NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Generating realistic nanorough surfaces via a Generative Adversarial Network

Vassileios G. Sioros

Supervisors: **George Giannakopoulos,** Research Fellow
Vassileios Constantoudis, Researcher
Panagiotis Stamatopoulos, Assistant Professor

ATHENS

OCTOBER 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δημιουργία ρεαλιστικών νανοδομημένων επιφανειών μέσω ενός Παραγωγικού Δικτύου Αντιπαράθεσης

Βασίλειος Γ. Σιώρος

Επιβλέποντες: **Γεώργιος Γιαννακόπουλος,** Επιστημονικός Συνεργάτης
Βασίλειος Κωνσταντούδης, Ερευνητής
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2021

BSc THESIS

Generating realistic nanorough surfaces via a Generative Adversarial Network

Vassileios G. Sioros
S.N.: 1115201500144

SUPERVISORS: **George Giannakopoulos,** Research Fellow
Vassileios Constantoudis, Researcher
Panagiotis Stamatopoulos, Assistant Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δημιουργία ρεαλιστικών νανοδομημένων επιφανειών μέσω ενός Παραγωγικού Δικτύου Αντιπαράθεσης

Βασίλειος Γ. Σιώρος

Α.Μ.: 1115201500144

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Γεώργιος Γιαννακόπουλος,** Επιστημονικός Συνεργάτης

Βασίλειος Κωνσταντούδης, Ερευνητής

Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής

ABSTRACT

Generating artificial nanorough surfaces in the context of a multi-physics simulation requires (1) identifying the structural feature space so that the generation of new nanorough surfaces is possible and (2) the reconstruction process to be property-preserving. In this work, we examine the possibility of providing multi-physics simulations with a computationally inexpensive way of integrating new nanorough surfaces similar to a predefined sample of surfaces. We focus on how a Generative Adversarial Network (GAN) based approach, given a nanorough surface data set, can learn to produce statistically equivalent samples. Additionally, we examine how pairing our model with a set of nanorough similarity metrics, can improve the realisticity of the resulting nanorough surfaces. We showcase via multiple experiments that our framework is able to produce sufficiently realistic nanorough surfaces, in many cases indistinguishable from real data. The complete source code is available at https://github.com/billsioros/RoughML.

SUBJECT AREA: Machine Learning

KEYWORDS: Nanotechnology, Machine Learning, Graph Theory

ΠΕΡΙΛΗΨΗ

Η δημιουργία τεχνητών νανοδομημένων επιφανειών στο πλαίσιο μιας φυσικο-χημικής προσομοίωσης απαιτεί (1) να προσδιοριστεί ο χώρος των δομικών χαρακτηριστικών, ώστε να επιτραπεί η ανακατασκευή νέων, ρελιστικών επιφανειών και (2) η διαδικασία ανακατασκευής να διατηρεί τις ιδιότητες των επιφανειών. Σε αυτή την εργασία, εξετάζουμε τη δυνατότητα να παρέχουμε στις φυσικο-χημικές προσομοιώσεις μια υπολογιστικά φθηνή λύση όσον αφορά την δημιουργία και ενσωμάτωση νέων νανοδομημένων επιφανειών παρόμοιων με ένα προκαθορισμένο σύνολο επιφανειών. Επικεντρωνόμαστε στο πώς ένα Παραγωγικό Δίκτυο Αντιπαράθεσης (ΠΔΑ), δεδομένου ενός συνόλου νανοδομημένων επιφανειών, είναι ικανό να μάθει να παράγει στατιστικά ισοδύναμα δείγματα επιφανειών. Στη συνέχεια, εξετάζουμε πώς ο συνδυασμός του μοντέλου μας με ένα σύνολο μετρικών ομοιότητας νανοδομημένων επιφανειών, έχει ως αποτέλεσμα πιο ρεαλιστικές νανοδοημένες επιφάνειες. Αποδεικνύουμε μέσω πολλαπλών πειραμάτων, ότι το σύστημά μας είναι σε θέση να παράγει αρκετά ρεαλιστικές νανοδομημένες επιφάνειες, τις οποίες, σε πολλές περιπτώσεις, είναι αδύνατον να διακρίνει κανείς από τις πραγματικές. Ο πλήρης πηγαίος κώδικας είναι διαθέσιμος στην ηλεκτρονική διεύθυνση https://github.com/billsioros/RoughML.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Νανοτεχνολογία, Μηχανική Μάθηση, Θεωρία Γράφων

ACKNOWLEDGEMENTS

I would like to wholeheartedly thank **Dr.Georgios Giannakopoulos** and **Dr.Vassilios Kostantoudis** for their valuable input and considerable support in completing this thesis.

CONTENTS

1. INTRODUCTION 13

2. BACKGROUND AND RELATED WORK 15

2.1 Nanotechnology . . . 15
2.1.1 Nanoelectronics . . . 15
2.1.2 Nanofabrication . . . 15

2.2 Nanometrology . . . 16
2.2.1 Structural Characteristics of Nanorough Surfaces . . . 16
2.2.1.1 Vertical Parameters . . . 16
2.2.1.2 Horizontal Parameters . . . 17

2.3 Machine Learning . . . 18

2.4 Deep Learning . . . 18
2.4.1 An Artificial Neuron . . . 18
2.4.2 Single-Layer Perceptron Network (SLP) . . . 19
2.4.3 Convolutional Neural Network (CNN) . . . 19
2.4.3.1 Convolution . . . 20
2.4.3.2 Convolutional Layer . . . 21
2.4.4 Activation Functions . . . 21
2.4.5 Back Propagation . . . 22
2.4.5.1 Mathematical Statement . . . 22
2.4.6 Gradient Descent . . . 24
2.4.6.1 Mathematical Statement . . . 25
2.4.7 Extensions and Variants of Gradient Descent . . . 26
2.4.7.1 Stochastic Gradient Descent . . . 26
2.4.7.2 Momentum . . . 27
2.4.7.3 RMSProp . . . 27
2.4.7.4 Adam . . . 27
2.4.8 Batch Normalization . . . 28
2.4.8.1 Mathematical Statement . . . 28
2.4.8.2 Limitations and Hindrances . . . 30
2.4.9 Generative Adversarial Network (GAN) . . . 30
2.4.9.1 Mathematical Statement . . . 30
2.4.9.2 Training . . . 31

2.5 N-Gram Graphs . . . 32
2.5.1 Similarity Metrics . . . 33
2.5.2 Variants . . . 33
2.5.2.1 Merged Model Graph . . . 33
2.5.2.2 Hierarchical Proximity Graph . . . 34

2.6 State of the Art . . . 35

3. METHODOLOGY **37**

3.1 Problem Definition . . . 37

3.2 System Overview . . . 38
3.2.1 Nanorough Surface Generation . . . 39
3.2.2 Nanorough Surface Quantization . . . 40
3.2.3 Content Similarity Metrics . . . 42
3.2.3.1 The N-Gram Graph Content Similarity Metric (NGG) . . . 42
3.2.3.2 The Two-Dimensional Array Graph Content Similarity Metric (A2G) . . . 43
3.2.3.3 The Hierarchical Proximity Graph Content Similarity Metric (HPS) . . . 44
3.2.3.4 The Fourier & Histogram Space Content Similarity Metric (FHS) . . . 46
3.2.4 Frameworks . . . 46
3.2.4.1 Single-Layer Perceptron GAN (SLPGAN) . . . 47
3.2.4.2 Deep Convolutional GAN (DCGAN) . . . 48
3.2.4.3 Training . . . 49

4. EXPERIMENTAL RESULTS **51**

4.1 Experimental Setup . . . 51
4.1.1 DCGAN Weight Initialization . . . 52
4.1.2 Evaluating the scalability of the Content Similarity Metrics . . . 53
4.1.2.1 Scalability of the N-Gram Graph Content Similarity Metric . . . 53
4.1.2.2 Scalability of the Two-Dimensional Array Graph Content Similarity Metric . . . 54
4.1.2.3 Scalability of the Hierarchical Proximity Graph Content Similarity Metric . . . 54
4.1.3 Evaluating the FHS Content Similarity Metric . . . 55

4.2 Results and Discussion . . . 56
4.2.1 SLPGAN paired with A2G . . . 56
4.2.2 DCGAN . . . 58
4.2.3 DCGAN paired with NGG . . . 59
4.2.4 DCGAN paired with A2G . . . 60
4.2.5 Evaluating the statistical significance of our results . . . 62
4.2.6 Training DCGAN paired with A2G on additional data sets . . . 63
4.2.7 Determining the minimal amount of training data required . . . 66
4.2.8 Assessing the scalability of our framework . . . 68

5. CONCLUSIONS AND FUTURE WORK **69**

ABBREVIATIONS - ACRONYMS **71**

REFERENCES **74**

LIST OF FIGURES

2.1 An artificial neuron . . . 19
2.2 Convolution with a stride of 1 and no padding . . . 20

3.1 System overview . . . 38
3.2 Comparing Uniform and Quantile Binning . . . 40
3.3 Nanorough Surface Quantization . . . 41
3.4 The SLPGAN framework . . . 47
3.5 The DCGAN framework . . . 48

