Generative Modeling for Conditioned Biomaterial Topography: A Generative Adversarial Network based Approach

Submitted September 2023, in partial fulfillment of
the conditions for the award of the degree **MSc Computer Science(AI).**

Tarun Kashyap
20498663

Supervised by Grazziela Figueredo

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Date: 08/09/2023

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Abstract

Biomaterial surface topographies play a crucial role in influencing microbe-material interactions, with specific surfaces essential for microbial attachment. The choice of materials is pivotal, with active materials potentially leading to fatal autoimmune responses, prompting a shift towards bio-compatible, non-active materials. The microstructure of these materials can, however, be designed to modulate microbial attachment. With the conventional design process being costly and time-consuming, this study delves into the generative potential of Generative Adversarial Networks (GANs) as a solution. By training GANs on existing attachment interaction data, this research seeks to expedite the biomaterial design process, generating structures with controlled behavior. The study employs three generative models: cGAN, DCGAN, and VAE, with the latter serving as a reference model. Preliminary results showcase the DCGAN model's superiority in generating 2D surface topography images, though a conclusive quality assessment of generated images remains elusive due to the nature of the input images. This work underscores the need for adaptive feedback loops in GAN-based modeling for real-world applications and paves the way for more nuanced and efficient biomaterial design processes, with vast implications for healthcare and research.

Acknowledgements

I am profoundly indebted to my esteemed supervisor, Dr. Grazziela Figueredo. Her unwavering support, insightful wisdom, and professional acumen have been indispensable throughout the course of this academic pursuit. The guidance and constructive feedback offered by Dr. Figueredo significantly elevated the caliber of this research.

I extend my sincere appreciation to the University of Nottingham for fostering a conducive environment for academic inquiry and innovation. The institution's robust resources, state-of-the-art facilities, and distinguished academic community have been instrumental in molding this research. I am deeply grateful for the myriad opportunities and enriching experiences the university has accorded me, reflecting its unwavering dedication to academic excellence.

To my family, for their steadfast support, immeasurable affection, and enduring patience have been a beacon during this journey. Their undying faith in my potential remains a perennial source of motivation.

I also wish to convey my heartfelt gratitude to my friends who have been a pillar of support, and companionship, and offered moments of respite amidst the rigors of my research. Their camaraderie has made this expedition both rewarding and memorable.

In conclusion, my acknowledgments would be incomplete without expressing gratitude to all the individuals, institutions, and resources that played a pivotal role in the fruition of this research. I cherish and value the support and the wealth of knowledge I have amassed on this journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The effect of material surface topography of biological implants relates to the microbe-material surface interaction, the microbes need a specific type of surface to attach to the material surface [24][8], the effect can also be modulated based on the type of material being used, but generally using an active material may result in the body rejecting the material due to autoimmune response[23], which can be fatal in some scenarios. This leaves us with the option of using chemically non-active material that is bio-compatible and doesn't lead to an auto-immune response. However, the material surface micro-structure can be used to modulate the microbe attachment to the surface. Various research [papers] have shown successful modulation of microbe attachment with the surface of biomaterial by changing the shape of 3D microstructures on the surface of the material. There are a multitude of factors affecting the interaction and also vary on the size and shape of the microbe, the factors include but are not limited to shape, orientation, space in-between, and stiffness of the microstructure[paper].

Researching and studying such complex factors and their impacts tends to be expensive and time-consuming. [ChemArchiChip]. Machine Learning models have been very useful in finding the relationship between microbe-material surface interaction[InnateImmune-ML]. The factors encoded the geometric properties of the objects, such as surface Area, Volume, or linear dimensions. With advancements in machine learning, the potential to

accelerate this development process emerges. Generative models, specifically Generative Adversarial Networks (GANs), offer a promising avenue. These models, conditioned with appropriate data, can generate innovative designs, potentially reducing both the time and cost of development. More so, they hold the capability to decipher the nuanced relationships between microbe-topography interactions, which can be instrumental in designing effective biomaterial surfaces.

1.2 Aims and Objectives

Our study proposes the use of GANs for expediting the biomaterial design process by training generative models on existing data with subsequent microbe attachment ratio, which can be used to condition our GAN training, allowing us to generate structures with controlled behavior. The study incorporates a dataset of 2000 2D topography images of micron-level 3D microstructures with corresponding bacteria attachment ratios of 2 types of bacteria A and B. Our goal is to perform an experiment on the use of GANs for this purpose. We designed three different generative models, a Conditional GAN (cGAN), a Deep Convolutional GAN (DCGAN), and a Variational AutoEncoder(VAE) as a reference model. Each of the models used has its own advantages and drawbacks, a cGAN is conditioned with the labels, helping guide the training process thus avoiding network oscillations. The DCGAN offers a deeper understanding of images due to the presence of Convolutional layers in the model which are quite good at capturing the in-depth features of an image. The VAE used in our study is a reference model, which is used to gauge the performance of the GAN models. The outputs of VAE and GANs are visually inspected and a Frechet Inception Distance (FID) scores are calculated.

The hypothesis behind our is that a good GAN-based generative model, which in general is known to work very well regardless of the type of image, can generate new topography samples with enough precision that they work within the range of attachment they are generated for, will provide a new opportunity in how auto-immune and bacteria resistant biomaterials are manufactured, which will not only bring down the price of development and the material itself but also pave a pathway for expanding the scope of generative

modeling. So far without research, we are limited to visible physical properties to guide the generation process, but chemical and biological properties of the material can also be considered to create an even more robust generator.

1.3 Description of the work

Our study resulted in successfully building generative models for generating 2D surface topography images. Various model enhancements were performed and the models were trained for more than 2000 epochs and were limited by resource constraints. The visual inspection suggests the best output by the DCGAN model at 1000 epochs of training with an FID score of 717.67, which is the lowest among the three models. Both the GANs performed better than the VAE model whose FID score was 1448.61. However, the visual inspection does not lead to a conclusive inference due to the unidentifiable nature of input images, which have very abstract shapes due to being the 2D topography of 3D microstructures, this is well explained in the paper[3][Matthew 2020]. Due to this issue, we can say that our model is performing better than VAE but cannot make a conclusive statement of how well it's working, as the FID score does not provide a specific set of ranges to define the quality of the generated image.

Our study's primary limitation is the uncertainty about the real-world functionality of the generated topographies, highlighting the need for an adaptive feedback loop in GAN-based modeling. Though, this research validates the potential of GANs in accelerating the design of biomaterial topographies, offering solutions for time-intensive and costly R&D in various fields. Future research could explore hybrid models, test different architectures, and utilize pre-trained models to improve generative outcomes.

Chapter 2

Background and Related Work

2.1 Microbe-Biomaterial Surface Interaction

The effect of interaction between Biomaterial and various microbial lifeforms is well studied. Various papers discuss the factors affecting the attachment ratios of microbes on different types of Biomaterials (both on chemical and physical aspects).

The paper [Saud, 2020] [8] provides a meticulous analysis of the interaction between bacteria and biomaterial surfaces, emphasizing the profound impact of surface topography on bacterial behavior. Surface topographies dictate how bacteria approach, attach, and proliferate on biomaterials. Factors such as the size, shape, and distribution of surface features play a crucial role in either facilitating or inhibiting bacterial adhesion. For instance, certain topographical patterns, when aligned with bacterial dimensions, can disrupt the attachment and motility of bacteria, thereby mitigating biofilm formation. Conversely, when the dimensions of the surface topographies exceed bacterial size, the attachment behavior often becomes independent of the surface features, indicating the nuanced role size plays in bacteria-biomaterial interactions.

Drawing from nature, the paper underscores how biomimetic designs, like those inspired by cicada wings, can be harnessed to create surfaces that actively damage bacteria upon contact. Such designs exploit the physical interactions between bacterial cell walls and sharp nanopillars to rupture and kill the bacteria. This exploration of bacteria-biomaterial interaction signifies that strategic manipulation of surface topography offers a promising

avenue to control bacterial attachment and proliferation, thereby providing a proactive solution to combat device-associated infections.

A similar paper [Li-Chong, 2020] [24], also discusses the Bacteria-Biomaterial interaction. The paper delves into the intricate dynamics of bacterial adhesion to biomaterial surfaces, a critical precursor to implant-related infections. Central to the discussion is the influence of the physical properties of the biomaterial on bacterial attachment.

Bacterial attachment unfolds in two sequential phases. Initially, there's a quick, unspecific, reversible phase driven by hydrodynamic and physicochemical interactions. This phase involves forces such as Lifshitz–van der Waals and electrostatic interactions. Following this, the irreversible phase takes over, characterized by specific molecular reactions between the bacteria and the material surface. In this phase, bacterial structures, including capsules and fimbriae, anchor the bacteria firmly to the surface.

The physical properties of the biomaterial surface, notably its chemistry, roughness, energy, charge, and hydrophobicity, emerge as significant determinants of bacterial adhesion. The paper underscores the pivotal role of surface chemistry in modulating bacterial adhesion. Interactions like van der Waals and electrostatic forces, rooted in the material's surface chemistry, regulate bacterial attachment. Theoretical models, such as the thermodynamic approach, have been proposed to predict bacterial adhesion based on these interactions.

Surface roughness is another influential factor. For isolating the influence of chemical interactions on bacterial adhesion, the material surface needs to be exceptionally smooth, typically with a root-mean-square roughness less than 2 nm. This ensures that the topographical effects don't confound the results.

The paper also highlights the dichotomy between hydrophilic and hydrophobic surfaces. Bacterial adhesion patterns vary based on the hydrophobicity of the surface. For instance, hydrophilic surfaces with positive charge characteristics typically exhibit maximum bacterial attachment. Conversely, hydrophilic surfaces with negative charge characteristics show the least bacterial adhesion.

Material modification emerges as a promising avenue to regulate bacterial adhesion. Experiments using self-assembled monolayers (SAMs) of thiol on gold substrates or alkyl silane layers on glass substrates have illuminated the influence of different functional groups on bacterial attachment. One study cited found that S. aureus exhibited the lowest adhesion on ethylene oxide-bearing surfaces. Another investigation on polyurethane surfaces revealed that hydrophilic modifications significantly curtailed bacterial adhesion compared to hydrophobic ones.

In conclusion, this paper offers valuable insights into the nexus between biomaterial surface properties and bacterial adhesion. It underscores the need to comprehend these interactions to design materials that can effectively resist bacterial attachment, thereby preventing implant-related infections.

