Università degli Studi di Padova

Master's Degree Thesis

Deep learning based methods for CMB analysis

Supervisor:

Author:
Lorenzo Pellizzari

Prof. Michele Liguori
Cosupervisor:

Andrea Ravenni

Università degli Studi di Padova
Master's Degree in Astrophysics and Cosmology
Academic Year: 2023/2024

Deep learning based methods for CMB analysis

Lorenzo Pellizzari

Università degli Studi di Padova

Master's Degree in Astrophysics and Cosmology

Academic Year: 2023/2024

September 15, 2024

Abstract

This thesis explores the application of Generative Adversarial Networks (GANs) in generating realistic cosmological maps, providing insight into their training dynamics, optimization challenges, and the statistical evaluation of the generated images. We present novel contributions through our custom GAN architecture, deep integration of coding procedures, and focus on evaluating the quality of our generator. The results show the promise of GANs in cosmology, while highlighting the areas requiring further investigation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Cosmology

1.1 Overview of Cosmology

Cosmology is the scientific study of the large-scale properties of the universe as a whole. It encompasses the origin, evolution, and eventual fate of the universe, grounded in the laws of physics and observations of cosmic phenomena. The modern cosmological model is based on the theory of General Relativity, which describes gravity as the curvature of spacetime caused by mass and energy.

At its core, cosmology seeks to answer fundamental questions: How did the universe begin? How does it evolve over time? What are its large-scale structures, and what will happen in the distant future? Observations, such as the cosmic microwave background (CMB) and the distribution of galaxies, help us understand these questions. The current framework that best explains these observations is known as the standard cosmological model, or the Lambda Cold Dark Matter (ΛCDM) model.

1.2 The Friedmann Universe

The Friedmann universe is a model that describes the expansion of the universe under the assumption that it is homogeneous and isotropic on large scales. This model is derived from Einstein's field equations of General Relativity and provides the foundation for the modern cosmological model [1].

The dynamics of the Friedmann universe are governed by the *Friedmann equations*, which describe the relationship between the expansion rate of the universe (the Hubble parameter $H(t)$) and the energy content of the universe, such as matter, radiation, and dark energy. The first Friedmann equation is given by:

$$H^2(t) = \frac{8\pi G}{3}\rho(t) - \frac{k}{a^2(t)} + \frac{\Lambda}{3} \tag{1.1}$$

where:

- $H(t)$ is the Hubble parameter, which measures the rate of expansion at time t.
- G is the gravitational constant.
- $\rho(t)$ is the energy density of the universe at time t, which includes contributions from matter, radiation, and dark energy.

- k is the curvature parameter, which determines the geometry of the universe ($k = 0$ for a flat universe, $k > 0$ for a closed universe, and $k < 0$ for an open universe).

- $a(t)$ is the scale factor, which describes how distances in the universe change with time.

- Λ is the cosmological constant, which represents dark energy.

The second Friedmann equation, which describes the acceleration of the universe's expansion, is:

$$\frac{\ddot{a}(t)}{a(t)} = -\frac{4\pi G}{3}\left(\rho(t) + 3p(t)\right) + \frac{\Lambda}{3} \tag{1.2}$$

where $p(t)$ is the pressure associated with the energy density. These equations can be used to describe the expansion history of the universe and predict its future evolution based on the relative contributions of matter, radiation, and dark energy.

The Friedmann universe plays a crucial role in our understanding of the early universe, particularly during the era of inflation and the formation of large-scale structures like galaxies and galaxy clusters. It provides a theoretical framework for the standard cosmological model [2].

1.3 The Standard Cosmological Model

The Standard Cosmological Model, also known as the ΛCDM model, is the prevailing theory that describes the structure and evolution of the universe. It is based on the principles of General Relativity and is supported by a wealth of observational evidence, including the cosmic microwave background (CMB), large-scale structure surveys, and supernova data.

The model assumes that the universe is homogeneous and isotropic on large scales, which means that, on average, it looks the same in all directions and at all locations. The primary components of the universe according to the ΛCDM model are:

- **Dark Energy (Λ):** This mysterious form of energy makes up about 70% of the universe's total energy density. It is thought to be responsible for the accelerated expansion of the universe, and is represented by the cosmological constant Λ in the Friedmann equations.

- **Cold Dark Matter (CDM):** Cold Dark Matter comprises approximately 25% of the universe's energy density. Unlike ordinary (baryonic) matter, dark matter does not interact with electromagnetic radiation (hence, it is "dark"), but it plays a critical role in the formation of cosmic structures by providing the gravitational scaffolding for galaxies and galaxy clusters.

- **Baryonic Matter:** The matter that makes up stars, planets, and all visible structures constitutes only about 5% of the universe's energy content. This ordinary matter is composed of protons, neutrons, and electrons.

- **Radiation:** Although now only a small fraction of the total energy density, radiation (including photons and neutrinos) dominated the energy budget of the early universe and played a crucial role in its evolution.

The ΛCDM model successfully explains several key observations:

- **Cosmic Microwave Background (CMB):** The CMB is the remnant radiation from the Big Bang and provides a snapshot of the universe when it was only about 380,000 years old. The small fluctuations in the CMB temperature are imprints of the initial density perturbations that later grew into galaxies and clusters of galaxies.
- **Large-Scale Structure:** The distribution of galaxies and clusters on large scales is consistent with the predictions of the ΛCDM model, which describes how dark matter and baryons evolved from the initial density fluctuations observed in the CMB [3].
- **Expansion of the Universe:** Observations of distant supernovae have shown that the universe's expansion is accelerating, a phenomenon attributed to the presence of dark energy. This acceleration is well described by the cosmological constant Λ.

One of the central features of the standard cosmological model is its ability to describe the universe's history from the Big Bang to the present day, as well as make predictions about its future. The evolution of the universe is governed by the Friedmann equations, and the expansion rate is determined by the balance between dark energy, dark matter, and radiation.

The model is not without its challenges. The nature of dark energy and dark matter remains one of the biggest unsolved mysteries in cosmology. However, despite these uncertainties, the ΛCDM model provides an accurate description of the universe's large-scale behavior and has become the foundation of modern cosmology [4].

1.4 Lensing and Lensing Maps

Gravitational lensing is a powerful tool in cosmology for studying the distribution of matter in the universe. It occurs when massive objects, such as galaxies or clusters of galaxies, bend the path of light from background objects due to the curvature of spacetime, as predicted by General Relativity. This effect can distort, magnify, or create multiple images of background objects [5].

There are two primary types of gravitational lensing:

- **Strong Lensing:** This occurs when the gravitational field of a foreground object is so strong that it produces multiple images or significant distortion of the background object. Strong lensing typically occurs in cases where the lens and source are closely aligned, often producing striking features like Einstein rings or arcs.
- **Weak Lensing:** In weak lensing, the effect is more subtle, causing slight distortions and shearing of the shapes of background galaxies. Although individual weak lensing signals are small, statistical analysis of large samples of galaxies can reveal information about the large-scale structure of the universe and the distribution of dark matter.

1.4.1 Weak Lensing in Cosmology

Weak lensing is particularly valuable for cosmology because it provides a direct way to map the distribution of dark matter. Since dark matter does not emit or absorb light, gravitational lensing is one of the few methods available to study its distribution. By measuring the distortions in the shapes of distant galaxies, scientists can infer the total matter distribution, including both visible and dark matter.

The weak lensing signal is typically characterized by two quantities:

- **Convergence (κ):** The convergence measures the projected surface density of matter along the line of sight and describes the magnification of background objects.
- **Shear (γ):** The shear represents the distortion in the shape of background objects due to the tidal gravitational field of the foreground mass distribution.

The weak lensing effect on the observed shapes of background galaxies is described by the lensing equation, which relates the observed position of a galaxy to its true position, taking into account the gravitational deflection by the lens.

$$\vec{\theta}_{\text{obs}} = \vec{\theta}_{\text{true}} + \vec{\alpha}(\vec{\theta}), \tag{1.3}$$

where $\vec{\alpha}(\vec{\theta})$ is the deflection angle due to the gravitational lensing effect of the intervening mass.

1.4.2 Lensing Maps

Lensing maps are constructed by observing the distortions in the shapes of large numbers of background galaxies over wide areas of the sky. These maps provide a detailed picture of the distribution of matter in the universe, including dark matter. By analyzing the convergence and shear fields, cosmologists can study the growth of structure over cosmic time, test models of dark energy, and constrain parameters of the standard cosmological model.

Weak lensing surveys, such as those conducted by the Dark Energy Survey (DES) and the Hyper Suprime-Cam (HSC) survey, have produced high-resolution lensing maps that cover large fractions of the sky. These maps are critical for understanding the evolution of the large-scale structure of the universe and for testing theories of gravity on cosmological scales.

Weak lensing also provides a way to study the relationship between galaxies and dark matter through galaxy-galaxy lensing, where the lens is a galaxy or a cluster, and the background sources are distant galaxies. By studying the distortions around galaxies, researchers can measure the mass distribution around galaxies and their dark matter halos.

Lensing maps are crucial in cosmology because they allow us to probe the invisible dark matter and test theoretical models of the universe's evolution. With increasing data from upcoming surveys like the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST) and the Euclid mission, the quality and resolution of lensing maps will continue to improve, providing even more precise insights into the structure of the universe [6].

1.5 Sunyaev-Zel'dovich (SZ) Effect and SZ Effect Maps

The Sunyaev-Zel'dovich (SZ) effect is a powerful observational tool in cosmology, providing insights into the large-scale distribution of galaxy clusters and the cosmic web. The SZ effect occurs when cosmic microwave background (CMB) photons pass through a hot, ionized gas, typically found in galaxy clusters, and are scattered by high-energy electrons. This interaction changes the energy distribution of the CMB photons, which results in a distortion of the CMB spectrum. The SZ effect is particularly useful because it allows us to detect and study galaxy clusters, the largest gravitationally bound structures in the universe [7].

1.5.1 The Thermal SZ Effect

The thermal SZ (tSZ) effect arises when CMB photons are scattered by hot electrons in galaxy clusters via inverse Compton scattering, transferring energy from the electrons to the photons. This effect causes a distortion in the CMB spectrum, which can be observed as a decrement in intensity at lower frequencies (e.g., in the microwave regime) and an increment at higher frequencies.

The tSZ effect is described by the following equation:

$$\frac{\Delta T_{\mathrm{SZ}}}{T_{\mathrm{CMB}}} = y \cdot f(x), \tag{1.4}$$

where:

- ΔT_{SZ} is the change in temperature of the CMB due to the SZ effect.
- T_{CMB} is the temperature of the CMB (about 2.725 K).
- y is the Compton y-parameter, which quantifies the strength of the SZ effect and depends on the integrated pressure of the electrons along the line of sight.
- $f(x)$ is a frequency-dependent function, where $x = h\nu/(k_B T_{\mathrm{CMB}})$, h is Planck's constant, ν is the observing frequency, and k_B is Boltzmann's constant.

The Compton y-parameter is given by:

$$y = \int \frac{k_B T_e}{m_e c^2} n_e \sigma_T dl, \tag{1.5}$$

where:

- T_e is the electron temperature.
- n_e is the electron number density.
- σ_T is the Thomson scattering cross-section.
- m_e is the electron mass.
- c is the speed of light.
- The integral is taken along the line of sight (dl).

The tSZ effect provides a nearly redshift-independent method for detecting galaxy clusters, as the SZ signal does not diminish with distance. This makes it an invaluable tool for identifying distant and faint clusters that are otherwise challenging to detect using optical or X-ray surveys.

1.5.2 The Kinetic SZ Effect

In addition to the thermal SZ effect, there is also a kinetic SZ (kSZ) effect. The kSZ effect is caused by the bulk motion of galaxy clusters relative to the CMB rest frame. When clusters move toward or away from the observer, the CMB photons are Doppler shifted, leading to a slight temperature shift in the CMB. Unlike the tSZ effect, the kSZ effect does not depend on the electron temperature but rather on the peculiar velocity of the galaxy cluster.

The change in temperature due to the kSZ effect is given by:

$$\frac{\Delta T_{\mathrm{kSZ}}}{T_{\mathrm{CMB}}} = -\int \frac{v_r}{c} n_e \sigma_T dl, \tag{1.6}$$

where v_r is the radial peculiar velocity of the cluster along the line of sight. Although the kSZ effect is much weaker than the tSZ effect, it provides valuable information about the peculiar velocities of galaxy clusters and the overall large-scale velocity field in the universe.

1.5.3 SZ Effect Maps

SZ effect maps are constructed by observing the distortions in the CMB caused by the tSZ and kSZ effects. These maps allow cosmologists to study the distribution of galaxy clusters across large areas of the sky and to probe the intracluster medium (ICM) within these clusters. The SZ effect provides an independent way of measuring the cluster mass and gas content, complementing other methods such as X-ray observations.

The tSZ effect is particularly sensitive to the integrated pressure of the ICM, which is directly related to the total mass of the galaxy cluster. By constructing SZ effect maps, cosmologists can estimate the masses of galaxy clusters, which are key tracers of the large-scale structure of the universe. These maps are also used to constrain cosmological parameters such as the amplitude of matter fluctuations (σ_8) and the mean density of matter (Ω_m).

1.5.4 The Role of SZ Effect Maps in Modern Cosmology

SZ effect maps are a crucial observational tool for modern cosmology. They allow for the detection and characterization of galaxy clusters over a wide range of redshifts, providing an important dataset for understanding the growth of structure in the universe. Moreover, SZ effect maps complement other cosmological probes such as gravitational lensing, X-ray observations, and CMB measurements.

In combination with machine learning techniques, SZ effect maps can be used to rapidly identify and classify galaxy clusters in large-scale surveys. Machine learning algorithms can analyze the complex patterns in SZ effect maps and predict properties such as cluster mass, redshift, and temperature, making them an efficient tool for cosmological studies.