4.1 Real nanorough surface samples ($\xi_y = 8, \xi_x = 8, \alpha = 1$) . . . 52
4.2 Training DCGAN with the conventional weight initialization scheme . . . 53
4.3 NGG scalability . . . 53
4.4 A2G scalability . . . 54
4.5 HPS scalability . . . 54
4.6 HPS scalability (Additional cases) . . . 55
4.7 Comparing the FHS values of real and artificial nanorough surface samples ($\xi_y = 8, \xi_x = 8, \alpha = 1$) . . . 56
4.8 Training SLPGAN . . . 56
4.9 Nanorough surface samples generated by SLPGAN . . . 57
4.10 A2G & FHS similarity scores in the case of SLPGAN ($\xi_y = 8, \xi_x = 8, \alpha = 1$) . . . 57
4.11 Training DCGAN . . . 58
4.12 Nanorough surface samples generated by DCGAN . . . 58
4.13 Training DCGAN paired with NGG . . . 59
4.14 Nanorough surface samples generated by DCGAN+NGG . . . 59
4.15 A2G & FHS similarity scores in the case of DCGAN+NGG ($\xi_y = 8, \xi_x = 8, \alpha = 1$) . . . 60
4.16 Training DCGAN paired with A2G . . . 60
4.17 Nanorough surface samples generated by DCGAN+A2G . . . 61
4.18 A2G & FHS similarity scores in the case of DCGAN+A2G ($\xi_y = 8, \xi_x = 8, \alpha = 1$) . . . 61
4.19 The FHS score populations used in the context of the Wilcoxon signed-rank tests . . . 62
4.20 Real nanorough surface samples ($\alpha = 0.5$) . . . 64
4.21 Nanorough surface samples generated by DCGAN+A2G ($\alpha = 0.5$) . . . 64
4.22 Real nanorough surface samples ($\alpha = 1$) . . . 65
4.23 Nanorough surface samples generated by DCGAN+A2G ($\alpha = 1$) . . . 65
4.24 Training DCGAN paired with A2G ($\xi_y = 4, \xi_x = 4, \alpha = 1$) . . . 66
4.25 Training DCGAN ($\xi_y = 4, \xi_x = 4, \alpha = 1$) . . . 66
4.26 Comparing nanorough surface samples generated by DCGAN+A2G ($\alpha = 1$) with respect to the size of the training data set . . . 67
4.27 A2G & FHS scores for different training data set sizes . . . 67
4.28 The generation cost as a function of the desired number of nanorough surfaces . . . 68

LIST OF TABLES

4.1 The results of the two-tailed Wilcoxon signed-rank test with regards to different sample combinations 63
4.2 The results of the one-tailed Wilcoxon signed-rank test with regards to different sample combinations 63

PREFACE

The present thesis was carried out in collaboration with the National Center of Scientific Research **Demokritos**.

1. INTRODUCTION

In this work, we examine the possibility of providing multi-physics simulations with a computationally inexpensive way of integrating new nanorough surfaces, similar to the ones being measured.

Modeling nanorough surfaces, requires (1) identifying the structural feature space so that the generation of new nanorough surfaces is possible and (2) the nanorough surface reconstruction process to be property-preserving, meaning that newly constructed nanorough surfaces should showcase structural properties similar to the the ones being modeled.

One additional requirement would be that the system is nanorough-surface-configuration-agnostic. This would enable the system to model a set of nanorough surfaces with no a priori knowledge of the underlying characteristics of the nanorough surfaces.

The essential idea is, given a set of nano-structures (in some digital format) to develop a method, which is able to learn the stochastic nature of their morphology by fitting a supervised learning model to the data set. This model can then be subsequently used to construct nano-structures with similar structural properties.

In this work:

1. We examine how Generative Adversarial Network (GAN) [1] based frameworks can be trained to generate realistic nanorough surfaces. We develop 2 different flavors of the GAN framework. We use a plethora of data sets corresponding to a variety of nanorough surface populations in order to train our models and examine how well they are able to adapt to different levels of stochasticity and correlation.

2. We develop 3 graph-based nanorough surface similarity metrics, which will provide additional feedback to the GAN model at hand, throughout the training process. We evaluate these metrics with regards to their computational cost and their effect on the realisticity of the resulting nanorough surfaces.

3. We design a novel nanorough surface similarity metric, which is used to evaluate the quality of the synthetic nanorough surfaces. Using the established similarity measures, we showcase that our method is able to generate nanorough surfaces virtually indistinguishable from real data.

4. We determine that a Deep Convolutional Generative Adversarial Network (DCGAN) [2] paired with one of our graph-based similarity metrics further improves on the case of merely utilizing a DCGAN, through multiple Wilcoxon signed-rank tests.

5. We investigate how the size of the training data set and the structural parameters of the nanorough surfaces making it up affect the quality of the synthetic nanorough surfaces, and note some limitations of our framework with regards to different degrees of correlation, stochasticity, and smoothness of the input nanorough surfaces.

This document is organized as follows; *Section 2* introduces the reader to basic concepts and ideas that are related to the problem at hand rather than completely reviewing the domain. *Section 3* provides a more formal description of the problem and elaborates on our technical approach. *Section 4* showcases the results of our various experiments. We look into (1) the computational cost of the various nanorough similarity metrics, (2) how different

combinations of GAN flavors and nanorough surface similarity metrics affect the quality of the synthetic nanorough surfaces, (3) the behavior of our framework when being trained on different data sets corresponding to varying amounts of correlation, stochasticity, and smoothness, and (4) the scalability of our framework with regards to nanorough surface generation. In *Section 5*, we conclude our work with findings regarding the advantages and limitations of our framework, and propose future research avenues.

2. BACKGROUND AND RELATED WORK

2.1 Nanotechnology

Nanotechnology may be defined as **the use of matter on an atomic, molecular, and supramolecular scale for industrial purposes**. To be more specific, nanotechnology encompasses the design, construction as well as operation of devices and systems, that contain materials the structural elements of which have dimensions less than 100 nm.

This definition reflects the fact that quantum mechanical effects are important at this quantum-realm scale, and so the definition shifted from a particular technological goal to a research category inclusive of all types of research and technologies that deal with the special properties of matter occurring below the given size threshold.

Nanotechnology can be separated into three main areas:

- **Nanoelectronics**, which is an evolution of microelectronics, refers to the use of nanotechnology in electronic components. The term covers a diverse set of devices and materials, with the common characteristic that they are so small that inter-atomic interactions and quantum mechanical properties need to be studied extensively.
- **Nanomedicine**, which is the medical application of nanotechnology and ranges from the medical applications of nanomaterials and biological devices to nanoelectronic biosensors.
- **Nanomaterials**, where materials with nanostructured surfaces or nanostructures are developed and investigated. Materials with structure at the nanoscale often have unique optical, electronic, thermo-physical, or mechanical properties.

2.1.1 Nanoelectronics

In 1965, Gordon Moore observed that the size of silicon transistors were undergoing a continual process of scaling downward, an observation which was later codified as **Moore's Law**. Since his observation, transistor minimum feature sizes have decreased from 10 micrometers to the 10nm range as of 2019.

The performance of an electronic device, and microchips, in particular, is determined by the number of transistors, which make them up. A large number of transistors corresponds to increased performance.

Nanoelectronics holds the promise of making computer processors more powerful than is possible with conventional semiconductor fabrication techniques. Several approaches are currently being researched, including new forms of nanolithography, as well as the use of nanomaterials such as nanowires or small molecules in place of traditional CMOS components.

2.1.2 Nanofabrication

There is no single accepted definition of nanofabrication, nor a definition of what separates nanofabrication from microfabrication. To meet the continuing challenge of shrinking

component size in microelectronics, new tools and techniques are being continuously developed. Component sizes went from tens of micrometers, to single-digit micrometers, to hundreds of nanometers, and finally to a few tens of nanometers where they stand today. As a result, what used to be called microfabrication was rebranded as nanofabrication, although the governing principles have remained essentially the same. The main driver of this technology has been the manufacture of integrated circuits, but there have been tremendous side benefits to other areas, including photonics.

Nanofabrication approaches can be separated into two main categories:

- **Bottom-up or self-assembly** approaches to nanofabrication use chemical or physical forces operating at the nanoscale to assemble basic units into larger structures. Researchers hope to replicate nature's ability to produce small clusters of specific atoms, which can then self-assemble into more-elaborate structures.
- **Top-down** approaches involve the breaking down of the bulk material into nanosized structures or particles. Top-down approaches are inherently simpler, compared to Bottom-up approaches. They depend either on the removal or division of bulk material or on miniaturization of bulk fabrication processes to produce the desired structure with appropriate properties.

2.2 Nanometrology

Nanometrology is a subfield of metrology, concerned with the science of measurement at the nanoscale level. Having manufactured a nanostructure, its structural characterization is required, before its application. Nanometrology significantly contributes to the production of accurate and reliable nanomaterials and devices.

The structural qualities of a nanostructure dramatically affect its functionality. The measurements, that are carried out in the context of **Nanometrology**, concern the geometric characteristics of measurements (height, width, roughness, etc.), as well as their chemical compounds, physical properties, and interactions with the environment.

2.2.1 Structural Characteristics of Nanorough Surfaces

The structural characteristics of a nanorough surface can be separated into those that characterize the distribution of its heights (or vertical parameters) and those that characterize the correlation of its points on a two-dimensional coordinate system (or horizontal parameters).