2.2 Modulating Microbe-Biomaterial Attachment Interaction

The paper[Matthew, 2020] [23]investigates the use of material topography to modulate the attachment and differentiation of human monocytes into macrophages. Macrophages are a type of immune cell that plays a central role in the body's response to foreign materials. They can either promote healing or inflammation, depending on their phenotype. The authors used a high-throughput screening approach to investigate the relationship between topography and monocyte attachment and phenotype. They generated a library of 2176 micropatterns with different shapes and sizes and then exposed human monocytes to these patterns.

The results showed that micropillars 5-10 µm in diameter were the most effective at promoting monocyte attachment. The authors also found that the combination of pillar size and density was important for modulating monocyte phenotype. For example, micropillars with a diameter of 5 µm and a density of 100 per mm2 promoted the differentiation of monocytes into anti-inflammatory M2 macrophages, while micropillars with a diameter

of 10 µm and a density of 200 per mm2 promoted the differentiation of monocytes into pro-inflammatory M1 macrophages.

The findings of this study suggest that the topography of a material can be used to control the immune response to that material. This could have implications for the design of biomaterials, such as medical implants, that are less likely to elicit an adverse immune response.

The author of this paper further delves into the intricate physical properties of the shape of micron-scale 3D objects, and develops a platform, ChemoArchiChip, to create and study new microstructures.

The paper[Matthew, 2023] [22] investigates the use of micron-scale 3D objects to modulate the behavior of innate immune cells, such as macrophages. The authors used a technique called two-photon polymerization to create arrays of surface-mounted, geometrically diverse 3D polymer objects. They then exposed these objects to macrophages and measured the cells' attachment, polarization, and gene expression.

The results of the study showed that the architecture and polymer chemistry of the 3D objects had a significant impact on the behavior of the macrophages. For example, objects with a high surface area-to-volume ratio promoted macrophage attachment, while objects with a high degree of curvature promoted macrophage polarization. The authors also found that the polymer chemistry of the objects could influence the expression of specific genes in the macrophages.

The authors used three different polymer chemistries for the 3D objects: GDGDA, GPOTA, and BDDA. These chemistries have different properties that can influence macrophage behavior. For example, GDGDA is a biocompatible polymer that has been shown to promote macrophage polarization towards an anti-inflammatory phenotype. GPOTA is a less biocompatible polymer that has been shown to promote macrophage polarization towards a pro-inflammatory phenotype. BDDA is a neutral polymer that has been shown to have no significant effect on macrophage polarization.

The authors also used two different shapes for the 3D objects: pillars and discs. Pillars

have a high surface area-to-volume ratio, while discs have a lower surface area-to-volume ratio. The authors hypothesized that pillars would promote macrophage attachment more than discs.

The results of the study supported the authors' hypotheses. Objects with a high surface area-to-volume ratio, such as pillars, promoted macrophage attachment more than objects with a lower surface area-to-volume ratio, such as discs. Objects with a high degree of curvature, such as pillars, also promoted macrophage polarization more than objects with a lower degree of curvature, such as discs. The polymer chemistry of the objects also affected macrophage polarization, with GDGDA promoting an anti-inflammatory phenotype, GPOTA promoting a pro-inflammatory phenotype, and BDDA having no significant effect.

The authors concluded that the architecture and polymer chemistry of 3D objects can have a significant impact on the behavior of macrophages. The ChemoArchiChip is a versatile platform that can be used to screen a large library of 3D objects to identify those that have the desired effects on macrophage behavior.

The findings of this study have important implications for the design of immunomodulatory implants. By carefully controlling the architecture and polymer chemistry of 3D objects, it may be possible to engineer implants that can specifically target and modulate the behavior of innate immune cells.

2.3 Generative Modelling

The goal of this paper [Goodfellow, 2014] [6] is to reduce the effort and cost of developing new microstructure design as studied in previous papers and use Machine Learning algorithms to learn the complex relationship of shape and orientation of the microstructures and their subsequent impact on microbe attachment. Such a method would expedite the development process.

Machine Learning algorithms, especially artificial neural networks are widely known for their exceptional performance in learning intricate correlations in data by mapping a very

complex estimator over its networks, in other words, they learn the latent distribution function of the data. ANNs are used for tasks such as classification, object detection, and pattern recognition and are trained over a large number of data points. This aspect of learning latent information from data can be used to generate new samples of the same data, this is known as Generative Modelling. The ANN-based generative model was introduced in the paper Generative Adversarial Nets [Goodfellow, 2014]

This paper proposed a novel approach to training generative models. GANs consist of two main components: a generator network and a discriminator network. The generator network learns to generate data that is similar to a given dataset, while the discriminator network learns to differentiate between real data from the dataset and fake data produced by the generator. Based on the minimax algorithm from game theory, the training process of GANs involves a game between these two networks. The generator tries to produce data that is indistinguishable from real data, while the discriminator tries to get better at distinguishing between real and fake data. This adversarial process leads to the generator improving its ability to generate realistic data over time. The breakthrough with GANs was their ability to generate high-quality, realistic data in various domains, including images, audio, and text. GANs have since become a foundational concept in the field of deep learning and have led to numerous applications such as image synthesis, style transfer, super-resolution, data augmentation, and more.

The introduction of Generative Adversarial Networks (GANs) by Goodfellow et al.[6] in 2014 ushered in a novel approach to generative modeling, addressing the challenge of approximating complex, intractable probabilistic computations. GANs consist of two components: a generator (G) that creates data samples, and a discriminator (D) that distinguishes between real data and generated data. The adversarial training process between G and D, relying solely on backpropagation for gradient computation, has become a powerful framework for generative modeling. GANs offer several advantages over traditional generative models, such as the avoidance of Markov chains, minimal inference during learning, and flexibility in modeling various factors and interactions. Moreover,

GANs have demonstrated the ability to produce state-of-the-art log-likelihood estimates and generate realistic samples.

In the context of generative models, unconditional models often lack control over the specific characteristics of the generated data. However, this limitation can be overcome by conditioning the generative model on additional information, allowing for the guided generation of data. Such conditioning can take various forms, including class labels, partial data for inpainting, or even data from different modalities. One significant contribution of this study is the exploration of Conditional Adversarial Nets (conditional GANs or cGANs)[Mirza, 2014][13]. cGANs extend the GAN framework to include conditioning information, which can be class labels, semantic attributes, or any auxiliary data. By incorporating this conditioning variable, cGANs offer the potential for precise control over the generated data, facilitating various applications.

In this study, image classification tasks have been completed by supervised neural networks, notably convolutional networks. However, challenges persist in scaling these models to handle a large number of output categories and in addressing one-to-many mappings. For instance, in image labeling, multiple tags or labels may apply to a single image, and different annotators may use varying terminologies to describe similar concepts. To tackle these issues, researchers have explored multi-modal learning and probabilistic one-to-many mappings, as exemplified by approaches like Deep Boltzmann Machines (DBNs) and multi-modal neural language models.

Multi-modal learning, particularly in image labeling tasks, has leveraged additional information from various modalities to improve classification performance. This approach benefits from the semantically meaningful geometric relations between image features and word representations, enabling better generalization and handling of synonymy. For instance, using natural language corpora to learn vector representations for labels has demonstrated improved performance. Conditional probabilistic generative models provide a solution to the one-to-many mapping problem by taking the input as the conditioning variable and modeling the mapping as a conditional predictive distribution. Researchers have employed this approach to train models on datasets like MIR Flickr 25,000, gener-

ating descriptive sentences for images and facilitating multi-modal learning.

The core of conditional GANs lies in extending the GAN framework to accommodate conditional information. Both the generator and discriminator in cGANs are conditioned on the auxiliary data, allowing for precise control over the generated data. The objective function in cGANs becomes a two-player minimax game, similar to traditional GANs but with conditional probabilities. The study includes experimental results showcasing the potential of cGANs in unimodal and multimodal settings. In the unimodal context, cGANs were trained on the MNIST dataset, conditioned on class labels. Despite being preliminary, the results demonstrated the model's capability to generate images corresponding to specific classes. However, further exploration of hyperparameters and architecture is needed to match or surpass the performance of non-conditional GANs.

In the multimodal scenario, cGANs were applied to automated image tagging using user-generated metadata. By conditioning the model on image features and tags, researchers aimed to generate tags for images. Preliminary results suggested the potential for generating descriptive tags, but more sophisticated models and extensive evaluations are required to validate the effectiveness of this approach.

This approach can be useful in our scenario as we are dealing with two different types of input data, one is the 2D topography images of the biomaterial and another is the corresponding bacteria attachment ratio for two different types of bacteria A and B. These attachment ratios can be used as labels, which are continuous values, so either they can be categorized in fixed ranges for conditioning or we can try to implement Continuous Conditional GANs [cCGAN].

The cGAN paper[Mirza, 2014][13] is quite revolutionary in terms of guiding the training process of GANs which are well-documented to be highly unstable and often lead to mode collapse. But the labels are not always categorical and can be continuous values, to handle such scenarios a new paper "CCGAN: Continuous Conditional Generative Adversarial Networks for Image Generation" was proposed by Ding et al.

In this paper[Ding, 2022][4], the author proposes a novel method for training cGANs to

generate images conditioned on continuous labels. The author calls their method CCGAN, which stands for Continuous Conditional Generative Adversarial Network. CCGAN addresses two fundamental problems that arise when conditioning a cGAN on a continuous label. The first one is, that the number of real images for some regression labels may be very small or even zero, and secondly, regression labels are scalar and infinitely many, so conventional label input methods are not applicable.

To address the first problem, the author proposes two empirical cGAN losses that are robust to the small number of real images. These losses are the hard vicinal discriminator loss (HVDL) and the soft vicinal discriminator loss (SVDL). HVDL penalizes the discriminator for misclassifying images that are close to real images, while SVDL is a smoothed version of HVDL that is less sensitive to outliers.

To address the second problem, the author proposes a novel method for incorporating regression labels into the generator and the discriminator. The author calls this method the improved label input (ILI) mechanism. ILI first maps the regression labels to a latent space using a neural network. Then, the latent space representations of the labels are concatenated with the hidden maps of the generator and the discriminator.

The author evaluated cCGAN on two image generation tasks: steering angle prediction and facial expression synthesis. The results showed that cCGAN outperforms existing cGANs on both tasks. The author believes that cCGAN is a promising new approach for training cGANs to generate images conditioned on continuous labels. It is effective in two image generation tasks, and it is a general method that can be applied to a wide range of continuous conditioning tasks.