As future surveys like the Simons Observatory and CMB-S4 come online, the resolution and sensitivity of SZ effect maps will continue to improve, enabling even more precise measurements of the properties of galaxy clusters and the large-scale structure of the universe.

1.6 New Techniques

Machine learning (ML) has emerged as a powerful tool in many scientific disciplines, and cosmology is no exception [8]. Cosmology, with its massive datasets from large-scale surveys and complex models, can benefit greatly from the efficiency and predictive power of ML algorithms. In this section, we discuss the motivations for using machine learning in cosmology and the specific problems it can address.

1.6.1 Dealing with Large Datasets

Modern cosmological surveys, such as the Sloan Digital Sky Survey (SDSS), the Dark Energy Survey (DES), and the upcoming Euclid and Vera C. Rubin Observatory (LSST), collect vast amounts of data, including images of millions of galaxies and clusters. Analyzing these large datasets using traditional methods can be computationally expensive and time-consuming. Machine learning provides a solution by automating the analysis of these datasets, identifying patterns, and extracting useful information at unprecedented speed and scale.

For instance, supervised learning models can be trained to classify objects in large sky surveys, distinguishing between galaxies, stars, and other celestial bodies. Deep learning, in particular, is well-suited for image-based tasks like galaxy morphology classification, where convolutional neural networks (CNNs) can efficiently recognize complex structures in images.

1.6.2 Accelerating Simulations

Cosmological simulations, such as N-body simulations and hydrodynamical simulations, play a critical role in understanding the evolution of the universe and the formation of large-scale structures like galaxies and galaxy clusters. However, these simulations are computationally intensive, requiring significant time and resources to run at high resolution.

Machine learning can help accelerate this process. By training generative models, such as Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs), on precomputed simulation data, it is possible to generate high-fidelity cosmological maps and simulations in a fraction of the time. These ML-based emulators can approximate the output of complex simulations without the need for full-scale computations, allowing researchers to explore a wider range of cosmological models and parameters.

For example, GANs have been used to generate weak lensing convergence maps and Sunyaev-Zel'dovich (SZ) effect maps, as discussed in previous sections. These generative models can produce high-resolution maps that match the statistical properties of the original simulations, providing a valuable tool for data augmentation and rapid exploration of cosmological scenarios.

1.6.3 Parameter Estimation and Inference

One of the key goals of cosmology is to constrain the fundamental parameters of the universe, such as the density of dark matter (Ω_m), the equation of state of dark energy (w), and the amplitude of matter fluctuations (σ_8). Traditionally, parameter estimation is done by comparing observational data with theoretical models through a process called likelihood-based inference. This method, while accurate, can be computationally expensive, particularly when exploring a large parameter space.

Machine learning offers a promising alternative for parameter estimation. Neural networks, for instance, can be trained to map observational data directly to cosmological parameters, bypassing the need for explicit likelihood calculations. Once trained, these models can perform rapid parameter estimation, providing a fast and efficient way to analyze large datasets.

In addition, ML-based methods, such as Bayesian neural networks or Gaussian processes, can incorporate uncertainty estimates into their predictions, allowing for robust parameter inference in the presence of noisy or incomplete data.

1.6.4 Improving Data Quality and Noise Reduction

Observational cosmology is often plagued by noise and other artifacts that can obscure or distort the underlying signal. For example, weak lensing surveys are susceptible to shape noise, which arises from the intrinsic ellipticity of galaxies and observational limitations. Similarly, cosmic microwave background (CMB) measurements can be affected by foreground contamination from galactic dust and synchrotron radiation.

Machine learning algorithms can be employed to mitigate these issues by improving the quality of observational data. Autoencoders, a type of unsupervised learning model, can be used to denoise cosmological images, removing noise while preserving important features. CNNs can also be trained to identify and remove foreground contaminants from CMB maps, enhancing the accuracy of cosmological measurements.

In weak lensing studies, machine learning can help reconstruct the underlying mass distribution more accurately by accounting for noise and observational biases. This leads to more precise measurements of the matter power spectrum, which is crucial for understanding dark matter and dark energy.

1.6.5 Why Machine Learning for Lensing and SZ Effect Maps?

The generation of weak lensing maps and SZ effect maps is computationally intensive, especially when dealing with high-resolution simulations over large sky areas. Machine learning, and in particular GANs, can drastically reduce the computational cost of generating these maps. Once trained, GANs can produce realistic lensing or SZ maps in seconds, compared to hours or even days for traditional simulations.

Moreover, machine learning models can learn to replicate the statistical properties of these maps, including their power spectra and higher-order statistics. This allows researchers to explore different cosmological models and predict the effects of varying parameters, such as the dark matter density or the amount of baryonic feedback, on lensing and SZ observations.

In summary, machine learning enables faster data generation, improved accuracy in parameter estimation, and the ability to uncover new insights from cosmological data, making it an indispensable tool for the future of cosmological research.

Chapter 2

Introduction to Machine Learning and Generative Adversarial Networks

2.1 Introduction to Machine Learning

Machine learning (ML) is a subfield of artificial intelligence (AI) that enables computers to learn from data and make decisions or predictions without being explicitly programmed for specific tasks. Instead of relying on hard-coded rules, machine learning algorithms build models based on patterns detected in datasets, allowing them to generalize their understanding and apply it to unseen data.

2.1.1 Key Concepts in Machine Learning

Features and Labels

A fundamental concept in ML is the relationship between *features* (input variables) and *labels* (target outcomes). Features represent the attributes of the data, while labels are the outcomes we aim to predict. For instance, in a weather prediction model, features might include temperature, humidity, and pressure, while the label could be whether it will rain or not.

Training and Generalization

The goal of machine learning is to develop a model that can learn from a *training set* and generalize well to *unseen* or *test data*. Generalization refers to the model's ability to apply learned patterns to new, unseen instances effectively. A key challenge in ML is balancing the trade-off between **underfitting**, where the model fails to capture the underlying structure in the data, and **overfitting**, where it learns the training data too closely, capturing noise as well as signal.

Supervised Learning

Supervised learning involves training a model on labeled data, where the input-output pairs are explicitly known. The model aims to map inputs to the correct output by minimizing a predefined *loss function*, which quantifies the difference between predicted and actual outcomes. Common examples include regression (predicting continuous values) and classification (assigning discrete labels).

Example of Label Training: The MNIST Dataset

A classic example of supervised learning is training a model to recognize handwritten digits using the MNIST dataset. The MNIST dataset consists of 70,000 grayscale images of handwritten digits, each of size 28×28 pixels. Each image is paired with a *label* that indicates the correct digit (0–9) represented in the image.

The task for the model is to learn the mapping from the image (features) to the correct digit (label). During training, the model sees many examples of handwritten digits with their corresponding labels and adjusts its weights using an optimization algorithm like gradient descent. Over time, the model becomes better at predicting the correct digit for new, unseen images.

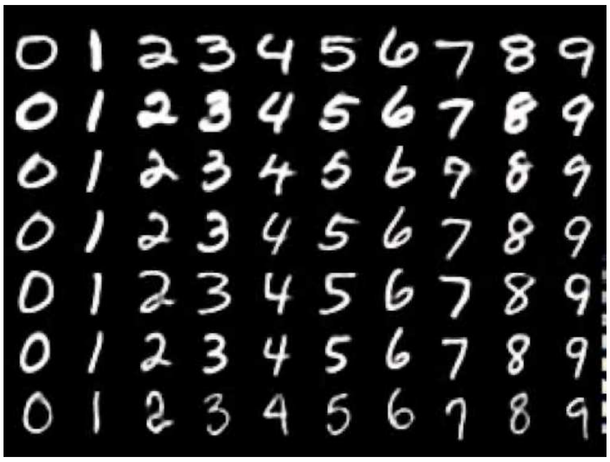


Figure 2.1: Example of handwritten digits from the MNIST dataset.The model learns to associate each image with its corresponding label (digit).

The MNIST dataset is widely used in machine learning research because it is simple yet provides valuable insights into the effectiveness of various learning algorithms. The same principle of label training applies to more complex datasets, including the cosmological maps used in this work.

Linear Models and Gradient Descent

A simple linear model assumes a linear relationship between the input features and the output labels. Mathematically, this can be represented as:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \tag{2.1}$$

where y is the predicted output, $x_1, x_2, \ldots, x_n$ are the input features, $w_1, w_2, \ldots, w_n$ are the weights, and b is the bias term.

The goal is to find the weights $w_1, w_2, \ldots, w_n$ and the bias b that minimize the difference between the predicted output y and the true output y_{true}. This difference is quantified by a loss function, typically the mean squared error (MSE):

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^{m} (y_i - y_{\text{true},i})^2 \tag{2.2}$$

To minimize the MSE, we use an optimization technique called *gradient descent*. Gradient descent iteratively adjusts the weights and bias in the direction that reduces the loss function:

$$w_j := w_j - \alpha \frac{\partial \text{MSE}}{\partial w_j} \tag{2.3}$$

$$b := b - \alpha \frac{\partial \text{MSE}}{\partial b} \tag{2.4}$$

where α is the learning rate, controlling the size of the steps taken during the optimization.

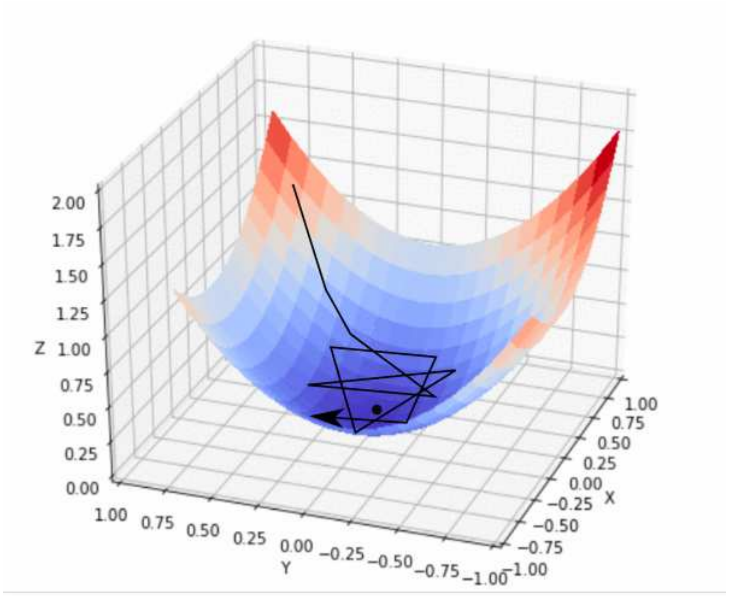


Figure 2.2: Illustration of gradient descent optimization.

In the context of the simple model described earlier, gradient descent was used to find the optimal weights and bias that best predicted the production of apples and oranges based on the features.

Unsupervised Learning

In unsupervised learning, the model is not provided with labeled outputs. Instead, it tries to learn the inherent structure of the input data. A common use case is clustering, where the algorithm groups data points based on similarities, such as in *k-means clustering*.

2.1.2 Applications

Machine learning is particularly powerful in scenarios where manual feature engineering and decision rules become impractical due to the complexity or volume of data. By leveraging statistical methods and computational power, ML algorithms automate pattern recognition tasks, enabling applications like fraud detection, image recognition, natural language processing, and, importantly for this work, generating simulations for scientific research.

2.2 Deep Learning and Neural Networks

2.2.1 Introduction to Deep Learning

Deep learning (DL) is a subset of machine learning that utilizes neural networks with multiple layers to model complex, high-dimensional data. These networks can automatically learn feature representations, making them well-suited for tasks like image classification, speech recognition, and data generation [9].

2.2.2 The Structure of Neural Networks

A neural network consists of layers of interconnected nodes, or *neurons*, that process data through weighted connections. The layers between the input and output, called *hidden layers*, allow the network to capture hierarchical patterns in data. Each layer applies a transformation, typically a combination of a linear transformation and a non-linear activation function, such as ReLU or Sigmoid.

Training Neural Networks

Neural networks are trained using *backpropagation*, an algorithm that computes the gradient of the loss function with respect to each weight in the network. The optimizer, commonly *stochastic gradient descent* (SGD), updates the weights in the direction that minimizes the loss. The learning rate, a key hyperparameter, controls the step size for these updates.

2.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are specialized neural networks designed for processing grid-like data, such as images. They employ convolutional layers that apply filters across the input data, detecting edges, shapes, and textures. Pooling layers reduce the spatial dimensions, making the network more computationally efficient and less sensitive to small translations in the input.

CNNs are a critical component of this thesis, as they form the backbone of the GAN architecture used to generate cosmological maps. Both the generator and discriminator rely on convolutional and transposed convolutional layers to create and evaluate realistic cosmological structures.

2.3 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow in 2014 [10] [11], consist of two networks—a generator (G) and a discriminator (D)—competing in a minimax game. The generator aims to create realistic data samples, while the discriminator tries to distinguish between real and generated samples.

The Generator

The generator takes a random noise vector as input and transforms it into data that mimics the real dataset. It employs deconvolutional layers to upscale the noise into a structured output, like an image.

The Discriminator

The discriminator, a CNN, evaluates the input data and classifies it as either real or generated. Its goal is to become better at detecting the fake data generated by the generator.

Training Process

GANs are trained by alternating between updating the generator and the discriminator. The discriminator's objective is to maximize its accuracy in distinguishing between real and fake data, while the generator's goal is to minimize the discriminator's ability to correctly identify its generated samples as fake. This interaction can be formalized as a **minimax problem**:

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

where:

- $D(x)$ represents the probability assigned by the discriminator to the real image x,
- $G(z)$ is the generator's output based on random noise z,
- The generator G tries to **minimize** the probability that the discriminator correctly identifies generated images as fake, while the discriminator D tries to **maximize** the probability of correctly classifying both real and fake images.