2.2.1.1 Vertical Parameters

Moments are a set of quantitative measures describing the shape of a given distribution. Assuming a multivariate real-valued discrete series $z(k,l)$, its n_{th} moment is given by:

$$u_n = \sum_{k=0}^{n-1} \sum_{l=0}^{m-1} \frac{(z_{k,l} - \bar{z})^n}{mn} \tag{2.1}$$

Using 2.1, we are able to calculate all the moments of $z(k,l)$. The first moment corresponds to the expected value, the second central moment to the variance, the third standardized moment to the skewness, and the fourth standardized moment to the kurtosis of $z(k,l)$:

$$\text{Mean} = \mu = \sum_{k=0}^{n-1}\sum_{l=0}^{m-1}\frac{z_{k,l}-\bar{z}}{mn} \tag{2.2}$$

$$\text{Standard Deviation} = \sigma = \sqrt{\sum_{k=0}^{n-1}\sum_{l=0}^{m-1}\frac{(z_{k,l}-\bar{z})^2}{mn}} \tag{2.3}$$

$$\text{Skewness} = \mathbb{S} = \sum_{k=0}^{n-1}\sum_{l=0}^{m-1}\frac{(z_{k,l}-\bar{z})^3}{\sigma^3} \tag{2.4}$$

$$\text{Kurtosis} = \mathbb{K} = \sum_{k=0}^{n-1}\sum_{l=0}^{m-1}\frac{(z_{k,l}-\bar{z})^4}{\sigma^4} \tag{2.5}$$

A nanorough surface can be interpreted as a multivariate real-valued discrete series $z(k,l)$, that maps different 2D coordinates y,x to the height of the nanorough surface on the specific coordinates $z(y,x)$. Hence, we can characterize it using the statistical moments 2.2-2.5.

2.2.1.2 Horizontal Parameters

Other than the aforementioned statistical measures, the correlation length of a given surface should be taken into consideration during its study. The correlation length can be defined as a measure of the constraint between height displacements of neighboring points of the surface. This constraint is expected to be significant if two points are well inside the correlation length and negligible outside it.

The correlation length of a given nanorough surface is determined by its **Autocorrelation Function (ACF)**. Autocorrelation is the correlation of a signal with a delayed copy of itself as a function of the delay.

Given the profile of a nanorough surface, i.e. a discrete normalized height function $y(x)$, the ACF is given by the following equation:

$$ACF(r_x) = \frac{1}{\sigma^2(l-r_x)}\sum_{1}^{l-r_x}(y(x)-\langle y\rangle)(y(x+r_x)-\langle y\rangle) \tag{2.6}$$

where l is the length of the profile in the direction of the horizontal axis and r_x is the distance between 2 points of the profile.

The ACF for small values of r_x can be expressed in exponential form as such:

$$ACF(r_x) = \exp(\frac{-r_x}{\beta}) \tag{2.7}$$

The **correlation length** of a given nanorough surface, is the length where the ACF has

decreased by a specific percentage compared to its original value. Usually, the desired percentage is around 10%, where $ACF = 0.1$.

2.3 Machine Learning

The term **Machine learning (ML)** describes a set of computer algorithms that can improve automatically through the use of data. Machine learning algorithms ”learn” to make predictions or decisions without being explicitly programmed to do so.

Machine learning algorithms aim at modeling complex functions and can be divided into two broad categories, the **Supervised Learning (SL)** and the **Unsupervised Learning (UL)** algorithms. We are going to be focusing on SL, and more specifically **Deep Learning** methods.

2.4 Deep Learning

Deep learning (also known as Deep Structured Learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning.

Representation learning or Feature Learning is a set of techniques that allows a system to automatically discover the representations needed for feature detection from raw data. This replaces manual feature engineering, which is the process of using domain knowledge to extract features (characteristics, properties, attributes) from raw data, and allows a machine to both learn the features and use them to perform a specific task. By the term feature, we refer to an individual measurable property or characteristic of a phenomenon. Features are usually numeric, but structural features such as strings and graphs can also be used. The concept of ”feature” is related to that of explanatory variables used in statistical techniques such as linear regression.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs, though are quite different from biological brains. ANNs are comprised of an input layer, one or more hidden layers, and an output layer. Each node or artificial neuron has inputs and produces a single output that can be sent to multiple other neurons. The inputs can be the feature values of a sample of external data, such as images or documents, or they can be the outputs of other neurons. The outputs of the final output neurons of the neural net accomplish the task, such as recognizing an object in an image.

An ANN wherein connections between the nodes do not form cycles or loops, is referred to as **Feed-Forward Neural Network**. The Feed-Forward Neural Network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward from the input nodes, through the hidden nodes (if any), and to the output nodes.

2.4.1 An Artificial Neuron

Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output. Each input is separately

weighted, and the sum is passed through a non-linear function known as an activation function.

Other than the neuron's weights, another term is added to the total sum before being passed through the activation function. This term is the so-called **bias**. Bias allows you to shift the activation function, analogously to a constant in the context of a linear function, whereby the line is effectively transposed by the constant value.

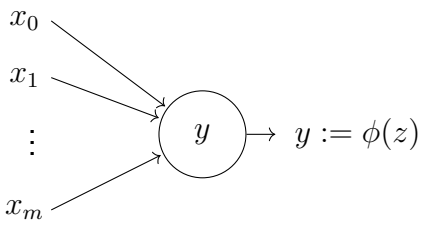


Figure 2.1: An artificial neuron. This visualization was produced using code adapted from *David Stutz*'s work [3].

2.4.2 Single-Layer Perceptron Network (SLP)

The simplest kind of neural network is a **Single-Layer Perceptron Network**, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. The sum of the products of the weights and the inputs is calculated in each node and passes through a, commonly non-linear, function. Single-layer perceptrons are only capable of learning linearly separable patterns.

For a given artificial neuron k, let there be $m + 1$ inputs with signals x_0 through x_m and weights $w_{k,0}$ through $w_{k,m}$. To achieve a bias inclusive representation, the x_0 input is assigned the value $+1$ and corresponds to the neuron's bias, with $w_{k,0} = b_k$. Then the output of neuron k is given by the following equation:

$$y_k = \phi(\sum_{j=0}^{m} w_{k,j} x_j) \tag{2.8}$$

where ϕ stands for the activation function of choice. This operation is demonstrated by *Figure 2.1*, where k is left out as we are demonstrating the case of a single neuron.

2.4.3 Convolutional Neural Network (CNN)

A **Convolutional Neural Network (CNN)** [4, 5] is a class of artificial neural networks, that take advantage of the hierarchical structure of data, assembling patterns of increasing complexity using smaller and simpler ones.

CNNs were inspired by the primary visual cortex of the brain, which is responsible for processing visual information. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the *receptive field*. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

2.4.3.1 Convolution

Assuming data with a grid-like topology, **Convolution** refers to the process of passing a sliding window of predetermined size over the data, and computing the dot product of a small matrix of numbers, better known as *kernel* or *filter*, with each sub-matrix of the input data. The resulting matrix is most commonly referred to as *feature map*.

Denoting the input as I, the kernel as K, and the feature map as F, convolution is described by the following equation:

$$F[m,n] = (I \cdot K)[m,n] = \sum_j \sum_k I[m-j, n-k] \times K[j,k] \tag{2.9}$$

The convolution process (*Figure 2.2*) is controlled by two hyperparameters, namingly the *padding* and the *stride*. Stride controls by how much we shift the convolution kernel. More specifically, for any integer $s > 0$ a stride s means that the kernel is translated s units at a time. A stride of 1 leads to heavily overlapping receptive fields between the columns, and a large output volume. A greater stride means a smaller overlap of receptive fields and smaller spatial dimensions of the output volume. In practice, $s \geq 3$ is quite rare. Sometimes, it is convenient to pad the input with zeros (or other values, such as the average of the respective region) on the border of the input volume. Padding determines the spatial size of the output volume.

Assuming stride s and padding p, the dimension of the output feature map is given by the following expression:

$$n_{out} = \lfloor \frac{n_{in} + 2 \times p - f}{s} + 1 \rfloor \tag{2.10}$$

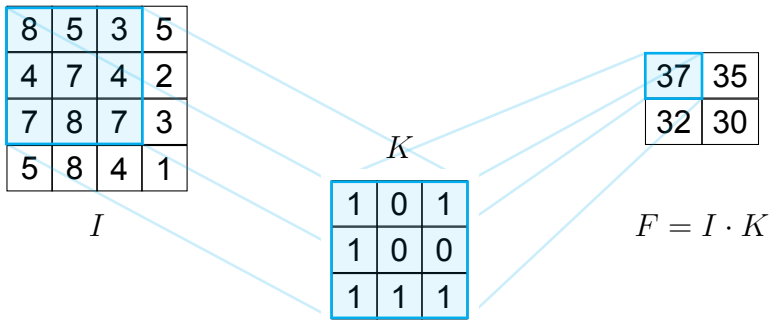


Figure 2.2: Convolution with a stride of 1 and no padding

In case, more than one kernel is to be applied, the convolution process is separately carried out for each one, and the results are stacked into a single three dimensional matrix. It is important that the kernel(s) have the same number of channels as the input. By *channels* we refer to the depth of the input data. Two dimensional data are treated as having a single channel. For example, in the case of RGB images, there are 3 channels, one for each color.

Denoting n as the size of the input data k as the kernel size, n_c as the number of channels of the input data, and n_k as the number of kernels to be applied on the data, the dimensions

of the output feature map are given by the following expression:

$$[n, n, n_c] \cdot [k, k, n_c] = [\lfloor \frac{n_{in} + 2 \times p - f}{s} + 1 \rfloor, \lfloor \frac{n_{in} + 2 \times p - f}{s} + 1 \rfloor, n_k] \tag{2.11}$$

2.4.3.2 Convolutional Layer

The **Convolutional Layer** is the core building block of the CNN architecture. A CNN is is constructed by stacking such along with other types (activation function, min/max/average pooling, etc.) of layers.

