x)] \quad (1.32)
\]
and using a real-valued discriminator $D(x; w) : \mathbb{R}^d \to \mathbb{R}$, referred to as the critic, constructs the following corresponding optimization:

$$V(\theta, w) = \mathbb{E}_{p_{\text{data}}(x)} [D(x; w)] - \mathbb{E}_{p_{\text{model}}(x)} [D(x; w)]$$

$$= \mathbb{E}_{p_{\text{data}}(x)} [D(x; w)] - \mathbb{E}_{p(z)} [D(G(z; \theta); w)]$$

$$w^* = \arg\max_w V(w, \theta)$$

$$\theta^* = \arg\min_\theta V(w^*, \theta)$$ (1.33)

where the 1-Lipschitz property is enforced on $D$ either through explicit weight clipping [4], or by adding the less restrictive and favorable Gradient Penalty to $D$'s objective [44]:

$$V_{\text{regul}} = -\mathbb{E}_{p_l(x)} [(||\nabla D(x)||_2 - 1)^2]$$ (1.36)

where $p_l(x)$ is induced by uniformly sampling from the line connecting a sample from $p_{\text{data}}(x)$ and a sample from $p_{\text{model}}(x)$.

In the few years since their invention, GANs have attracted an enormous body of research, and have advanced at a rapid pace, going from generating jagged hand-written digits to sharp high-resolution faces, objects, and scenes in just a couple of years. Arguably, the main advancement in GANs can be attributed to the use of novel normalization techniques [56, 88, 59], regularization techniques [60, 85] and finally the more complex and larger architectures whose stable training were enabled by these techniques [60, 11, 58, 142, 104]. Interestingly, the choice of objective function, a myriad of which have been proposed as briefly reviewed above, have little effect in general on the performance of GANs, often only marginally outperforming one another [82, 85].
GANs have been used across many applications and domains, ranging from image manipulation to text generation, becoming one of the most prevalent and successful generative models of the recent years. Unconditional GANs, which transform noise inputs (e.g. randomly sampled from Uniform distribution) to samples from a specific target distribution, have been used for generating images [58, 11], text [93, 33], audio [23, 31] and video [128, 112], with unprecedented realism and quality, extending their influence to the domain of creative and artistic image generation [30]. Conditional GANs, which transform a specific source distribution to that of the target distribution, have been used in a wide range of manipulation applications, including interactive image editing [20, 144], generating images from outlines and sketches [115, 16], applying specific textures to images [136], transferring images from one style into another [145] and even modifying content of images by changing semantic labels [135]. GANs have been incorporated into many image processing tasks as well, such as image in-painting [139, 55], denoising [15], and super-resolution [75, 102]. GANs have also been used to improve semi-supervised learning [18, 113], zero-shot learning [101, 146], and even black-box optimization [45, 127].

It is exactly this unparalleled popularity and success, rapid growth, and extensive incorporation into various pipelines, that makes understanding GANs’ limitations and restrictions very important. Developing this understanding is key for designing task-specific GANs efficiently and effectively, without having to spend months, if not years, on a tedious trial and error design process. More importantly, it will help identify the points of failure in GANs, that is, the type of information that might be lost or misrepresented when using GANs to learn a distribution and later represent it as part of larger decision pipelines.

Since the very advent of GANs, a large body of research has investigated its limitations, both in theory and in practice. Arjovsky et al. [44] showed that the Jensen Shannon Divergence is maxed out when the generated and real data
distributions have disjoint support, resulting in vanishing or unstable gradient when training GANs. Several works investigated the loss of variation in the distribution learnt by GANs compared to the true distribution [79, 121, 87, 5]. Most notably, Arora et al. studied the generalization of GANs, that is, how well GANs trained on an empirical estimate of the true distribution (based on the limited number of samples) can in fact learn the true distribution, and showed important limitations in learning the complete support of the true distribution in theory [5] and in practice [6].

Studying the characteristics of GAN’s training, and proposing various regularization techniques to control potential causes of its instability, constitutes another major direction of research into GAN’s limitations [95, 88, 44, 104]. In the same direction, Nagrajan et al. [91] proved that the training of GANs is locally convergent when generated and real data distributions are equal near the equilibrium point, and Mescheder et al. [85] showed the necessity of this condition on a prototypical example. Recently, Zhang et al. [141], proposed adding an attention mechanism to the convolutional neural networks used in GANs which aggregates features in each layer of the network in order to improve networks’ ability to recognize global patterns in each layer, and Karras et al. [60] analyzed the effect of adaptive instance normalization on generating image artifacts.

While many limitations of GANs have been studied, and various objectives and theories have been proposed to improve the guarantees of learning arbitrary probability distributions, little emphasis is placed on the geometric shape or frequency content of these distributions, and how these properties might affect GANs’ performance. Specifically, whether GANs are indifferent to the connectedness of the support of a target distribution or not, and to the prominence of high frequencies in the manifold over which the target distribution is defined, remain understudied. It is at this juncture that our research was introduced, the contributions of which this dissertation will extend and elaborate, advocating a
thread of research on GAN performance from a geometric and spectral point of view, in contrast to the prevalent point of view of probability theory.
Chapter 2
Difficulties of Learning Disconnected Manifolds

2.1 Introduction to Disconnected Manifolds

Consider two natural images, picture of a bird and picture of a cat for example, can we continuously transform the bird into the cat without ever generating a picture that is not neither bird nor cat? In other words, is there a continuous transformation between the two that never leaves the manifold of "real looking" images? It is often the case that real world data falls on a union of several disjoint manifolds and such a transformation does not exist, i.e. the true data distribution is supported on a disconnected manifold, and an effective generative model needs to be able to learn such manifolds.

For the remainder of this chapter, we assume the true data distribution is supported on a manifold $S$ which is a union of disjoint globally connected manifolds each denoted by $M_i$; we refer to each $M_i$ as a submanifold (note that we are overloading the topological definition of submanifolds in favor of brevity):

$$S_r = \bigcup_{i=1}^{n_r} M_i \quad \forall i \neq j : M_i \cap M_j = \emptyset$$

Furthermore, we assume that these submanifolds are compact, to rule out the pathological case of submanifolds being arbitrarily close to one another. The question is then whether GANs can learn such true data distributions well.

One major assumption for the convergence of GANs is that the generator and discriminator both have unlimited capacity [42, 4, 121, 49], and modeling
them with neural networks is then justified through the Universal Approximation
Theorem. However, we should note that this theorem is only valid for continuous
functions. Moreover, neural networks are far from universal approximators in
practice. In fact, we often explicitly restrict neural networks through various
regularizers to stabilize training and enhance generalization. Therefore, when
generator and discriminator are modeled by stable regularized neural networks,
they may no longer enjoy a good convergence as promised by the theory.

In this chapter, we elaborate on the challenge of learning distributions with
disconnected support, and show how limitations of neural networks in model-
ing discontinuous functions can cause difficulties in learning such distributions
with GANs. We study why these difficulties arise, and what consequences they
have in practice.

2.2 The Connectedness of Support in GANs

A GAN as proposed by Goodfellow _et al._ [42], and most of its successors (e.g.
[4, 44]) learn a continuous $G : \mathcal{Z} \rightarrow \mathbb{R}^d$, which receives samples from some prior
$p(z)$ as input and generates true data as output. The prior $p(z)$ is often a stan-
dard multivariate normal distribution $\mathcal{N}(0, I)$ or a bounded uniform distribution
$\mathcal{U}(-1, 1)$. This means that $p(z)$ is supported on a globally connected subspace
of $\mathcal{Z}$. Since a continuous function always keeps the connectedness of space in-
tact [61], the probability distribution induced by $G$ is also supported on a glob-
ally connected space. Thus $G$, a continuous function by design, can not correctly
model a union of disjoint manifolds in $\mathbb{R}^d$. We highlight this fact in Figure 2.1
using an illustrative example where the support of true data is $\{+2, -2\}$. We will
look at some consequences of this shortcoming in the following sections.
Figure 2.1: Illustrative example of continuous generator $G(z) : \mathcal{Z} \to \mathbb{R}^d$ with prior $z \sim \mathcal{U}(-1, 1)$, trying to capture true data coming from $p(x) = \frac{1}{2}(\delta(x - 2) + \delta(x + 2))$, a distribution supported on union of two disjoint manifolds. (a) shows an example of what a stable neural network is capable of learning for $G$ (a continuous and smooth function), (b) shows an optimal generator $G^*(z)$. Note that since $z$ is uniformly sampled, $G(z)$ is necessarily generating off manifold samples (in $[-2, 2]$) due to its continuity.

### 2.3 Off-Manifold Samples

In GANs, the generator is in constant competition with the discriminator, trying to generate samples that look as close as possible to the true samples. When the generator is restricted to a globally connected support, it will try to cover all submanifolds of data with a single globally connected manifold, in order to make sure that it is generating those samples that the discriminator has access to as true observations (true data). In other words, the generator will try to move its support to those regions where the discriminator output is high, and since it cannot break its support, it will end up learning a cover of the submanifolds of true data.

Therefore, the generator will inevitably generate off true-manifold samples. These samples will look like a mash-up of the concepts carried by each submanifold of the true data, deteriorating the sample quality, and are in fact regularly observed even when training high resolution large-scale GANs, particularly on multi-class datasets [11, 142]. Such samples essentially become outliers for any
down-stream task using GANs as part of its pipeline.

Crucially, note that to avoid such off true-manifold regions, one should push the generator to learn a higher frequency function, the learning of which is explicitly avoided by stable training procedures and means of regularization. Therefore the GAN model in a stable training, in addition to real looking samples, will also generate low quality off true-manifold samples. See Figure 3.1 for an example of this problem.

### 2.4 Mode Dropping and Mode Collapse

Mode dropping and model collapse are one of the primarily challenges that generative models face. While these terms are used to refer to a wide range of different phenomena, in this dissertation, we explicitly use the term *mode dropping* to refer to the situation where one or several submanifolds of true data are not completely covered by the support of the generated distribution. Note that mode collapse then is a special case of this definition where all but a small part of a single submanifold are dropped.

When the generator can only learn a distribution with globally connected support, it has to learn a cover of the true data submanifolds, in other words, the generator can not reduce the probability density of the off true-manifold space beyond a certain value. However, the generator can try to minimize the volume of the off true-manifold space to minimize the probability of generating samples there. As an example, see how in Figure 3.1b the learned globally connected manifold has minimum off true-manifold volume: it does not, for example, learn a cover that crosses the center (the same manifold is learned in 5 different runs). So, in learning the cover, there is a trade off between covering all true data submanifolds, and minimizing the volume of the off true-manifold space in the cover. This trade off means that the generator may sacrifice certain submanifolds,
entirely or partially, in favor of learning a cover with less off true-manifold volume, hence mode dropping.

The issue of mode dropping has grave consequences for any down-stream task using GANs as models of reality, particularly in situations where this failure is not easily discernible by a manual investigation the generated samples (i.e. in situations where a partial mode drop has happened). For example, an unbiased and well-designed classifier might be trained on a dataset augmented by GANs, and as a result it might end up learning a biased classification model, since in the augmented version of the dataset the contribution of certain modes have been diminished by GANs. Furthermore, certain marginal modes of variation might be completely dropped, making down-stream tasks that use GAN-based simulations to understand and analyze a phenomenon blind to certain possibilities [45, 21, 90].

2.5 Local Convergence

GANs are known for being hard to train, and in practice it is often the case thier gradient-based optimization does not lead to convergence. Despite some practical advancement, the training dynamics of GANs’ optimization and the reasons behind its instability are still not well-understood. One property that is particularly of interest, is the local convergence of GANs, that is, assuming GANs come sufficiently near the equilibrium point in the parameter space where $p_{model}(x) = p_{data}(x)$, will they be able to converge to this point or will they oscillate, or even worse, diverge?

Mescheder et al. [86] and Nagarajan & Kolter [91] have shown that the eigenvalues of the Jacobian of the vector field associated with the parameter update operator can characterize the local convergence and certain stability properties of GAN training: if the eigenvalues all have negative real parts at the equilibrium
point, GAN training converges locally, given a small enough learning rate; however, if any of the eigenvalues are on the imaginary axis, it is generally not locally convergent. Importantly, Nagarajan & Kolter [91] subsequently proved that the training of GANs is locally convergent when generated and true data distributions are equal near the equilibrium point, and Mescheder et al. [85] showed the necessity of this condition on a prototypical example.