In this setup, the generator and discriminator are locked in a **zero-sum game**: the generator attempts to fool the discriminator by reducing its loss, and the discriminator works to maximize its ability to correctly distinguish between real and fake images. Training continues until a balance, often referred to as a **Nash equilibrium**, is achieved, where the discriminator can no longer easily distinguish between real and fake images, and the generator produces realistic-looking images.

The primary challenge in training GANs lies in maintaining a balance between the two networks. Instability can arise if either the generator or discriminator significantly outperforms the other [12].

2.4 Why GANs for Cosmology?

GANs are well-suited for generating high-dimensional, structured data such as cosmological maps. Traditional methods for simulating cosmological data are computationally expensive and time-consuming, making GANs an attractive alternative for efficiently generating realistic simulations that capture the statistical properties of the universe.

Chapter 3

Analysis of the Existing Work and Dataset

3.1 Referral paper

In 2019, Mustafa et al. presented an innovative application of Generative Adversarial Networks (GANs) in cosmology [13], focusing specifically on weak gravitational lensing maps. Their work aimed to address the computational challenges posed by traditional simulations in cosmology. By leveraging the power of deep generative models, they were able to significantly reduce the computational cost of producing high-resolution simulations while maintaining statistical fidelity to the underlying physical processes.

Weak gravitational lensing refers to the deflection of light from distant galaxies by massive cosmic structures, such as dark matter. This bending of light allows cosmologists to create convergence maps that depict the distribution of mass in the universe. Typically, these maps are generated using N-body simulations, which are computationally intensive. Mustafa et al. introduced a framework for using GANs to generate these maps more efficiently without sacrificing the statistical accuracy required for scientific analysis.

3.2 Dataset and Simulation Procedure

The dataset used by Mustafa et al. for training the GAN was derived from high-fidelity cosmological simulations using the Gadget2 N-body simulation code. These simulations were designed to replicate the large-scale structure of the universe, particularly focusing on weak gravitational lensing effects. The key cosmological parameters used in the simulations were:

- $\sigma_8 = 0.798$ (amplitude of matter fluctuations)
- $w = -1.0$ (dark energy equation of state)
- $\Omega_m = 0.26$ (matter density parameter)
- $\Omega_\Lambda = 0.74$ (dark energy density parameter)
- $n_s = 0.96$ (scalar spectral index)
- $H_0 = 0.72$ (Hubble constant)

A total of 45 simulations were performed, each containing 512^3 particles within a box size of $240\,h^{-1}$ Mpc. The simulations were processed using the Inspector Gadget ray-tracing pipeline to generate 1000 weak lensing shear and convergence maps at a redshift of $z = 1.0$. Each map covered an area of 12 square degrees and was initially rendered at a resolution of 2048×2048 pixels. These maps were subsequently downsampled to 1024×1024 pixels for analysis.

3.2.1 Weak Lensing Convergence

Weak gravitational lensing can be described by the Jacobian matrix, which maps the distortion effects on light as it passes through matter in the universe:

$$A(\theta) = \begin{pmatrix} 1-\kappa-\gamma_1 & -\gamma_2 \\ -\gamma_2 & 1-\kappa+\gamma_1 \end{pmatrix}$$

Here, κ represents the convergence (mass density) and γ represents the shear (distortion). The resulting convergence maps reflect the mass distribution in the universe along the observer's line of sight.

3.2.2 Data Preprocessing

To prepare the dataset for GAN training, each of the 1000 original maps was cropped into 200 smaller maps, each with a resolution of 256×256 pixels. This provided a total of 200,000 individual maps for training. The preprocessing steps also included the following:

- **Normalization:** Pixel values were checked, but as the probability of a pixel value falling outside the $[-1.0, 1.0]$ range was less than 0.9%, the data was used in its natural format without normalization.
- **Data Augmentation:** Maps were rotated and flipped randomly to augment the dataset and improve generalization during training.
- **Dataset Splitting:** The dataset was divided into training, validation, and testing sets, where 70% of the maps were used for training, 15% for validation, and 15% for testing.

3.3 GAN Breakdown

In GANs, the generator is tasked with producing fake data samples, while the discriminator evaluates whether a given sample is real or generated. The objective is for the generator to create data that is indistinguishable from the real dataset, thus "fooling" the discriminator.

Mustafa et al. implemented a Deep Convolutional GAN (DCGAN) for generating weak lensing convergence maps. The DCGAN architecture is designed to handle grid-like data, such as images, and employs convolutional layers for both the generator and the discriminator.

3.4 Generator and Discriminator Architecture

3.4.1 Discriminator Architecture

The discriminator is tasked with differentiating real cosmological maps from those generated by the GAN. It functions as a binary classifier, outputting a probability that the input map is real.

Convolutional Layers: These layers progressively reduce the spatial dimensions of the input, extracting increasingly complex features. A convolution operation can be mathematically expressed as:

$$y(i,j) = \sum_m \sum_n x(i-m, j-n) w(m,n)$$

where $x(i,j)$ is the input image, $w(m,n)$ is the filter, and $y(i,j)$ is the output feature map. This allows the discriminator to learn hierarchical features from the input maps.

Batch Normalization: After the second, third, and fourth convolutional layers, batch normalization is applied to stabilize the training and prevent overfitting.

Leaky ReLU Activation: The Leaky ReLU activation function is used after each convolutional layer. Unlike ReLU, Leaky ReLU allows a small gradient for negative inputs, defined as:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.2x & \text{otherwise} \end{cases}$$

This prevents the network from "dying" by ensuring that some gradient information passes even when the input is negative.

Fully Connected Layer: The final layer of the discriminator is a fully connected layer that outputs a scalar value representing the probability that the input map is real.

The discriminator's architecture is detailed in Table 3.1.

Layer	**Input Size**	**Output Size**	**Kernel Size**	**Stride**	**Padding**
Conv Layer 1	(1, 256, 256)	(64, 128, 128)	5x5	2	2
Conv Layer 2	(64, 128, 128)	(128, 64, 64)	5x5	2	2
Conv Layer 3	(128, 64, 64)	(256, 32, 32)	5x5	2	2
Conv Layer 4	(256, 32, 32)	(512, 16, 16)	5x5	2	2
Fully Connected	(512 * 16 * 16)	(1)	-	-	-

Table 3.1: Discriminator architecture detailing layer dimensions, kernel sizes, strides, and padding.

Both the generator and discriminator utilize batch normalization and activation functions strategically to improve convergence, avoid gradient-related issues, and stabilize training in the context of GANs.

Kernel Size: The kernel size refers to the dimensions of the filter applied during the convolution. For example, a 5x5 kernel means the filter covers a 5x5 region of the input at a time. The kernel slides over the input data, performing element-wise multiplications to extract features from the image.

Stride: The stride is the number of pixels by which the filter moves across the input data. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 skips every other pixel, effectively downsampling the input.

Padding: Padding refers to the addition of extra pixels around the input data, usually to control the output dimensions after the convolution. 'Same' padding keeps the output size the same as the input by adding zeros around the edges, while 'valid' padding means no padding is applied, resulting in a smaller output.

3.4.2 Generator Architecture

The generator is designed to transform a random noise vector sampled from the latent space into a realistic cosmological map. This transformation is achieved progressively through a series of transposed convolutional layers (also referred to as deconvolutional layers), which upsample the input noise into a high-resolution image. Each layer increases the spatial dimensions, moving from a latent vector to a final 256x256 output map.

Transposed Convolutional Layers: These layers perform the reverse of regular convolutions, increasing the spatial size of the input. For example, the first transposed convolutional layer transforms the input from a 16x16 representation to a 32x32 representation. The kernel size and stride control the amount of upscaling, while padding ensures the correct output dimensions.

Batch Normalization: After each transposed convolution, batch normalization is applied to normalize the activations. This improves the training speed and helps prevent overfitting.

ReLU Activation: Each transposed convolutional layer (except the last one) is followed by the ReLU activation function, defined as:

$$f(x) = \max(0, x)$$

This activation prevents negative values and mitigates the vanishing gradient problem.

Tanh Activation: The final layer uses the Tanh activation function, which scales the output values between -1 and 1:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

This ensures the output values match the range of the real data after preprocessing.

The generator's architecture is summarized in Table 3.2.

Layer	**Input Size**	**Output Size**	**Kernel Size**	**Stride**	**Padding**
Latent Vector (z)	(64)	(512, 16, 16)	Fully Connected	-	-
Deconv Layer 1	(512, 16, 16)	(256, 32, 32)	5x5	2	2
Deconv Layer 2	(256, 32, 32)	(128, 64, 64)	5x5	2	2
Deconv Layer 3	(128, 64, 64)	(64, 128, 128)	5x5	2	2
Deconv Layer 4	(64, 128, 128)	(1, 256, 256)	5x5	2	2

Table 3.2: Generator architecture detailing layer dimensions, kernel sizes, strides, and padding.

3.4.3 Training Procedure

The GAN was trained using the Adam optimizer with a learning rate of 2×10^{-4} and $\beta_1 = 0.5$. A batch size of 64 maps was used during training. The networks were trained for 45 epochs, during which the generator and discriminator were updated alternately.

To prevent the discriminator from overpowering the generator early in the training process, label flipping was employed, where real labels were occasionally assigned to fake samples and vice versa with a 1% probability. This technique helped balance the training dynamics between the two networks.

One of the main challenges in GAN training is its inherent instability, particularly in the later stages of training. Mustafa et al. noted that the training process often led to oscillatory behavior, where the performance of the generator would vary unpredictably. To address this, they employed early stopping based on statistical tests, such as the KS test, and careful tuning of the learning rate.

3.5 Evaluation Metrics

To evaluate the performance of the GAN, Mustafa et al. used several key metrics, focusing on both Gaussian and non-Gaussian statistical properties of the generated maps.

3.5.1 Pixel Intensity Distribution

The pixel intensity distribution of the generated maps was compared to that of the real maps using the Kolmogorov-Smirnov (KS) test. The generated maps showed an excellent match to the real maps, with a KS p-value greater than 0.999, indicating that the generator successfully captured the underlying distribution of pixel intensities.

3.5.2 Power Spectrum

The power spectrum, which measures the correlation of matter density fluctuations at different length scales, is a critical tool in cosmology. Mustafa et al. evaluated the power spectra of the generated and real maps at 248 Fourier modes. The results showed that the GAN was able to reproduce the power spectrum of the real maps with high accuracy, especially at large scales. Deviations at smaller scales were minimal and within acceptable statistical bounds.

3.5.3 Minkowski Functionals

Minkowski functionals provide a non-Gaussian statistical analysis of the generated maps. These functionals (area, perimeter, and Euler characteristic) measure the topological properties of the maps and offer insights into their geometrical structure. Mustafa et al. evaluated the Minkowski functionals of the generated maps and found that they closely matched those of the real maps, further validating the GAN's ability to replicate non-Gaussian features.

3.6 Comments

Looking forward, Mustafa et al. suggested several avenues for future research, including the development of conditional GANs that can generate maps based on specific cosmological parameters. This would allow for more targeted simulations and a deeper understanding of how different parameters affect the structure of the universe.

Mustafa et al.'s work represents a significant advancement in the use of GANs for cosmological simulations. By demonstrating that GANs can generate high-fidelity weak lensing maps that match the statistical properties of fully simulated maps, they have laid the groundwork for future research into the use of machine learning models as computationally efficient alternatives to traditional simulations. While challenges remain, particularly in terms of training stability and capturing small-scale structures, their results show great promise for the future of cosmological simulations.

Chapter 4

New Implementation

4.1 Transition from TensorFlow to PyTorch

While the initial approach was to use the original GAN implementation in TensorFlow, it quickly became clear that this was not feasible due to the significant advances in machine learning frameworks over the past few years. The original code, written in 2019, was outdated and incompatible with modern GPU drivers and CUDA versions provided by CloudVeneto. After several attempts to create a suitable environment, including setting up virtual environments with older TensorFlow versions, the CUDA and NVIDIA driver issues persisted, causing crashes and compatibility problems.

Recognizing the limitations of continuing with TensorFlow, we decided to rewrite the entire codebase using PyTorch, a more modern and flexible framework widely supported by the latest hardware and software environments. This transition required a learning curve as we undertook the task of mastering PyTorch from scratch. To achieve this, we relied on online resources and tutorials, where I learnt to manipulate tensors, create linear models, and build convolutional neural networks (CNNs) for image processing. This foundational knowledge of CNNs directly translated to our GAN implementation since both generators and discriminators rely heavily on convolutional layers to process and create images.

The beauty of the GAN framework is its versatility. The same code that could be used to generate number images (e.g., MNIST dataset) in basic tutorials could be seamlessly adapted to generate cosmological lensing maps. The architecture itself does not "know" what kind of images it is working with; it simply learns the underlying patterns in the data it is provided with.

4.1.1 Developing Original Code

The goal in developing the PyTorch code was to build a system that was flexible, modular, and easy to experiment with. We used several key features that are state of the art:

- **Adjustable Learning Rate:** We have control over the learning rate, the most important parameter. We experimented with tools like `ReduceLROnPlateau` to adjust the learning rate dynamically, but more often than not, we manually adjusted it as the situation required. This feature is essential, allowing us to stabilize the generator and prevent mode collapse.

- **Model Saving and Checkpointing:** We implemented a system that saved model checkpoints after each epoch, allowing recovery from crashes or disconnections and facilitating experimentation with different stages of the model's development. This feature was crucial for long training runs spanning multiple days, ensuring that we didn't lose progress in case of hardware failures or other issues.
- **Loss Monitoring and Logging:** We developed a comprehensive system to log the generator and discriminator losses after each step. This feature allowed us to closely monitor the training process, diagnose problems like vanishing gradients, and detect mode collapse early on. By tracking the losses over time, we could make informed decisions about when to intervene in the training process.