A convolutional layer receives a block of input feature maps, convolves it using a set of learnable kernels, and generates a block of output feature maps. These kernels activate when the convolutional layer detects a specific type of feature at some spatial position in the input. Different kernels learn to activate for different features. A certain combination of features in a certain area can signal a larger, more complex feature. For example, in the case of visual imagery, detecting a set of curves might result in detecting a set of circles (a combination of curves), which consequently might result in detecting a bicycle (a combination of line and circle features), and so on.

During the forward pass, each kernel is convolved across the width and height of the input volume. A bias term is optionally added to expression 2.9, and the result is stored in the output feature map. Every entry in the output volume can thus be interpreted as an output of a neuron that examines only a small region of the input data and shares parameters with other neurons in the same feature map. Moreover, the number of output channels determines the number of neurons that connect to the same region of the input volume. Hence, not all neurons in two consecutive layers are connected to each other.

Fully connected feed-forward neural networks are generally impractical for large inputs, such as high-resolution images, where each pixel is a relevant input feature, as it would require a tremendous number of neurons, even in the case of a shallow architecture. In this scenario, CNNs are a preferable option as connections are local in space and neurons of the same feature map share weights, thus reducing the number of free parameters, and allowing the network to be deeper. Furthermore, CNNs, contrary to traditional neural network architectures which treat input values that are far apart the same way as values that are close together, do take the spatial structure of data into account. This renders them ideal for data with a grid-like topology.

2.4.4 Activation Functions

Activation functions are a way of introducing non-linearity to a neural network. In the absence of an activation function, no matter how many layers there are in a neural network, the last layer is going to be a linear function of the first. As a result, the neural network degenerates into a linear regression model with limited expressive capabilities.

Other than non-linear, activation functions are often monotonically increasing, continuous, differentiable, and bounded.

Popular choices include the **Rectified Linear Unit (**ReLU**)**, **Hyperbolic Tangent (**TanH**)**, the **Sigmoid (**σ**)** and $\mathrm{LeakyReLU}$ activation functions:

$$\mathsf{ReLU}(x) = \mathsf{max}(0, x) \tag{2.12}$$
$$\mathsf{TanH}(x) = \mathsf{tanh}(x) \tag{2.13}$$
$$\sigma(x) = (1 + e^{-x})^{-1} \tag{2.14}$$
$$\mathsf{LeakyReLU}_{\alpha} = \mathsf{max}(0, x) + \alpha \cdot \mathsf{min}(0, x) \tag{2.15}$$

2.4.5 Back Propagation

Backpropagation [6] is a widely used algorithm for training feed-forward neural networks, wherein the gradient of a loss function is computed with respect to the weights of the network for a specific input-output instance.

The gradient of a scalar-valued differentiable function f of several variables is the vector field (or vector-valued function) ∇f whose value at a point p is the vector whose components are the partial derivatives of f at p. That is, for $f\colon \mathbb{R}^n \to \mathbb{R}$, its gradient $\nabla f\colon \mathbb{R}^n \to \mathbb{R}^n$ is defined at the point $p = (x_1, \ldots, x_n)$ in n-dimensional space as the vector:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix} \tag{2.16}$$

A loss function calculates the difference between the network output and its expected output after a training example has propagated through the network. Loss functions are not fixed and are chosen depending on the task at hand.

During backpropagation the gradients are computed one layer at a time, iterating backward from the last layer.

During model evaluation, the weights are fixed, while the inputs vary and the target output is unknown. Whereas during model training, the input-output pairs are fixed and the weights vary.

Backpropagation requires the derivatives of the activation functions to be known at network design time. Additionally, the loss function must be expressible as a function of the outputs of the neural network as well as an average over individual error functions $Q_i(w)$, where each summand function Q_i is typically associated with the i^{th} observation in a n-large training data set:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \tag{2.17}$$

2.4.5.1 Mathematical Statement

Given a feed-forward neural network architecture, let x be the neural network's input, which is a vector of features, y be the target output, C be the loss function, L be the number of layers that make up the neural network, $W^l = (w^l_{j,k})$ be the weights, where $w^l_{j,k}$ is the weight between the k^{th} node in layer $l-1$ and the j^{th} node in layer l and f^l be the activation functions at layer l.

Assuming that nodes in each layer are connected only to nodes in the immediate next layer, without skipping any layers, the overall network can be mathematically described as a combination of function composition and matrix multiplication, as such:

$$g(x) := f^L(W^L f^{L-1}(W^{l-1} \cdots f^1(W^1 x) \cdots)) \tag{2.18}$$

Given an input-output pair (x, y) the loss function is:

$$C(y, g(x)) = C(y, f^L(W^L f^{L-1}(W^{l-1} \cdots f^1(W^1 x) \cdots))) \tag{2.19}$$

The derivative of the loss in terms of the inputs is given by the **chain rule** as:

$$\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdots \frac{\partial a^1}{\partial z^1} \cdot \frac{\partial z^1}{\partial x} \tag{2.20}$$

The **chain rule** is a formula that expresses the derivative of the composition of two differentiable functions f and g in terms of the derivatives $f^{'}$ and $g^{'}$. To elaborate, if a variable z depends on a variable y, which itself depends on a x, then z depends on x as well, via the intermediate variable y and the chain rule states that:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.21}$$

Taking into consideration that:

$$(f^l)' = \frac{\partial a^l}{\partial z^l} \tag{2.22}$$

$$W^l = \frac{\partial W^l a^{l-1}}{\partial a^{l-1}} = \frac{\partial z^l}{\partial a^{l-1}} \tag{2.23}$$

2.20 can be rewritten as:

$$\frac{\partial C}{\partial a^L} \cdot (f^L)' \cdot W^L \cdot (f^{L-1})' \cdot W^{L-1} \cdots (f^1)' \cdot W^1 \tag{2.24}$$

The gradient (∇) is the transpose of the derivative of the output in terms of the input. The transpose of a matrix is an operator which flips a matrix over its diagonal; that is, it switches the row and column indices of the matrix A by producing another matrix, often denoted by A^T. The transpose of a product of matrices is the product, in the reverse order, of the transposes of the factors:

$$(AB)^T = B^T A^T \tag{2.25}$$

Using 2.24 and 2.25 we can calculate the gradient as such:

$$\nabla_x C = (W^1)^T \cdot (f^1)' \cdots (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \tag{2.26}$$

We shall now introduce the auxiliary quantity δ^l, which stands for the *"error at level l"* and is defined as the gradient of the input values at level l:

$$\delta^l = (f^l)' \cdot (W^{l+1})^T \cdots (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \tag{2.27}$$

The gradient of the weights in layer l is then:

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T \tag{2.28}$$

δ^l is multiplied by a factor of a^{l-1}, as the weights W^l, between levels $l-1$ and l, affect level l proportionally to the inputs.

2.27 can be rewritten as:

$$\delta^{l-1} = (f^{l-1})' \circ (W^l)^T \cdot \delta^l \tag{2.29}$$

where $\circ$ is the *Hadamard product*, that is a binary operation that takes two matrices of the same dimensions, and produces another matrix where each element i, j is the product of elements i, j of the original two matrices.

Backpropagation essentially consists of utilizing expression 2.29 to recursively evaluate expression 2.28, starting at the last layer and working our way to the first layer.

Backpropagation is capable of efficiently computing the gradient by avoiding duplicate calculations and not computing unnecessary intermediate values. Computing δ^{l-1} in terms of δ^l avoids the duplicate multiplication of layers $l, l+1, \cdots, L-1, L$. Propagating the error backwards means that each step simply multiplies the vector δ^l by the matrices of weights $(W^l)^T$ and derivatives of activation functions $(f^{l-1})'$. By contrast, multiplying forwards, starting from the changes at an earlier layer, means that each multiplication multiplies a matrix by a matrix. This is much more expensive and corresponds to tracking every possible path of a change in one layer l forward to changes in the layer $l+2$ (for multiplying W^{l+1} by W^{l+2}, with additional multiplications for the derivatives of the activation functions), which unnecessarily computes the intermediate quantities of how weight changes affect the values of hidden nodes.

The term *Backpropagation* strictly refers to the process of computing the gradients, and not how they are used.

2.4.6 Gradient Descent

Gradient descent optimization algorithms [7] are usually used jointly with backpropagation to train multi-layer networks, updating the network's weights and thus minimizing the network's loss. One of the more popular ones is the **Gradient Descent** method.

Gradient Descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as gradient ascent.

Gradient Descent works in spaces of any number of dimensions, even in infinite-dimensional ones.

2.4.6.1 Mathematical Statement

Gradient Descent is based on the observation that if the multi-variable function F is defined and differentiable in a neighborhood of a point a, then $F(x)$ decreases fastest if one goes from a in the direction of the negative gradient of F at a, $-\nabla F(a)$. It follows that if

$$a_{n+1} = a_n - \gamma \nabla F(a) \tag{2.30}$$

for $\gamma \in \mathbb{R}^+$ small enough, then $F(a_n) \geq F(a_{n+1})$. In other words, the term $\gamma \nabla F(a)$ is subtracted from a, because we want to move against the gradient, toward the local minimum. With this observation in mind, one starts with a guess x_0 for a local minimum of F and considers the sequence $x_0, x_1, x_2 \ldots$ such that

$$x_{n+1} = x_n - \gamma \nabla F(x_n), n \geq 0 \tag{2.31}$$

We have a monotonic sequence

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \ldots \tag{2.32}$$

so, hopefully, the sequence $(\mathsf{x_n})$ converges to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration.