Moreover, in the case of distributions whose support is a disconnected manifold, a generator will not be locally convergent if it cannot learn such a support. Thus, GANs, restricted by the connectedness of their support and unable to learn the correct support of such distributions, will not be able to converge to the equilibrium point even if they succeed in reaching a very close neighborhood of it. In practice, this means the generator’s support keeps oscillating near the true data manifold. To address this issue, and the ones before, naturally we have to look for possible ways that can enable GANs to generate disconnected supports, a challenge we will undertake in the next chapter.
Chapter 3
Disconnected Manifold Learning

3.1 Ensemble of Generators

In order to overcome the issues discussed in the previous chapter, we need to enable the GAN’s support in $\mathbb{R}^d$ to become disconnected. There are two ways to achieve this disconnectedness in $\mathbb{R}^d$: making $Z$ disconnected, or making $G : Z \rightarrow \mathbb{R}^d$ discontinuous. The former needs considerations for how to make $Z$ disconnected, for example adding discrete dimensions [17], or using a mixture of Gaussians [46]. The latter solution can be achieved by introducing a collections of independent neural networks as $G$. In this work, we investigate the latter solution since it is more suitable for parallel optimization and can be more robust to bad initialization. In this chapter, we first propose an ensemble model for learning a disconnected support, and then, in Section 3.2, we will see the importance of learning a correct prior over the ensemble and introduce an EM-based approach to achieve this. In Section 3.3 we discuss the evaluation metrics used for judging the performance of our proposed method on natural images, and in the following Section 3.4 present the experiments. Finally, in Section 3.5 we review the related works and discuss their relation to our findings and proposal.

We first introduce a set of generators $G_c : Z \rightarrow \mathbb{R}^d$ instead of a single one, independently constructed on a uniform prior in the shared latent space $Z$. Each generator can therefore potentially learn a separate connected manifold. However, we need to encourage these generators to each focus on a different submanifold of the true data, otherwise they may all learn a cover of the submanifolds and
experience the same issues of a single generator GAN. Intuitively, we want the
samples generated by each generator to be perfectly unique to that generator,
in other words, each sample should be a perfect indicator of which generator it
came from. Naturally, we can achieve this by maximizing the mutual information
$I(c; x)$, where $c$ is generator id and $x$ is generated sample. As suggested by
Chen et al. [17], we can implement this by maximizing a lower bound on mutual
information between generator ids and generated images:

$$I(c; x) = H(c) - H(c|x)$$

$$= H(c) + \mathbb{E}_{c \sim p(c), x \sim p_g(x|c)} \left[ \mathbb{E}_{c' \sim p(c'|x)} \ln p(c'|x) \right]$$

$$= H(c) + \mathbb{E}_{x \sim p_g(x)} \left[ KL(p(c|x) || q(c|x)) + \mathbb{E}_{c \sim p(c), x \sim p_g(x|c), c' \sim p(c'|x)} \ln q(c'|x) \right]$$

$$\geq H(c) + \mathbb{E}_{c \sim p(c), x \sim p_g(x|c), c' \sim p(c'|x)} \left[ \ln q(c'|x) \right]$$

$$= H(c) + \mathbb{E}_{c \sim p(c), x \sim p_g(x|c)} \left[ \ln q(c|x) \right]$$

where $q(c|x)$ is the distribution approximating $p(c|x)$, $p_g(x|c)$ is induced by each
generator $G_c$, $KL$ is the Kullback Leibler divergence, and the last equality is a
consequence of Lemma 5.1 in [17]. Therefore, by modeling $q(c|x)$ with a neural
network $Q(x; \gamma)$, the encoder network, maximizing $I(c; x)$ boils down to mini-
mizing a cross entropy loss:

$$L_c = -\mathbb{E}_{c \sim p(c), x \sim p_g(x|c)} \left[ \ln q(c|x) \right]$$  (3.1)

Utilizing the Wasserstein GAN [4] objectives, discriminator (critic) and generator
maximize the following, where $D(x; w): \mathbb{R}^d \rightarrow \mathbb{R}$ is the critic function:

$$V_d = \mathbb{E}_{x \sim p_{data}(x)} [D(x; w)] - \mathbb{E}_{c \sim p(c), x \sim p_g(x|c)} [D(x; w)]$$  (3.2)

$$V_g = \mathbb{E}_{c \sim p(c), x \sim p_g(x|c)} [D(x; w)] - \lambda L_c$$  (3.3)
Additionally, to stabilize training [44], we add the single sided version of penalty gradient regularizer with a weight $\lambda'$ to the discriminator/critic objectives of DMWGAN and all baselines:

$$V_{\text{regul}} = -\mathbb{E}_{x \sim p_l(x)} \left[ \max(\|\nabla D(x)\|_2 - 1, 0)^2 \right]$$

(3.4)

where $p_l(x)$ is induced by uniformly sampling from the line connecting a sample from $p_{\text{data}}(x)$ and a sample from $p_g(x)$. We call this model Disconnected Manifold Learning WGAN (DMWGAN). Our proposed approach can similarly be applied on top of any other GAN model with ease, by using the corresponding generator and discriminator objectives in place of $V_d$ and $V_g$.

The original convergence theorems of Goodfellow et al. [42] and Arjovsky et al. [4] hold for the proposed disconnected manifold learning versions respectively, because all our modifications concern the internal structure of the generator, and can be absorbed into the unlimited capacity assumption. More concretely, all generators together can be viewed as a unified generator where $p(c)p_g(x|c)$ becomes the generator probability, and $L_c$ can be considered as a constraint on the generator function space incorporated using a Lagrange multiplier. While most multi-generator models consider $p(c)$ as a uniform distribution over generators, this naive choice of prior can cause certain difficulties in learning a disconnected support. We will discuss this point, and also introduce and motivate the metrics we use for evaluations, in the next two sections.

### 3.2 Prior Learning

In practice, we cannot assume that the true number of submanifolds in the true data distribution is known a priori. So let us consider two cases regarding the number of generators $n_g$, compared to the true number of submanifolds in data $n_r$, under a fixed uniform prior $p(c)$. If $n_g < n_r$, then some generators have to cover
several submanifolds of the true data, thus partially experiencing the same issues discussed in Section 2.2. If \( n_g > n_r \), then some generators have to share one true submanifold, and since we are forcing the generators to maintain disjoint supports, this results in partially covered true submanifolds, causing mode dropping. See Figures 3.1c and 3.2c for examples of this issue. Note that an effective solution to the latter problem reduces the former problem into a trade off: the more the generators, the better the cover. We can address the latter problem by learning the prior \( p(c) \) such that it vanishes the contribution of redundant generators. Even when \( n_g = n_r \), what if the distribution of data over submanifolds are not uniform? Since we are forcing each generator to learn a different submanifold, a uniform prior over the generators would result in a suboptimal distribution. This issue further shows the necessity of learning the prior over generators.

We are interested in finding the best prior \( p(c) \) over generators. Notice that \( q(c|x) \) is implicitly learning the probability of \( x \in \mathbb{R}^d \) belonging to each generator \( G_c \), hence \( q(c|x) \) is approximating the true posterior \( p(c|x) \). We can take an EM approach to learning the prior: the expected value of \( q(c|x) \) over the true data distribution gives us an approximation of \( p(c) \) (E step), which we can use to train the DMGAN model (M step). Instead of using empirical average to learn \( p(c) \) directly, we learn it with a model \( r(c; \zeta) \), which is a softmax function over parameters \( \{\zeta_i\}_{i=1}^{n_g} \) corresponding to each generator. This enables us to control the learning of \( p(c) \), the advantage of which we will discuss shortly. We train \( r(c) \) by minimizing the cross entropy as follows:

\[
H(p(c), r(c)) = -\mathbb{E}_{c \sim p(c)} \left[ \log r(c) \right] \\
= -\mathbb{E}_{x \sim p_{data}(x), c \sim p(c|x)} \left[ \log r(c) \right] \\
= \mathbb{E}_{x \sim p_{data}(x)} \left[ H(p(c|x), r(c)) \right]
\]

where \( H(p(c|x), r(c)) \) is the cross entropy between model distribution \( r(c) \) and
true posterior $p(c|x)$ which we approximate by $q(c|x)$. However, learning the prior from the start, when the generators are still mostly random, may prevent most generators from learning by vanishing their probability too early. To avoid this problem, we add an entropy regularizer and decay its weight $\lambda^\prime\prime$ with time to gradually shift the prior $r(c)$ away from uniform distribution. Thus the final loss for training $r(c)$ becomes:

$$
L_{prior} = \mathbb{E}_{x \sim p_{data}(x)} [H(q(c|x), r(c))] - \alpha t \lambda^\prime\prime H(r(c)) \quad (3.5)
$$

where $H(r(c))$ is the entropy of model distribution, $\alpha$ is the decay rate, and $t$ is training timestep. The model is not very sensitive to $\lambda^\prime\prime$ and $\alpha$, any combination that insures a smooth transition away from uniform distribution is valid. We call this augmented model Disconnected Manifold Learning GAN with Prior Learning (DWMGAN-PL) in our experiments. See Figures 3.1 and 3.2 for examples showing the advantage of learning the prior.

Finally, as discussed in Section 3.1, maximizing mutual information (MI) between generated samples and the generator ids helps prevent separate generators from learning the same submanifolds of data and experiencing the same issues of a single generator model. It is important to note that even without the MI term in the objective, the generators are still “able” to learn the disconnected support correctly. However, since the optimization is non-convex, using the MI term to explicitly encourage disjoint supports for separate generators can help avoid undesirable local minima. Figure 3.3 shows the importance of the MI term in practice by removing the term from the generator objective of DWMGAN-PL, we call this variant DWMGAN-PL-MI0.
Figure 3.1: Comparing Wasserstein GAN (WGAN) and its Disconnected Manifold version with and without prior learning (DMWGAN-PL, DMWGAN) on disjoint line segments dataset when $n_g = 10$. Different colors indicate samples from different generators. Notice how WGAN-GP fails to capture the disconnected manifold of true data, learning a globally connected cover instead, and thus generating off manifold samples. DMWGAN also fails due to incorrect number of generators. In contrast, DMWGAN-PL is able to infer the necessary number of disjoint components without any supervision and learn the correct disconnected manifold of true data. Each figure shows 10K samples from the respective model. We train each model 5 times, the results shown are consistent across different runs.
Figure 3.2: Comparing WGAN-GP, DMWGAN and DMWGAN-PL convergence on unbalanced disjoint line segments dataset when $n_g = 10$. The true data is the same line segments as in Figure 3.1, except the top right line segment has higher probability. Different colors indicate samples from different generators. Notice how DMWGAN-PL (d) has vanished the contribution of redundant generators without any supervision. Each figure shows 10K samples from the respective model. We train each model 5 times, the results shown are consistent across different runs.
Figure 3.3: (a, b) shows DMWGAN-PL-MI0 (without MI), at 30K and 500K training iterations respectively. (c, d) shows the same for DMWGAN-PL (with MI). See how MI encourages generators to learn disjoint supports, leading to learning the correct disconnected manifold.
Algorithm 1 Disconnected Manifold Learning WGAN with Prior Learning (DMWGAN-PL).

**Precondition:** \( p(z) \) prior on \( Z \), \( m \) batch size, \( k \) number of discriminator updates, \( n_g \) number of generators, \( \lambda = 1, \lambda' = 10 \) and \( \lambda'' = 1000 \) are weight coefficients, \( \alpha = 0.999 \) is decay rate, and \( t = 0 \)

1: repeat
2:   for \( j \in \{1 \ldots k\} \) do
3:     \( \{x^i\}_{i=1}^m \sim p_{\text{data}}(x) \) \hspace{1cm} \( \triangleright \) A batch from true data
4:     \( \{z^i\}_{i=1}^m \sim p(z) \) \hspace{1cm} \( \triangleright \) A batch from \( Z \) prior
5:     \( \{c^i\}_{i=1}^m \sim r(c; \zeta) \) \hspace{1cm} \( \triangleright \) A batch from generator’s prior
6:     \( \{x^i\}_{i=1}^m \leftarrow G(z^i; \theta_c) \) \hspace{1cm} \( \triangleright \) Generate batch using selected generators
7:     \( g_w \leftarrow \nabla_w \frac{1}{m} \sum_i [D(x^i; w) - D(x^i_g; w) + \lambda \lambda' V_{\text{regul}}] \)
8:     \( w \leftarrow \text{Adam}(w, g_w) \) \hspace{1cm} \( \triangleright \) Maximize \( V_d \) wrt. \( w \)
9:   end for
10: \( \{x^i\}_{i=1}^m \sim p_{\text{data}}(x) \)
11: \( \{z^i\}_{i=1}^m \sim p(z) \)
12: \( \{c^i\}_{i=1}^m \sim r(c; \zeta) \)
13: \( \{x^i_g\}_{i=1}^m \leftarrow G(z^i; \theta_c) \)
14: for \( j \in \{1 \ldots n_g\} \) do
15:     \( g_{\theta_j} \leftarrow \nabla_{\theta_j} \frac{1}{m} \sum_i [D(x^i_g; \gamma) - \lambda \ln Q(x^i_g; \gamma)] \) \hspace{1cm} \( \triangleright \) \( \theta_j \) is short for \( \theta_c \)
16:     \( \theta_j \leftarrow \text{Adam}(g_{\theta_j}, \theta_j) \) \hspace{1cm} \( \triangleright \) Maximize \( V_g \) wrt. \( \theta \)
17: end for
18: \( g_{\gamma} \leftarrow \nabla_{\gamma} \frac{1}{m} \sum_i \ln Q(x^i_g; \gamma) \)
19: \( \gamma \leftarrow \text{Adam}(g_{\gamma}, \gamma) \) \hspace{1cm} \( \triangleright \) Minimize \( L_c \) wrt. \( \gamma \)
20: \( g_{\zeta} \leftarrow \nabla_{\zeta} \frac{1}{m} \sum_i [H(Q(x^i; \gamma), r(c; \zeta))] - \alpha \lambda'' H(r(c; \zeta)) \)
21: \( \zeta \leftarrow \text{Adam}(g_{\zeta}, \zeta) \) \hspace{1cm} \( \triangleright \) Minimize \( L_{\text{prior}} \) wrt. \( \zeta \)
22: \( t \leftarrow t + 1 \)
23: until convergence.
3.3 Evaluation Metrics

We require metrics that can assess inter-mode variation, intra-mode variation and sample quality. The common metric, Inception Score [113], computes the entropy of the discrete distribution of images over classes, as classified by a pre-trained classifier, as a measure of quality. The intuition is that a good generative model should be able to generate samples from all modes of the data (roughly assumed to be all the classes). However, this metric has several drawbacks [7, 81], most notably it is indifferent to intra-class variations and favors generators that achieve close to uniform distribution over classes of data. Instead, we consider a set of more direct metrics.