These features made the code ready for experimentation, hyperparameter fine-tuning and extended training periods.

4.2 Challenges and Adjustments

- **Strong Discriminator:** One of the key challenges was the discriminator becoming too strong, pushing the generator out of the game early on. The increased label flipping, as mentioned, helped to mitigate this by occasionally confusing the discriminator and giving the generator a chance to "catch up."
- **Learning Rate Tuning:** We avoided using a single, static learning rate throughout the training process. Instead, we dynamically adjusted the learning rate, especially when we detected mode collapse or instability. This fine-tuning was necessary to maintain a balance between the generator and discriminator. While the original paper trained for 45 epochs at a fixed learning rate, our approach was more dynamic. We trained for fewer epochs, with the learning rate gradually decreasing over time. Of course though, stopping training too early led to suboptimal results, so patience was required for the model to fully converge.
- **Time and computational optimization:** We also settled on a batch size of 64 to make the best use of available GPU memory and speed up the training process. While we experimented with label smoothing, it either slowed down the training too much or led to balanced losses without improving image quality. Our framework, combined with the hardware resources available, reduced the training time significantly. Where the original paper required 100 hours for a complete training cycle, we were able to complete 45–50 epochs in just 28 hours.

4.2.1 Instability

The GAN training process is inherently unstable. The generator and discriminator are constantly "competing," and in this adversarial setup, the generator tries to fool the discriminator, while the discriminator tries to become better at detecting fakes. During our training experiments, the discriminator often became too strong too quickly, leading to poor-quality images from the generator.

To address this, we increased the **label flipping** probability from 1% (as suggested in the original paper) to 3%. Label flipping involves intentionally mislabeling some real images as fake and vice versa, which forces the discriminator to learn in a more robust manner. By increasing the flipping probability, we gave the generator more breathing room to learn and prevented the discriminator from overpowering it too early.

The Delicate Balance: This dynamic, however, was not without its own risks. A high label-flipping probability could destabilize the discriminator, causing it to get "confused" and potentially allow poor-quality images to pass as real. This is what we experienced with trial and error, first with a label flipping probability of 5%. The challenge was to balance both networks' strengths and ensure that neither side "won" too quickly. This adversarial process is what makes GANs so powerful but also challenging to train.

4.2.2 Hyperparameter Tuning

One of the main challenges in implementing GANs is choosing the right hyperparameters, particularly the learning rate. In the original paper by Mustafa et al., the learning rate for both the generator and discriminator was set at 2×10^{-4}. However, we found that this rate was unsuitable for our implementation. With this high learning rate, the generator produced poor-quality images, and the discriminator became too powerful, leading to unstable training.

After extensive experimentation, we settled on a starting learning rate of 2×10^{-5}. This change allowed the generator to produce more stable and realistic results. However, learning rate tuning was not a one-time adjustment; throughout the training process, we needed to manually adjust the rate depending on the behavior of the GAN. For example, we occasionally had to lower the learning rate further to prevent mode collapse or instability.

Another recurring issue was **mode collapse**, where the generator would produce images that were visually similar with little variation. Already at early epochs, if the learning rate was not carefully chosen or too high, like the one suggested by the original paper, we could see signs of total mode collapse. To combat this, we manually adjusted the learning rate whenever we detected mode collapse. Sometimes, extending the training time allowed the generator to escape mode collapse and produce more diverse images, but this required careful monitoring. Additionally, overtraining at a low learning rate could cause overfitting, where the generator simply memorized the training data without generating novel variations.

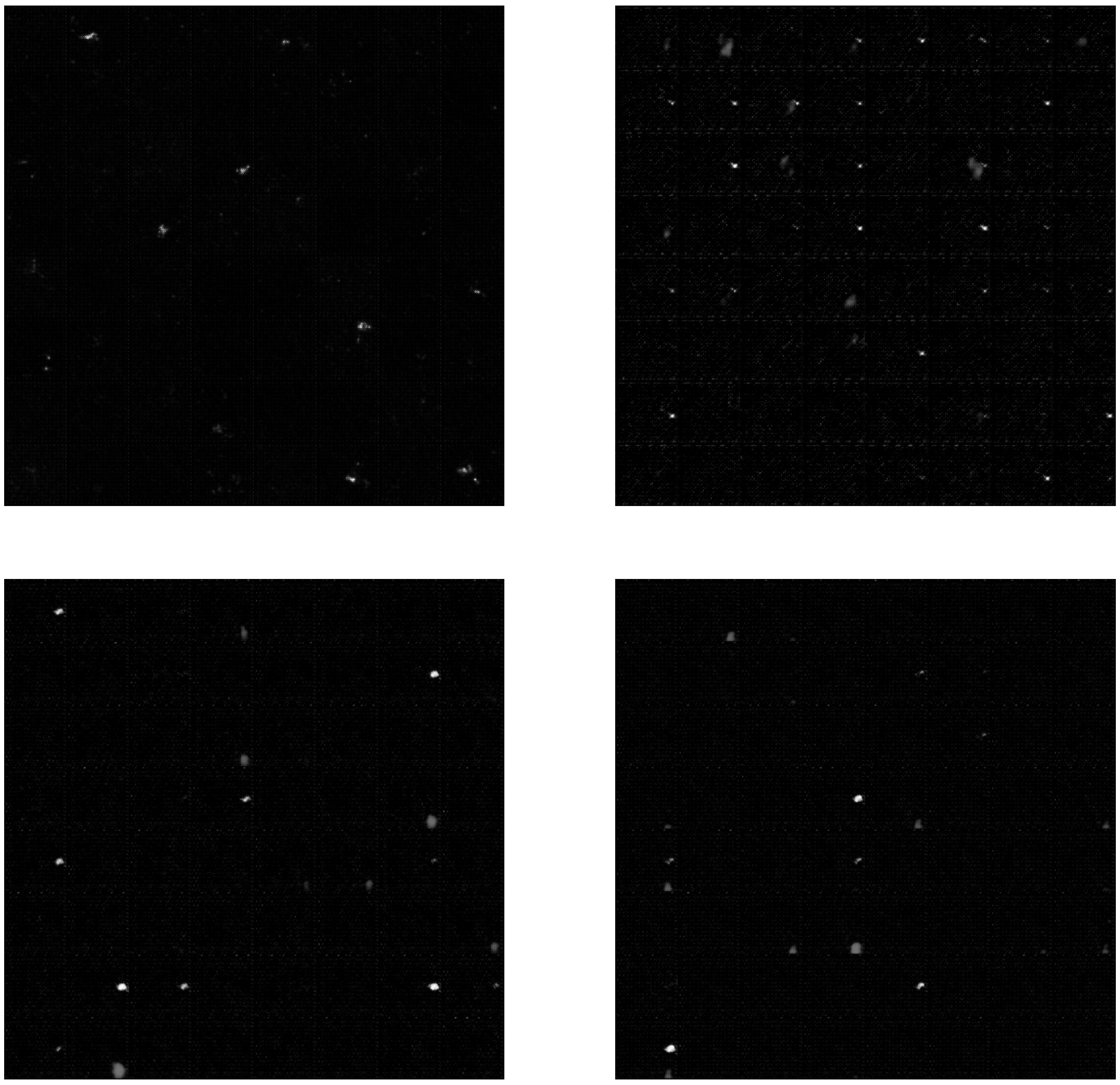
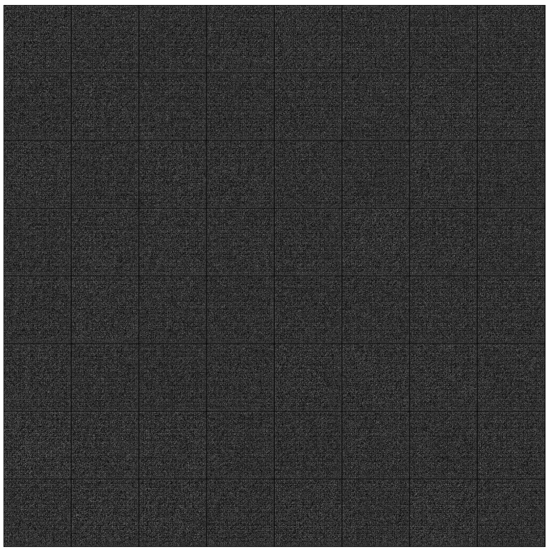
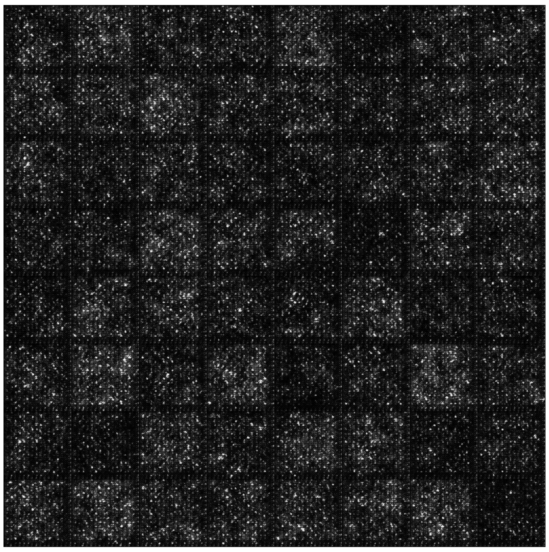
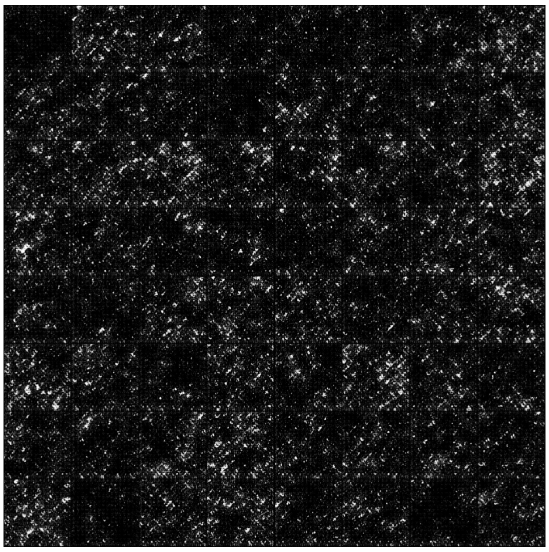
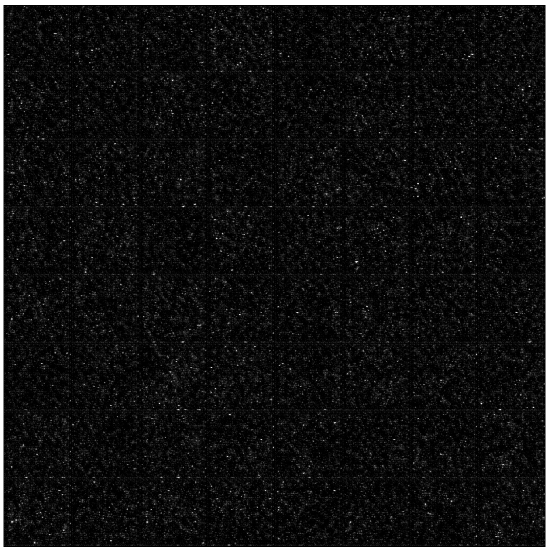


Figure 4.1: Mode collapse examples: repeated features after several epochs stop the improvement of the generator.


(a) Noise


(b) Early Training


(c) Intermediate Training


(d) Late Training

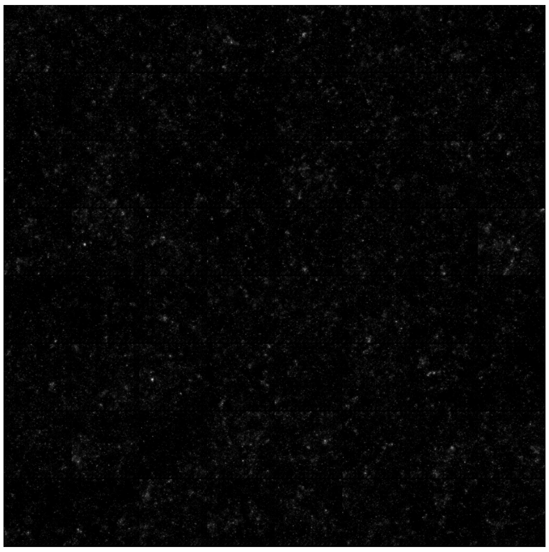

(e) Final Quality

Figure 4.2: Improvement in generated images over time, from noise (top-left) to best results (bottom).

4.2.3 Techniques and Experiments

In this section, we present and discuss several techniques that were experimented with during the training of our GAN model. These techniques were aimed at stabilizing training and improving convergence.

Label Flipping

Label flipping involves intentionally mislabeling a small percentage of real and generated samples during training. Specifically, we flipped the labels of real images to fake and vice versa with a small probability. This technique prevents the discriminator from becoming too confident early in training, encouraging the generator to improve. Label flipping adds noise to the learning process, making it harder for the discriminator to perfectly distinguish between real and fake data

Label Smoothing

Label smoothing is used to soften the labels provided to the discriminator. Instead of labeling real images as 1 and fake images as 0, real labels are smoothed to values close to 1 (e.g., 0.9), and fake labels can be slightly greater than 0 (e.g., 0.1). This helps prevent the discriminator from becoming overconfident and can also mitigate overfitting. In addition, it encourages the discriminator to maintain some uncertainty in its predictions, promoting a more balanced training process. After several attempts, we did not settle with label smoothing because it made the generator overpower the discriminator

Learning Rate Scheduler

During our experimentation, we implemented a learning rate scheduler to dynamically adjust the learning rate throughout the training process. The logic behind this approach is that reducing the learning rate as training progresses can help the model converge more smoothly. Early on, larger updates enable the model to explore the solution space more freely, while later in training, smaller updates can help fine-tune the model.