Since using a step size γ that is too small would slow convergence, and a γ too large would lead to divergence, finding a good setting of γ is an important practical problem. Other than the step size γ, one could also alter the direction of the descent. Whilst using a direction that deviates from the steepest descent direction may seem counter-intuitive, the idea is that the smaller slope may be compensated for by being sustained over a much longer distance. Let's consider the more general update rule with direction p_n and step size γ_n:

$$a_{n+1} = a_n - \gamma_n p_n \tag{2.33}$$

Finding good settings of p_n and γ_n requires a little thought. First of all, we would like the update direction to point downhill. Mathematically, letting θ_n denote the angle between $\nabla F(a_n)$ and p_n, this requires that $\cos\theta_n > 0$. Under the fairly weak assumption that F is continuously differentiable, we may prove that:

$$F(a_{n+1}) \leq F(a_n) - \gamma_n \left\|\nabla F(a_n)\right\|_2 \left\|p_n\right\|_2 \left[\cos\theta_n - \max_{t\in[0,1]} \frac{\left\|\nabla F(a_n - t\gamma_n p_n) - \nabla F(a_n)\right\|_2}{\left\|\nabla F(a_n)\right\|_2}\right] \tag{2.34}$$

This inequality implies that the amount by which we can be sure the function F is decreased depends on a trade-off between the two terms in square brackets. The first term in square brackets measures the angle between the descent direction and the negative gradient. The second term measures how quickly the gradient changes along the descent direction.

In principle, this inequality could be optimized over p_n and γ_n to choose an optimal step size and direction. The problem is that evaluating the second term in square brackets requires

evaluating $\nabla F(a_n - t\gamma_n p_n)$, and extra gradient evaluations are generally expensive and undesirable.

With certain assumptions on the function F and particular choices of γ, convergence to a local minimum can be guaranteed, for example, when the function F is convex, all local minima are also global minima, so in this case, gradient descent can converge to the global solution.

2.4.7 Extensions and Variants of Gradient Descent

Various Gradient Descent variants have been designed through the years, which improve upon different aspects or tackle limitations of the original Gradient Descent method. We are going to be exploring a few of these.

2.4.7.1 Stochastic Gradient Descent

As previously mentioned in 2.4.5, training a neural network effectively evaluates to minimizing an objective function that can be expressed as:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \tag{2.35}$$

where the parameter w which minimizes $Q(w)$ is to be estimated. Each summand function Q_i is typically associated with the i^{th} observation in the training data set.

When used to minimize the above function, the *standard (or "batch")* gradient descent method would perform the following iterations:

$$w := w - \gamma \nabla Q(w) = w - \frac{\gamma}{n}\sum_{i=1}^{n} \nabla Q_i(w) \tag{2.36}$$

where γ is a step size or *learning rate*.

In stochastic (or *"on-line"*) gradient descent, the true gradient of $Q(w)$ is approximated by a gradient at a single example:

$$w := w - \gamma \nabla Q_i(w) \tag{2.37}$$

Especially in high-dimensional optimization problems, this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate. As the algorithm sweeps through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm converges.

A compromise between computing the true gradient and the gradient at a single example is to compute the gradient against more than one training example (called a "mini-batch") at each step. This can perform significantly better than the "true" stochastic gradient descent described because the code can make use of vectorization libraries rather than computing each step separately. It may also result in smoother convergence, as the gradient computed at each step is averaged over more training examples.

2.4.7.2 Momentum

Stochastic gradient descent with momentum [8] keeps track of the update Δ_w at each iteration, and determines the next update as a linear combination of the gradient and the previous update:

$$\left.\begin{aligned} \Delta_w &:= \alpha\Delta_w - \gamma\nabla Q_i(w) \\ w &:= w + \Delta_w \end{aligned}\right\} \implies w := w - \gamma\nabla Q_i(w) + \alpha\Delta_w \tag{2.38}$$

where α is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the weight change.

Momentum allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space.

2.4.7.3 RMSProp

In **Root Mean Square Propagation (RMSProp)** [9], the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight.

So, first the running average is calculated in terms of means square:

$$v(w,t) := \beta v(w,t-1) + (1-\beta)(\nabla Q_i(w))^2 \tag{2.39}$$

where β is the *forgetting* factor. And the parameters are updated as such:

$$w := w - \frac{\gamma}{\sqrt{v(w,t)}}\nabla Q_i(w) \tag{2.40}$$

2.4.7.4 Adam

Adaptive Moment Estimation (Adam) [10] is an extension of RMSProp. In Adam, running averages of both the gradients and the second moments of the gradients are used.

Given parameters $w^{(t)}$ and a loss function $L^{(t)}$, where t indicates the current training iteration, the parameters are updated as such:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)} \tag{2.41}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2 \tag{2.42}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1} \tag{2.43}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2} \tag{2.44}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \gamma\frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon} \tag{2.45}$$

where ϵ is a small scalar (e.g. 10^{-8}) used to prevent division by 0, and β_1 (e.g. 0.9) and β_2 (e.g. 0.999) are the *forgetting* factors for gradients and second moments of gradients, respectively.

2.4.8 Batch Normalization

Each layer of a neural network has inputs with a corresponding distribution, which is affected during the training process by the randomness in the parameter initialization and the randomness in the input data. The effect of these sources of randomness on the distribution of the inputs to internal layers during training is described as *internal covariate shift*. Although a clear-cut precise definition seems to be missing, the phenomenon observed in experiments is the change in means and variances of the inputs to internal layers during training.

Batch normalization [11] was initially proposed to mitigate internal covariate shift. During the training stage of networks, as the parameters of the preceding layers change, the distribution of inputs to the current layer changes accordingly, such that the current layer needs to constantly readjust to new distributions. This problem is especially severe for deep networks because small changes in shallower hidden layers will be amplified as they propagate within the network, resulting in a significant shift in deeper hidden layers. Therefore, the method of batch normalization is proposed to reduce these unwanted shifts to speed up training and to produce more reliable models. Some scholars have argued that batch normalization does not reduce internal covariate shift, but rather smooths the objective function, which in turn improves performance.

Besides reducing internal covariate shift, batch normalization is believed to introduce many other benefits. With this additional operation, the network can use a higher learning rate without vanishing or exploding gradients. Furthermore, batch normalization seems to have a regularizing effect such that the network improves its generalization properties. This prevents the model from corresponding too closely to a particular set of data and therefore failing to fit additional data or predict future observations reliably (also known as *Overfitting*).

It has been observed also that batch normalization the network becomes more robust to different initialization schemes and learning rates.

2.4.8.1 Mathematical Statement

In a neural network, batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs. Ideally, the normalization would be conducted over the entire training set, but to use this step jointly with stochastic optimization methods, it is impractical to use the global information. Thus, normalization is restrained to each mini-batch in the training process.

Use B to denote a mini-batch of size m of the entire training set. The empirical mean and variance of B could thus be denoted as

$$\mu_B = \frac{1}{m}\sum_{i-1}^{m} x_i \tag{2.46}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i-1}^{m}(x_i - \mu_B)^2 \tag{2.47}$$

For a layer of the network with d-dimensional input, $x = (x^{(1)}, ..., x^{(d)})$, each dimension of its input is then normalized (i.e. re-centered and re-scaled) separately

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}} \tag{2.48}$$

where $k \in [1, d]$ and $i \in [1, m]$; $\mu_B^{(k)}$ and $\sigma_B^{(k)^2}$ are the per-dimension mean and variance, respectively. ϵ is added in the denominator for numerical stability and is an arbitrarily small constant.

The resulting normalized activation $\hat{x}^{(k)}$ have zero mean and unit variance *if ϵ is not taken into account*. To restore the representation power of the network, a transformation step then follows as

$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)} \tag{2.49}$$

where the parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are subsequently learned in the optimization process.

Formally, the operation that implements batch normalization is a transform, the **Batch Normalization Transform**

$$\mathsf{BN}_{\gamma^{(k)},\beta^{(k)}} : x_{1...m}^{(k)} \rightarrow y_{1...m}^{(k)} \tag{2.50}$$

The output of the BN transform $y^{(k)} = \mathsf{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ is then passed to other network layers, while the normalized output $\hat{x}_i^{(k)}$ remains internal to the current layer.

During inference, the normalization step is computed with the population statistics such that the output could depend on the input in a deterministic manner.

$$\mathsf{E}[x^{(k)}] = \mathsf{E}_B[\mu_B^{(k)}] \tag{2.51}$$

$$\mathsf{Var}[x^{(k)}] = \frac{m}{m-1}\mathsf{E}_B[\sigma_B^{(k)^2}] \tag{2.52}$$

The BN transform in the inference step thus becomes

$$y^{(k)} = BN^{\mathsf{inf}}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)}) = \frac{\gamma^{(k)}}{\sqrt{\mathsf{Var}[x^{(k)}] + \epsilon}}x^{(k)} + \left(\beta^{(k)} - \frac{\gamma^{(k)}E[x^{(k)}]}{\sqrt{\mathsf{Var}[x^{(k)}] + \epsilon}}\right) \tag{2.53}$$

where $y^{(k)}$ is passed on to future layers instead of $x^{(k)}$. Since the parameters are fixed in this transformation, the batch normalization procedure is essentially applying a linear transform to the activation.

2.4.8.2 Limitations and Hindrances

When activation functions are used whose derivatives can take on larger values, one risks encountering the **exploding gradient problem**, which refers to accumulating gradients resulting in very large updates to neural network model weights during training. This renders the model unstable and unable to learn from the training data.