In computer vision, CNNs [14] have demonstrated exceptional performance in image classification and object recognition. However, the application of CNNs to unsupervised learning has received relatively less attention.

The paper[Radford, 2016] [16]addresses the challenge of learning reusable feature representations from vast, unlabelled datasets, such as extensive collections of images and videos. The fundamental idea is to harness the immense amount of unlabelled data to acquire in-

termediate image representations that can be applied to diverse supervised learning tasks, particularly image classification. While traditional maximum likelihood techniques have been employed for this purpose, the authors propose an innovative approach centered around Generative Adversarial Networks (GANs) [6]. GANs are known for their capacity to learn complex data distributions without the need for heuristic cost functions, making them attractive candidates for representation learning.

A critical aspect of this research is the introduction of architectural constraints for CNNs, designed to enhance their stability during training. These constraints give rise to the concept of Deep Convolutional Generative Adversarial Networks (DCGANs), which exhibit greater resilience during training, addressing one of the main challenges associated with GANs. The authors systematically evaluate these architectural constraints, demonstrating their effectiveness in training CNNs for unsupervised learning tasks.

One of the primary contributions of this work is the application of the trained discriminators from DCGANs to image classification tasks. The results show that these discriminators can achieve competitive performance compared to other unsupervised learning algorithms, thus highlighting the utility of DCGANs in learning informative image representations.

Moreover, the research provides valuable insights into the internal workings of CNNs. The authors visualize the filters learned by GANs and empirically demonstrate that specific filters have learned to represent distinct objects. This visualization offers a glimpse into the hierarchical representations that these networks acquire, shedding light on their interpretability, which is often a challenge in neural network-based approaches.

Another intriguing aspect explored in this work is the vector arithmetic manipulation of generated samples. The research reveals that DCGANs possess unique vector arithmetic properties, enabling the effortless manipulation of semantic attributes in generated images. This feature holds promise for applications in image generation and editing, offering novel ways to control and modify image content.

In the broader context of unsupervised representation learning, this research aligns with established techniques such as clustering, autoencoders, and deep belief networks. How-

ever, it distinguishes itself by leveraging the power of GANs to learn representations and extends their applicability to a wide array of tasks beyond generative modeling.

Additionally, the work contributes to the field of generative image models by demonstrating the capabilities of DCGANs [16] in generating natural images. This aspect addresses a longstanding challenge in the generation of realistic and high-quality images from learned representations, further expanding the potential applications of DCGANs.

The work on DCGANs represents a significant advancement in the intersection of unsupervised learning and computer vision. Their introduction of architectural constraints, exploration of representation learning, visualization of CNN internals, and novel vector arithmetic manipulation properties open new avenues for research and application development in image generation, representation learning, and computer vision as a whole. This research serves as a cornerstone for future advancements in unsupervised learning with CNNs and their diverse applications in computer vision tasks.

Chapter 3

Dataset

3.1 Data Source

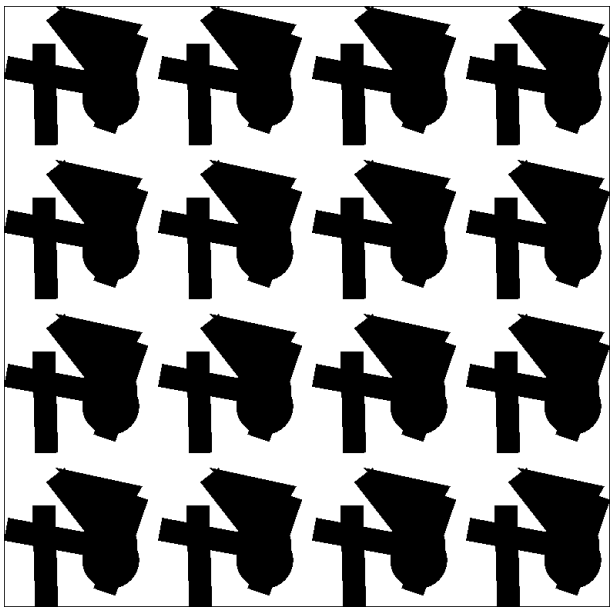

Figure 3.1: Sample Image from Dataset

The paper [4] theorized a relationship between microbe-biomaterial attachment and proposed a functional relationship between the two. The paper used machine learning to generate images based on the proposed relationship by tweaking various physical parameters of the micron surface, such as shape and orientation. That same function was used to generate a set of 2D topography images and their respective bacterial attachment ratios for our research, a total of 2,177 images are provided in the dataset, of which, the

attachment ratios are provided for 2,101 images. The provided attachment ratios are for two distinct bacteria under consideration, namely A and B.

3.2 Data Preparation

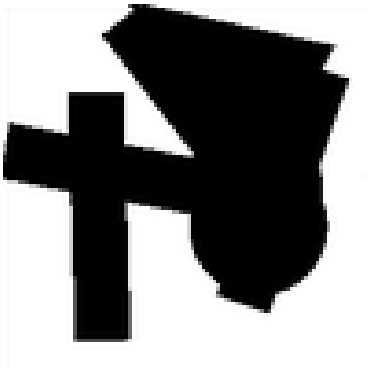

Figure 3.2: Sample Processed Image

Each image is of four by a repeating pattern, and the images are of non-uniform shapes ranging between 300x300 to 1200x1200. Thus, image resizing will be required before any processing. Since each image has a repeating 4x4 pattern, each image has the same pattern 16 times, thus, we can crop the first section of the images to be used for any analysis or modeling. All the images are single-channel grayscale images. The cropped images are resized to a uniform shape of 200x200. For the DCGAN model[], due to the huge number of parameters, the image was reshaped to standard 28x28 to compare with the standard MNIST dataset.

Image files are provided with IDs as their filenames, which correspond to the IDs in the CSV(comma-separated values) files containing attachment ratios of bacteria. Not all image has an attachment ratio as a label, thus we will be dropping those images.

Chapter 4

Generative Adversarial Nets

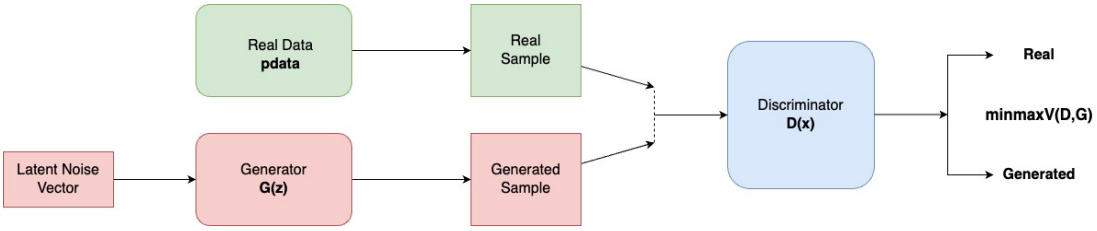

Figure 4.1: GAN(Generative Adversarial Networks) Diagram

Generative Adversarial Networks (GANs) [6] are a novel paradigm in machine learning that frames the process of generative modeling as a game between two distinct entities: a Generator and a Discriminator. These entities, often realized as neural networks compete in a two-player minimax game. To truly grasp the innovation behind GANs, it is essential to delve into the mathematical formulation that underpins them.

The Generator, denoted as G, seeks to transform random noise z into synthetic data samples. It is mathematically represented by the function $G(z;\theta_g)$, where θ_g are its parameters. On the other hand, the Discriminator, denoted as D aims to distinguish between genuine and synthetic data samples. It is represented as $D(x;\theta_d)$, with θ_d being its parameters. The output is a scalar estimating the probability that the sample x originates from the real data distribution.

The Minimax Game

The essence of GANs is captured by the value function ($V(D,G)$) of the minimax game[1]:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{4.1}$$

Here's an elucidation of the components:

1. $E_{x\sim p_{data}(x)}[\log D(x)]$: This segment captures the expected log probability that the Discriminator assigns to real data samples. Ideally, the Discriminator should recognize real samples, translating to this value being maximized.

2. $E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$: Conversely, this segment embodies the expected log-probability that the Discriminator assigns to the samples crafted by the Generator. The Discriminator's goal is to identify these as fake, maximizing this value. The Generator, in its adversarial role, endeavors to minimize this, implying its aim to craft samples that can deceive the Discriminator.

With this foundational understanding of the GAN framework, we will next introduce the concept of Conditional GANs, an advanced variant that offers more directed and controlled data generation capabilities.

4.1 Model Architecture

4.1.1 Model 1 Conditional GAN

Building on the foundational concepts of GANs[6], Conditional GANs (cGANs)[13] introduce an additional layer of complexity and control by conditioning the generation process on external information. This conditioning allows for directed data generation, making cGANs immensely powerful for tasks where specificity in generation is desired. The model presented in this research builds upon the foundational GAN architecture, seamlessly incorporating conditioning variables to drive the data generation process in a directed manner. Our cGAN consists of two primary entities: the Generator and the Discriminator, both augmented with conditioning information.

The Generator, within the cGAN framework, is responsible for generating synthetic data.