However, in our specific case, using a learning rate scheduler did not lead to any noticeable improvement. In fact, it sometimes accelerated issues like mode collapse, particularly when the learning rate was reduced too quickly. The idea behind the scheduler is to adjust the learning rate based on fixed steps or certain triggers (such as validation loss), but for our GAN, these adjustments failed to provide a more comprehensive view of the training dynamics.

One reason for this could be that the training process of GANs is highly sensitive and complex. A steady reduction in learning rate, especially when following a pre-determined schedule, doesn't necessarily correspond to the rapidly changing dynamics between the generator and discriminator. In some instances, reducing the learning rate prematurely might have restricted the generator's ability to adapt, leading to poor performance or mode collapse.

For this reason, we opted not to include a learning rate scheduler in our final model. Instead, we maintained a constant learning rate for the majority of the training, which allowed for more stability across the process. This decision highlighted the need for more sophisticated adaptive strategies tailored specifically to the unique challenges of GAN training, namely fine tune it following the logic we mentioned.

Different Learning Rates for Generator and Discriminator

In some of our experiments, we tried using different learning rates for the generator and discriminator, often assigning a lower learning rate to the generator and a higher one to the discriminator. The idea behind this approach is rooted in the typical imbalance between the two networks: the discriminator tends to learn faster than the generator, especially in the early stages of training. By giving the generator a lower learning rate, we aimed to slow down the discriminator's learning and give the generator more time to catch up.

However, this technique did not lead to any significant improvements in our case. In fact, the results often showed greater instability, with the model converging to suboptimal equilibria or experiencing premature mode collapse. The key issue seemed to be that introducing an artificial imbalance in the learning rates exaggerated the already delicate dynamics between the generator and discriminator.

In GAN training, a fine balance between the two networks is crucial. If the discriminator becomes too strong too quickly, the generator can struggle to make meaningful progress, producing low-quality outputs or collapsing altogether. On the other hand, if the generator improves too fast, the discriminator may not have enough time to adjust, leading to poor classification performance. Finding the right balance is difficult, and adjusting the learning rates separately added more complexity without yielding better results.

For these reasons, we ultimately decided to keep the learning rates equal for both networks, as it provided more stable training dynamics.

4.2.4 Overfitting and Underfitting

In machine learning, overfitting and underfitting are common challenges when training models. Overfitting in the context of GANs refers to the phenomenon where the generator begins to produce images that are too closely tailored to the training data, failing to generalize to new unseen examples. While early stages of training focus on generating realistic images that align with the true data distribution, prolonged training can lead to a degradation in performance, as the generator effectively "memorizes" the training data and loses its ability to produce diverse and general representations of the target distribution. Overfitting was particularly noticeable when the generated images lacked variation or the GAN produced similar structures repeatedly.

Pixel Intensity Distribution: Evidence of Overfitting

The provided plots 4.8 depict the pixel intensity distribution of generated images compared to real images over several epochs. These distributions provide insight into how well the generator is capturing the overall statistical properties of the real images. The horizontal axis represents the pixel intensity, while the vertical axis shows the density on a log scale.

- **Initial Improvement:** In the first four plots, we observe that the generated images gradually improve in matching the pixel intensity distribution of the real images. Earlier epochs show larger discrepancies between generated and real images, particularly in the higher intensity ranges (right side of the plot). As training progresses, these discrepancies reduce, and the generator starts producing images that better align with the real data distribution. This is the desired effect of training where the generator is effectively learning the underlying patterns in the dataset.

- **Peak Alignment:** At a certain point in training (visible in the mid-range epochs), the generator achieves a close match to the real data. The pixel intensity distributions of the generated and real images nearly overlap, indicating that the generator has successfully learned to mimic the real data's characteristics. This can be considered the optimal point of training where the generator is generalizing well.

- **Degradation:** However, after this point, the later epochs begin to show signs of degradation. The last few plots reveal that the generator starts to diverge from the real data distribution again. This could be due to overfitting—where the generator has over-optimized for specific patterns in the training data, leading to a failure to generalize. The generated images now exhibit a pixel intensity distribution that no longer aligns with the real data, particularly in the high-intensity ranges. This indicates that the generator has memorized certain aspects of the data instead of learning the underlying distribution, a classic sign of overfitting.

Impact of Overfitting on Image Quality

The visual representation of this degradation is subtle but significant. The quality of the generated images may appear to improve initially, but as overfitting sets in, the variability and richness of the generated images decrease. This is reflected in the pixel intensity distribution plots, where the generated images start producing overly smooth or repetitive patterns, and the diversity of pixel intensities diminishes.

In summary, these plots clearly demonstrate the challenge of overfitting in GAN training. The early stages show improvement and alignment with real data, while later epochs exhibit the negative effects of prolonged training, where the generator diverges from the true data distribution. This reinforces the importance of careful monitoring during training and the need for stopping criteria or regularization techniques to prevent overfitting.

In addition to the visual degradation observed in the pixel intensity distributions, the Kolmogorov-Smirnov (KS) statistic provides a quantitative metric to evaluate the similarity between the pixel intensity distributions of the generated and validation maps. The KS statistic measures the maximum difference between the cumulative distribution functions (CDFs) of two datasets, which in our case are the pixel intensities of the generated maps and the validation maps.

Mathematically, the KS statistic is defined as follows:

$$D_{n,m} = \sup_x |F_n(x) - F_m(x)|$$

where $F_n(x)$ and $F_m(x)$ are the empirical cumulative distribution functions (CDFs) of the pixel intensities of the generated and validation maps, respectively, and $\sup_x$ denotes the supremum (or maximum) value over all points x in the pixel intensity range. Essentially, this measures the largest vertical distance between the two CDFs.

The CDF of a distribution is given by:

$$F(x) = P(X \leq x)$$

where X represents a random variable (in this case, pixel intensity), and $P(X \leq x)$ is the probability that X takes a value less than or equal to x.

A smaller KS statistic, i.e., a smaller $D_{n,m}$, indicates that the pixel intensity distributions of the generated maps are closer to those of the validation maps, implying that the generator has successfully captured the distribution of real data. A KS statistic of 0.036, as observed in our best-performing generator, suggests a high degree of similarity between the pixel intensity distributions of the generated and validation maps, indicating that the generator is effectively reproducing the true data characteristics.

During training, the KS statistic initially declined, reflecting improvements in the generator's ability to approximate the validation data distribution. However, as the model trained further, the KS statistic started to increase, signifying overfitting. This trend suggests that while the generator learned to replicate certain features of the validation data, it eventually began to memorize specific details rather than generalizing well.

Even after adjusting the learning rate, the KS statistic worsened at later stages of training, highlighting that the generator was producing images that diverged from the true data distribution. Mathematically, this increase in the KS statistic, $D_{n,m}$, implies that the CDFs of the pixel intensities between the generated and validation maps grew further apart, indicating that the generator's outputs were less faithful to the real data distribution.

Therefore, the KS statistic, combined with visual and other statistical measures, serves as a powerful tool for detecting overfitting and assessing the model's generalization ability.

Underfitting

Underfitting happens when the model fails to capture the underlying patterns in the data. This is often due to insufficient training or an overly simple model. Underfitting in our GAN manifested when the generator produced blurry or low-quality images, indicating that it had not learned enough detail from the data.

Balancing between overfitting and underfitting required a careful adjustment of the learning rate, training epochs, and the regular use of validation data to monitor the model's performance on unseen images. Additionally, early stopping mechanisms and careful monitoring of the validation loss were used to prevent the model from overfitting.

In summary, our journey to fine-tune the GAN involved numerous iterations, adjustments, and careful monitoring of hyperparameters. While GANs offer incredible flexibility in generating realistic data, their training is an art that requires constant balancing between the competing forces of the generator and discriminator. Through dynamic learning rate adjustments, label flipping, and improved loss monitoring, we were able to generate realistic cosmological maps with greater time efficiency than the older implementation.

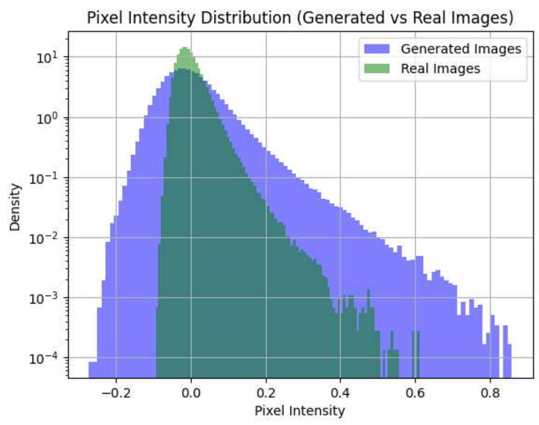


Figure 4.3: Pixel Intensity Distribution - Epoch 30

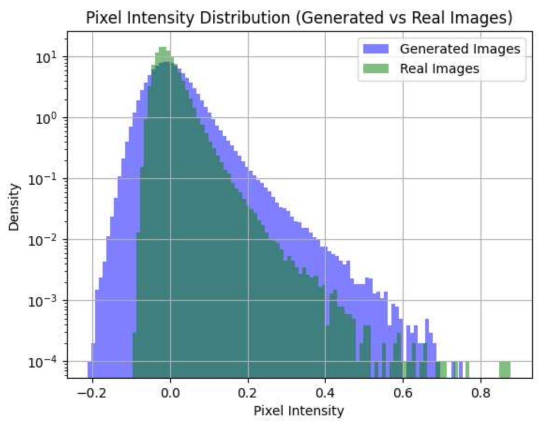


Figure 4.4: Pixel Intensity Distribution - Epoch 40

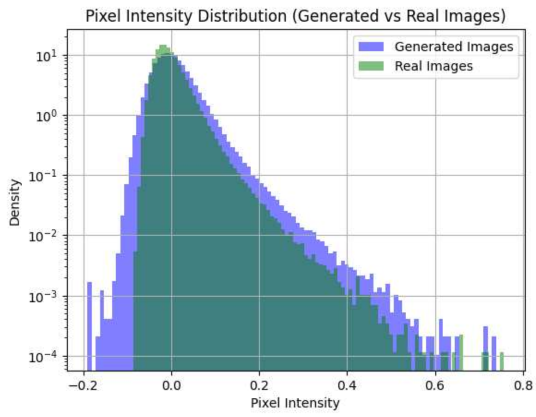


Figure 4.5: Pixel Intensity Distribution - Epoch 45

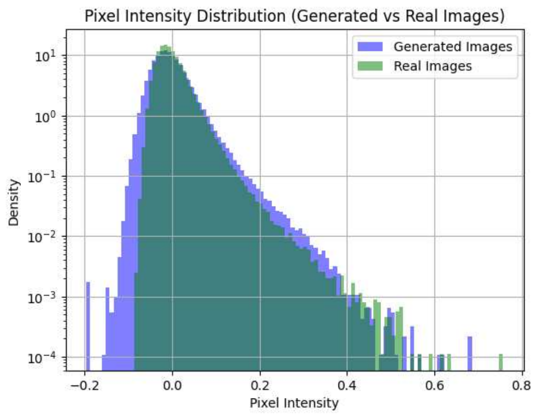


Figure 4.6: Pixel Intensity Distribution - Epoch 50

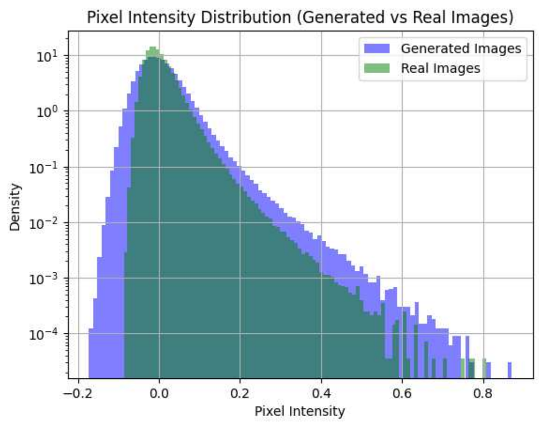


Figure 4.7: Pixel Intensity Distribution - Epoch 55

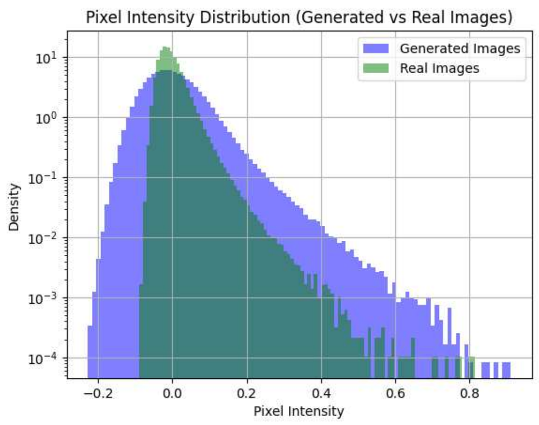


Figure 4.8: Pixel Intensity Distribution - Epoch 60

Chapter 5

Results

5.1 Our generator

In this section, we present the results of our trained GAN model, focusing on the qualitative and quantitative performance, as well as statistical analyses. The results were achieved after extensive training and fine-tuning over several epochs to ensure a meaningful comparison between generated and validation maps.

The architecture of our generator was based on the model used by Mustafa et al., as previously described, with key adjustments made for label flipping and training stability. Specifically, we implemented a label flipping probability of 3%, as discussed earlier, to prevent the discriminator from overpowering the generator during the early stages of training. This adjustment helped balance the dynamics between the two networks and contributed to more stable training.

The GAN model was trained using a NVIDIA Tesla T4 GPU, which provided the necessary computational power for efficient training. The training process was carried out over a total of 50 epochs, with a decreasing learning rate to gradually fine-tune the model as follows:

- 30 epochs at an initial learning rate of 2×10^{-5},
- 5 epochs at a reduced learning rate of 5×10^{-6},
- 5 epochs at 2×10^{-6},
- 5 epochs at 1×10^{-6},
- and the final 5 epochs at 5×10^{-7}.