A good overall score, in place of the Inception Score, is the Frechet Inception Distance (FID) [47], which first extracts features from the two sets of images corresponding to the two distributions under comparison, that is, the true data distribution \( p_{\text{data}}(x) \) and the generated data distribution \( p_{\text{model}}(x) \) – using a latent layer of an Inception model classifier pre-trained on ImageNet as the feature extractor – and then computes the Frechet distance between two multivariate Gaussian distributions fit to each set of features respectively, that is, \( \mathcal{N}(\mu_{\text{data}}, \Sigma_{\text{data}}) \) and \( \mathcal{N}(\mu_{\text{model}}, \Sigma_{\text{model}}) \), as a measure of quality:

\[
d(N(\mu_{\text{data}}, \Sigma_{\text{data}}), \mathcal{N}(\mu_{\text{model}}, \Sigma_{\text{model}}))^2 = ||\mu_{\text{data}} - \mu_{\text{model}}||^2 + \text{Tr} \left( \Sigma_{\text{data}} + \Sigma_{\text{model}} - 2(\Sigma_{\text{data}}\Sigma_{\text{model}})^{\frac{1}{2}} \right) \]

(3.6)

FID essentially compares the first and second moments of the “useful” features extracted from the two distributions under comparison (useful in the sense of discriminative capability). FID has been shown to be consistent with increasing disturbances and human judgment [47].
For inter mode variation, we use the Jensen Shannon Divergence (JSD) between the class distribution of a pre-trained classifier over real data and generator’s data. This can directly tell us how well the distribution over classes are captured. JSD is favorable to KL due to being bounded and symmetric. For intra mode variation, we define mean square geodesic distance (MSD): the average squared geodesic distance between pairs of samples classified into each class. To compute the geodesic distance, Euclidean distance is used in a small neighborhood of each sample to construct the Isomap graph [137] over which a shortest path distance is calculated. This shortest path distance is an approximation to the geodesic distance on the true image manifold [124]. Note that average square distance, for Euclidean distance, is equal to twice the trace of the Covariance matrix, i.e. sum of the eigenvalues of covariance matrix, and therefore can be an indicator of the variance within each class:

\[ \mathbb{E}_{x,y}[||x - y||^2] = 2\mathbb{E}_x [x^T x] - 2\mathbb{E}_x [x]^T \mathbb{E}_x [x] = 2Tr(Cov(x)) \]

In our experiments, we choose the smallest \( k \) for which the constructed \( k \) nearest neighbors graph (Isomap) is connected in order to have a better approximation of the geodesic distance (\( k = 18 \)).

Another concept we would like to evaluate is sample quality. Given a pre-trained classifier with small test error, samples that are classified with high confidence can be reasonably considered good quality samples. We plot the ratio of samples classified with confidence greater than a threshold, versus the confidence threshold, as a measure of sample quality: the more off real-manifold samples, the lower the resulting curve. Note that the results from this plot are exclusively indicative of sample quality and should be considered in conjunction with the aforementioned metrics. For example, a generator that memorizes only one image from the dataset, can perform very good with respect to this metric,
because although the generator is generating only one sample, that sample has very good quality.

What if the generative model memorizes the dataset that it is trained on? Such a model would score perfectly on all our metrics, while providing no generalization at all. First, note that a single generator GAN model can not memorize the dataset because it can not learn a distribution supported on \( N \) disjoint components as discussed in Section 2.2. Second, while our modifications introduces disconnectedness to GANs, the number of generators we use in our proposed modifications are in the order of data submanifolds which is several orders of magnitude less than common dataset sizes. Note that if we were to assign one unique point of the \( \mathcal{Z} \) space to each dataset sample, then a neural network could learn to memorize the dataset by mapping each selected \( z \in \mathcal{Z} \) to its corresponding real sample (we have introduced \( N \) disjoint component in \( \mathcal{Z} \) space in this case), however this is not how GANs are modeled. Therefore, the memorization issue is not of concern for common GANs and our proposed models (note that this argument is addressing the very narrow case of dataset memorization, not over-fitting in general).

### 3.4 Experiments

In this section we present several experiments to investigate the issues and proposed solutions mentioned in Sections 2.2 and 3.1 respectively. The same network architecture is used for the discriminator and generator networks of all models under comparison, except we use \( \frac{1}{4} \) number of filters in each layer of multi-generator models compared to the single generator models, to control the effect of complexity. In all experiments, we train each model for a total of 200 epochs with a five to one update ratio between discriminator and generator. \( Q \), the encoder network, is built on top of discriminator’s last hidden layer, and is trained simultaneously
with generators. Each data batch is constructed by first selecting 32 generators according to the prior \( r(c; \zeta) \), and then sampling each one using \( z \sim U(-1, 1) \). See the Appendix for details of our networks and the hyperparameters.

**Disjoint line segments.** This dataset is constructed by sampling data with uniform distribution over four disjoint line segments to achieve a distribution supported on a union of disjoint low-dimensional manifolds. See Figure 3.1 for the results of experiments on this dataset. In Figure 3.2, an unbalanced version of this dataset is used, where 0.7 probability is placed on the top right line segment, and the other segments have 0.1 probability each. The generator and discriminator are both MLPs with two hidden layers, and 10 generators are used for multi-generator models. We choose WGAN-GP as the state of the art GAN model in these experiments (we observed similar or worse convergence with other flavors of single generator GANs). MGAN achieves similar results to DMWGAN.

**MNIST dataset.** MNIST [74] is particularly suitable since samples with different class labels can be reasonably interpreted as lying on disjoint manifolds (with minor exceptions like certain 4s and 9s). The generator and discriminator are DCGAN like networks [104] with three convolution layers. Figure 3.6 shows the mean squared geodesic distance (MSD) and Table 3.1 reports the corresponding divergences in order to compare their inter mode variation. 20 generators are used for multi-generator models. Results demonstrate the advantage of adding our proposed modification on both GAN and WGAN.

**Face-Bed dataset.** We combine 20K face images from CelebA dataset [80] and 20K bedroom images from LSUN Bedrooms dataset [138] to construct a natural image dataset supported on a disconnected manifold. We center crop and resize images to \( 64 \times 64 \). 5 generators are used for multi-generator models. Figures 3.6b, 3.7 and Table 3.1 show the results of this experiment.
Figure 3.4: DMWGAN-PL prior learning during training on MNIST with 20 generators (a, b) and on Face-Bed with 5 generators (c, d). (a, c) show samples from top generators with prior greater than 0.05 and 0.2 respectively. (b, d) show the probability of selecting each generator $r(c; \zeta)$ during training, each color denotes a different generator. The color identifying each generator in (b) and the border color of each image in (a) are corresponding, similarly for (d) and (c). Notice how prior learning has correctly learned probability of selecting each generators and dropped out redundant generators without any supervision.
Table 3.1: Inter-class variation measured by Jensen Shannon Divergence (JSD) with true class distribution for MNIST and Face-Bedroom dataset, and square root of FID score for Face-Bedroom (smaller is better). We run each model 5 times with random initialization, and report average values with one standard deviation interval.

<table>
<thead>
<tr>
<th>Model</th>
<th>JSD MNIST ×10^{-2}</th>
<th>JSD Face-Bed ×10^{-4}</th>
<th>FID Face-Bed</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGAN-GP</td>
<td>0.13 std 0.05</td>
<td>0.23 std 0.15</td>
<td>8.30 std 0.27</td>
</tr>
<tr>
<td>MIX+GAN</td>
<td>0.17 std 0.08</td>
<td>0.83 std 0.57</td>
<td>8.02 std 0.14</td>
</tr>
<tr>
<td>DMWGAN</td>
<td>0.23 std 0.08</td>
<td>0.46 std 0.25</td>
<td>7.96 std 0.08</td>
</tr>
<tr>
<td>DMWGAN-PL</td>
<td>0.06 std 0.02</td>
<td>0.10 std 0.05</td>
<td>7.67 std 0.16</td>
</tr>
</tbody>
</table>

Figure 3.5: Intra-class variation in MNIST. Bars show the mean square distance (MSD) within each class of the dataset. On average, DMWGAN-PL outperforms WGAN-GP in capturing intra class variation, as measured by MSD, with larger significance on certain classes. We run each model 5 times with random initialization, and report average values with one standard deviation intervals in both figures. 10K samples are used for metric evaluations.
Figure 3.6: (a) Shows the sample quality in MNIST experiment. (b) Shows sample quality in Face-Bed experiment. Notice how DMWGAN-PL outperforms other models due to fewer off real-manifold samples. We run each model 5 times with random initialization, and report average values with one standard deviation intervals in both figures. 10K samples are used for metric evaluations.
Figure 3.7: Samples randomly generated by GAN models trained on Face-Bed dataset. Notice how WGAN-GP generates combined face-bedroom images (red boxes) in addition to faces and bedrooms, due to learning a connected cover of the real data support. DMWGAN does not generate such samples, however it generates completely off manifold samples (red boxes) due to having redundant generators and a fixed prior. DMWGAN-PL is able to correctly learn the disconnected support of real data. The samples and trained models are not cherry picked.
3.5 Related Works

Several recent works have directly targeted the mode collapse problem by introducing a network $F : \mathbb{R}^d \rightarrow \mathcal{Z}$ that is trained to map back the data into the latent space prior $p(z)$. It can therefore provide a learning signal if the generated data has collapsed. ALI [25] and BiGAN [24] consider pairs of data and corresponding latent variable $(x, z)$, and construct their discriminator to distinguish such pairs of true and generated data. VEEGAN [121] uses the same discriminator, but also adds an explicit reconstruction loss $\mathbb{E}_{z \sim p(z)} \left[ \| z - F_{\theta}(G_{\gamma}(z)) \|_2^2 \right]$. The main advantage of these models is to prevent loss of information by the generator (mapping several $z \in \mathcal{Z}$ to a single $x \in \mathbb{R}^d$). However, in case of distributions with disconnected support, these models do not provide much advantage over common GANs and suffer from the same issues we discussed in Section 2.2 due to having a single generator.

Another set of recent works have proposed using multiple generators in GANs in order to improve their convergence. MIX+GAN [5] proposes using a collection of generators based on the well-known advantage of learning a mixed strategy versus a pure strategy in game theory. MGAN [49] similarly uses a collection of $k$ generators in order to model a mixture distribution, and train them together with a $k$-class classifier to encourage them to each capture a different component of the real mixture distribution. MAD-GAN [36], also uses $k$ generators, together with a $k + 1$-class discriminator which is trained to correctly classify samples from each generator and true data (hence a $k + 1$ classifier), in order to increase the diversity of generated images. While these models provide reasons for why multiple generators can model mixture distributions and achieve more diversity, they do not address why single generator GANs fail to do so. In this work, we explain why it is the disconnectedness of the support that single generator GANs are unable to learn, not the fact that real data comes from a mixture distribution.
Moreover, all of these works use a fixed number of generators and do not have any prior learning, which can cause serious problems in learning of distributions with disconnected support as we discussed in Section 3.2 (see Figures 3.1c and 3.2c for examples of this issue).