Even though batch normalization was originally introduced to alleviate gradient vanishing or explosion problems, a deep batch normalization network suffers from gradient explosion at initialization time, no matter what it uses for non-linearity. Thus the optimization landscape is very far from smooth for a randomly initialized, deep batch normalization network. More precisely, if the network has L layers, then the gradient of the first layer weights has norm $> c\lambda^L$ for some $\lambda > 1$, $c > 0$ depending only on the non-linearity. For any fixed non-linearity, λ decreases as the batch size increases. For example, for ReLU, λ decreases to $\frac{\pi}{\pi-1} \approx 1.467$ as the batch size tends to infinity. Practically, this means deep batch normalization networks are untrainable. This is only relieved by skip connections in the fashion of residual networks. Note that, the gradient explosion depends on **stacking** batch normalization layers typical of modern deep neural networks.

2.4.9 Generative Adversarial Network (GAN)

A **Generative Adversarial Network (GAN)** [1] is a machine learning framework wherein, two models, namely the generator and the discriminator are simultaneously trained and play a minimax two-player game.

The generative model captures the data distribution and generates candidates, while the discriminative network, given a sample, estimates the probability that it originates from the training data rather than the generative model.

The contest operates in terms of data distributions. Typically, the generative network learns to map from a latent space to a data distribution of interest.

GANs are implicit generative models, which means that they do not explicitly model the likelihood function nor provide means for finding the latent variable corresponding to a given sample.

2.4.9.1 Mathematical Statement

The adversarial modeling framework is most straightforward to apply when the models are both multi-layer perceptrons. To learn the generator's distribution p_g over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a multi-layer perceptron with parameters θ_g. We also define a second multi-layer perceptron $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1\boxtimes D(G(z)))$.

In other words, D and G play the following two-player minimax game with value function

$V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)} \Big[\log D(x) \Big] + \mathbb{E}_{z \sim p_z(z)} \Big[1 - \log D(G(z)) \Big] \tag{2.54}$$

In [1] *Ian Goodfellow et al.* proved that this minimax game has a global optimum for $p_g = p_{data}$.

2.4.9.2 Training

The generative network's training objective is to increase the error rate of the discriminative network, by producing novel candidates that the discriminator fails to distinguish from real data.

A known data set serves as the initial training data for the discriminator. Training involves presenting it with samples from the training data set until it achieves acceptable accuracy. The generator trains based on whether it succeeds in fooling the discriminator. Typically the generator is seeded with randomized input that is sampled from a predefined latent space (e.g. a multivariate normal distribution). Thereafter, candidates synthesized by the generator are evaluated by the discriminator. Independent backpropagation procedures are applied to both networks so that the generator produces better samples, while the discriminator becomes more skilled at flagging synthetic samples. The gradient-based updates can use any standard gradient-based learning rule, but we are going to be presenting a mini-batch stochastic gradient descent (along with momentum) approach, as this is the one explored in [1]:

Algorithm 1 Mini-batch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter

1: **for** `number of training iteration` **do**
2: **for** `k steps` **do**
3: `Sample mini-batch of m noise samples` $\{z(1) \cdots z(m)\}$ `from noise prior` $p_g(z)$.
4: `Sample mini-batch of m examples` $\{x(1) \cdots x(m)\}$ `from data generating distribution` $p_{data}(x)$.
5: `Update the discriminator by ascending its stochastic gradient:`

$$\nabla_{\theta_d} \frac{1}{m} \sum_1^m \Big[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \Big]$$

6: `Sample mini-batch of m noise samples` $\{z(1) \cdots z(m)\}$ `from noise prior` $p_g(z)$.
7: `Update the generator by descending its stochastic gradient:`

$$\nabla_{\theta_g} \frac{1}{m} \sum_1^m \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 \boxtimes D(G(z)))$ saturates. Rather

than training G to minimize $\log(1\boxtimes D(G(z)))$ we can train G to maximize $\log D(G(z))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite data sets would result in overfitting. Instead, we alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough.

This algorithm optimizes function 2.54, thus obtaining the desired result of $p_g \approx p_{data}$.

2.5 N-Gram Graphs

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, or base pairs according to the application.

Assuming a text (T^l), an elementary way of extracting its corresponding set of n-grams SS^n is described by *Algorithm 2*:

Algorithm 2 Extracting n-grams from a text

1: $SS^n \leftarrow \emptyset$
2: **for** `all` i `in` $[1, length(T) - n + 1]$ **do**
3: $\quad SS^n \leftarrow SS^n \bigcup S_{i,i+n-1}$

In [12] *George Giannakopoulos et al.* proposed the **n-gram graph** model, which is a language-neutral, statistical approach of representing a text document. In [13], the authors, using the n-gram graph model, designed an automatic summary evaluation system.

The n-gram graph is a graph $G = \{V, E, L, W\}$, where V is the set of vertices, E is the set of edges, L is a one-to-one function assigning a label to each vertex and to each edge and W is a function assigning a weight to every edge. The graph has n-grams as its vertices $v \in V$ and edges $e \in E$ connecting them. The weights $w \in W$ of the edges indicate either the distance or the number of co-occurrences of two n-grams, within a given window D_{win}, in the original text. The meaning of distance and window size changes by whether we use character or word n-grams.

In [12] 3 different weighting approaches were presented based on different types of windows. Denoting a random n-gram as N_0 located at position p_0, the various approaches are described below:

- **The non-symmetric approach** where, then the window will span from $p_0 - D_{win}$ to $p_0 - 1$, taking into account only preceding n-grams. Every neighbor contributes equally to the corresponding edge's weight.
- **The symmetric approach** where, then the window will span from $p_0 - \lfloor \frac{D_{win}}{2} \rfloor$ to $p_0 + \lfloor \frac{D_{win}}{2} \rfloor$, taking into account both preceding and succeeding n-grams. Every neighbor contributes equally to the corresponding edge's weight.
- **The Gauss-normalized symmetric approach** where, then the window will span from $p_0 - \lfloor \frac{3 \times D_{win}}{2} \rfloor$ to $p_0 + \lfloor \frac{3 \times D_{win}}{2} \rfloor$, taking into account both preceding and suc-

ceeding n-grams. Each neighbor contribution is weighted based on that neighbor's distance to the target n-gram.

2.5.1 Similarity Metrics

In [12] *George Giannakopoulos et al.* introduced a variety of metrics aimed at determining the similarity between two n-gram graphs. These metrics include the **Value Similarity (VS)**, the **Size Similarity (SS)** and the **Normalized Value Similarity (NVS)**.

Assuming two n-gram graphs $G_1 = \{V_1, E_1, L_1, W_1\}$ and $G_2 = \{V_2, E_2, L_2, W_2\}$, then the *Value Similarity* is defined as:

$$\mathrm{VS}(G_1, G_2) = \frac{\sum_{e \in E_1 \cap E_2} \mathrm{VR}(e)}{max(|E_1|, |E_2|)} \tag{2.55}$$

where $|E_i|$ stands for the cardinality of E_i. VR stands for *Value Ratio* and is defined as:

$$\mathrm{VR}(e) = \frac{min(w_1^e, w_2^e)}{max(w_1^e, w_2^e)} \tag{2.56}$$

where w_1^e and w_2^e correspond to the weights of edge e in graphs G_1 and G_2 respectively.

Size Similarity is defined as:

$$\mathrm{SS}(G_1, G_2) = \frac{min(|E_1|, |E_2|)}{max(|E_1|, |E_2|)} \tag{2.57}$$

Finally, *Normalized Value Similarity* uses both 2.55 and 2.57 and is defined as:

$$\mathrm{NVS}(G_1, G_2) = \frac{\mathrm{VS}(G_1, G_2)}{\mathrm{SS}(G_1, G_2)} \tag{2.58}$$

2.5.2 Variants

In [13] *George Giannakopoulos et al.* presented two variants of n-gram graphs, targeting the task of summarization evaluation. The first method, referred to as **Merged Model Graph (MeMoG)**, utilizes a single n-gram graph to represent a set of documents, while the second method, referred to as **Hierarchical Proximity Graph (HPG)**, utilizes a hierarchy of graphs to represent a set of documents with different granularity levels.

2.5.2.1 Merged Model Graph

The Merged Model Graph approach allows modeling a whole set of documents using one representative graph. Given a set D_N of N documents, the construction of the representative graph comprises of:

1. Constructing N individual graphs, one for each document in D_N
2. Merging these N graphs into one representative graph

Merging the individual graphs is carried out using the *Update Operator* $\bigcup(G_1, G_2, l)$, which takes as input two graphs, one that is considered to be the pre-existing graph G_1 and one that is considered to be the new graph G_2. The operator also expects a parameter referred to as the *learning factor* $l \in [0, 1]$, which determines the sensitivity of G_1 to changes in G_2. More precisely:

- A value of $l = 0$ indicates that G_1 will completely ignore the changes introduced by G_2.
- A value of $l = 1$ indicates that the weights of the edges of G_1 will be overwritten by the weights of the edges of G_2.

Assuming two graphs G_1 and G_2, applying U results in the following weight update:

$$W^i(e) = W^1(e) + (W^2(e) - W^1(e)) \times l \tag{2.59}$$

where $W^1(e)$ and $W^2(e)$ is the weight of edge e on graphs G_1 and G_2 respectively and $W^l(e)$ corresponds to the weight of edge e on the resulting graph.