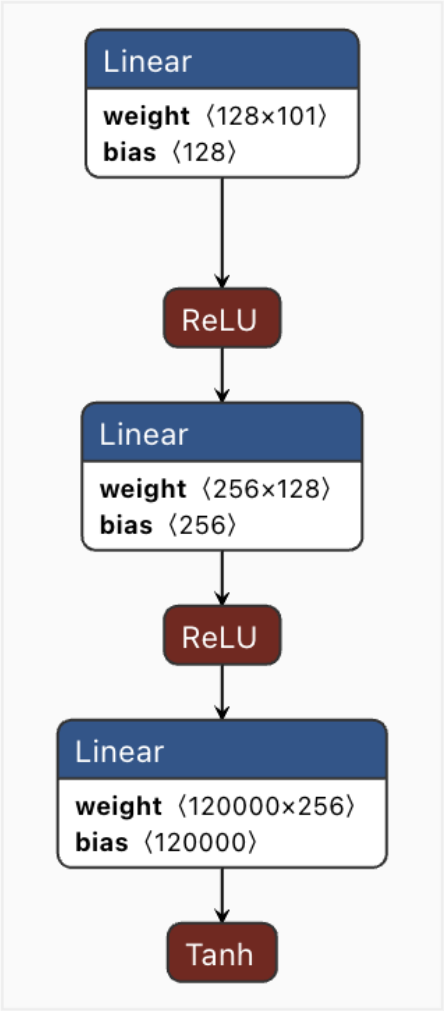

Figure 4.2: cGAN Generator Architecture

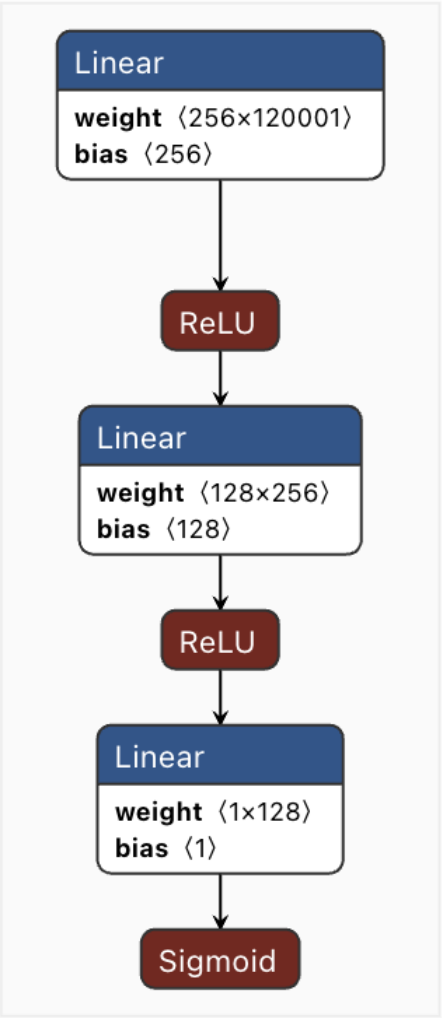

Figure 4.3: cGAN Discriminator Architecture

Its input comprises a random noise vector, z, amalgamated with a conditioning variable y through concatenation, resulting in the composite input $x = [z, y]$. This augmented input traverses a series of transformations, beginning with a linear transformation governed by the equation $h1 = W1x + b1$. The output undergoes the Rectified Linear Unit (ReLU) activation [5], producing $a1 = ReLU(h1)$. This pattern of linear transformation followed by a non-linear activation persists, with the data flowing through another such layer defined by $h2 = W2a1 + b2$ and its subsequent ReLU activation $a2 = ReLU(h2)$. The final transformation layer, characterized by $h3 = W3a2 + b$, culminates in a Tanh activation[5], resulting in the synthetic image $img = tanh(h3)$. The Tanh activation ensures the generated data values lie within a specific range, typically between -1 and 1.

Contrasting the Generator's creative essence, the Discriminator embodies the critical evaluator of the duo. Tasked with discerning the authenticity of data samples, the Discriminator, similar to the Generator, also receives the conditioning variable y. The data sample, termed as img, is concatenated with y to produce $x = [img, y]$. This combined input undergoes a series of transformations, analogous to the Generator but tailored for classification. The initial transformation is captured by $h1 = W1x + b1$, followed by the ReLU activation $a1 = ReLU(h1)$. The processed data then proceeds through another layer, $h2 = W2a1 + b2$, with its ReLU counterpart $a2 = ReLU(h2)$. The Discriminator's final layer, encapsulated by $h3 = W3a2 + b3$, employs a sigmoid activation [5] to produce a scalar output, $validity = (h3)$. This output represents the probability of the input data being genuine.

The above architecture was realized in the PyTorch framework, which leverages its

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=101, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=120000, bias=True)
    (5): Tanh()
  )
)
```

Figure 4.4: Layers of cGAN Generator

```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=120001, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

Figure 4.5: Layers of cGAN Discriminator

nn.Module class is a foundational building block that facilitates the creation of custom neural network architectures.

The **Generator** class is a descendant of the module and encapsulates the functionality

required to produce synthetic data samples. Its constructor initializes a sequential model defined using Sequential. This utility in PyTorch allows for the straightforward chaining of layers and operations in a prescribed order. The Generator begins by accepting an input of dimension, which is the concatenated size of the random noise vector z and the conditioning variable ground_truth. The first layer, nn.Linear(input_dim, 128), linearly transforms this input to a 128-dimensional space. This is followed by a non-linear transformation via the nn.ReLU() activation function. Subsequent layers continue this pattern of linear transformation and activation, ultimately culminating in a nn.Tanh() activation. This final activation ensures that the generated data, lies within the range of [-1, 1], a common normalization practice for neural networks.

The forward propagation method of the Generator, forward(self, z, ground_truth), takes both the noise vector z and the conditioning variable ground_truth as inputs. These inputs are concatenated using PyTorch's concatenation function along the last dimension, forming a single input tensor. This combined tensor is then passed through the sequential model, resulting in the synthetic image.

The **Discriminator** class is also a child of nn.Module, is designed to assess the authenticity of data samples. Its architecture mirrors the Generator in its use of the nn.Sequential utility, tailored for binary classification. The Discriminator's input dimension corresponds to the combined size of the data sample image and the conditioning ground truth. Its initial layer, projects this input into a 256-dimensional space, followed by a ReLU activation. The subsequent layers gradually reduce dimensionality, with the final layer being a single neuron with a Sigmoid activation. This neuron outputs the probability indicating the perceived authenticity of the input data sample.

The Discriminator's forward method functions similarly to that of the Generator. It concatenates the data sample image with the conditioning ground truth and processes this combined tensor through the sequential model. The output, termed validity, represents the probability of the input data sample being real.

4.1.2 Model 2 Deep Convolutional GAN

The architecture proposed in this section employs a Deep Convolutional Generative Adversarial Network (DCGAN)[16], which builds upon the foundational concepts of GANs[6]. The DCGAN design capitalizes on the spatial hierarchies present in image data, harnessing convolutional layers to effectively capture and replicate intricate patterns found in real images. In this section, we delve into the architectural details of the model, encompassing both the Generator and Discriminator components. In our research, we adopted PyTorch, a prominent deep learning framework, to implement the DCGAN architecture. PyTorch offers a dynamic computation graph, which is especially suitable for complex architectures like GANs, where forward and backward passes are interwoven with training logic.

Generator Architecture

The Generator's primary objective is to transform an input of random noise into a coherent image that mirrors the characteristics of our dataset. This transformational journey, from a latent space to a tangible image, is facilitated through a series of transposed convolutional layers[14]. Initiating the process, the Generator first maps the 100-dimensional random noise to a tensor with 128 channels, through the convolutional transpose operation. Sequentially, this tensor undergoes transformations, reducing channel size while upscaling spatial dimensions. Specifically, a second layer reduces the channel count to 64, followed by a third layer that further compresses it to 32 channels. Ultimately, the tensor is transformed into a single-channel output, representing the generated image. Each transition between layers is accompanied by Batch Normalization[18], ensuring stable and accelerated training. This step is pivotal to stabilize the activations and facilitate faster and more stable training. By normalizing the activations to have zero mean and unit variance, Batch Normalization mitigates internal covariate shift[18], a phenomenon where the distribution of inputs changes during training, potentially hampering convergence. Furthermore, the ReLU activation[5] function introduces non-linearity, apart from the final layer which employs the Tanh activation function to constrain output values between [-1, 1].

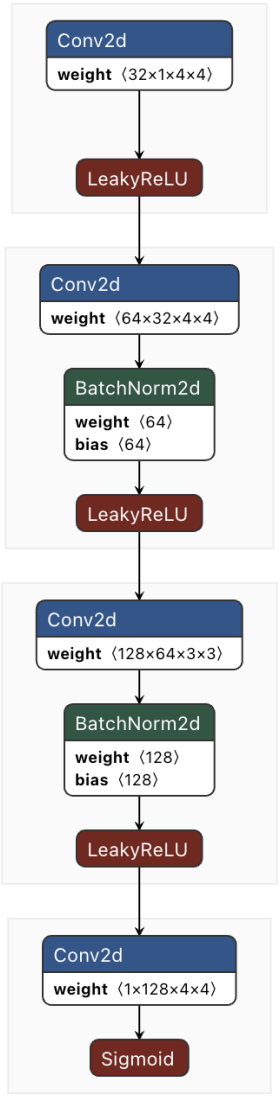

Figure 4.6: DCGAN Discriminator Architecture

The Generator, on the other hand, is implemented as a Python class inheriting from, the base class for all neural network modules in PyTorch. Within the Generator, we define a method that serves a similar purpose to the Discriminator's helper function but focuses on transposed convolutions. The method uses, facilitating the upscaling of its input tensor, essentially achieving the reverse of what the Discriminator's convolution layers do. Similar to the Discriminator, we incorporate batch normalization for stabilization, and the activation functions are either the hyperbolic tangent (tanh) for the final layer or the rectified linear unit (ReLU) for intermediate layers.

The forward method of the Generator class ensures that any input is passed through the defined layers, producing the final generated image as the output.

Discriminator Architecture

```
Generator(
  (layers): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(100, 128, kernel_size=(4, 4), stride=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (3): Sequential(
      (0): ConvTranspose2d(32, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): Tanh()
    )
  )
)
```

Figure 4.7: DCGAN Generator Layers

```
Sequential(
  (0): Sequential(
    (0): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (1): Sequential(
    (0): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (3): Sequential(
    (0): Conv2d(128, 1, kernel_size=(4, 4), stride=(2, 2), bias=False)
    (1): Sigmoid()
  )
)
```

Figure 4.8: DCGAN Discriminator Layers

Acting as the critical evaluator, the Discriminator discerns the authenticity of images, distinguishing between original images from the dataset and the synthesized counterparts generated by the Generator. This binary classification is facilitated through a series of convolutional layers that progressively downsample the input while amplifying its depth. The initial layer, taking input as a single-channel image, outputs a tensor with 32 channels. This depth is subsequently increased to 64 in the next layer and further boosted to 128 channels in the third convolutional layer. These transformations refine the spatial context and also enrich the feature representation of the input. The culmination

of this process is the final layer that outputs a single-channel probability score, indicating the likelihood of the input image being genuine. Throughout the Discriminator, the LeakyReLU [5] activation function ensures the propagation of gradients, especially for lower values. However, the final layer adopts the Sigmoid activation function, rendering a probability score that provides a definitive output on the image's authenticity.
The Discriminator is designed using a helper function, which efficiently creates a sequence of layers. This function takes in parameters such as the number of input channels, output channels, kernel size, stride, padding, and a few others to customize the behavior of the layer. At its core, the function utilizes, a 2D convolutional layer provided by PyTorch, which serves to extract features from the input image and downsample it. To ensure stability during training and to accelerate convergence, batch normalization is conditionally appended to the layer sequence based on the parameter. The activation function used is either a sigmoid (for the final layer) or a LeakyReLU with a negative slope of 0.2, which introduces non-linearity and helps in propagating gradients even for negative values.