Finally, several works have targeted the problem of learning the correct manifold of data. MDGAN [14], uses a two step approach to closely capture the manifold of real data. They first approximate the data manifold by learning a transformation from encoded real images into real looking images, and then train a single generator GAN to generate images similar to the transformed encoded images of previous step. However, MDGAN can not model distributions with disconnected supports. InfoGAN [17] introduces auxiliary dimensions to the latent space $\mathcal{Z}$, and maximizes the mutual information between these extra dimensions and generated images in order to learn disentangled representations in the latent space. DeLiGAN [46] uses a fixed mixture of Gaussians as its latent prior, and does not have any mechanisms to encourage diversity. While InfoGAN and DeLiGAN can generate disconnected manifolds, they both assume a fixed number of discreet components equal to the number of underlying classes and have no prior learning over these components, thus suffering from the issues discussed in Section 3.2.
Chapter 4
Spatial Frequency Bias

4.1 Introduction to Spatial Frequencies

The information in an image is carried by a set of spatial frequencies, that is, a set of planar sinusoids with unique frequencies and directions. Intuitively, we associate the high frequencies with the details of an image, and the low frequencies with its general form; however, neither frequencies should be treated as more important by a generative model seeking to learn a distribution. To make this more clear, consider a two-dimensional planar cosine wave defined over a $128 \times 128$ image, and assume that we sample the magnitude of this static wave from a Gaussian distribution. Whether this wave completes 64 periods across the image (i.e. high spatial frequency), or 3 periods (i.e. low spatial frequency), should not affect a generative model’s learning of the underlying Gaussian distribution.

Convolutional GANs are the state-of-the-art generative models for generating natural image distributions, yet their spectral limitations remain mostly unexplored. The theory of GANs [42], and its many variants [4, 89, 94], do not reveal any spectral limitation. Moreover, the recent success of large-scale GANs in learning fine details in high resolution images seems to support this notion that GANs are not sensitive to the spectral composition [60, 58]. In contrast, the progression of GAN research over the recent years reflects a constant effort for generating better details while generating general form and color seems to be quite easy [60, 59, 58, 135, 54]. Can this apparent difficulty of generating details in practice be linked to an explicit spectral bias?
In this chapter, we first present definitions and preliminaries with regard to spatial frequency components (Section 4.2) and then provide a toy experiment that illustrates the change in a GAN’s performance when trained on isolated low spatial frequencies versus high spatial frequencies, motivating a spatial frequency bias (Section 4.3). Next, we show that convolutional GANs trained on natural images do not learn high spatial frequencies as well as low spatial frequencies, suggesting the existence of a spatial frequency bias (Section 4.4). Then, we show how the same distribution when primarily carried by high spatial frequencies becomes harder to learn for convolutional GANs, versus when carried by low spatial frequencies, confirming the bias (Section 4.5). Finally, the bias is theoretically explained as a manifestation of linear dependencies contained in the spectrum of filters of a generative Convolutional Neural Network (CNN) (Section 4.6). In the following chapter, we develop an efficient method for manipulating this spatial frequency bias towards other spatial frequencies (Section 5.1) and also review the related works and their relation to our findings and proposals (Section 5.3).

Importantly, note that we are only considering GANs that use CNNs as their generator in this chapter, and for brevity, we refer to these models simply as GANs. We use three popular convolutional GAN models in our studies: WGAN-GP [44] serves as a simple but fundamental GAN model; and Progressively Growing GAN (PG-GAN) [58] and StyleGAN2 [60] serve as state-of-the-art models with large capacity and complex structure, incorporating state-of-the-art normalization and regularization techniques. Since our goal is to compare the performance of GANs on high versus low spatial frequencies, and not to compare the overall quality of the generated samples with one another or the state-of-the-art, we chose to use PG-GAN and StyleGAN2 with a slightly smaller capacity in our training (corresponding to the capacity used in Section 6.1 of [58] for ablation studies). Refer to the Appendix for the details of each model.
4.2 Frequency Components

According to Inverse Discrete Fourier Transform [40], every periodic discrete 2D signal $I(x, y)$ with $x \in \{0, 1, 2, \ldots, m - 1\}$ and $y \in \{0, 1, 2, \ldots n - 1\}$, can be written as a sum of several complex sinusoids as follows:

$$I(x, y) = \frac{1}{mn} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} C(u, v) e^{j2\pi\left(\frac{ux}{m} + \frac{vy}{n}\right)} = \frac{1}{mn} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} C(u, v) e^{j2\pi(\hat{u}, \hat{v}).(x,y)} \quad (4.1)$$

We denote each complex sinusoid a spatial frequency component which can be expressed by a vector $(u, v)$ over the pixel locations $(x, y)$. In the above equation, $C(u, v)$ is the complex amplitude of each frequency component, $(\hat{u}, \hat{v}) = (\frac{u}{m}, \frac{v}{n})$ defines the direction of propagation on the 2D plane and its magnitude defines the spatial frequency in that direction, and $m, n \in \mathbb{N}$ are the periods of $I$ in $x$ and $y$ direction respectively. Every channel of a digital 2D image can be assumed periodic beyond the image boundaries, and therefore represented by Eq. 4.1, with periods $m$ and $n$ being the length and width of the image respectively. In that case, the vector $(\hat{u}, \hat{v})$ would define the spatial frequency of a sinusoid in terms of cycles per pixel, in $x$ and $y$ direction respectively, with $\hat{u}, \hat{v} \in [0, 1)$. The maximum frequency in each direction is 0.5 corresponding to the Nyquist frequency (the shortest period needs at least two pixels to be represented, hence the maximum frequency is half cycle per pixel). In favor of clarity, and without loss of generality, we will assume $\hat{u}, \hat{v} \in [-0.5, 0.5)$ throughout this work. Additionally, we loosely refer to the spatial frequency components with $|\hat{u}|$ or $|\hat{v}|$ close to 0.5 as high frequencies, and with $\hat{u}$ or $\hat{v}$ close to 0 as low frequencies. Whenever displaying power spectrums $|C(\hat{u}, \hat{v})|^2$, for better visualization, we drop the DC power, apply Hann window, normalize by the maximum power, and apply log, such that the most powerful frequency always has value 0. Also, $\hat{u}$ and $\hat{v}$ are placed on horizontal and vertical axes respectively, such that low frequencies are placed close to the center, while high frequencies close to the corners.
4.3 Isolated Spatial Frequency Components

In this section, we investigate how well WGAN-GP can learn the spectrum of an isolated spatial frequency component, and whether its performance changes on high frequencies compared to low frequencies. Let us consider a single spatial frequency component on a $128 \times 128$ image, that is, a planar cosine wave

$$I(x,y) = A \cos(2\pi(\hat{u}x + \hat{v}y) + \phi)$$

with spatial frequency $\sqrt{\hat{u}^2 + \hat{v}^2}$, and $x,y \in \{0, 1, 2 \ldots 127\}$. We assume a Normal distribution on its amplitude, i.e. $A \sim N(0.5, 0.1)$, and a Bernoulli on its phase, i.e. $\phi \sim \{0, \pi\}$; then we collect 50K true samples for each of the following spatial frequencies $(\hat{u}, \hat{v})$: $(0, 0)$, $(\frac{3}{128}, \frac{3}{128})$, $(\frac{61}{128}, \frac{61}{128})$, and $(\frac{1}{2}, \frac{1}{2})$.

Having four datasets of the same amplitude and phase distributions, but carried by different spatial frequencies, we train WGAN-GP separately on each dataset and observe the power spectrum of the learnt image distributions. Fig. 4.1 shows the normalized power spectrum for true images on the top row, and for GAN generated images on the bottom row. In each column, ideally, we would expect to observe the same spectrum at the top and the bottom. While this is the case for low spatial frequency datasets (Fig. 4.1a and 4.1b), WGAN-GP fails to learn the correct spectrum for high spatial frequency datasets (Fig. 4.1c and 4.1d). Additionally, to have a quantitative measure of this shortcoming, we report Leakage Ratio (LR) in Table 4.1, which intuitively computes what percentage of the power in the true spectrum has been mistakenly transformed to non-existent spatial frequencies in the GAN generated spectrum. More concretely, we first convert the average power spectrums of true and GAN generated images into power distributions, by dividing each with its own total power, and then compute the total variation [38] between the two power distributions.
Figure 4.1: Frequency leakage in WGAN-GP. The top row shows average power spectrum (DC not dropped) for true data, and the bottom row shows the same for the GAN generated data. Notice the amount of difference between the top and bottom spectrums in high spatial frequencies (c and d), compared to low spatial frequencies (a and b). WGAN-GP fails to learn the true data spectrum for an isolated high spatial frequency.

Table 4.1: Leakage Ratio (LR) in WGAN-GP. The average and standard deviation of LR over three random training runs are reported for each spatial frequency component. The higher the spatial frequency, the larger the leakage into nonexistent frequencies.

<table>
<thead>
<tr>
<th>Component $(\hat{u}, \hat{v})$</th>
<th>$(0, 0)$</th>
<th>$(\frac{3}{128}, \frac{3}{128})$</th>
<th>$(\frac{61}{128}, \frac{61}{128})$</th>
<th>$(\frac{64}{128}, \frac{64}{128})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leakage Ratio (%)</td>
<td>0.09 std 0.04</td>
<td>0.41 std 0.13</td>
<td>1.56 std 0.63</td>
<td>97.43 std 0.76</td>
</tr>
</tbody>
</table>

4.4 Spectral Frechet Inception Distance

We want to observe whether GANs trained on natural images learn the information carried by high frequencies as well as low frequencies. One approach is to directly compare the average power spectrums of GAN generated images with true images [28, 26]. However, the average power of a spatial frequency component is not very informative of how well the distribution carried by that component is learnt. A more informative alternative is to compare how well the distribution of image features carried by high frequencies are learnt compared to that carried by
low frequencies. Frechet Inception Distance (FID) [47], as detailed in Section 3.3, provides a reliable measure of mismatch between the distributions of features extracted from two sets of images: the larger the FID, the larger the mismatch. FID is particularly suitable for our purpose because of its sensitivity to a wide range of high and low frequency features [9, 82]. We propose an extension to this metric, denoted FID Levels, in which we plot FID between two sets of images after gradually removing low frequency bands from both sets. Each point on the FID Levels plot shows FID computed after applying a high-pass Gaussian filter, with the cut-off specified on the horizontal axis, to both the GAN generated and the true images. As a baseline for comparison, we also compute FID Levels between two disjoint subsets of the true images, denoted True FID Levels. For completeness, and as a direct measure of spectral difference, we also report total variation [38] between the GAN generated and the true average power spectrums normalized into density functions, denoted Leakage Ratio (LR).

If a generative model is learning low and high frequencies equally well, then gradually removing spatial frequency bands from generated and true images should result in a generally declining FID between the two sets, as the total amount of information is gradually reduced. To illustrate this behavior, we consider a noisy version of True FID Levels, where we manually introduce mismatch at different spatial frequencies by perturbing the frequencies of one of the two disjoint sets of true images. Specifically, we perturb a frequency component by adding normal noise with mean 0 and variance equal to its power. The total noise added to each image is normalized such that a fixed signal to noise power ratio (SNR) is maintained. Figure 4.2 shows that when a fixed amount of noise (in terms of SNR) is introduced at low frequencies, or at all frequencies, the FID Levels declines; in contrast, when the same amount of noise is introduced at high frequencies, the FID Levels increases.
Given the observations in Figure 4.2, we return to the case of GANs. Figure 4.3 shows FID Levels of GANs trained on two 128 × 128 image datasets: CelebA [80] and LSUN-Bedrooms [138]. The GANs exhibit an increase in FID Levels on both datasets, similar to the behavior observed in Figure 4.2 (middle), suggesting that the learnt high frequencies contain more mismatch than the low frequencies. In contrast, the True FID Levels remains approximately constant in both datasets. As such, the GANs appear to have a bias against learning high frequencies. Without such a bias, we would expect a declining FID Levels as previously observed in Figure 4.2 (right). There remains a caveat, however; since much of the information in natural images is concentrated at low frequencies [34], this bias could be attributed to the scarcity of high frequencies during training. We will see that this cannot be a sufficient explanation in the next section.
Figure 4.2: Sensitivity of FID Levels to the presence of mismatch in different spatial frequency bands. The FID Levels between two disjoint subset of CelebA images are plotted after adding spectral noise to one of the sets, with various SNRs, at a low frequency band $[0, \frac{1}{8}]$ (left), at a high frequency band $[\frac{3}{8}, \frac{1}{2}]$ (middle), and at all frequencies (right). The blue curve depicts the True FID Levels (no added noise).
Figure 4.3: FID Levels of GANs trained on CelebA and LSUN-Bedrooms. The farther to the right on the horizontal axis, the more low frequencies are removed prior to FID computation. Notice the transient increase in FID (worsening performance) as low frequencies are removed. All plots will eventually decline to zero if we continue removing frequencies. In all figures, the blue curve depicts the True FID Levels of the corresponding dataset as a baseline. All figures show average FID with one standard deviation error bars (dashed line), over three random training runs.
4.5 Frequency Shifted Datasets

If GANs are not biased against high frequencies, their performance should remain indifferent to shifting the frequency contents of the datasets. In other words, whether the information in a dataset is primarily carried by high frequencies or low frequencies should not affect how well GANs can learn the underlying image distribution. In order to test this hypothesis, we can create high frequency versions of CelebA and LSUN-Bedrooms by applying a frequency shift operator, that is, multiplying every image in each dataset with \( \cos(\pi (x+y)) \) channel-wise, to create Shifted CelebA (ScelebA) and Shifted LSUN-Bedrooms (SBedrooms) respectively. In effect, all we are doing is swapping the low and high frequency contents of these datasets. Note that the frequency shift operator is a homeomorphism and therefore the distributions of SCelebA and SBedrooms have the same topological properties as CelebA's and LSUN-Bedroom's, and therefore the GANs' performance should remain unchanged from a purely probabilistic perspective.