This gradual reduction in the learning rate was implemented to allow the model to converge more smoothly towards an optimal solution, reducing the risk of instability or mode collapse during the later stages of training. This strategy also helped fine-tune the generator's ability to produce realistic maps by making smaller, more refined updates to the model parameters as training progressed.

5.1.1 Training Dynamics and Stability

The training process highlighted several challenges related to the stability of the GAN. Given the complexity of the training, some degree of instability is expected, and, as noted in the original paper, determining the optimal stopping point is somewhat arbitrary. This is because, in this architecture, the discriminator tends to surpass the generator over time, leading to diminishing improvements with prolonged training.

Figure 5.1 illustrates the loss curves for both the generator and discriminator throughout the training epochs. Notably, at epoch 45, the generator's loss spiked significantly, indicating potential overfitting. By continuously monitoring the loss curves, we were able to intervene and halt the training if such instability occurred. To select the best generator, we applied the KS test, analyzed the pixel intensity distribution, evaluated the power spectrum, and visually inspected the generated images for overall quality.

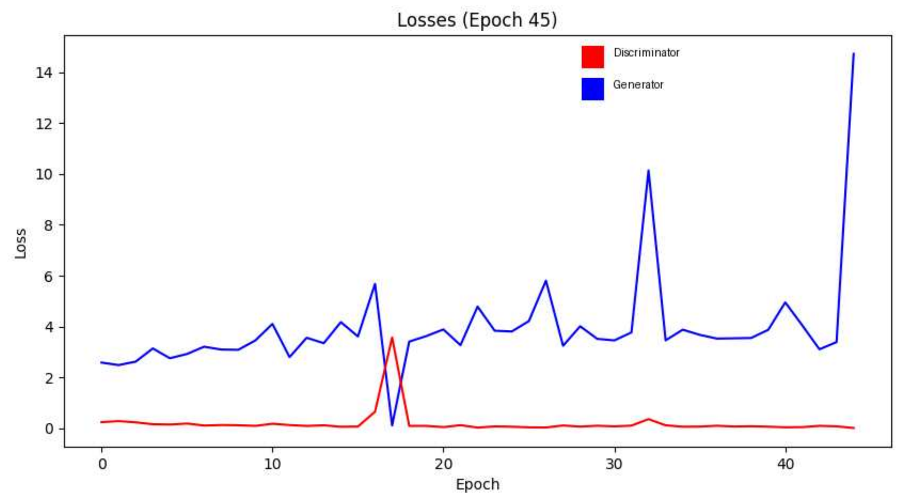


Figure 5.1: Training dynamics for generator and discriminator losses over epochs.

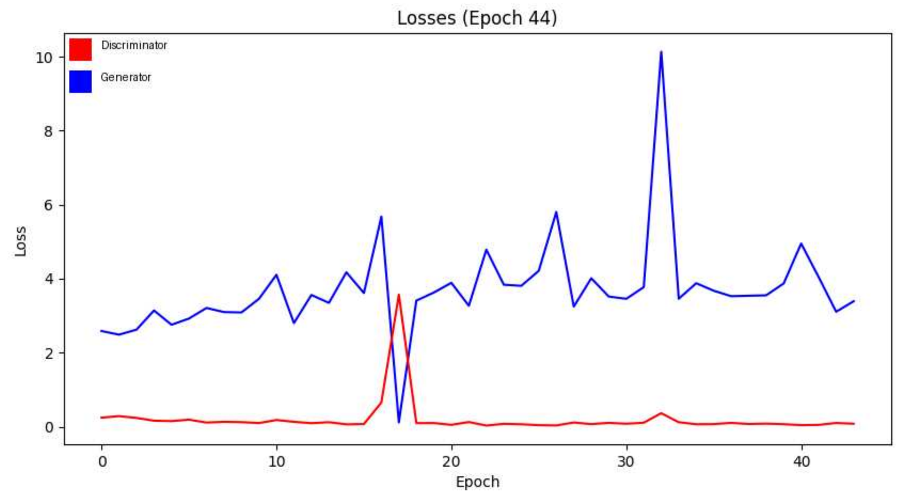


Figure 5.2: Going back to previous stable point and trying different hyperparameters.

5.1.2 Visual Quality of Generated Images

The visual quality of the generated images provides the first indication of the model's performance. Our generator successfully replicated the complex structures seen in cosmological maps. Despite the challenges, we observe that the core features of the maps are well-preserved and the generated maps achieve a high level of fidelity when compared to the validation set.

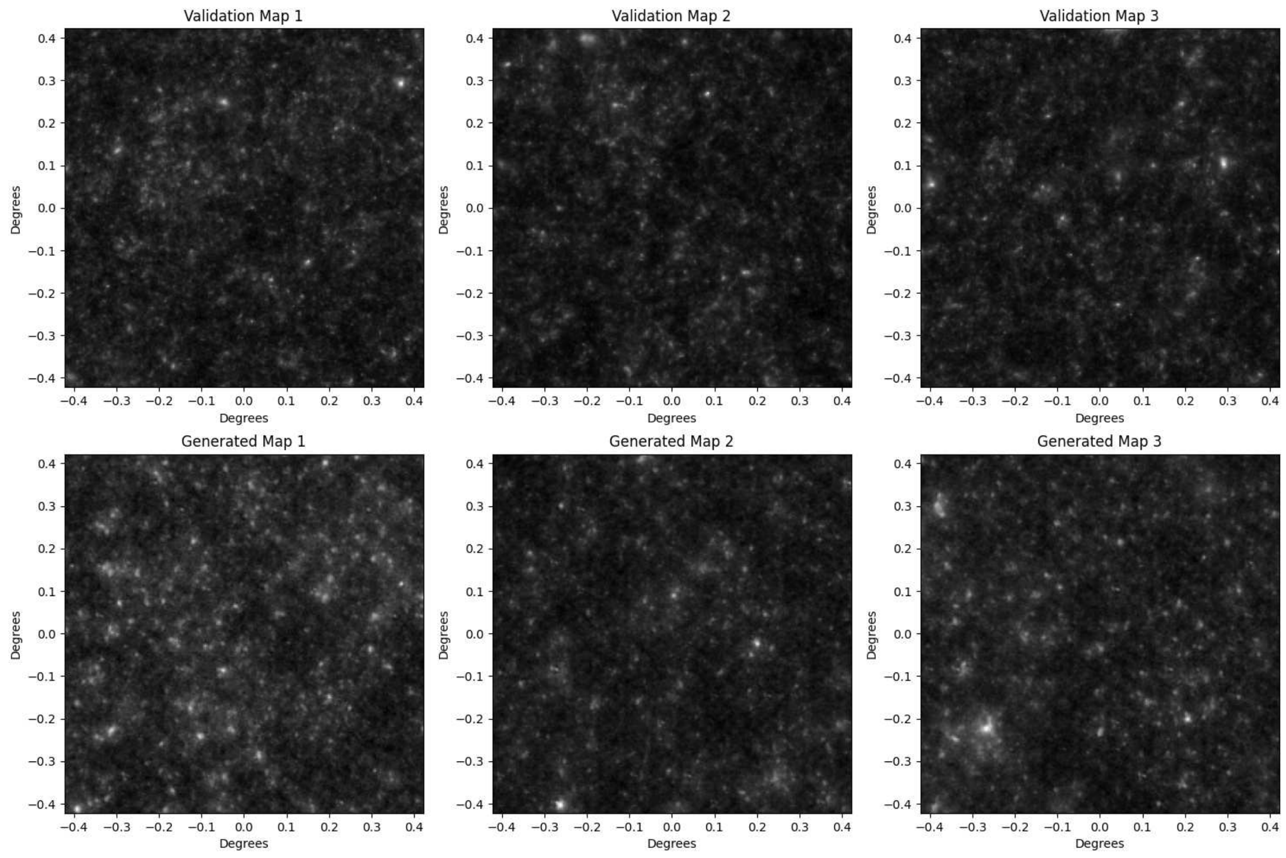


Figure 5.3: Random samples from validation dataset (top row) and generated images (bottom row).

5.1.3 Pixel Intensity Distribution

One important quantitative metric is the pixel intensity distribution, which gives insight into the overall fidelity of the generator. Figure 5.4 shows that the generated maps' pixel intensity distribution closely matches the validation maps. Small deviations, particularly at the tails of the distribution, suggest that the GAN may overfit certain regions. This issue was also reported in the original paper. In some instances, our generator achieved higher concordance in the high-intensity pixel regions compared to the 2019 results, although this was accompanied by a slight deterioration in other statistics. In the end we decided to present this version of it.

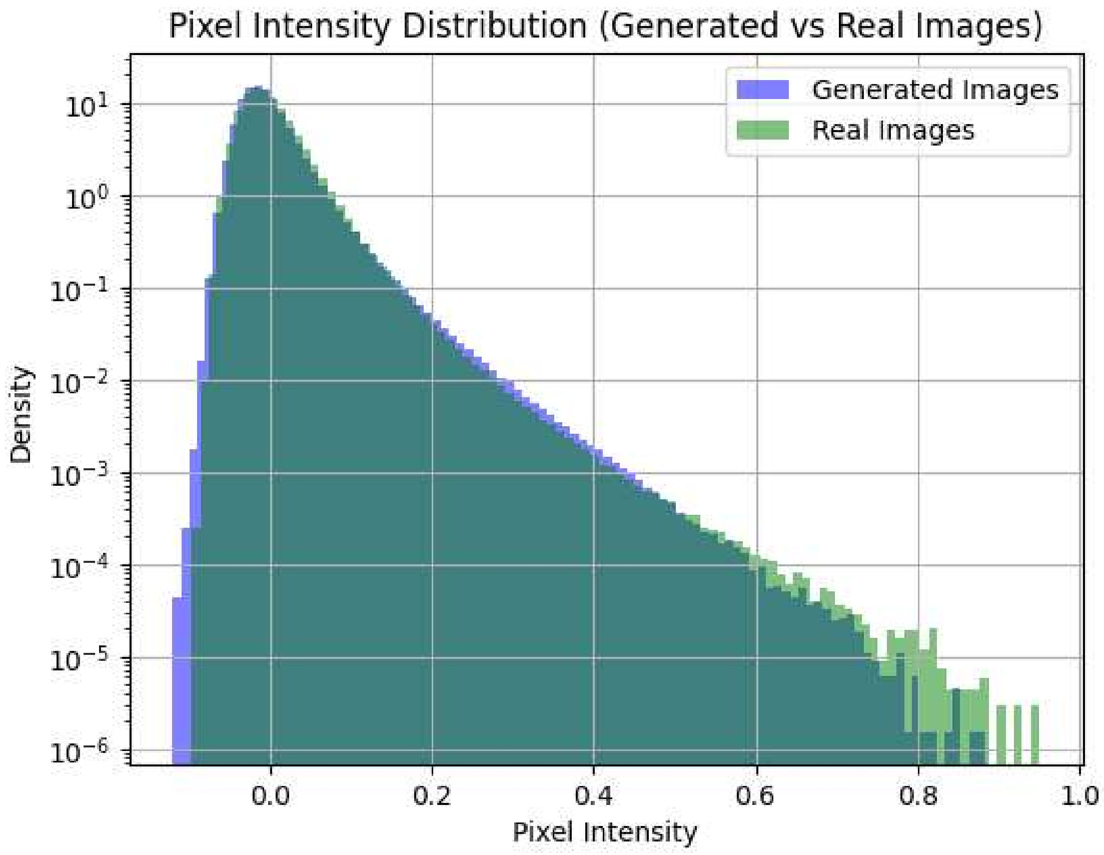


Figure 5.4: Pixel intensity distribution comparison between validation and generated maps.

5.1.4 Power Spectrum Analysis

The **power spectrum** is a fundamental statistical tool in cosmology that quantifies the distribution of matter and energy in the universe across different scales. More specifically, it measures the variance of fluctuations in the map as a function of spatial scale. These fluctuations are typically expressed in terms of their **Fourier modes**, which break down the spatial variations of the map into components with different wavelengths. Large-scale structures (e.g., cosmic voids, superclusters) correspond to low-frequency (large-wavelength) modes, while smaller-scale structures correspond to high-frequency (small-wavelength) modes.

The power spectrum $P(k)$ quantifies the variance in the field as a function of wave number k, where k is related to the spatial frequency:

$$P(k) = \langle \tilde{\delta}(k)\tilde{\delta}^*(k) \rangle$$

where:

- $\tilde{\delta}(k)$ is the Fourier transform of the density contrast $\delta(x)$, which represents fluctuations in the map.
- $\langle \cdot \rangle$ denotes the ensemble average.

In terms of **multipole moments** l for a 2D map (as is typical in cosmological weak lensing studies), the angular power spectrum C_l is often used:

$$C_l = \frac{1}{2l+1} \sum_{m=-l}^{l} |a_{lm}|^2$$

where a_{lm} are the spherical harmonic coefficients of the field. The multipole moment l is related to the angular scale of the features in the map, with larger l corresponding to smaller angular scales.

To evaluate the performance of the GAN, we computed the **power spectrum** for both the generated and validation maps. This was done using 6400 samples from each set, comparing the **Fourier modes** according to the procedure outlined in the reference article. We used **LensTools** to calculate the power spectra for a range of multipole moments l, focusing on scales from $l \approx 10^3$ to $l \approx 5 \times 10^4$.