4.1.3 Model 3 Variational Autoencoder

The Variational Autoencoder (VAE) [15] is a class of generative models that are based on the principle of probability and variational inference to learn a latent representation of data. In the context of our dataset, which focuses on biomaterial topography image analysis, the VAE offers distinct advantages for capturing the underlying latent features of the images. This section delves into the specifics of the VAE architecture proposed for analyzing these biomaterial topography images and serves as a foundation for comparing its performance and characteristics with Generative Adversarial Networks (GANs).
The VAE model is a type of autoencoder[3] designed to learn probabilistic mappings between the input data and a latent space. It operates by encoding the input data into a lower-dimensional latent space and then decoding it back to its original form. What sets the VAE apart from traditional autoencoders is its ability to learn a probabilistic distribution for the latent variables, allowing for more robust representations and sample

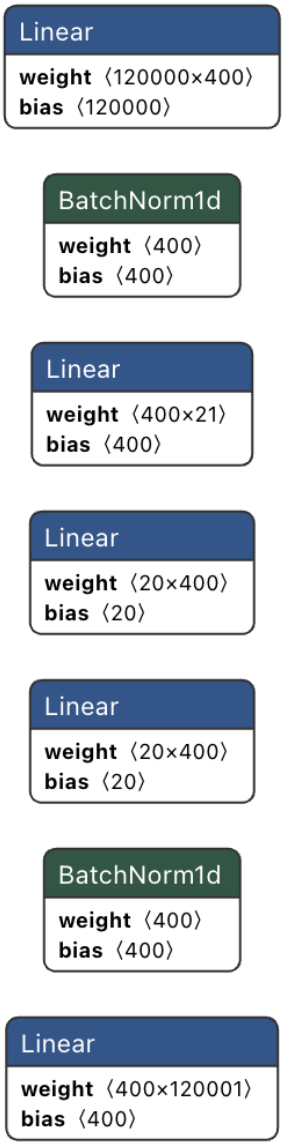

Figure 4.9: VAE Model Architecture

generation.

The key challenge in VAEs is to compute or approximate the posterior distribution $p\theta(zx)$. Direct computation is intractable due to the denominator term, which involves an integral over the entire latent space. To tackle this, VAEs turn to variational inference.

Variational inference optimizes a surrogate objective called the Evidence Lower BOund (ELBO)[15]. For any choice of approximate posterior $q\phi(zx)$ the ELBO is given by:

$$ELBO(\phi, \theta; x) = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{4.2}$$

The first term encourages the reconstructed data (via the decoder) to be close to the original data, acting like a traditional reconstruction loss. The second term, the Kullback-Leibler (KL)[19] divergence, regularizes the learned approximate posterior to be close to the prior, ensuring a smooth and meaningful latent space.

In contrast with GANs, VAEs operate in a probabilistic framework, ensuring that the

```
VAE(
  (fc1): Linear(in_features=120001, out_features=400, bias=True)
  (bn1): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc21): Linear(in_features=400, out_features=20, bias=True)
  (fc22): Linear(in_features=400, out_features=20, bias=True)
  (fc3): Linear(in_features=21, out_features=400, bias=True)
  (bn2): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc4): Linear(in_features=400, out_features=120000, bias=True)
)
```

Figure 4.10: VAE Model Layers

latent space has a smooth and continuous structure. This can be advantageous when the focus is on understanding the underlying data distribution or when interpolation between data points is desired. GANs, on the other hand, excel at producing sharp, high-quality images by setting up a game-theoretic framework between a generator and a discriminator. However, GANs can sometimes produce unrealistic samples due to mode collapse and may not always guarantee a smooth latent space.

The model definition is as follows:

1. **Input Compression**:

$$h1 = ReLU(BatchNorm1(fc1(x)))$$

2. **Latent Variable Calculation**:

$$\mu = fc21(h1)$$

$$\log(var) = fc22(h1)$$

3. **Reparameterization**:

$$\sigma = \exp(0.5 \times \log(var))$$

$$z = \mu + \sigma \odot \epsilon$$

4. **Decoding the Latent Variable**:

$$h3 = ReLU(BatchNorm2(fc3(z)))$$

5. **Output Reconstruction**:

$$x' = sigmoid(fc4(h3))$$

The provided Variational Autoencoder (VAE) is designed using the PyTorch framework and offers an elegant way to process and generate images. At its core, the VAE takes in an input image and an additional piece of information, representing ground truth. Starting with the encoder, the input undergoes a compression where it's transformed into a more manageable size. This compressed form then branches out into two pathways. One pathway discerns the average structure or the "mean" of the data, and the other identifies how spread out or varied this data is. These two aspects together give us a sense of the underlying probability distribution of our input data in a space known as the "latent space." Directly using this latent space for further computations can be challenging. So, the VAE employs a method known as the "reparameterization trick"[10]. Instead of directly using the computed latent space, it introduces a bit of randomness, ensuring the entire process remains smooth and differentiable, which is crucial for the learning phase.

Once we have this reparameterized latent space, the decoder steps in. It takes this latent representation and starts decoding it, aiming to reconstruct the original input image. As it reconstructs, the decoder ensures the output values are suitable for image data, lying between 0 and 1.

This Variational Autoencoder (VAE) model serves as a reference for our Generative Adversarial Network (GAN) models. It provides a standard against which the training

dynamics and performance metrics of the GANs will be gauged, ensuring a comparative evaluation of their effectiveness and efficiency. Using the VAE as a reference point enables a more informed assessment of the GANs, highlighting their relative strengths and areas for potential improvement.

4.2 Model Training

4.2.1 Model 1 Conditional GAN

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 128]          13,056
              ReLU-2                  [-1, 128]               0
            Linear-3                  [-1, 256]          33,024
              ReLU-4                  [-1, 256]               0
            Linear-5               [-1, 120000]      30,840,000
              Tanh-6               [-1, 120000]               0
         Generator-7               [-1, 120000]               0
================================================================
Total params: 30,886,080
Trainable params: 30,886,080
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 2.75
Params size (MB): 117.82
Estimated Total Size (MB): 120.57
----------------------------------------------------------------
```

Figure 4.11: cGAN Generator Model Parameters

The model training for cGAN[13] was performed by first combining the multimodal data into a single source of data by creating a custom data loader class in PyTorch. The primary role of this class is to retrieve and process images based on their identifiers from the loaded data. Each image, once fetched, is converted to the 3-channel format and then subjected to a series of transformations, which include resizing to a unified dimension of 200x200 and normalization. Such transformations are essential to ensure that the data fed into the network is consistent in size and scale. Normalization, in particular, is crucial for neural networks as it ensures that the range of input features is similar, leading to more

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 256]      30,720,512
              ReLU-2                 [-1, 256]               0
            Linear-3                 [-1, 128]          32,896
              ReLU-4                 [-1, 128]               0
            Linear-5                   [-1, 1]             129
           Sigmoid-6                   [-1, 1]               0
     Discriminator-7                   [-1, 1]               0
================================================================
Total params: 30,753,537
Trainable params: 30,753,537
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.46
Forward/backward pass size (MB): 0.01
Params size (MB): 117.32
Estimated Total Size (MB): 117.78
----------------------------------------------------------------
```

Figure 4.12: cGAN Discriminator Model Parameters

stable and faster convergence during training. The DataLoader utility not only batches and shuffles the data but also ensures its efficient retrieval, minimizing any potential I/O bottlenecks during the training process. Batching is essential for leveraging the parallel processing capabilities of modern hardware while shuffling the data helps in breaking any inherent order in the data, promoting better model generalization.

Once the data preparation for the model is complete, the focus shifts to the training of the Conditional Generative Adversarial Network. Key hyperparameters, such as the number of epochs, batch size, learning rate, and latent dimension size, are set to guide the training process. The Generator and Discriminator networks are instantiated with their respective architectures. The Binary Cross-Entropy loss is chosen for the Discriminator whose primary role in a GAN is to classify whether a given sample is from the real dataset or is generated by the Generator. This task is inherently a binary classification problem: "real" or "fake". The Binary Cross-Entropy loss [12] is specifically designed for such binary classification tasks, i.e. discerning real from generated samples. The BCE loss provides a measure of the difference between the true labels and the predicted probabilities. When the prediction aligns closely with the true label, the BCE loss is low, and when the prediction is far from the true label, the BCE loss is high. This provides a gradient that helps the Discriminator improve its classification accuracy over time For optimiz-

ing the network weights, the Adam optimizer [9] is chosen, given its proven track record in handling non-convex optimization problems commonly encountered in deep learning. Adam (short for Adaptive Moment Estimation) is an optimization algorithm that computes adaptive learning rates for each parameter. In the context of deep learning, and especially GANs, the optimization landscape can be riddled with saddle points, local minima, and non-convex regions. In GANs, the training of the Generator and Discriminator is very delicate. If one vastly outperforms the other, training can become unstable. Adam's adaptive nature can often bring a semblance of stability to this process [9].

As the training process continues, for each batch of data, the Generator generates synthetic images. These images are produced using a combination of random noise and the associated labels from the batch. The Discriminator's role is then to evaluate these generated images alongside the real images from the batch. Its objective is dual: correctly identify real images and label the synthetic images as generated. On the other hand, the Generator aims to enhance its synthesis capabilities to produce images that even the Discriminator perceives as real. The training process also includes logging loss values and FID(Frechet Inception Distance) scores [7] of both Generator and Discriminator. This later on helps evaluate the performance and training of the GAN model.