Table 4.2 compares the GANs' performance on SCelebA and SBedrooms versus the original CelebA and LSUN-Bedrooms.\(^1\) Their performance has worsened significantly (larger FID and LR) on the high frequency datasets, showing that the GANs perform considerably better when the same image distribution is carried primarily by low frequencies. This observation rejects our earlier hypothesis, that is, the GANs’ performance is not indifferent to frequency shift. Additionally, this shows that the bias against high frequencies we observed in Section 4.4 cannot be explained by the scarcity of high frequencies in natural images: even though the unbalancedness in the distribution of power has remained unchanged in the high frequency versions of the datasets, the GANs’ performance has worsened significantly. We conclude that this bias is indeed a “spatial frequency bias” against high frequencies, regardless of how abundant or scarce they are.

\(^1\)In SCelebA and SBedrooms, true and generated images are re-shifted before computing FID so that the values are comparable with the FID results on CelebA and LSUN-Bedrooms.
Table 4.2: Performance drop (increase in both FID and LR) when GANs are
trained on the high frequency versions of CelebA and LSUN-Bedrooms, denoted
SCelebA and SBedrooms respectively. Average measures with standard deviation
(sd) are reported over three random training runs.

<table>
<thead>
<tr>
<th>Model</th>
<th>CelebA</th>
<th>SCelebA</th>
<th>Bedrooms</th>
<th>SBedrooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGAN-GP</td>
<td>FID 20.97 sd 0.70</td>
<td>328.72 sd 9.70</td>
<td>55.14 sd 1.29</td>
<td>283.02 sd 7.06</td>
</tr>
<tr>
<td></td>
<td>LR% 2.29 sd 0.31</td>
<td>59.04 sd 5.09</td>
<td>1.99 sd 0.48</td>
<td>42.42 sd 4.32</td>
</tr>
<tr>
<td>PG-GAN</td>
<td>FID 8.66 sd 0.41</td>
<td>23.12 sd 2.08</td>
<td>36.65 sd 0.97</td>
<td>69.03 sd 10.28</td>
</tr>
<tr>
<td></td>
<td>LR% 1.06 sd 0.21</td>
<td>3.93 sd 0.70</td>
<td>1.51 sd 0.29</td>
<td>3.12 sd 0.16</td>
</tr>
<tr>
<td>StyleGAN2</td>
<td>FID 6.08 sd 0.11</td>
<td>343.57 sd 53.59</td>
<td>22.49 sd 2.00</td>
<td>260.84 sd 4.03</td>
</tr>
<tr>
<td></td>
<td>LR% 1.55 sd 0.42</td>
<td>43.03 sd 34.10</td>
<td>1.28 sd 0.19</td>
<td>7.32 sd 1.86</td>
</tr>
</tbody>
</table>

4.6 On the Causes of the Bias

Let us consider the structure of a typical generative CNN. A 2D generative CNN
$G(x, y; W, H^1)$, with parameters $W \in W$, input features $H^1 \in \mathbb{R}^{d_0 \times d_0 \times c_0}$, and
output space $\mathbb{R}^{d \times d}$, can be modeled as a series of affine convolution layers Conv$^l_i: \mathbb{R}^{d_{l-1} \times d_{l-1} \times c_{l-1}} \rightarrow \mathbb{R}^{d_l \times d_l \times c_l}$ as follows $^2$:

$$H_i^{l+1} = \text{Conv}^l_i(H^l) = b_i + \sum_c F_{ic}^l \ast \text{Up}(\sigma(H_c^l)) \quad (4.2)$$

where $l$ indices the layer (depth), $i$ the output channels (width), $c$ the input
channels, $F_{ic}^l \in \mathbb{R}^{k_l \times k_l}$ is a parametric 2D filter, $b_i \in \mathbb{R}$ is the bias, Up(.) denotes
the upsampling operator, and $\sigma(.)$ is a non-linearity. If we restrict $\sigma$ to rectified
linear units (ReLU), then in a neighborhood of almost any parameter $W$, we can
consider the combined effect of Up($\sigma(.)$) as a fixed linear operation:

**Proposition 1.** At any latent input $H^1$ of a finite size ReLU-CNN, almost every-
where on the parameter space, there exists a neighborhood in which ReLUs are

$^2$Transposed Conv layers are sufficiently represented by an appropriate choice of the Up(.) operator.
equivalent to fixed binary masks.

Proof. Let $f(W) : \mathbb{R}^n \rightarrow \mathbb{R}$ denote a scalar output of a Conv layer of a finite ReLU-CNN with an n-dimensional parameter space ($\mathcal{W}$), at a parameter vector $W$, and at a latent input $H^1 \in \mathbb{R}^{d_0 \times d_0}$. We want to show that for all $W \in \mathcal{W}$, except a measure zero set, there exists a neighborhood of $W$ in which either $f(\hat{W}) \geq 0$ for all members $\hat{W}$ or $f(\hat{W}) \leq 0$ for all members. To do so, it suffices to show the following set has measure zero in $\mathcal{W}$:

$$G = \{W \in \mathcal{W} \mid \forall \mathcal{N}(W) \exists U, V \in \mathcal{N}(W) : f(U) < 0 < f(V)\} \quad (4.3)$$

where $\mathcal{N}(W)$ denotes a neighborhood of $W$. Incidentally, since $f$ is a continuous function, this is the set of zero-crossings of $f(W)$ over the parameter space $\mathcal{W}$. Since $f(W)$ has a finite set of neurons, it has a finite set of ReLU activations, and therefore at any $W$ it will be equivalent to one of a finite set of possible polynomials on $\mathcal{W}$ (corresponding to the finitely many binary permutations of ReLUs). A polynomial function on $\mathbb{R}^n$ to $\mathbb{R}$ has a measure zero set of zero-crossings in the parameter space [13]. Therefore, a finite set of such polynomials also has a measure zero set of zero-crossings, which concludes that $G$ has measure zero. Finally, note that the same argument holds for any scalar output $f$ of the CNN, at any spatial location in any layer, and given the finite number of these outputs, the measure of $G$ for all outputs $f$s is also zero, which completes the proof.

Therefore, in this neighborhood, improving the output spectrum is only achievable through adjusting the spectrum of filters $F_{ic}^l$. Intuitively, the filters try to carve out the desired spectrum out of the input spectrum which is distorted by ReLUs (as binary masks), and aliased by upsampling. In the following theorem, we investigate how freely can these filters adjust their spectrum. Specifically, we will show how the filter size $k_l$ and the spatial dimension $d_l$ of a Conv layer affect
the correlation in the spectrum of its filters. Note that more correlation in a
filter’s spectrum means more linear dependency, and thus reduces its effective ca-
pacity, in other words, the filter can not freely adjust specific frequencies without
affecting the adjacent correlated frequencies.

**Theorem 1.** Let $U = \mathcal{F}\{F_{ic}\}(u_0, v_0)$ and $V = \mathcal{F}\{F_{ic}\}(u_1, v_1)$ be any two spatial
frequency components on the spectrum of any 2D filter of the $l$-th Conv layer, with
spatial dimension $d_l \in \mathbb{N}$ and filter size $k_l \in \mathbb{N}$, such that $1 < k_l \leq d_l$. Assuming
i.i.d. weight initialization, the magnitude of the complex correlation coefficient
between $U$ and $V$, at any point during the training, is given by:

$$|\text{corr}(U, V)| = \left| \frac{\text{Sinc}(u_0 - u_1, v_0 - v_1)}{k_l^2} \right|$$

s.t. $\text{Sinc}(u, v) = \frac{\sin\left(\frac{\pi u k_l}{d_l}\right) \sin\left(\frac{\pi v k_l}{d_l}\right)}{\sin\left(\frac{\pi u}{d_l}\right) \sin\left(\frac{\pi v}{d_l}\right)}$ (4.4)

**Proof.** Let us consider a single 2D filter $F^l \in \mathbb{R}^{k_l \times k_l}$ in the $l$-th Conv layer. Since
the spatial dimension of the layer’s output is $d_l$, a filter $G^l \in \mathbb{R}^{d_l \times d_l}$ can sufficiently
model any spectrum in the layer’s output space. So we can write $F^l$ as a restriction
of $G^l$, that is, the multiplication of $G^l$ with the 2D pulse $P \in \mathbb{R}^{d_l \times d_l}$ of area $k_l^2$:

$$F^l = P.G^l$$

$$P(x, y) = \begin{cases} 1 & 0 \leq x, y < k_l \\ 0 & \text{otherwise} \end{cases}$$

(4.6)

Therefore, by convolution theorem, the spectrum of $F^l$ is equal to the spectrum
of $G^l$ convolved by the spectrum of $P$:

$$\mathcal{F}\{F^l\} = \mathcal{F}\{P.G^l\} = \mathcal{F}\{P\} \ast \mathcal{F}\{G^l\} = \text{Sinc} \ast \mathcal{F}\{G^l\}$$

(4.7)

where $\mathcal{F}\{\cdot\}$ denotes the $d_l$-point DFT, $\ast$ is the circular convolution, and Sinc is
the aliased sinc function:

\[
\text{Sinc}(u, v) = \frac{\sin(\frac{\pi u k_l}{d_l}) \sin(\frac{\pi v k_l}{d_l})}{\sin(\frac{\pi u}{d_l}) \sin(\frac{\pi v}{d_l})} e^{-j\pi(u+v)(k_l-1)/d_l} \quad d_l, k_l \in \mathbb{N} \quad \text{and} \quad 1 < k_l \leq d_l
\]

(4.8)

Now, let \( U = \mathcal{F}\{F^l\}(u_0, v_0) \) and \( V = \mathcal{F}\{F^l\}(u_1, v_1) \) be two spatial frequency components on the spectrum of the filter, located at the \((\frac{u_0}{d_l}, \frac{v_0}{d_l})\) and \((\frac{u_1}{d_l}, \frac{v_1}{d_l})\) frequencies respectively. We are interested in the correlation between these two components. Note that we are defining \( G^l \) as the hypothetical unrestricted filter, such that any possible filter that can be learnt during training becomes a restriction of \( G^l \). Therefore, without loss of generality, we can assume \( G^l \) has independent spatial frequency components with variance \( \sigma^2 \). Considering these assumptions, we can find the complex correlation coefficient [100] between \( U \) and \( V \) by first computing the complex covariance:

\[
\text{Cov}[U, V] = \text{Cov}\left[\text{Sinc} \ast \mathcal{F}\{G^l\}(u_0, v_0), \text{Sinc} \ast \mathcal{F}\{G^l\}(u_1, v_1)\right]
\]

(4.9)

\[
\begin{align*}
&= \text{Cov}\left[\sum_{u, v} \text{Sinc}(u, v)\mathcal{F}\{G^l\}(u_0 - u, v_0 - v), \sum_{\hat{u}, \hat{v}} \text{Sinc}(\hat{u}, \hat{v})\mathcal{F}\{G^l\}(u_1 - \hat{u}, v_1 - \hat{v})\right] \\
&= \sum_{u, v} \sum_{\hat{u}, \hat{v}} \text{Sinc}(u, v)\text{Sinc}^\ast(\hat{u}, \hat{v}) \\
&\quad \times \text{Cov}\left[\mathcal{F}\{G^l\}(u_0 - u, v_0 - v), \mathcal{F}\{G^l\}(u_1 - \hat{u}, v_1 - \hat{v})\right] \\
&= \sum_{u, v} \text{Sinc}(u, v)\text{Sinc}(u_0 - u - 1 - u, v_0 - v - 1 - v) \\
&\quad \times \text{Var}\left[\mathcal{F}\{G^l\}(u_0 - u, v_0 - v)\right] \\
&= \sigma^2 \text{Sinc} \ast \text{Sinc}(u_0 - u_1, v_0 - v_1) \\
&= \sigma^2 d_l^2 \text{Sinc}(u_0 - u_1, v_0 - v_1)
\end{align*}
\]

(4.10)
Then computing the variance:

\[
\text{Var}[U] = \text{Var}[\text{Sinc} * \mathcal{F}\{G^l\} (u_0, v_0)]
\]

\[
= \text{Var}\left[ \sum_{u,v} \text{Sinc}(u, v)\mathcal{F}\{G^l\} (u_0 - u, v_0 - v) \right]
\]

\[
= \sum_{u,v} |\text{Sinc}(u, v)|^2 \text{Var}[\mathcal{F}\{G^l\} (u_0 - u, v_0 - v)]
\]

\[
= \sigma^2 \sum_{u,v} |\text{Sinc}(u, v)|^2
\]

\[
= \sigma^2 d_l^2 k_l^2
\]

To finally reach at:

\[
\text{corr}(U, V) = \frac{\text{Cov}[U, V]}{\sqrt{\text{Var}[U] \text{Var}[V]}} = \frac{\text{Sinc}(u_0 - u_1, v_0 - v_1)}{k_l^2}
\]

\[
\text{corr}(U, V) = \frac{\sin^2\left(\frac{\pi k_l}{d_l}\right)}{k_l^2 \sin^2\left(\frac{\pi}{d_l}\right)}
\]

Note that all the expectations are taken over the probability space of all unrestricted 2D filters of a specific output dimension \(d_l\), that is the probability space of \(G^l\) when considered as a random variable, and that \(\text{Sinc}^*\) represents the complex conjugate of \(\text{Sinc}\).