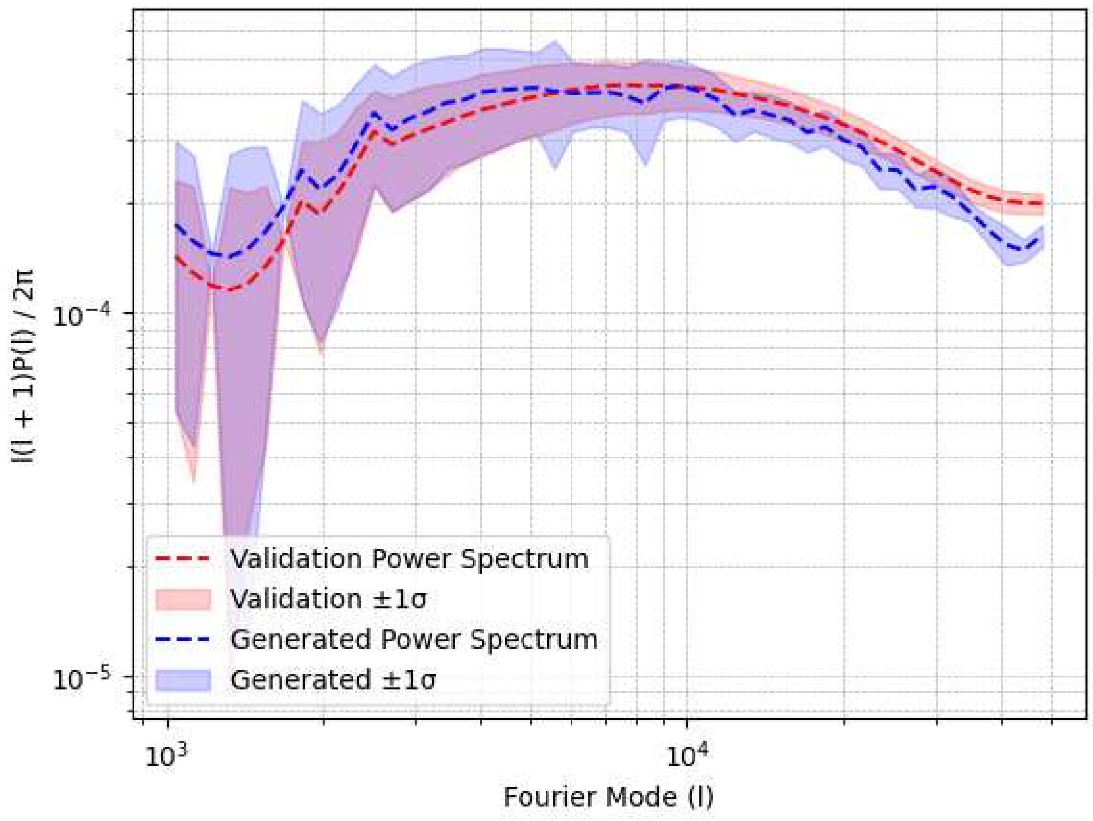


Figure 5.5: Power spectrum comparison between validation and generated maps.

The **power spectrum comparison** reveals that the generated maps align well with the validation maps on **large scales** (corresponding to low l or low-frequency modes). This suggests that the GAN is successfully capturing the broad, large-scale structures of the universe, such as cosmic voids and superclusters, as these are easier to reproduce.

However, discrepancies become apparent at **smaller scales** (high l or high-frequency modes), where the generated maps begin to deviate from the validation maps. These small-scale deviations could be attributed to a few potential causes:

- **Overfitting:** The generator may overfit certain features, leading to unrealistic patterns at high frequencies (small scales), which are harder to model correctly. The smaller-scale structures, such as galaxy clusters, are more sensitive to the fine details of the underlying cosmology.

- **Resolution Limitations:** At higher values of l, we are probing finer details in the maps, and the GAN may struggle to reproduce these details accurately due to limitations in resolution or training complexity.

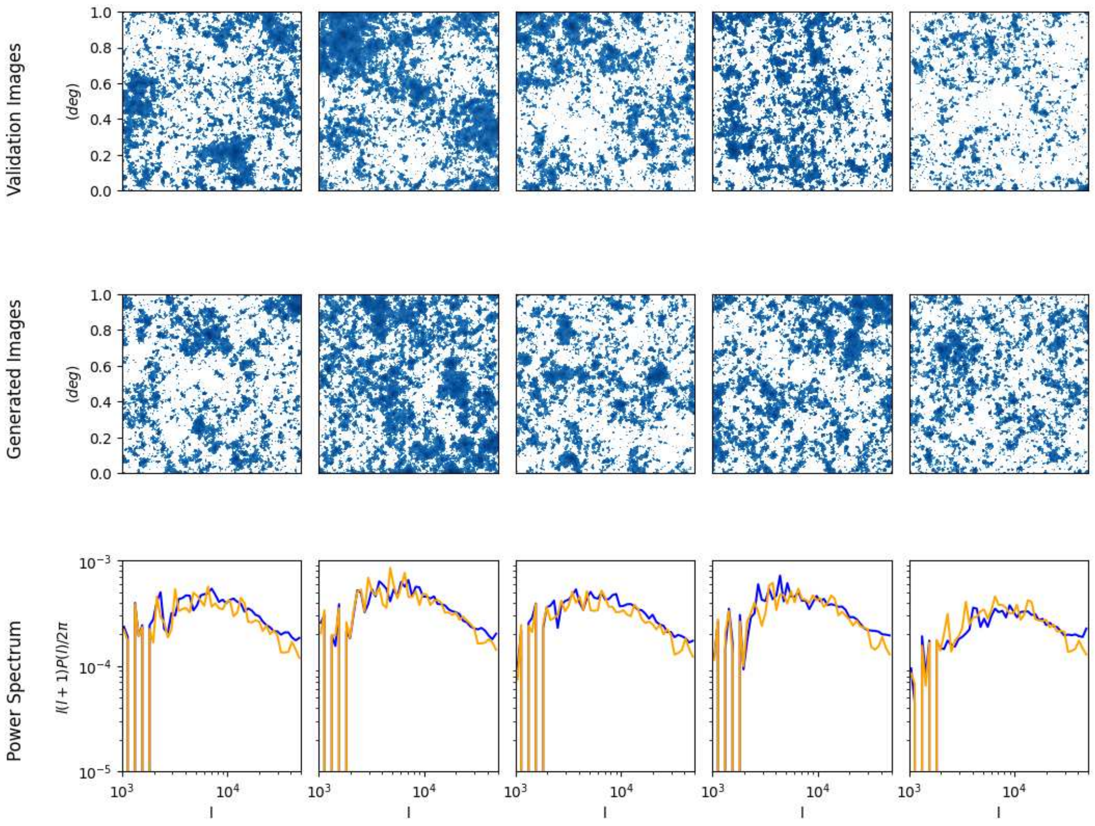
It is well known that **small scales are inherently more difficult to probe** due to noise, finite resolution, and the complexity of the underlying physics. In this first attempt, we did not surpass the accuracy achieved in the original paper for small-scale structures, despite efforts to fine-tune the training and learning rates.

The power spectrum remains one of the most direct and robust ways to compare the statistical properties of generated and real cosmological maps. While the GAN successfully replicates large-scale structures, the small-scale discrepancies indicate areas where further work is needed to improve the generator's capacity to model fine details. Future work could explore regularization techniques or alternative architectures to mitigate overfitting and enhance the accuracy at small scales.

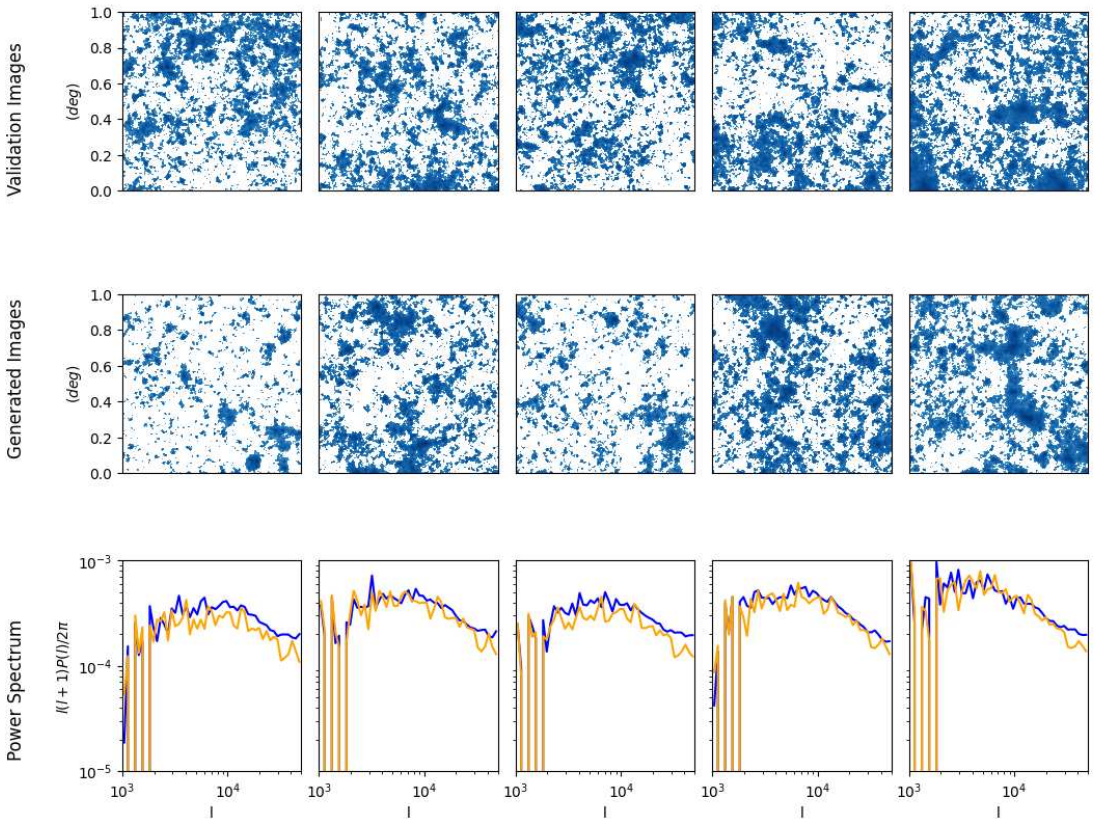
The power spectrum comparison reveals that the generated maps generally match the validation maps well, particularly on large scales (low Fourier modes). However, discrepancies appear at smaller scales, where generated maps exhibit a small deviation, probably due to overfitting. It is reported that the smaller scales are more difficult to probe and we didn't find a useful solution in this first attempt of ours to get a better result than the original paper.

5.1.5 Visual Comparisons Based on Euclidean Distance

To ensure that the GAN does not simply memorize the training dataset, we performed a Euclidean distance-based comparison between the generated images and their closest validation counterparts. This method confirms that the generator is producing novel outputs, not duplicates of the validation maps. We present 10 pairs of generated and validation images with their power spectra.



(a)



(b)

Figure 5.6: Visual comparison between validation maps (top) and generated maps (bottom), selected by minimum Euclidean distance.

5.1.6 Non-Gaussian Analysis via Minkowski Functionals

Minkowski functionals provide a powerful statistical tool for quantifying the geometrical and topological features of cosmological maps. In the context of cosmological data, they are particularly useful for detecting non-Gaussianities in the large-scale structure of the universe, as Gaussian statistics alone are insufficient to describe the complex, non-linear processes that give rise to the observed structures.

For a two-dimensional field, such as the convergence maps used in our analysis, there are three Minkowski functionals:

- V_0: the Area,
- V_1: the Perimeter,
- V_2: the Euler Characteristic.

These functionals are calculated over excursion sets, which are regions where the field exceeds a certain threshold, ν. As we vary ν, the functionals provide a way to track how the morphology of the field changes at different intensity levels, thus offering insight into the underlying geometry of the field. Each of the Minkowski functionals provides a specific type of geometric information:

- **Area** (V_0): The area of the excursion set is the simplest Minkowski functional and is defined as:
$$V_0(\nu) = \frac{1}{A}\int_{\Sigma(\nu)} dA$$
where A is the total area of the map, and $\Sigma(\nu)$ is the region where the field exceeds the threshold ν. This functional essentially measures the fraction of the map area covered by regions where the field exceeds the threshold.

- **Perimeter** (V_1): The perimeter of the excursion set is given by the length of the boundary between regions above and below the threshold:
$$V_1(\nu) = \frac{1}{4A}\int_{\partial\Sigma(\nu)} ds$$
where $\partial\Sigma(\nu)$ represents the boundary of the excursion set and ds is the differential length element. This functional quantifies the complexity of the boundary, which reflects the level of small-scale structures.

- **Euler Characteristic** (V_2): The Euler characteristic is a topological invariant that counts the number of connected regions minus the number of holes in the excursion set:
$$V_2(\nu) = \frac{1}{2\pi A}\int_{\partial\Sigma(\nu)} \kappa ds$$
where κ is the curvature of the boundary. The Euler characteristic tracks the topological complexity of the field and reveals information about the connectivity and holes in the structure.

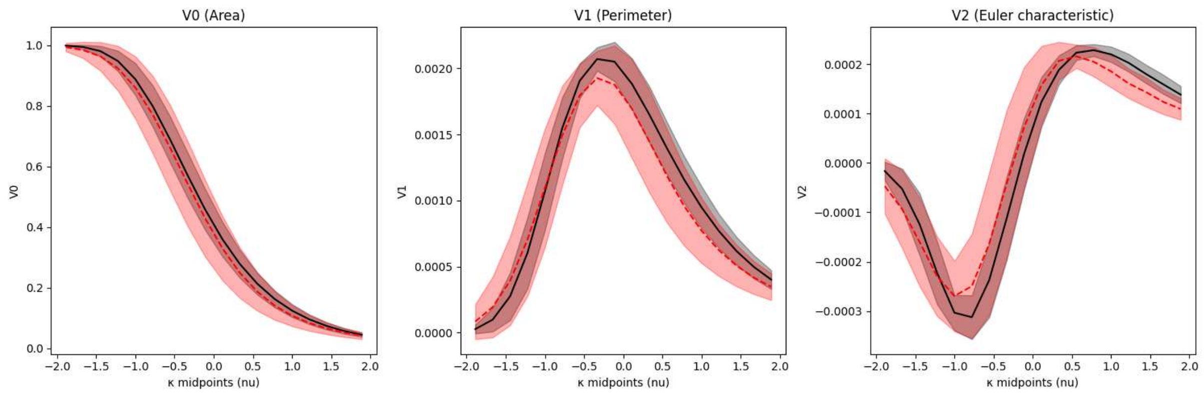


Figure 5.7: Minkowski functional comparison (V_0, V_1, and V_2) between validation and generated maps.