4.2.2 Model 2 Deep Convolutional GAN

Following the detailed discussion on our DCGAN's architecture [16], the subsequent step is the actual model training. Before the training, it's important to ensure that our data is correctly sourced and preprocessed. We utilize the PyTorch framework, which provides efficient tools for data handling. The dataset comprised of images, recognizing the importance of uniformity for neural networks, all images are preprocessed to maintain a consistent size of 28x28 pixels. For the DCGAN model 28x28 input shape was selected to reduce the number of trainable parameters, as due to multiple convolutional layers, the sheer size of a number of parameters, made to hard to train a model even on a fast

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 128, 4, 4]         204,800
       BatchNorm2d-2            [-1, 128, 4, 4]             256
              ReLU-3            [-1, 128, 4, 4]               0
   ConvTranspose2d-4             [-1, 64, 7, 7]          73,728
       BatchNorm2d-5             [-1, 64, 7, 7]             128
              ReLU-6             [-1, 64, 7, 7]               0
   ConvTranspose2d-7            [-1, 32, 14, 14]         32,768
       BatchNorm2d-8            [-1, 32, 14, 14]             64
              ReLU-9            [-1, 32, 14, 14]              0
  ConvTranspose2d-10             [-1, 1, 28, 28]            512
             Tanh-11             [-1, 1, 28, 28]              0
================================================================
Total params: 312,256
Trainable params: 312,256
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.27
Params size (MB): 1.19
Estimated Total Size (MB): 1.47
----------------------------------------------------------------
```

Figure 4.13: DCGAN Generator Model Parameters

GPU-accelerated machine. This particular shape was used as it is a standard shape for gauging deep learning models in computer vision, it is based on the MNIST dataset, which is mostly used for performance evaluation of any deep learning convolutional model. Such consistency aids in the network's ability to generalize across the dataset.

To facilitate the data input during training, we employ a data loader. This mechanism batches the dataset into smaller chunks, making it manageable for the model to process and learn from in each iteration. The transformation pipeline applied to the dataset ensures grayscale conversion, tensor conversion, and normalization [18]. Grayscale conversion is essential to maintain channel consistency, given that the DCGAN is designed for single-channel images. Normalization, by scaling pixel values to a range of [-1, 1], ensures that the neural network's weights and biases adjust at a consistent rate during training, aiding in faster convergence.

With data preparation in place, our focus shifts to the main protagonists: the Discriminator and the Generator. Both models are initialized, ensuring their weights are set with normal initialization. This weight initialization strategy is crucial because it prevents any

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             512
         LeakyReLU-2          [-1, 32, 32, 32]               0
            Conv2d-3          [-1, 64, 16, 16]          32,768
       BatchNorm2d-4          [-1, 64, 16, 16]             128
         LeakyReLU-5          [-1, 64, 16, 16]               0
            Conv2d-6           [-1, 128, 8, 8]          73,728
       BatchNorm2d-7           [-1, 128, 8, 8]             256
         LeakyReLU-8           [-1, 128, 8, 8]               0
            Conv2d-9             [-1, 1, 3, 3]           2,048
          Sigmoid-10             [-1, 1, 3, 3]               0
================================================================
Total params: 109,440
Trainable params: 109,440
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.02
Forward/backward pass size (MB): 1.06
Params size (MB): 0.42
Estimated Total Size (MB): 1.50
----------------------------------------------------------------
```

Figure 4.14: DCGAN Discriminator Model Parameters

layer's weights from either vanishing or exploding during the training iterations, ensuring a stable training process.

In the training procedure of the Discriminator and the Generator. The Discriminator takes the stage first. For each batch of images, it's tasked with distinguishing between genuine images from the dataset and synthesized ones from the Generator. The learning process involves adjusting its weights based on the error of its predictions. The training loss for the Discriminator is a combination of its performance on real images and its performance on the synthesized images. Subsequently, the Generator, its objective is to create images that can deceive the Discriminator. Thus, its training process involves generating images from random noise and then evaluating the Discriminator's decision on these images. The Generator then adjusts its weights to improve the quality of its next set of images. During training, the optimization algorithm, Adam [9] in this case, incrementally adjusts the model weights to minimize the respective loss functions. Specific hyperparameters, like learning rates and betas, are set to ensure a balance between training speed and stability. The iterative process of training the Discriminator and Gen-

erator continues for a predetermined number of epochs. With each epoch, the Generator becomes more adept at producing realistic images, while the Discriminator enhances its ability to distinguish between real and synthetic images.

4.2.3 Model 3 Variational Autoencoder

Following the architecture design of our Variational Autoencoder (VAE) [15], the model training process is implemented to learn the intrinsic patterns and relationships from the data.

The primary dataset originates from a CSV file, which consists of ground truth data corresponding to images. The images, stored separately, are matched with this dataset using their unique IDs. Once the ground truth data and the images are consolidated into a unified dataset, a custom dataset handler is designed. This handler is pivotal for seamlessly fetching and processing data during training. By leveraging PyTorch's Dataset class, the custom dataset handler is equipped to resize images to a consistent 200×200 resolution, convert them into tensors, and normalize them. Normalization ensures that the pixel values of the images are scaled to a standard range, which aids in stabilizing the training dynamics. Given the nature of neural network training, it's essential to supply data in manageable batches. This is where the concept of dataloading comes into play. As discussed earlier in the section DataLoader utility in PyTorch facilitates this by segmenting the dataset into smaller batches, ensuring that each batch is efficiently loaded into the memory during training iterations. It also offers the advantage of shuffling the data, promoting more generalized learning.

A crucial aspect of training VAEs is the loss function. The chosen loss function here amalgamates the binary cross-entropy loss, which measures the difference between the original and the reconstructed images, and the Kullback-Leibler divergence [10], which quantifies the difference between the learned latent variable distribution and a standard normal distribution, since, VAEs aim to not just reconstruct input data but also to learn a probabilistic mapping of data into a latent space. This latent space is assumed to be