**Corollary 1.1.** If \(U\) and \(V\) are two diagonally adjacent spatial frequency components of \(F^l_{ic}\), then:

\[
|\text{corr}(U, V)| = \frac{\sin^2\left(\frac{\pi k_l}{d_l}\right)}{k_l^2 \sin^2\left(\frac{\pi}{d_l}\right)}
\]

Now, in each Conv layer, note that the maximum spatial frequency that can be generated is limited by the Nyquist frequency, that is, \(\text{Conv}^l\) can only adjust image spatial frequencies in \([0, \frac{d_l}{2}]\) without aliasing\(^3\). This means that high frequencies

\(^3\)Aliasing here refers to the process of generating high frequencies by replicating low frequencies in the expanded spectrum introduced by upsampling. Since this makes duplicates in
are primarily generated by the CNN’s outer layers where \( d_i \) is larger. According to Eq. 4.21, given a fixed filter size \( k_i \), the larger the \( d_i \), the larger the correlation in the filter’s spectrum (see the equation’s graph in Figure 4.4), and consequently the smaller its effective capacity. Therefore, the outer layers responsible for generating high frequencies are more restricted in their spectrum compared to the inner layers with smaller \( d_i \). We hypothesize that this is the main cause of the spatial frequency bias. This in turn implies that the issue of spatial frequency bias is not limited to GANs. Indeed, high frequency discrepancies between CNN generated images and true images have been observed both in L2 reconstructions [21, 78, 129] and Variational Autoencoders (VAEs) [28], however, in these tasks, the spatial frequency bias of generative CNNs is not easily distinguishable from the known spectral biases inherent to the respective objective functions.

**On the effect of increasing depth.** One way to counter the correlation is to replace an individual Conv layer with a stack of Conv layers, resulting in a deeper CNN. This can increase the effective filter size \( k_l \) operating on each spatial dimension, thus reducing the correlation. However, note that while only outer layers can generate high frequencies without aliasing, low frequencies can be generated by all layers without aliasing. As such, low frequencies will always enjoy a larger end-to-end filter size compared to high frequencies, and thus less correlation (see the spectrums of effective filters in Figure 4.4).

**On the effect of increasing width.** Another way to counter the correlation is to simply include more filters in a Conv layer, resulting in a wider CNN. However, this becomes particularly costly at the outer layers with larger spatial dimensions. Moreover, making the outer layers wider will increase the capacity of generating both high and low frequencies equally, and not exclusively that of

---

high frequency bands of the spectrum, it’s ability to control high frequencies is minimal. Many upsampling methods explicitly attenuate such aliased frequencies.
high frequencies, as discussed in the previous paragraph, thus, the spatial frequency bias remains.

**On the effect of increasing resolution.** It is key to note that whether a signal contains high spatial frequencies or not is directly related to its sampling rate. For example, consider the continuous-valued image of a bird formed on a camera’s sensor, whose feathers change color from white to black 64 times over the length of the image. If this image is sampled into a $128 \times 128$ picture, the feathers would form a high frequency component ($\frac{1}{2}$ cycles per pixel). If the same image is instead sampled into a $1024 \times 1024$ picture, the feathers now form a low frequency component ($\frac{1}{16}$ cycles per pixel). Therefore, one solution to the spatial frequency bias is to simply use data at a very high resolution, such that no high frequency component remains, and train a larger scale CNN on the high resolution data. However, note that the larger scale CNN still contains the spatial frequency bias, which can be revealed when trained on a dataset with prominent high frequencies. For example, see Figure 4.5 where we train a large-scale StyleGAN2 (config-e) on a high resolution fractal dataset.

What all the aforementioned solutions have in common, is an appeal to the Universal Approximator Theorem: a neural network, equipped with hidden units and non-linearities, can model any continuous function given a large enough number of hidden units (i.e. increase in depth and/or width). However, in case of generative CNNs, even though the capacity of generating high frequencies can be increased to any desirable amount by the aforementioned solutions, low frequencies will always receive more capacity than high frequencies. This introduces a redundancy in generative CNNs, and comes at the cost of computational resources and generalization (a larger model demands more data). Naturally, one wonders if there is a way to more directly assign capacity to high frequencies, or to any spatial frequency of interest. The next chapter will explore this idea.
Figure 4.4: (a) Dashed blue line shows the predicted correlation between diagonally adjacent spatial frequencies in Eq. 4.21 for different layers ($d_l$) given a fixed filter size ($k_l = 5$), and the orange points show its empirical evaluation (average correlation with one standard deviation computed over the filters of WGAN-GP trained on CelebA). (b) The impulse response spectrum of effective filters operating on each layer in WGAN-GP trained on CelebA. Notice how the spectrum of the effective filters that operate on inner layers, i.e., control the generation of low frequencies (top row), are much sharper than the ones that operate on outer layers, i.e., control the generation of high frequencies (bottom row). Smoothness is an indication of larger correlation in the effective filters.
Figure 4.5: Average power spectrum of large-scale GANs trained on a fractal-based dataset clearly reveals how low frequencies (closer to center) are matched much more accurately than the high frequencies (closer to corners). (c) Average power spectrum of randomly rotated Koch snowflakes of level 5 and size $1024 \times 1024$. (a, b) Average power spectrum of PG-GAN and StyleGAN2 trained on the latter. PG-GAN completely fails. A representative patch from the perimeter of true and generated fractals are also displayed.
Figure 4.6: Power spectrum for the output of convolving a pure cosine filter \( \cos(\frac{\pi}{4}x) \) of varying filter sizes, with an impulse input of fixed 64x64 size. Notice how the very sharp spectrum produced by the large filter in (a) starts to “leak” as the filter size is gradually reduced in (b), (c) and (d). While the filter in (a) is capable of affecting individual frequency components of the output, the filter loses this spectral resolution as the filter size is reduced, that is, its effects on the output’s frequency components becomes increasingly correlated. The pattern of leakage follows a Sinc function, corresponding to the Fourier transform of the hypothetical window that is limiting the full-size filter.
Figure 4.7: Power spectrum for the output of convolving an impulse input of varying sizes with a pure cosine filter $\cos(\frac{\pi}{4}x)$ of fixed 16x16 size. In (a), where the image and filter have the same size, the output’s spectrum is very sharp, that is, the filter can affect individual frequency components of the output independently, hence having a high spectral resolution. However, as the image size increases and thus the relative filter size is reduced, from (b) to (d), the filter gradually loses its spectral resolution due to frequencies “leaking” into each other, that is, its effects on the output spectrum becomes increasingly correlated.
Chapter 5
Frequency Shifted Generators

5.1 Spectral Shift

In section 4.6, we observed that GANs have a spatial frequency bias, favoring the learning of low frequencies, however, is it possible to manipulate this bias such that it favors other frequencies? If so, this would make it possible to explicitly target specific frequencies of interest in a dataset. In this section, we show how this can be achieved without any increase in training resources. Instead of inherently generating high frequencies, a generative model $G(x, y)$ can first generate a signal with prominent low frequencies and then transform the signal such that these prominent frequencies are shifted towards a desired frequency $(\hat{u}_t, \hat{v}_t)$. This can be achieved by a frequency shift operator:

$$G(x, y)e^{j2\pi(\hat{u}_tx+\hat{v}_ty)} = \frac{1}{mn} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} C(u, v)e^{j2\pi(\hat{u}u+\hat{v}v),(x,y)}$$

(5.1)

where $\hat{u}_t, \hat{v}_t \in [-0.5, 0.5)$. After the frequency shift, the frequency components previously close to $(0, 0)$ are now placed close to $(\hat{u}_t, \hat{v}_t)$. However, since $G$ is generating a real signal and the spectrum of real signals are symmetric, it can not sufficiently represent a high frequency band, that is, $G$ can only represent symmetric frequency bands. Note that while natural images are real signals and have symmetric spectrum with respect to zero, a specific band of their spectrum is not necessarily symmetric. In order to generate a non-symmetric frequency band, we can use two neural networks to generate a real image ($G_r$) and an imaginary
image \((G_i)\), which together compose the complex generated image \((G_c)\). The complex image is then shifted to \((\hat{u}_t, \hat{v}_t)\) according to Eq. 5.1 to construct the shifted generator \(G_s\) as follows:

\[
G_s(x, y) = G_c(x, y)e^{j2\pi(\hat{u}_tx + \hat{v}_ty)} = [G_r(x, y) + jG_i(x, y)]e^{j2\pi(\hat{u}_tx + \hat{v}_ty)}
\]  

(5.2)

The real part of \(G_s\) is now generating an image which can sufficiently represent any frequency band, and has a spatial frequency bias favoring the desired component \((\hat{u}_t, \hat{v}_t)\):

\[
\Re[G_s(x, y)] = G_r(x, y)\cos(2\pi(\hat{u}_tx + \hat{v}_ty)) - G_i(x, y)\sin(2\pi(\hat{u}_tx + \hat{v}_ty))
\]

(5.3)

Frequency Shifted Generators (FSGs) can be used to efficiently target specific spatial frequency components in a dataset. Figure 5.1(a to c) shows the effect of different target frequencies on the power spectrum of the generator (at the start of training when the effect of frequency shift is very easy to observe).

Table 5.1 shows the results of training GANs using FSG with \((\hat{u}_t, \hat{v}_t) = (\frac{1}{2}, \frac{1}{2})\) on SCelebA and SBedrooms, and Figures 5.2, 5.3 and 5.4 show the respective samples compared to original GANs. The use of FSG has considerably improved the GANs’ performance on these high frequency datasets, without any increase in training resources. This also provides an interesting insight: the discriminator is able to effectively guide a capable generator towards learning high frequencies, therefore, the spatial frequency bias must be primarily rooted in the GAN’s generator and not the discriminator.
Table 5.1: Performance gain (decrease in both FID and LR) on the high frequency datasets achieved by using FSG in GANs. Average measures with sd are reported over three random training runs.

<table>
<thead>
<tr>
<th>Model</th>
<th>SCelebA</th>
<th>SBedrooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGAN-GP</td>
<td>328.72 sd 9.70</td>
<td>59.04 sd 5.09</td>
</tr>
<tr>
<td>WGAN-FSG</td>
<td>20.70 sd 0.44</td>
<td>1.93 sd 0.57</td>
</tr>
<tr>
<td>PG-GAN</td>
<td>23.12 sd 2.08</td>
<td>3.93 sd 0.70</td>
</tr>
<tr>
<td>PG-GAN-FSG</td>
<td>17.91 sd 0.74</td>
<td>2.96 sd 0.55</td>
</tr>
<tr>
<td>StyleGAN2</td>
<td>343.57 sd 53.59</td>
<td>43.03 sd 34.10</td>
</tr>
<tr>
<td>StyleGAN2-FSG</td>
<td>7.17 sd 0.07</td>
<td>1.41 sd 0.10</td>
</tr>
</tbody>
</table>
Figure 5.1: Average power spectrum (DC not dropped) of initialized WGAN-GP generators at the start of training, at 128x128 resolution. (a) corresponds to the original generator’s spectrum, (b) to a FSG with $(\hat{u}, \hat{v}) = (\frac{1}{2}, \frac{1}{2})$, (c) to a FSG with $(\hat{u}, \hat{v}) = (\frac{1}{4}, \frac{1}{4})$, and (d) to a multiple-FSG model formed by adding shifted generators at $(0, \frac{1}{16})$, $(\frac{1}{16}, 0)$, $(\frac{1}{16}, \frac{1}{16})$ and $(-\frac{1}{16}, \frac{1}{16})$ to the WGAN-GP generator.
Figure 5.2: WGAN-GP random samples compared to WGAN-GP-FSG on SCelebA and SBedrooms. Samples are re-shifted for visualization.
Figure 5.3: PG-GAN random samples compared to PG-GAN-FSG on SCelebA and SBedrooms. Samples are re-shifted for visualization.
Figure 5.4: StyleGAN2 random samples compared to StyleGAN2-FSG on SCelebA and SBedrooms. Samples are re-shifted for visualization.
5.2 Ensemble of Shifted Generators

We observed that constructing the FSG version of a GAN can improve its performance near a single specific frequency, a natural next step then is to consider whether multiple FSGs, with smaller network capacity, can be added to the main generator of GANs to improve performance on various frequencies. To test this hypothesis, we add four FSGs (four pairs of generators), each with a smaller capacity than the original generator, to each of the three GANs under consideration, and train them on CelebA. Figure 5.1(d) shows the effect of utilizing multiple target frequencies on the power spectrum of the generator (at the start of training when the effect of frequency shift is very easy to observe).