The Minkowski functionals are computed for a range of threshold values ν, and their dependence on ν provides a detailed characterization of the structure of the field. For a perfectly Gaussian field, the Minkowski functionals follow well-defined analytical forms. Deviations from these forms can indicate the presence of non-Gaussian features, which are important for understanding non-linear gravitational evolution and the physics of structure formation.

In our case, the Minkowski functional analysis demonstrates that the GAN captures many of the key geometrical structures found in the validation maps. However, slight deviations occur, particularly in the perimeter (V_1) and Euler characteristic (V_2) at higher threshold values. These discrepancies point to missing fine-scale details and differences in the topological structure of the generated maps compared to the real data. For example, the higher Euler characteristic values at large thresholds suggest that the generated maps contain more disconnected regions or isolated structures than the validation maps.

Therefore, Minkowski functionals provide a robust way to assess the quality of the generated maps, complementing other metrics such as the power spectrum and pixel intensity distribution. The combined use of these tools offers a comprehensive picture of how well the GAN replicates the true cosmological structures.

5.1.7 Efficiency and Scalability

Our training pipeline was optimized to significantly improve efficiency and scalability. By utilizing advanced hardware (such as NVIDIA Tesla T4 GPUs) and optimizing our codebase for parallelization and faster data loading, we reduced the training time for 50 epochs from the previously reported 100 hours to just 28 hours. These improvements were achieved without sacrificing the quality or features of the generated maps, allowing the GAN to converge more quickly while maintaining high fidelity.

Additionally, the scalability of our model is noteworthy. The generator can produce thousands of unique cosmological maps within seconds, which is crucial for large-scale cosmological simulations where many realizations of the universe are required. This capability demonstrates that our approach can be efficiently scaled to handle even larger datasets or more complex tasks, such as incorporating additional physical effects or generating higher-resolution maps in future studies

5.2 Conclusions

This thesis set out with the ambitious goal of generating cosmological maps using GANs, focusing on weak lensing maps with the eventual hope of adapting the model to generate SZ effect maps. While we initially aimed to diversify the range of cosmological phenomena our GAN could simulate, we found that working with weak lensing maps alone provided enough challenges to focus our efforts on a single objective. The generation of high-quality weak lensing maps represents a crucial first step, demonstrating the power and flexibility of generative models in a field like cosmology, where data is often expensive to obtain or simulate.

The process was not without its difficulties, especially in stabilizing the training process of the GAN. We employed various techniques, and adjusted the architecture and training parameters to ensure that both the generator and discriminator developed the necessary sophistication to produce realistic images. Despite these adjustments, the results show certain limitations, especially when it comes to fine-scale features as seen in the power spectrum and Minkowski functional comparisons. This indicates that while our GAN framework was successful in many ways, it could benefit from further refinement, particularly through even more regularization methods.

Nevertheless, this work has achieved its core objective: the generation of weak lensing maps that not only approximate real cosmological data but also offer a promising path forward for future research. These results could serve as a base for more advanced studies, including the generation of SZ effect maps, as originally envisioned.

Looking back, the effort invested in understanding and adapting the GAN to cosmological applications has been rewarding, not just in terms of the maps generated but in the insights gained into how machine learning can interface with complex scientific data. The progress made here, while representing a first attempt, lays the groundwork for more sophisticated models and more diverse datasets in future research.

The challenge now is to ensure that these generated maps are scientifically ”safe” and useful for real-world applications. Their potential lies not just in augmenting datasets for machine learning but in offering new ways to simulate and explore the universe. However, this is still a model that generates maps based on what it has learned from existing data, and this data-driven process must be critically examined in future studies. What are the limitations of using GAN-generated maps in a scientific framework? This remains a key question for future exploration.

In conclusion, while the future applications still lie ahead, this thesis marks a successful first venture into applying GANs to cosmology. The results, though imperfect, demonstrate both the promise of generative models and the opportunities for further research and refinement. As GANs continue to evolve, so too will their ability to contribute meaningfully to cosmology.

Bibliography

[1] P. Peebles, *Principles of Physical Cosmology.* Princeton University Press, 1993.

[2] S. Dodelson, *Modern Cosmology.* San Diego, CA: Academic Press, 2003.

[3] P. Collaboration *et al.*, "Planck 2018 results. vi. cosmological parameters," *Astronomy & Astrophysics*, vol. 641, p. A6, 2020.

[4] E. W. Kolb and M. S. Turner, *The Early Universe.* Addison-Wesley, 1990.

[5] M. Bartelmann and P. Schneider, "Weak gravitational lensing," *Physics Reports*, vol. 340, no. 4-5, pp. 291–472, 2001. [Online]. Available: https://arxiv.org/abs/astro-ph/9912508

[6] D. Collaboration, "Dark energy survey year 1 results: Cosmological constraints from galaxy clustering and weak lensing," *Monthly Notices of the Royal Astronomical Society*, 2018.

[7] R. A. Sunyaev and Y. B. Zel'dovich, "The velocity of clusters of galaxies relative to the microwave background - the possibility of its measurement," *Monthly Notices of the Royal Astronomical Society*, vol. 190, no. 3, pp. 413–420, 1980.

[8] K. Schawinski, C. Zhang, H. Zhang, L. Fowler, and G. K. Santhanam, "Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit," *Monthly Notices of the Royal Astronomical Society*, vol. 467, no. 1, pp. L110–L114, 2017. [Online]. Available: https://doi.org/10.1093/mnrasl/slx008

[9] S. Ravanbakhsh, J. B. Oliva, S. Fromenteau, L. Matthey, B. Moews, D. Sullivan, ..., and B. Poczos, "Estimating cosmological parameters from the dark matter distribution," *Proceedings of the National Academy of Sciences*, vol. 114, no. 48, pp. 12 496–12 501, 2017. [Online]. Available: https://arxiv.org/abs/1711.02033

[10] I. Goodfellow *et al.*, "Generative adversarial networks," *Communications of the ACM*, 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3422622#core-history

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* Cambridge, MA: MIT Press, 2016.

[12] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015. [Online]. Available: https://arxiv.org/pdf/1511.06434

[13] M. Mustafa *et al.*, “Cosmogan: Creating high-fidelity weak lensing convergence maps using generative adversarial networks,” *Computational Astrophysics and Cosmology*, vol. 6, no. 1, pp. 1–14, 2019. [Online]. Available: https://arxiv.org/abs/1706.02390

Appendix A

Training Code

A.1 Training Code

The following code shows the full process for training the GAN used in this project:

```


import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader, Subset
from torchvision.utils import make_grid, save_image
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim
import time

data_dir = '/mnt/new_volume/gan_training/data'
dataset_files = [f'cosmogan_maps_256_8k_{i}.npy' for i in range(1,
    16)]
sample_dir = '/mnt/new_volume/gan_training/generated'
os.makedirs(sample_dir, exist_ok=True)

class CosmoDataset(Dataset):
    def __init__(self, data_dir, dataset_files):
        self.data_dir = data_dir
        self.dataset_files = dataset_files
        self.data_lengths = []
        self.total_length = 0

        for dataset_file in dataset_files:
            data = np.load(os.path.join(data_dir, dataset_file),
                mmap_mode='r')
            length = data.shape[0]
            self.data_lengths.append(length)
            self.total_length += length

    def __len__(self):
        return self.total_length

    def __getitem__(self, idx):
        cumulative_length = 0
        for i, length in enumerate(self.data_lengths):
            if idx < cumulative_length + length:
                file_idx = i
                sample_idx = idx - cumulative_length
                break
            cumulative_length += length

        dataset_file = os.path.join(self.data_dir, self.dataset_files[file_idx])
        data = np.load(dataset_file, mmap_mode='r')
        sample = data[sample_idx]
        sample = torch.tensor(np.array(sample[np.newaxis, :, :]), dtype=torch.float32)
        return sample

def show_images(images, nmax=64):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(images.detach()[:nmax], nrow=8).permute(1, 2, 0), cmap='gray')
    plt.show()

def get_default_device():
    return torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

device = get_default_device()

def to_device(data, device):
    if isinstance(data, (list, tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        return len(self.dl)

class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=5, stride=2, padding=2, bias=False)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=5, stride=2, padding=2, bias=False)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=5, stride=2, padding=2, bias=False)
        self.bn3 = nn.BatchNorm2d(256)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=5, stride=2, padding=2, bias=False)
        self.bn4 = nn.BatchNorm2d(512)
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(512 * 16 * 16, 1)

    def forward(self, x):
        x = F.leaky_relu(self.conv1(x), 0.2, inplace=True)
        x = F.leaky_relu(self.bn2(self.conv2(x)), 0.2, inplace=True
            )
        x = F.leaky_relu(self.bn3(self.conv3(x)), 0.2, inplace=True
            )
        x = F.leaky_relu(self.bn4(self.conv4(x)), 0.2, inplace=True
            )
        x = self.flatten(x)
        x = self.fc(x)
        return x

class Generator(nn.Module):
    def __init__(self, latent_size):
        super(Generator, self).__init__()
        self.fc = nn.Linear(latent_size, 512 * 16 * 16, bias=False)
        self.bn1 = nn.BatchNorm1d(512 * 16 * 16)
        self.relu = nn.ReLU(True)

        self.deconv1 = nn.ConvTranspose2d(512, 256, kernel_size=5,
            stride=2, padding=2, output_padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(256)
        self.deconv2 = nn.ConvTranspose2d(256, 128, kernel_size=5,
            stride=2, padding=2, output_padding=1, bias=False)
        self.bn3 = nn.BatchNorm2d(128)
        self.deconv3 = nn.ConvTranspose2d(128, 64, kernel_size=5,
            stride=2, padding=2, output_padding=1, bias=False)
        self.bn4 = nn.BatchNorm2d(64)
        self.deconv4 = nn.ConvTranspose2d(64, 1, kernel_size=5,
            stride=2, padding=2, output_padding=1, bias=False)
        self.tanh = nn.Tanh()

    def forward(self, x):
        x = self.fc(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = x.view(-1, 512, 16, 16)
        x = self.relu(self.bn2(self.deconv1(x)))
        x = self.relu(self.bn3(self.deconv2(x)))
        x = self.relu(self.bn4(self.deconv3(x)))
        x = self.tanh(self.deconv4(x))
        return x

latent_size = 100
batch_size = 64

discriminator = to_device(Discriminator(), device)
generator = to_device(Generator(latent_size), device)

loss_fn = nn.BCEWithLogitsLoss()

lr = 1e-7
opt_d = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5,
    0.999))
```

```
opt_g = optim.Adam(generator.parameters(), lr=lr, betas=(0.5,
    0.999))

def train_discriminator(real_images, opt_d):
    opt_d.zero_grad()

    real_images = real_images.to(device)
    real_preds = discriminator(real_images)
    real_targets = torch.ones(real_images.size(0), 1, device=device
        )
    real_loss = loss_fn(real_preds, real_targets)
    real_score = torch.mean(torch.sigmoid(real_preds)).item()

    latent = torch.randn(batch_size, latent_size, device=device)
    fake_images = generator(latent).detach()
    fake_preds = discriminator(fake_images)
    fake_targets = torch.zeros(fake_images.size(0), 1, device=
        device)
    fake_loss = loss_fn(fake_preds, fake_targets)
    fake_score = torch.mean(torch.sigmoid(fake_preds)).item()

    loss = real_loss + fake_loss
    loss.backward()
    opt_d.step()

    return loss.item(), real_score, fake_score

def train_generator(opt_g):
    opt_g.zero_grad()
    latent = torch.randn(batch_size, latent_size, device=device)
    fake_images = generator(latent)
    preds = discriminator(fake_images)
    targets = torch.ones(batch_size, 1, device=device)
    loss = loss_fn(preds, targets)
    loss.backward()
    opt_g.step()
    return loss.item()

generator.load_state_dict(torch.load('/mnt/new_volume/gan_training/
    generated/generator_final4p1.pth', map_location=device))
discriminator.load_state_dict(torch.load('/mnt/new_volume/
    gan_training/generated/discriminator_final4p1.pth', map_location
    =device))

train_dataloader = DataLoader(CosmoDataset(data_dir, dataset_files)
    , batch_size=batch_size, shuffle=True, num_workers=8)
train_device_dataloader = DeviceDataLoader(train_dataloader, device
    )

def fit(epochs, lr, train_dataloader):
    history = []
    start_time = time.time()

    for epoch in range(epochs):
        for i, real_images in enumerate(train_dataloader):
            loss_d, real_score, fake_score = train_discriminator(
                real_images, opt_d)
            loss_g = train_generator(opt_g)

            if (i+1) % 100 == 0:
                elapsed_time = time.time() - start_time
                latent_tensors = torch.randn(batch_size, latent_size, device=device)
                save_samples(epoch*len(train_dataloader) + i, latent_tensors)

        val_loss = validate(generator, train_dataloader, device)
        history.append((loss_d, loss_g, real_score, fake_score, val_loss))

    return history

fit(epochs=30, lr=lr, train_dataloader=train_device_dataloader)
```