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 400]      48,000,800
       BatchNorm1d-2                  [-1, 400]             800
            Linear-3                   [-1, 20]           8,020
            Linear-4                   [-1, 20]           8,020
            Linear-5                  [-1, 400]           8,800
       BatchNorm1d-6                  [-1, 400]             800
            Linear-7                [-1, 120000]      48,120,000
================================================================
Total params: 96,147,240
Trainable params: 96,147,240
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.46
Forward/backward pass size (MB): 0.93
Params size (MB): 366.77
Estimated Total Size (MB): 368.16
----------------------------------------------------------------
```

Figure 4.15: VAE Model Parameters

governed by some known distribution, usually a standard normal distribution. The idea is that by enforcing this distribution on the latent space, the model can generate new samples by simply sampling from this known distribution and decoding them. The KL divergence measures how one probability distribution is different from a second, reference probability distribution. In the context of VAEs, the KL divergence quantifies how much the learned latent variable distribution deviates from the desired (standard normal) distribution. By incorporating the KL divergence into the loss function, the VAE is penalized if the latent variable distribution strays too far from the standard normal distribution. This combined loss ensures that the VAE not only reconstructs images accurately but also learns a meaningful latent space. For an objective evaluation of the model's performance, the Frechet Inception Distance (FID) [7] is computed between real and generated images during training. The FID serves as a metric to gauge the similarity between the distributions of real and generated images, with a lower FID indicating better model performance. To calculate the FID, activations from an intermediate layer of a pre-trained Inception model are extracted for both real and generated images. These activations capture the high-level features of the images, and their distributions are then compared to compute the FID. In each epoch, the VAE is trained on the data, following which a set of images

is generated. The FID score between these generated images and a batch of real images is then calculated and monitored.

4.3 Model Evaluation

The common method of evaluating the Generative Model for images is Visual Inspection. By simply comparing the generated image with a realistic image, a decision can be made. However, in our case, the images do not really make sense to the human eye, as they are just patterns in a 2D image. There must be evaluation metrics that can provide a more numerical approach to gauging the model performance. We use 3 different methods of evaluating the performance of our generative models.

1. Visual Inspection

As discussed earlier, the first and the most rudimentary approach to gauging a model's performance is just by looking at the generated sample and deciding how similar it looks to a real image. However, this is not a very empirical approach, and cannot be justified in a paper. Nonetheless, this is an important approach to solving the problem statement of any generative task.

2 Frechet Inception Distance (FID)

Frechet Inception Distance (FID) is a metric used to assess the quality of images generated by generative models, including Generative Adversarial Networks (GANs) and Variational AutoEncoders (VAEs). It offers a measure of similarity between the distribution of generated images and the distribution of real images. Unlike some other metrics, which might only assess image quality based on pixel values or other low-level features, FID takes into account higher-level features and structures in the images.

The FID metric involves using a pre-trained Inception network [20] (typically used for image classification tasks) to extract features from an intermediate layer. Both the real and generated images are passed through this network, and their respective features are extracted. Once the features are extracted, the mean and covariance of these features are computed for both sets of images. The FID is then computed as the Frechet distance

between these two multivariate Gaussian distributions (defined by the means and covariances).

Specifically, if $m1$, $m2$ are the means and $C1$, $C2$ are the covariances of the real and generated distributions, respectively, the FID is given by:

$$FID = \|m_1 - m_2\|^2 + Tr\left(C_1 + C_2 - 2C_1C_2 * 0.5\right) \tag{4.3}$$

Here, Tr stands for the trace of a matrix. A lower FID score indicates that the two sets of images (real and generated) are more similar, implying that the generative model is producing higher-quality images that more closely resemble the real data. Conversely, a higher FID score suggests a greater disparity between the real and generated images.

3.Loss Plot

In a generative model, the loss does not necessarily provide any performance metrics,

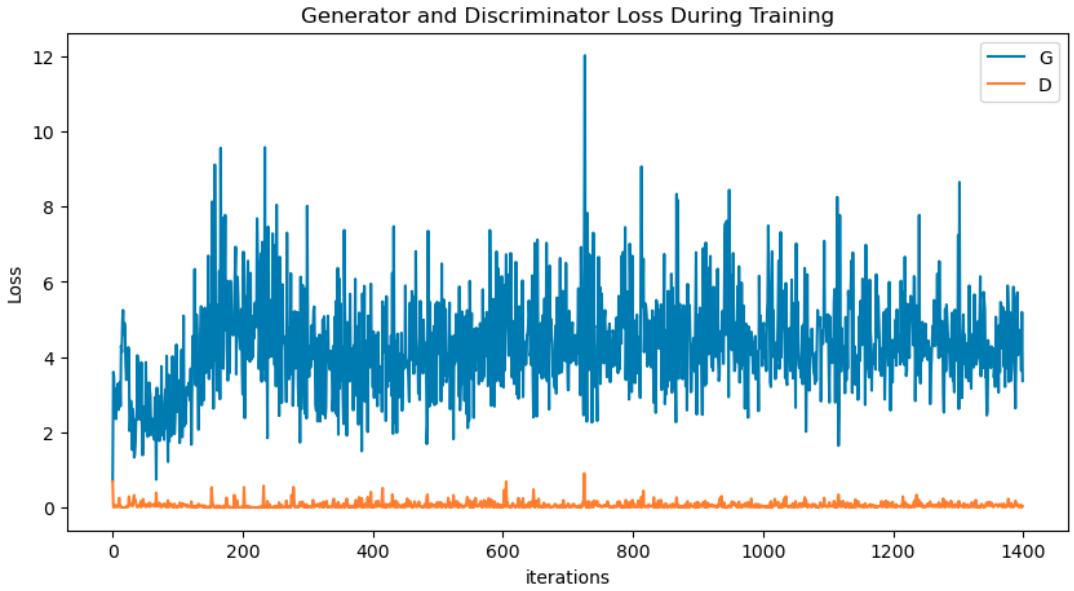

Figure 4.16: cGAN Generator Discriminator Loss over Epochs

but it does provide model training stability and coherence. Generative models especially GANs have a very delicate balance between the discriminator and generator. And there are a lot of common issues a GAN can face during training:

3.a Mode Collapse [21]: This is a situation where the generator produces a limited variety of outputs, i.e., it always generates the same or very similar samples. This happens

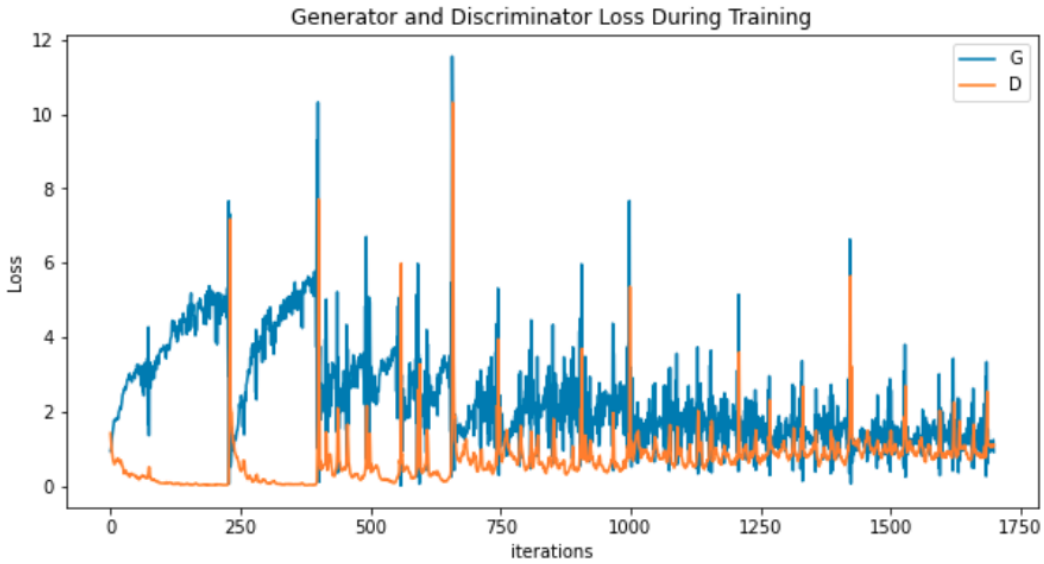

Figure 4.17: DCGAN Generator Discriminator Loss over Epochs

when the generator discovers a particular sample that the discriminator thinks is real, and the generator starts producing only that sample.

3.b Vanishing Gradients [21]: Especially in deep GAN architectures, the discriminator can become so good that the generator gradient vanishes and it cannot learn anymore.

3.c Oscillations [11]: The generator and discriminator losses can oscillate without the network reaching a stable equilibrium.
The generator loss vs. discriminator loss can be very useful in identifying such issues during and after training, giving us a good idea of where the model is failing.

Chapter 5

Results and Conclusion

5.1 Results

Model	FID Score	No. of Epochs
cGAN	1379.80	1000
DCGAN	717.67	1000
VAE	1448.61	1000

Table 5.1: Comparison of models by FID Score and No. of Epochs

In our exploration of the effectiveness of generative models in designing biomaterial surface topographies, we employed three distinct models: cGAN, DCGAN, and VAE. The subsequent sections detail their performance and comparative efficacy.

5.1.1 Analysis of the Generative Models

The cGAN model, after rigorous training, yielded an FID score of 1379.80. This score, while respectable, indicated room for improvement in generating precise topography images. On the other hand, the DCGAN model performed relatively well with a FID score of 717.67, reflecting its superior capability in capturing intricate patterns and relationships within the data. The VAE model, which we employed as our reference model, lagged behind with an FID score of 1448.61. However, its importance transcended mere performance metrics, as discussed later.

A comparative assessment of the three models underscored the DCGAN's prowess in

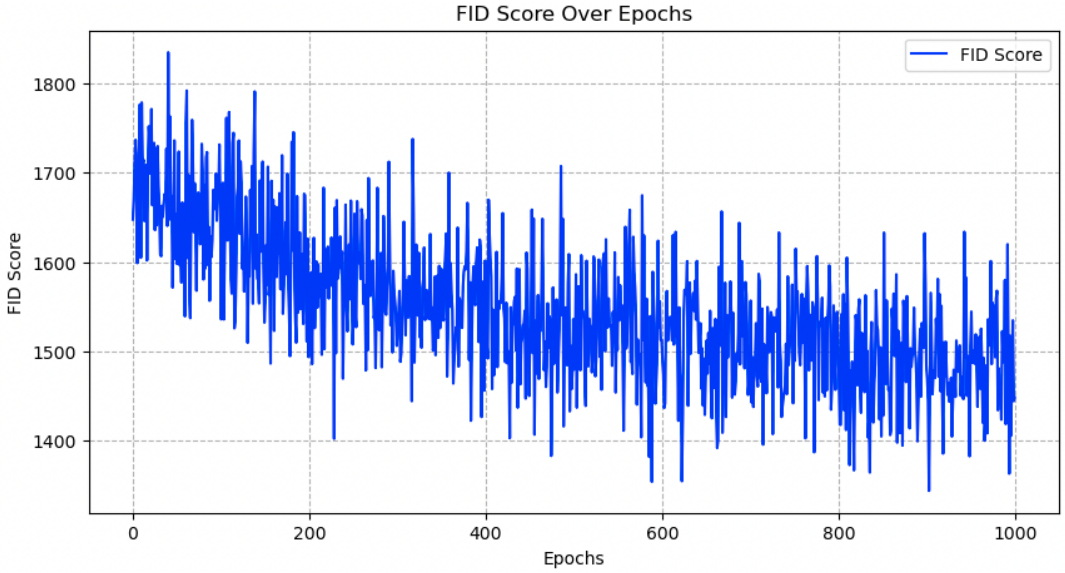

Figure 5.1: VAE FID Score over Epochs

generating high-fidelity topography images. While the cGAN demonstrated promise, certain network oscillations during its training hinted at potential areas of optimization and the use of alternate loss functions such Wasserstein loss function [2] which offers more stability to the training. The VAE, with its higher FID score, did not compete with the GANs in terms of performance. Yet, it offered invaluable insights, serving as a benchmark and providing context to the capabilities and potential of the GAN models.

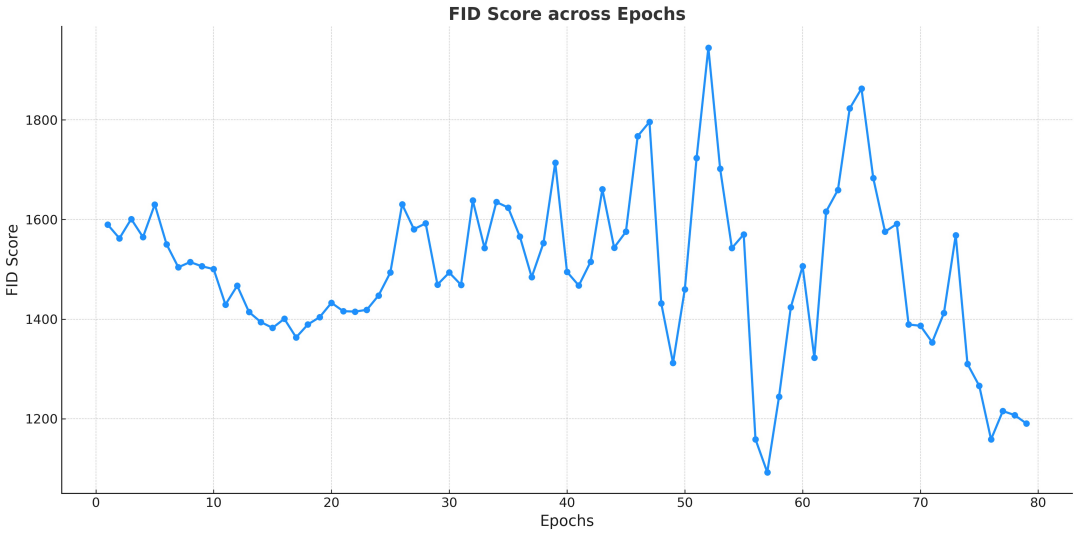

Figure 5.2: FID Score of DCGAN over Epochs

5.2 Conclusion