Figure 5.5 shows the improvement in GANs trained on CelebA when their respective generators are enhanced by adding multiple FSGs with \((\hat{u}_t, \hat{v}_t)\) at \((\frac{1}{16}, 0)\), \((0, \frac{1}{16})\), \((-\frac{1}{16}, \frac{1}{16})\), and \((\frac{1}{16}, \frac{1}{16})\) (refer to the Appendix for details of the networks). Interestingly, the added FSGs have specialized towards their respective target frequency \((\hat{u}_t, \hat{v}_t)\), without any explicit supervision during training, as evident in Figures 5.6, 5.8 and 5.10 for the respective GANs. This provides further evidence for the existence of the spatial frequency bias: if unbiased, the added FSGs would have no incentive to specialize towards any specific frequency.

To make sure the observed improvements are not merely due to an increase in the size of the overall network, we also add the exact same FSGs, but without applying the shift, to the main generator, resulting in a modeled denoted FSG-noshift. As apparent from Figure 5.5, FSG-noshift does not provide the same performance improvements, despite having the same size as multiple FSGs models. In fact, it performs worse than the original GAN alone in WGAN-GP and PG-GAN. We hypothesize that this is due to the inability of the added generators to become sufficiently disjoint: when multiple generators are added together, the instant effect is that the gradients received by each generator become less reliable.
than if it was alone, due to the changes simultaneously applied by the other generators to the output. This unreliable gradient directly degrades the training, unless the generators can successfully divide the image generation between themselves, such that the gradients received by different generators become disentangled (for example one generator generating the colors, another the edges, another the textures, and so forth). In practice, we observe that no-shift FSGs added to the original generator fail to achieve this separation well, except in StyleGAN2, as can be seen when comparing Figure 5.11 with Figures 5.7 and 5.9.
Figure 5.5: Improvements in the FID Levels of GANs when enhanced by adding multiple FSGs, trained on CelebA. Note that adding the FSGs without the shift (FSG-noshift) does not yield the same improvements, suggesting that the improvements are not merely due to the increase in the overall generative capacity of the CNNs.
Figure 5.6: Each column corresponds to one sample of a WGAN-GP enhanced by adding multiple FSGs, trained on CelebA. The first four rows, from top to bottom, show the outputs of the FSGs with $(\hat{u}_t, \hat{v}_t)$ at $(\frac{1}{16}, 0)$, $(0, \frac{1}{16})$, $(-\frac{1}{16}, \frac{1}{16})$, and $(\frac{1}{16}, \frac{1}{16})$ respectively. The fifth row shows the main generator, and the final row the output of the compound generator (sum of all the preceding rows).
Figure 5.7: WGAN-GP’s FSG-noshift samples, trained on CelebA. The first eight rows correspond to the outputs from each of the FSG-noshift generators. The ninth row is the sum of the outputs of FSG-noshift generators (sum of all the eight rows above), middle is the original generator, and bottom is the final output (sum of the two preceding rows).
Figure 5.8: Each column corresponds to one sample of a PG-GAN enhanced by adding multiple FSGs, trained on CelebA. The first four rows, from top to bottom, show the outputs of the FSGs with \((\hat{u}_t, \hat{v}_t)\) at \((\frac{1}{16}, 0), (0, \frac{1}{16}), (-\frac{1}{16}, \frac{1}{16})\), and \((\frac{1}{16}, \frac{1}{16})\) respectively. The fifth row shows the main generator, and the final row the output of the compound generator (sum of all the preceding rows).
Figure 5.9: PG-GAN’s FSG-noshift samples, trained on CelebA. The first eight rows correspond to the outputs from each of the FSG-noshift generators. The ninth row is the sum of the outputs of FSG-noshift generators (sum of all the eight rows above), middle is the original generator, and bottom is the final output (sum of the two preceding rows).
Figure 5.10: Each column corresponds to one sample of a StyleGAN2 enhanced by adding multiple FSGs, trained on CelebA. The first four rows, from top to bottom, show the outputs of the FSGs with $(\hat{u}_t, \hat{v}_t)$ at $(\frac{1}{16}, 0)$, $(0, \frac{1}{16})$, $(-\frac{1}{16}, \frac{1}{16})$, and $(\frac{1}{16}, \frac{1}{16})$ respectively. The fifth row shows the main generator, and the final row the output of the compound generator (sum of all the preceding rows).
Figure 5.11: StyleGAN2’s FSG-noshift samples, trained on CelebA. The first eight rows correspond to the outputs from each of the FSG-noshift generators. The ninth row is the sum of the outputs of FSG-noshift generators (sum of all the eight rows above), middle is the original generator, and bottom is the final output (sum of the two preceding rows).
5.3 Related Works

Recent works on fully-connected neural networks have discovered a spectral bias against learning high frequency functions [105, 8], which can be addressed by using a proper high dimensional embedding of the input space [123]. However, while these works define a frequency component as a periodic change in a single output of the network with respect to changes in the input space, we define a frequency component as a periodic change across the adjacent outputs of the network (hence a \textit{spatial} frequency component). Note that these two notions of frequency are independent by definition, that is, one can be mathematically defined while the other is not and vice versa, therefore a bias in one does not readily imply a bias in the other, and the arguments do not carry over.

Spectral limitations have been observed in different tasks when using generative Convolutional Neural Networks (CNNs). In L2-reconstruction tasks, there is a bias against high frequencies, primarily attributed to the vanishing gradient of the L2 loss on low powered frequencies, which is often counteracted by emphasizing the low-power frequencies through additional loss terms [21, 78, 129]. In Auto Encoders (AEs) and Variational Auto Encoders (VAEs), there is a similar bias, primarily attributed to the distribution assumptions in their objectives [53, 73]. In contrast, GAN’s objective does not impose any such spectral limitations. In theory, GANs must be able to learn any suitable distribution regardless of the carrier spatial frequencies. Therefore, while a spectral bias in generative CNNs could be obscured by the inherent spectral biases of AEs, VAEs and L2 tasks, GANs provide a clear lens for observing such biases.

Very relevant to this work, two recent concurrent works [28, 26] have observed that high frequency discrepancies can be utilized to easily distinguish GAN generated images from natural images. Specifically, Dzanic \textit{et al.} [28] have shown that these discrepancies can be removed by directly modifying the spectrum of
generated images in post-processing. Our findings in this work are consistent with these observations, but further suggest that such discrepancies are the consequence of an underlying spatial frequency bias against learning high frequencies. As such, the issue is not merely that GANs generate high frequency artifacts on datasets with little to no high frequency content (e.g. high resolution natural images), which can be removed by modifying the generated spectrum after training, rather that GANs tend to lose the information carried by high frequencies, which is not recoverable in post-processing. Additionally, Durall et al. [26] connect the discrepancies to the aliasing introduced by the up-sampling operator in the expanded spectrum. However, note that if the up-sampling does not introduce aliased power in the expanded frequency spectrum, the following layer will not have any power to filter-out and shape the desired higher frequencies. In contrast, in Theorem 1 (Section 4.6), we showed that the more likely culprit is the linear dependencies in the Conv filter’s spectrum that cause spectral limitations, and explained why this affects high frequencies more severely than low frequencies, causing a systemic bias against high frequencies, not just artifacts in the power spectrum.

The prevalent metrics for evaluating GANs, most notably Inception Score [113], FID [47], and MS-SSIM [96], consider all spatial frequency components at once when comparing GAN generated and true images, thus lacking any spectral resolution. Most relevant to our proposed metric, Karras et al. [58] propose computing sliced Wasserstein distance between patches extracted from true and generated images at different levels of a Laplacian pyramid (SWD). Interestingly, evaluating GANs with SWD shows approximately similar performance across frequency bands [58]. We conjecture that this discrepancy comes from the fact that small differences between patches in the pixel space, can result in large differences in the more meaningful feature space used by FID. As such, FID Levels is a more informative measure of GANs’ performance across spatial frequencies.
Several models have incorporated multiple discriminators and/or generators to improve the training of GANs. Durugkar et al. [27] use an ensemble of discriminators on the input image in order to provide a more reliable learning signal to the generator. Wang et al. [135] use multiple discriminators at different spatial resolutions of the image in order to increase the receptive field of the discriminator and enhance its ability to detect global features. Conversely, Arora et al. [5] propose using a collection of generators based on the well-known advantage of learning a mixed strategy versus a pure strategy in game theory. Similarly, Hoang et al. [49] use a collection of $k$ generators in order to model a mixture distribution, and train them together with a $k$-class classifier to encourage them to each capture a different component of the real mixture distribution. Ghosh et al. [37], also uses $k$ generators, together with a $k+1$-class discriminator which is trained to correctly classify samples from each generator and true data (hence a $k+1$ classifier), in order to increase the diversity of generated images. Hierarchical structures have also been proposed for GANs [22, 54, 142, 143] which are built on the idea that breaking the GAN’s training into smaller steps could stabilize training and improve GAN’s performance. While these models bring about many promising enhancements to the convergence of GANs from a probability distribution point of view, none investigate the performance of GANs in presence of different frequency components. From the frequency perspective, using multiple discriminators would not affect the frequency bias, since even a single discriminator can sufficiently distinguish high frequencies as well as low frequencies, as we observe in Section 4.5, and the issue appears to be isolated to the generator; using multiple generators, on the other hand, does not improve the frequency bias because all the generators share the same frequency bias, although it is indeed possible to modify them such that each have a unique frequency bias and can therefore affect the frequency bias as an ensemble, an approach discussed in Section 5.1; hierarchical GANs, specifically the model by Denton et al. [22] which
trains independent conditional GANs on levels of a Laplacian pyramid such that each generator explicitly focuses on a specific frequency band, would still suffer from the frequency bias because, as we showed in Section 4.5, the main issue is not the lack of focus on a specific frequency band during training: the sharp decline in performance on high frequencies is still observed even when such frequencies are much more prominent than low frequencies.

Finally, several recent works have achieved astonishing results by combining various old and new tricks of GAN’s training with very large neural networks. Most notably, Karras et al. [58] used a progressively growing training procedure for GANs, in conjunction with several normalization tricks and large capacity neural networks, to learn very high resolution images with astonishing details. Brock et al. [11], in the same spirit, trained a GAN model, based on the model in [141], at the largest scale attempted to date, and showed promising results at high resolution generation, while also identifying the shortcomings causing instability for the training at such large scales. The results of our research could benefit small and large GAN models alike as explained in Section 4.6.
Chapter 6

General Conclusion

In a time of a revived interest in unsupervised learning, where the vast amount of unlabeled data available online is seen as a great potential for building useful representations of the world – representations that can subsequently be used to solve many down-stream supervised tasks with only a fraction of the labeled data otherwise required – Generative Adversarial Networks (GANs) are viewed as a prime way to capture the inner workings of real-world phenomena, such as the human facial movement or the human voice or the folding of proteins. Once a GAN is trained, recreating and manipulating these phenomena become a matter of searching and structuring the latent space of the GAN: to animate a still portrait, one can search the latent space of a face-generating GAN for the code to that photograph and then use structures on this latent space to transform it to a smiling expression; to read an arbitrary message in a famous voice, one can search the latent space of a speech-generating GAN for the code of the desired voice and then use structures on this space to utter different words; same goes for simulating the evolution of a cosmological object according to cosmological observations and theories, or discovering proteins that fold in a desirable way. GANs are to become the most meticulous and talented painters of our world, such that creating any desirable picture becomes merely a matter of learning to talk to them, that is, to search and structure their latent space. While this appears promising in many respects, we must note that GANs, despite their great expressivity compared to their predecessors, still contain certain limitations and biases, and therefore if they are to be used as models of reality, we must make
sure we understand the extent to which such a reality is accurately represented by GANs. This last argument is the main lesson that can be gleaned from this dissertation, that even though GANs have come a long way and can generate distributions with unprecedented fidelity, there are still certain geometric and spectral blind spots, possibly many more, that must be dealt with and carefully considered in any down-stream task.

The initial motivation for the projects that shaped this dissertation was the wondering, and perhaps frustration, at why GANs tend to generate samples, albeit occasionally, that look nothing like the phenomenon they were trained on, and why such failures affect high frequencies much more than low frequencies.

Analyzing the former motivating question, led to the study of GANs’ performance on distributions supported on disconnected compact manifolds. We have shown that the common single-generator GANs can not correctly learn such distributions due to the smoothness of the underlying generative DNN, and this shortcoming can have certain consequences in practice, namely mode dropping, loss of local convergence, and generating off-manifold samples. Furthermore, we proposed a multi-generator GAN as an effective solution to this limitation. Importantly, we showed that learning a prior over the generators rather than using a fixed prior is crucial for the success of such multi-generator GANs. However, it is important to emphasize that in this work we have assumed the disconnectedness of the true data support. Verifying this assumption in major datasets, and studying the topological properties of these datasets in general, are interesting avenues for future research. Extending the prior learning to other methods, such as learning a prior over shape of the latent space, also remains as an exciting future path for research.