In the case of MeMoG, the representative graph's edges are required to hold weights averaging the weights of all of its constituent graphs. In order to achieve that, the i^{th} graph update should contribute to the representative graph with a learning factor of $l = \frac{1}{i}$, $i > 1$.

Finally, MeMoG is structured as a standard n-gram graph, hence similarity metrics 2.56-2.58 can be used to compare two MeMoGs.

2.5.2.2 Hierarchical Proximity Graph

An HPG of $L \in \mathbb{N}^*$ levels is a hierarchy H of proximity graphs, where subgraphs of symbols from a lower level $l - 1$ serve as the symbols of the level $l \in [1, L]$. Each level H_l holds a proximity graph J_l and an index of symbols I_l.

Given a source text, in order to create the first level, we need to extract its corresponding set of n-grams. Afterwards, the index I_1 maps, through a bijection, every distinct n-gram to an integer symbol, and vertices are created in the proximity graph J_1, one for each symbol. Considering the source text as a sequence of symbols Z and given a window D_{win}, all symbols found within a maximum distance (the number of n-grams between two n-grams of interest) of D_{win} in Z have their vertices linked by an edge.

The aforementioned process is repeated for every symbol in Z, resulting in the construction of multiple graphs, referred to as *s-neighborhood*s, each one connecting a symbol to its neighbors. Each s-neighborhood serves as a symbol for the next level of graphs and is, thus, mapped to an integer in the corresponding index.

Constructing levels $1 < l \leq L$ of the HPG, consists of

- Retrieving the symbol sequence corresponding to level $l - 1$.
- Constructing the s-neighborhoods of the current level and adding them to index I_l.
- Generating the current level proximity graph J_l.

The resulting hierarchy of proximity graphs $\mathbb{H} =< J_1, \cdots, J_L >$ is referred to as a **Hierarchical Proximity Graph (HPG)**.

Note that, every level uses a different window size and more specifically, the window size increases linearly with the level, $D_{winl} = \lfloor D_{win} \times l \rfloor$. This is based on the intuition, that the notion of neighborhood changes completely, going from a word to a paragraph, as entities that are further away should be now considered neighboring.

Lastly, the similarity between two HPGs H_1 and H_2 is given by the weighted normalized sum of value similarities between the corresponding levels of H_1 and H_2:

$$\frac{\sum_{l\in[1,L]} l \times \mathrm{VS}^{\mathrm{l}}(H_1, H_2)}{\sum_{l\in[1,L]} l} \tag{2.60}$$

where $VS^l(H_1, H_2)$ is the *Value Similarity* (2.55) of the l level proximity graphs of H_1 and H_2.

2.6 State of the Art

Deep generative networks have already been successfully applied in the field of Nanotechnology and more specifically to the task of characterizing microstructures and synthesizing artificial ones.

Ahmet Cecen et al. [14] employ a 3D convolutional neural network in order to reliably link a microstructure to its properties. The learned 3D-CNN features are then used alongside other spatial metrics to estimate higher-order statistics leading to improved accuracy in terms of property prediction. *Zijiang Yang et al.* [15] employ a convolutional neural network in order to predict the micro-scale elastic strain field of a three-dimensional voxel-based microstructure of a high-contrast two-phase composite. The model is trained on multiple data sets corresponding to varying degrees of contrast and is able to significantly outperform state-of-the-art methods. *Antonios Stellas et al.* [16] showcased that deep neural networks, as well as other machine learning models, can efficiently predict a nanosurface's active area given its corresponding structural parameters, such as the RMS, association length(s) etc.

Satoshi Noguchi et al. [17] utilize a Variational Auto-Encoder [18] in order to map material microstructures to a latent space and subsequently use a PixelCNN in order to generate statistically equivalent microstructures based on these latent features. *Zijiang Yang et al.* [19] develop a three-dimensional CNN aiming to model elastic homogenization linkages for three-dimensional high-contrast composite material system which improves on past physics-inspired approaches. *Ruijin Cang et al.* [20] employ a convolutional deep belief network [21] aiming to establish a two-way conversion between microstructures and their corresponding lower-dimensional feature representations. The proposed model is applied to a wide spectrum of heterogeneous material systems and is able to produce material reconstructions that are close to the original samples with respect to two-point correlation functions and mean critical fracture strength while achieving an 1000-fold dimensional reduction from the microstructure space. *Daria Fokina et al.* [22] utilize a StyleGAN [23] in their efforts to synthesize larger microstructures from several smaller samples. The authors use image quilting between the borders of two nearby patches to generate realistically looking samples of a larger size. The method is tested on microstructure synthesis and porous media reconstruction and it is shown that the generated structures closely re-

semble the real ones with regards to their effective properties. *Lukas Mosser et al.* [24] evaluate the application of generative adversarial neural networks [1] for stochastic image reconstruction of porous media and show through various measures that the model is able to capture the statistical and physical behavior of the training data. *Andrea Gayon-Lombardo et al.* [25] implement a deep convolutional generative adversarial network [2] with the goal of generating realistic n-phase micro-structural data. The model is successfully applied on two different three-phase microstructures, namely a lithium-ion battery cathode and a solid oxide fuel cell anode and it is able to produce artificial microstructures that are virtually indistinguishable from real data.

The motivation behind this work is developing a data-driven framework aiming at the stochastic reconstruction of nanorough surfaces. We employ a DCGAN [2] with the goal of improving upon its sole performance by utilizing a novel graph-based nanorough surface similarity metric. This similarity metric is going to alter the generator's applied loss and effectively guide the model throughout the training course. We consider the generation of nanorough surfaces as a data-driven supervised learning task, where nanorough surfaces of predefined structural parameters serve as the training data set.

3. METHODOLOGY

3.1 Problem Definition

A nanorough surface can be approximated by a height map, that is a matrix containing the height values of different point samples. Note that such a representation entails limitations regarding its representative capabilities due to utilizing a discrete number of points, as described in [26]:

$$S = \begin{bmatrix} s_{1,1} & \cdots & s_{1,n} \\ \vdots & \ddots & \vdots \\ s_{m,1} & \cdots & s_{m,n} \end{bmatrix} \tag{3.1}$$

A nanorough surface can be characterized by its RMS, horizontal and vertical correlation lengths (ξ_x, ξ_y), as well as other metrics. We are going to be referring to the set of values

$$\mathbb{C} =< RMS, \xi_x, \xi_y > \tag{3.2}$$

as the **configuration** of the nanorough surface.

Our goal is to generate nanorough surfaces of a specific configuration, without a priori knowledge of their specific configuration, but only of nanorough surface samples. Hence, we need to determine a function, mapping a set of nanorough surfaces S_{C^*} to another set of nanorough surfaces of the same configuration $\mathbb{C}^*$:

$$\mathbb{F}\colon U_{C^*} \rightarrow U_{C^*} \tag{3.3}$$

where U_{C^*} is the set containing every possible nanorough surfaces with a configuration of C^*.

3.2 System Overview

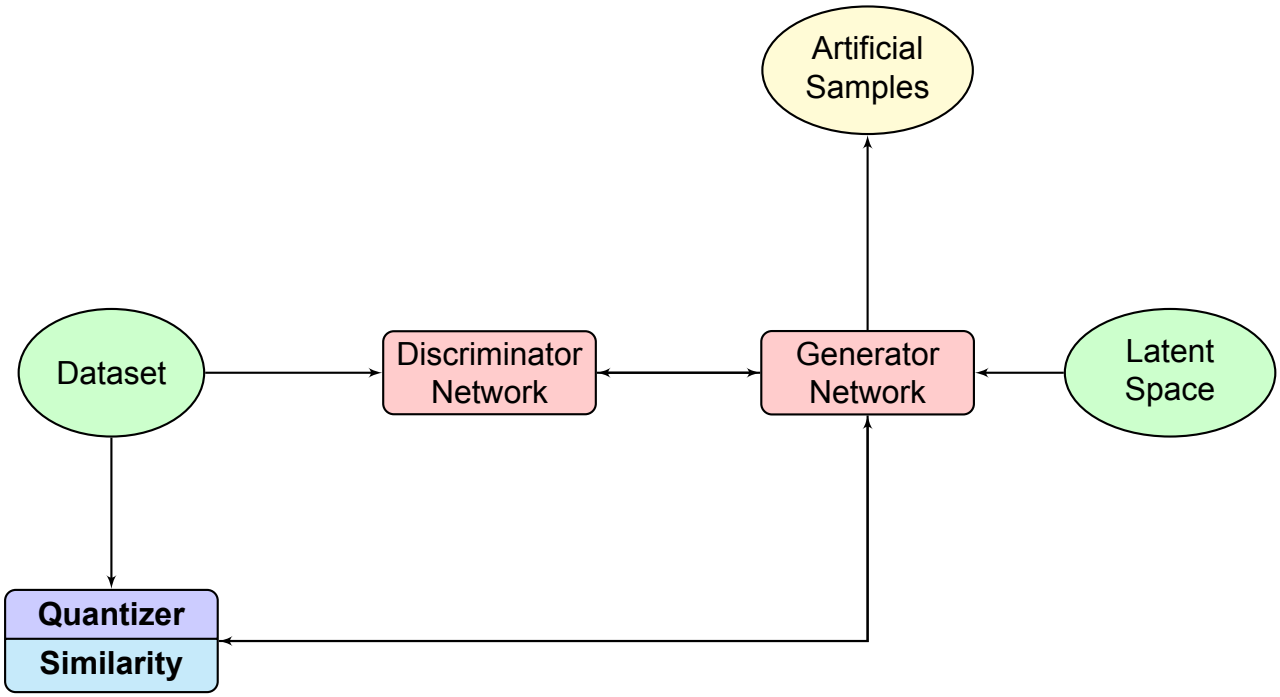


Figure 3.1: System overview

Our framework employs a CNN-based GAN. More specifically, the generator, which is a convolutional neural network competes with the discriminator, which is a deconvolutional neural network. The models are trained on a set of nanorough surfaces corresponding to a specific parameter configuration $\mathbb{C}^*$.