Based on our study's primary objectives, the aim was to harness the power of generative models for biomaterial surface design. Our findings, encapsulated in the results, are both enlightening and promising.

The DCGAN model emerged as a front-runner in our experiments, demonstrating the vast potential of deep convolutional networks in understanding and generating intricate surface topographies. Its superior FID score stands testament to its prowess. While the cGAN lagged slightly behind, its performance was commendable, suggesting that with further optimization, it too could reach similar heights.

Interestingly, the VAE, despite its weaker performance, played a pivotal role. Serving as a reference model, it provided a baseline against which we could gauge the advancements and capabilities of the GAN models. Its relatively higher FID score underlined the advancements that GAN architectures bring to the table, especially in tasks as complex as biomaterial surface design.

However, our study wasn't without challenges. The cGAN, for instance, displayed oscillatory behavior during training, possibly due to an imbalance between its generator and discriminator. The DCGAN, while powerful, required extensive computational resources, posing a barrier for extended epochs or more granular training. The VAE, beyond its role as a reference, was limited in capturing the depth of features, evident from its higher FID score.

Our experiment provides a proof of study for the use of GANs for the controlled generation of new biomaterial topographies that can help expedite the expensive and time-consuming process of developing new designs in laboratories. This study, once refined for its concept and performance, can help solve similar problems in other fields where research and development take a lot of time and cost.

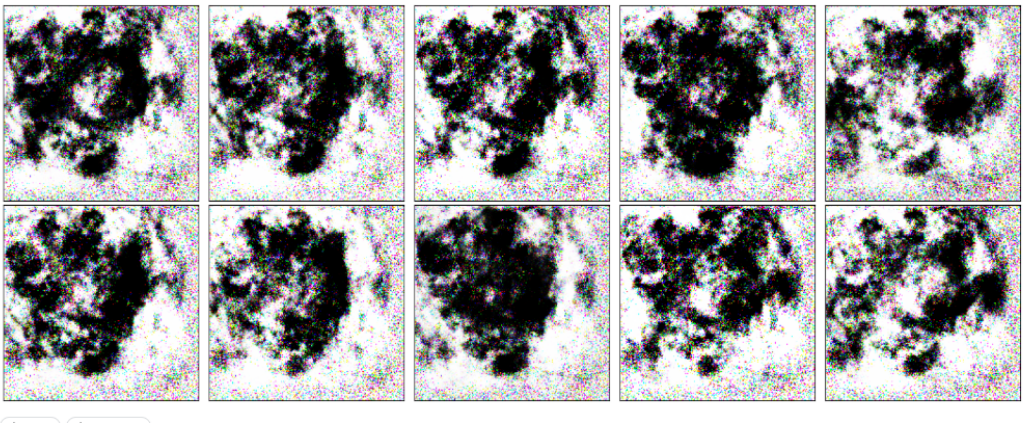

Figure 5.3: cGAN Output at 1000 Epochs of Training

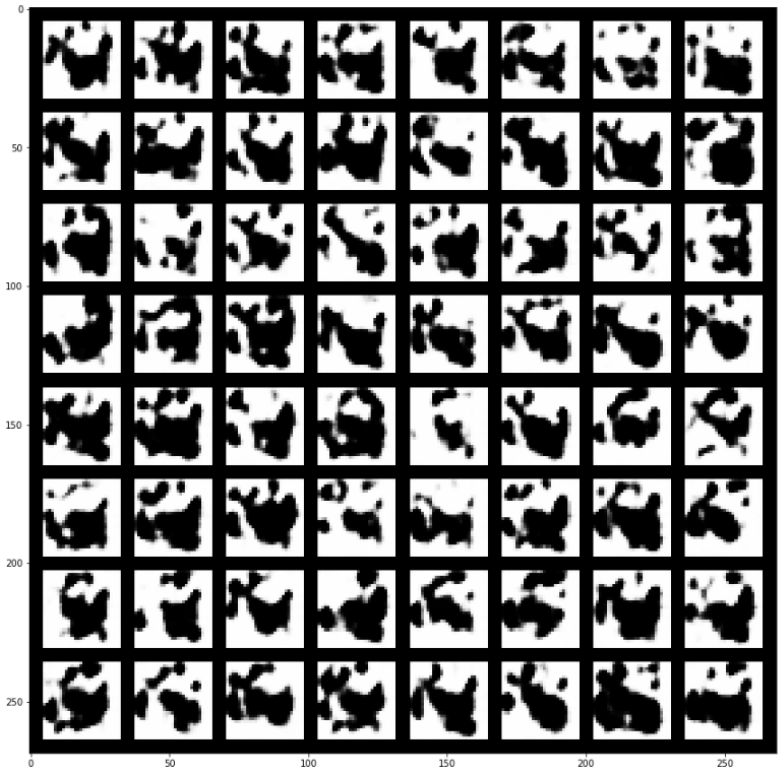

Figure 5.4: DCGAN Output at 1000 Epochs of Training

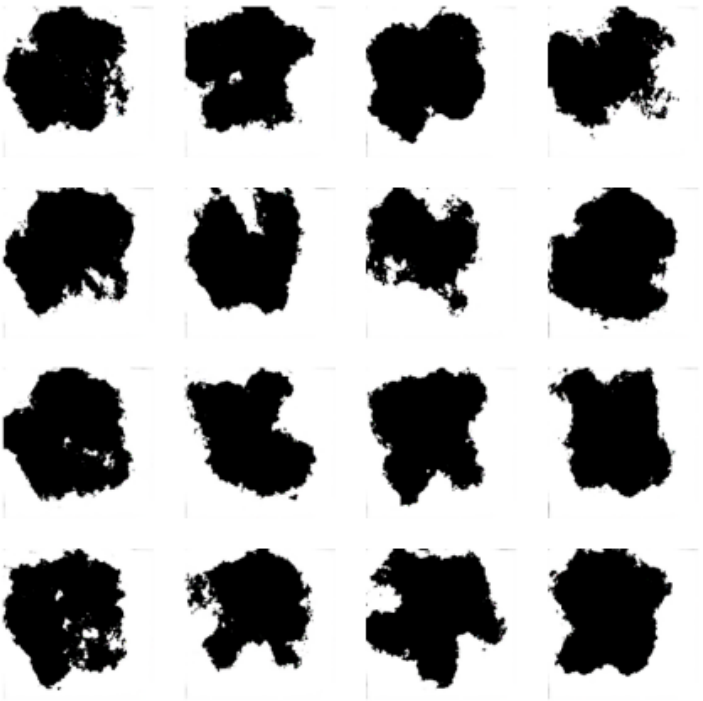

Figure 5.5: VAE Output at 1000 Epochs of Training

Chapter 6

Limitations and Future Work

The main drawback of the study is unable to conclude the quality of the image generated, in terms of functionality, i.e. if the generated topography was realized to a physical form, would it perform the way it was generated to perform? There is a requirement for an adaptive feedback loop in the development process of GAN-based generative modeling of real-world applications, which can generate more data from the testing of previously generated models, which in turn can be used to further train the model in a supervised manner.

The proposed models, though generated discernable topography images, lacked the sharpness of the real input data, which could have been the result of the relatively low depth of the neural networks implemented in the models. This can be solved by implementing deeper models and tuning and testing the model with different sets of layers and filters. GANs in general are known to be very delicate and very hard to train to reach stability. Mode Collapse is also one of the biggest issues while training a GAN, which occurs when a GAN fails to capture the full diversity of the target data distribution and instead produces only a limited set of similar or repeated samples. This was faced during the training of the DCGAN model. Another issue with GANs is training instability, where the model sometimes exhibits oscillations or instability in the loss function, which can be mitigated by using smoothening methods on labels, or by using alternative loss functions such as Wasserstein loss or hinge loss [2].

One of the influential limitations faced in our study was the lack of data points, the avail-

able dataset only has 2100 trainable images [17] and corresponding labels. The GANs are known to work better if provided with a large amount of data. GANs trained on a small dataset may struggle to generalize to new, unseen data. The generator may produce unrealistic or poor-quality samples when faced with data points outside the training set distribution.

A proposed future study can be hybridizing multiple models that are specific to their task such as utilizing the probabilistic latent learning of VAE to guide the training of GAN. Different architectures can be tested, since the functionality of convolutional layers is very hard to understand, a more researched architecture may perform better. Another aspect of further study will be implementing pre-trained models, and fine-tuning them for our applications, Pre-trained models with weights are known to be very accurate at understanding structures and require a very small number of parameters to be fine-tuned.

Bibliography

[1] , B. S., R, V., and R, s. *Analysis of Minimax Algorithm Using Tic-Tac-Toe*. 11 2020.

[2] Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan, 2017.

[3] Bank, D., Koenigstein, N., and Giryes, R. Autoencoders, 2021.

[4] Ding, X., Wang, Y., Xu, Z., Welch, W. J., and Wang, Z. J. Continuous conditional generative adversarial networks: Novel empirical losses and label input mechanisms, 2022.

[5] Dubey, S. R., Singh, S. K., and Chaudhuri, B. B. Activation functions in deep learning: A comprehensive survey and benchmark, 2022.

[6] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014.

[7] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[8] Khalid, S., Gao, A., Wang, G., Chu, P. K., and Wang, H. Tuning surface topographies on biomaterials to control bacterial infection. *Biomater. Sci. 8* (2020), 6840–6857.

[9] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization, 2017.

[10] Kingma, D. P., and Welling, M. Auto-encoding variational bayes, 2022.

[11] Liang, K. J., Li, C., Wang, G., and Carin, L. Generative adversarial network training is a continual learning problem, 2018.

[12] Mao, A., Mohri, M., and Zhong, Y. Cross-entropy loss functions: Theoretical analysis and applications, 2023.

[13] Mirza, M., and Osindero, S. Conditional generative adversarial nets, 2014.

[14] O'Shea, K., and Nash, R. An introduction to convolutional neural networks, 2015.

[15] Panguluri, K., and Kamarajugadda, K. Image generation using variational autoencoders. *IJITEE (International Journal of Information Technology and Electrical Engineering) 9* (03 2020).

[16] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[17] Saad, M. M., O'Reilly, R., and Rehmani, M. H. A survey on training challenges in generative adversarial networks for biomedical image analysis, 2023.

[18] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization?, 2019.

[19] Shao, H., Yao, S., Sun, D., Zhang, A., Liu, S., Liu, D., Wang, J., and Abdelzaher, T. ControlVAE: Controllable variational autoencoder. In *Proceedings of the 37th International Conference on Machine Learning* (13–18 Jul 2020), H. D. III and A. Singh, Eds., vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 8655–8664.

[20] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions, 2014.

[21] Thanh-Tung, H., and Tran, T. On catastrophic forgetting and mode collapse in generative adversarial networks, 2020.

[22] Vassey, M., Ma, L., Kämmerling, L., Mbadugha, C., Trindade, G., Figueredo, G., Pappalardo, F., Hutchinson, J., Markus, R., Rajani, S., Hu, Q., Winkler, D., Irvine, D., Hague, R., Ghaemmaghami, A., Wildman, R., and Alexander, M. Innate immune cell instruction using micron-scale 3d objects of varied architecture and polymer chemistry: The chemoarchichip. *Matter 6* (03 2023).

[23] Vassey, M. J., Figueredo, G. P., Scurr, D. J., Vasilevich, A. S., Vermeulen, S., Carlier, A., Luckett, J., Beijer, N. R. M., Williams, P., Winkler, D. A., de Boer, J., Ghaemmaghami, A. M., and Alexander, M. R. Immune modulation by design: Using topography to control human monocyte attachment and macrophage differentiation. *Advanced Science 7*, 11 (2020), 1903392.

[24] Xu, L.-C., and Siedlecki, C. A. Chapter 2 - bacterial cell–biomaterials interactions. In *Handbook of Biomaterials Biocompatibility*, M. Mozafari, Ed., Woodhead Publishing Series in Biomaterials. Woodhead Publishing, 2020, pp. 11–42.