Shifting our focus to the latter motivating question, led to the study of GANs’ spectral performance. We identified a bias against high spatial frequencies in convolutional GANs, and then proposed an approach for manipulating this bias and
reducing its adverse effects. Being aware of this bias is of paramount importance when using GANs in applications concerned with intricate patterns. Moreover, our findings suggest that the information carried by high frequencies is much more likely to be missed by GANs, a critical consideration when using GANs for data-augmentation or in semi-supervised learning. We also observed that the spatial frequency bias primarily affects GAN’s generator and not its discriminator. This gives the discriminator an advantage which can be the root of certain instabilities in GAN training. Investigating this connection between the spatial frequency bias and unstable GAN training, as well as extending the introduced theoretical analysis to incorporate the effect of various normalization and stabilization techniques, are interesting directions for future research. We also proposed Frequency Shifted Generators (FSGs) as a way to match the bias of GANs to the known biases of a dataset. However, note that while FSGs exploit the bias to improve the performance, they do not remove this bias. Devising solutions for removing the bias is another potential direction for future research.

Finally, we would like to conclude this document by emphasizing the importance and potential of explicit ensembles, the shared pillar upon which both of our proposed solutions in this dissertation stand. While the value of implicit ensembles is well-known, per the success of sparse neural networks in particular and latent-variable models in general, the value of explicit ensembles is often overlooked. The general trend in machine learning is presently towards building larger deeper models, that is, implicit ensembles of millions of experts, however, our observations throughout the development of this dissertations suggest that there is a lot of potential in using many instances of smaller models together (i.e. explicit ensembles), compared to a single instance of a larger model. The advantage of explicit ensembles comes from the separability of the different DNNs simultaneously trained in the ensemble: this allows for more control when incorporating prior knowledge into a task since we can modify the structure of certain
DNNs without affecting the other members of the ensemble; can mitigate the effects of bad initializations and bad local optima since if one member becomes hindered by such issues, it cannot directly affect the many other members in the ensemble; and finally, makes the overall model much more interpretable since the contribution of each DNN in the ensemble can be observed and analyzed independent of the other members.
Appendix A

Networks of Disconnected Manifold Learning

The pre-trained classifier used for metrics is the ALL-CNN-B model from [120] trained to test accuracy 0.998 on MNIST and 1.000 on Face-Bed. The GAN architecture is based on the WGAN-GP detailed in [44] using the architectures described in Tables A.1 through A.6 with the following hyper-parameters: Batch Normalization [56] is used on the pre-activation latent layer of $Q(x)$; Adam [66] optimizer is used with a batch size of 32, learning rate of 0.0002, and first and second moment moving average factor of $\beta_1 = \beta_2 = 0.5$; Leaky ReLU when ever used has a negative slope of 0.2; weight initialization is due to Xavier Initialization [39] and biases are initialized to zero. Finally, for Gradient Penalty [44], a weight of 10 is used.
Table A.1: Single Generator Models on MNIST.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G(z)$: $z \sim U[-1, 1]$</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>8 x 8 x 64</td>
<td>ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>4 x 4 x 128</td>
<td>8 x 8 x 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td></td>
<td></td>
<td>16 x 16 x 64</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>16 x 16 x 32</td>
<td>ReLU</td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td></td>
<td></td>
<td>28 x 28 x 32</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>8 x 8 x 1</td>
<td>Tanh</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>28 x 28 x 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Multiple Generator Models on MNIST.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
<th>Shared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_c(z)$: $z \sim U[-1, 1]$</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>8 x 8 x 16</td>
<td>ReLU</td>
<td>N</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>4 x 4 x 32</td>
<td>8 x 8 x 32</td>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td></td>
<td></td>
<td>16 x 16 x 16</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>16 x 16 x 8</td>
<td>ReLU</td>
<td>N</td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td></td>
<td></td>
<td>28 x 28 x 8</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>28 x 28 x 1</td>
<td>Tanh</td>
<td>N</td>
</tr>
</tbody>
</table>

$Q(x), D(x)$: 28 x 28 x 1

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
<th>Shared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Connected</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>1 x 1</td>
<td>4 x 4 x 128</td>
<td>Leaky ReLU</td>
<td>Y</td>
</tr>
<tr>
<td>Q Fully Connected</td>
<td>$n_g$</td>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>
Table A.3: Single Generator Models Face-Bed.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G(z) ): ( z \sim U[-1, 1] )</td>
<td></td>
<td></td>
<td>100</td>
<td>ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>8 x 8 x 512</td>
<td>1 x 1</td>
<td>16 x 16 x 256</td>
<td>ReLU</td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td>32 x 32 x 256</td>
<td>32 x 32 x 128</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td>64 x 64 x 128</td>
<td>64 x 64 x 3</td>
<td>Tanh</td>
<td></td>
</tr>
</tbody>
</table>

\( D(x) \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Connected</td>
<td>32 x 32 x 3</td>
<td>32 x 32 x 128</td>
<td>Leaky ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>2 x 2</td>
<td>16 x 16 x 256</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>2 x 2</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td></td>
<td></td>
<td>1</td>
<td>N</td>
</tr>
</tbody>
</table>

Table A.4: Multiple Generator Models on Face-Bed.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
<th>Shared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_c(z) ): ( z \sim U[-1, 1] )</td>
<td></td>
<td></td>
<td>100</td>
<td>ReLU</td>
<td>N</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>8 x 8 x 128</td>
<td>16 x 16</td>
<td>16 x 16 x 64</td>
<td>ReLU</td>
<td>N</td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td>32 x 32 x 64</td>
<td>32 x 32 x 32</td>
<td>ReLU</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Nearest Upsample</td>
<td>64 x 64 x 32</td>
<td>64 x 64 x 3</td>
<td>Tanh</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

\( Q(x), D(x) \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
<th>Shared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Connected</td>
<td>64 x 64 x 3</td>
<td>32 x 32 x 128</td>
<td>Leaky ReLU</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>2 x 2</td>
<td>16 x 16 x 256</td>
<td>Leaky ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>5 x 5</td>
<td>2 x 2</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
<td></td>
</tr>
<tr>
<td>D Fully Connected</td>
<td>1</td>
<td></td>
<td>1</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Q Convolution</td>
<td>5 x 5</td>
<td>2 x 2</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
<td></td>
</tr>
<tr>
<td>Q Fully Connected</td>
<td></td>
<td></td>
<td>n_g</td>
<td>Softmax</td>
<td>N</td>
</tr>
</tbody>
</table>
Table A.5: Single Generator Models on Disjoint Line Segments.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G(z): z \sim U[-1, 1]$</td>
<td>100</td>
<td></td>
<td></td>
<td>ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>128</td>
<td></td>
<td></td>
<td>ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>64</td>
<td></td>
<td></td>
<td>ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D(x)$</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fully Connected</td>
<td>64</td>
<td></td>
<td></td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>128</td>
<td></td>
<td></td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.6: Multiple Generator Models on Disjoint Line Segments.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
<th>Shared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_c(z): z \sim U[-1, 1]$</td>
<td>100</td>
<td></td>
<td></td>
<td>ReLU</td>
<td>N</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>32</td>
<td></td>
<td>ReLU</td>
<td>ReLU</td>
<td>N</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>16</td>
<td></td>
<td>ReLU</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Fully Connected</td>
<td>2</td>
<td></td>
<td>ReLU</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>$Q(x), D(x)$</td>
<td>2</td>
<td></td>
<td>Leaky ReLU</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Fully Connected</td>
<td>64</td>
<td></td>
<td>Leaky ReLU</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Fully Connected</td>
<td>128</td>
<td></td>
<td>Leaky ReLU</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>D Fully Connected</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q Fully Connected</td>
<td>128</td>
<td></td>
<td>Leaky ReLU</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Q Fully Connected</td>
<td>$n_g$</td>
<td></td>
<td>Softmax</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

Networks of Frequency Shifted Generators

For WGAN-GP, we use the network in Table B.1, and train according to the specifications of [44] for 200 epochs. For PG-GAN, we use the network in Table B.2 and train according to the specifications of [58] for 10M images. For StyleGAN2, we use the network and training setup of config-e in [60], except that we use the same progression of CNN filter channel sizes as in Table B.2, and train for 10M images. On the $1024 \times 1024$ fractal dataset of Figure 4.5, the exact network and setup of config-e is used for the large-scale StyleGAN2.

In multiple FSGs experiments, for WGAN-GP, each FSG has the same network as WGAN-GP itself, except it has only up-sampling instead of the last layer, so that FSGs have less capacity than the main generator. For PG-GAN and StyleGAN2, each FSG shares all the Conv blocks of the corresponding main generator network except the two outer most blocks, which are replaced by a single Conv block followed by up-sampling to the output spatial dimension, that is, the FSGs are essentially branching out of the main generator at the start of the $64 \times 64$ Conv layer. In StyleGAN2, the added FSGs share the mixing input to the last block of the main generator.

For FID calculation, 50K images are sampled per distribution in CelebA and SCelebA experiments, and 10K images are sampled in LSUN-Bedroom and SBedrooms experiments, due to the limited number of available true images. For spectrum visualization, 1K images are sampled.
Table B.1: WGAN-GP’s networks.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G(z)$: $z \sim U[-1, 1]$</td>
<td></td>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fully Connected</td>
<td>$8 \times 8 \times 512$</td>
<td></td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Upsample</td>
<td>$16 \times 16 \times 512$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$1 \times 1$</td>
<td>$16 \times 16 \times 256$</td>
<td>ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td>$32 \times 32 \times 256$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$1 \times 1$</td>
<td>$32 \times 32 \times 128$</td>
<td>ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td>$64 \times 64 \times 128$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$1 \times 1$</td>
<td>$64 \times 64 \times 64$</td>
<td>ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td>$128 \times 128 \times 64$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$1 \times 1$</td>
<td>$128 \times 128 \times 3$</td>
<td>Tanh</td>
</tr>
<tr>
<td>$D(x)$: $x$</td>
<td></td>
<td>128</td>
<td>$128 \times 128 \times 3$</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$2 \times 2$</td>
<td>$64 \times 64 \times 64$</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$2 \times 2$</td>
<td>$32 \times 32 \times 128$</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$2 \times 2$</td>
<td>$16 \times 16 \times 256$</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>$5 \times 5$</td>
<td>$2 \times 2$</td>
<td>$8 \times 8 \times 512$</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Table B.2: PG-GAN's and StyleGAN2's networks.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G(z): z \sim \mathcal{N}(0, 1) )</td>
<td></td>
<td></td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>4 x 4</td>
<td>1 x 1</td>
<td>4 x 4 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>4 x 4 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td></td>
<td></td>
<td>8 x 8 x 512</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td></td>
<td></td>
<td>16 x 16 x 512</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>16 x 16 x 256</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>16 x 16 x 256</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td></td>
<td></td>
<td>32 x 32 x 256</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>32 x 32 x 128</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>32 x 32 x 128</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td></td>
<td></td>
<td>64 x 64 x 128</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>64 x 64 x 64</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>64 x 64 x 64</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Upsample</td>
<td></td>
<td></td>
<td>128 x 128 x 64</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>128 x 128 x 32</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>128 x 128 x 32</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>1 x 1</td>
<td>1 x 1</td>
<td>128 x 128 x 3</td>
<td></td>
</tr>
<tr>
<td>( D(x): x )</td>
<td></td>
<td></td>
<td>128 x 128 x 3</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>1 x 1</td>
<td>1 x 1</td>
<td>128 x 128 x 32</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>128 x 128 x 32</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>128 x 128 x 64</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Downsampler</td>
<td></td>
<td></td>
<td>64 x 64 x 64</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>64 x 64 x 64</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>64 x 64 x 128</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Downsampler</td>
<td></td>
<td></td>
<td>32 x 32 x 128</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>32 x 32 x 128</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>32 x 32 x 256</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Downsampler</td>
<td></td>
<td></td>
<td>16 x 16 x 256</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>16 x 16 x 256</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>16 x 16 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Downsampler</td>
<td></td>
<td></td>
<td>8 x 8 x 512</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>8 x 8 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Downsampler</td>
<td></td>
<td></td>
<td>4 x 4 x 512</td>
<td></td>
</tr>
<tr>
<td>Minibatch stddev</td>
<td></td>
<td></td>
<td>4 x 4 x 513</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td>4 x 4 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Convolution</td>
<td>4 x 4</td>
<td>4 x 4</td>
<td>1 x 1 x 512</td>
<td>Leaky ReLU</td>
</tr>
<tr>
<td>Fully Connected</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
References


IEEE conference on computer vision and pattern recognition, pages 1526–1535, 2018.