The training set is loaded and the nanorough surfaces are processed in batches. Following *Algorithm 1*, the generator and the discriminator are trained separately via backpropagation. The discriminator is trained on real and artificially generated data, while the generator on the other hand is trained based on the discriminator's output. More specifically, the discriminator outputs a value in the range $[0, 1]$, which corresponds to the likelihood of a surface originating from the real nanorough surface distribution.

A set of novel n-gram graph-based metrics are used in conjunction with the discriminator, providing additional feedback to the generator model, regarding a surface's origins, that is whether or not it originates from the real data distribution. All similarity metrics require that their input consists of a fixed set of symbols. Hence, the nanorough surfaces must undergo a processes referred to as *Quantization*, wherein the data is translated from a 2D real matrix to a 2D symbol matrix representation.

Our framework was implemented in *Python*. Nanorough surface generation described in *Section 3.2.1*, utilizes *NumPy* [27], *SciPy* [28] and *SymPy* [29]. Nanorough surface quantization described in *Section 3.2.2* utilizes *scikit-learn* [30]. The implementation of the graph-based content similarity metrics described in *Section 3.2.3* utilizes the *PyINSECT* [31] module. Both flavors of the GAN framework described in *Section 3.2.4* were implemented using the *PyTorch* [32] ML framework. The visualizations showcased throughout *Chapter 4* were created using *Matplotlib* [33] and *Plotly* [34]. The complete source code can be found at https://github.com/billsioros/RoughML.

3.2.1 Nanorough Surface Generation

It was required that we generate a large number of nanorough surfaces to serve as the training data to our models. For this, we simply ported the algorithm used by *Antonios Stellas et al.* in [26] to *Python*.

The nanorough surface generation algorithm can be configured by modifying on of following 9 parameters:

- **n_points**: the square root of the actual nanorough surface size. The resulting surfaces will be square matrices of size $n\_points \times n\_points$.
- **rms**, **skewness** and **kurtosis**: the root mean square error, skewness and kurtosis of the height value distribution.
- **corlength_x** and **corlength_y**: the desired correlation lengths (ξ_x, ξ_y) of the resulting nanorough surfaces.
- **alpha**, **beta_x** and **beta_y**: the smoothing hyper-parameters α, β_x and β_y (β_x and β_y are only present on the *Bessel* variant of the algorithm).

The nanorough surface generation algorithm consists of:

1. Populating R, an $N \times N$ matrix, with values resulting from the provided *auto-correlation function* (more on that later).
2. Calculating the power spectrum, $FR(R)$, of R based on the *Wiener–Khinchin* theorem, as well as the expression $AMPR(R) = \sqrt{d_x^2 + d_y^2} \times |FR(R)|$.
3. Generating an $N \times N$ matrix, corresponding to white noise, normalizing it and calculating its Fourier transform, XF.
4. Calculating the inverse Fourier transform of the product $XF \times \sqrt{AMPR(R)}$, extracting the real part, normalizing and scaling it by RMS.
5. Generating an $N \times N$ matrix, z_{ngn}, corresponding to a *Pearson type III* continuous random variable.
6. Flattening both z and z_{ngn}, sorting their values in descending order, reordering z_{ngn} based on the order of z and reshaping z_{ngn} into a 2D matrix v_ngs^*.
7. The conjugate transpose of v_ngs^* (z_ngs) corresponds to a non-Gaussian correlated nanorough surface.

The two nanorough surface generation algorithm flavors are differentiated only by the auto-correlation functions they utilize:

$$\text{Standard}(x, y) = RMS^2 \times e^{-\left|\sqrt{\frac{x}{\xi_x}^2 + \frac{y}{\xi_y}^2}\right|^{2\times\alpha}} \tag{3.4}$$

$$\text{Bessel}(x, y) = \text{Standard}(x, y) \times \text{J}_0(2\pi\sqrt{\frac{x}{\beta_x}^2 + \frac{y}{\beta_y}^2}) \tag{3.5}$$

where $J_0(x)$ is the Bessel function of the first kind, for $n = 0$. In this work, we only experimented with the so-called *Standard* version of the algorithm.

3.2.2 Nanorough Surface Quantization

The n-gram graph model, as well as its variants, were originally applied in the field of *Natural Language Processing* and more specifically in the context of *Summary Evaluation*. Consequently, they were initially designed to operate on strings, that is sequences of symbols from some alphabet. On the other hand, a nanorough surface is represented as a two-dimensional real matrix, whose values correspond to the height of the nanorough surface in different sampling coordinates. Therefore, in order to represent a nanorough surface as an n-gram graph, the corresponding two-dimensional real matrix must be translated to a two-dimensional matrix of symbols. The process responsible for achieving this is commonly referred to as **Quantization** (or **Binning**).

The term *Binning* describes the process of, given some data points, replacing the original data points which fall into different value ranges (i.e. bins) by a value representative of that interval. There is a plethora of Binning methods, differentiated mainly by the way the bin edges are calculated. Two prominent examples of Binning methods are:

- **Uniform Binning**, wherein all bins have identical widths.
- **Quantile Binning**, wherein we assign the same number of observations to every bin.

Figure 3.2 showcases how 1000 samples drawn from a standard normal distribution are segregated, in both cases, using 5 bins. First of all, the data points are sorted in ascending numerical order. Afterwards, the edges of each and every bin are calculated. Finally, every data point is replaced by the index of its respective bin.

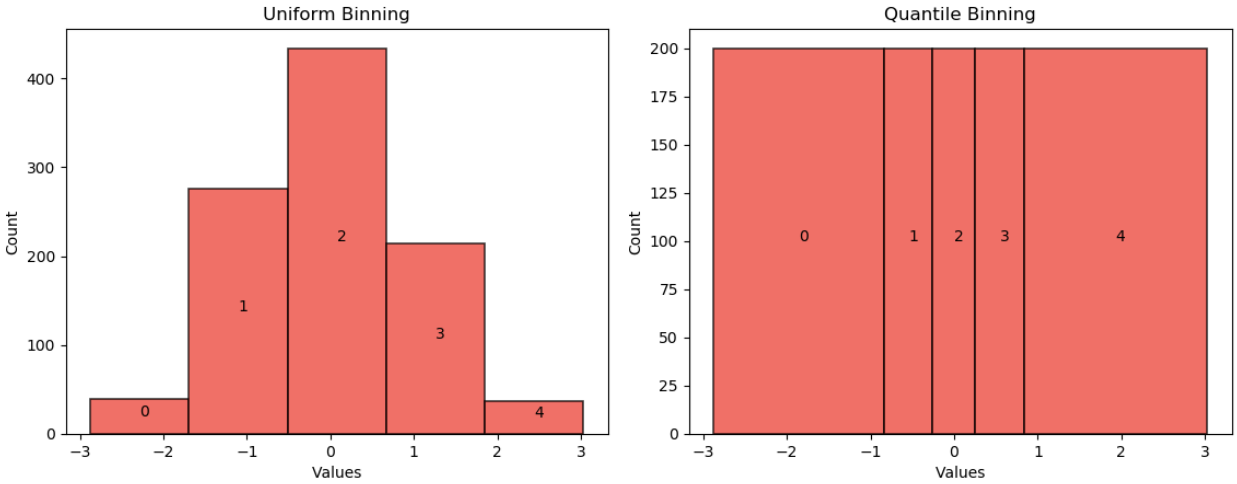


Figure 3.2: Comparing Uniform and Quantile Binning. In the Uniform Binning case, the bin edges are $[-2.874, -1.694, -0.514, 0.665, 1.845, 3.025]$. Whereas, in the Quantile Binning case, the bin edges are $[-2.874, -0.838, -0.259, 0.247, 0.841, 3.025]$.

For our purposes, we opted for 5 bins and the Quantile Binning approach, as more height values being mapped to the same symbol, entails sparser graph representations and consequently renders the process of calculating the appropriateness of a given nanorough surface less computationally expensive.

In the case of Quantile Binning, the edges of the bins are calculated using *Algorithm 3* [35, 36]. More specifically, when using 5 bins the 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0 quantiles are calculated. These 6 values delimit the edges of the 5 bins, and any value that falls within the range of a given bin, is replaced by its corresponding index, e.g. a value in the range $[\text{quantile}(0.0), \text{quantile}(0.2)]$ will be replaced by 0.

Algorithm 3 Calculating the p^{th} quantile of an n-large set of real numbers $\mathbb{R}$ (It is assumed that the values have already been sorted).

1: $m = n \times p$
2: $m_{integer} = \lfloor m \rfloor$
3: $m_{float} = m - m_{integer}$
4: **if** $m_{float} \approx 0$ **then**
5: **return** $\mathbb{R}[m_{integer}]$
6: **else**
7: **return** $\frac{\mathbb{R}[m_{integer}-1]+\mathbb{R}[m_{integer}]}{2}$

The edges of the bins are initially calculated using the training data set in its entirety. The training data set is first transformed into a 2D matrix, where every row corresponds to a different nanorough surface. Binning is then carried out separately on every column (feature) of this matrix.

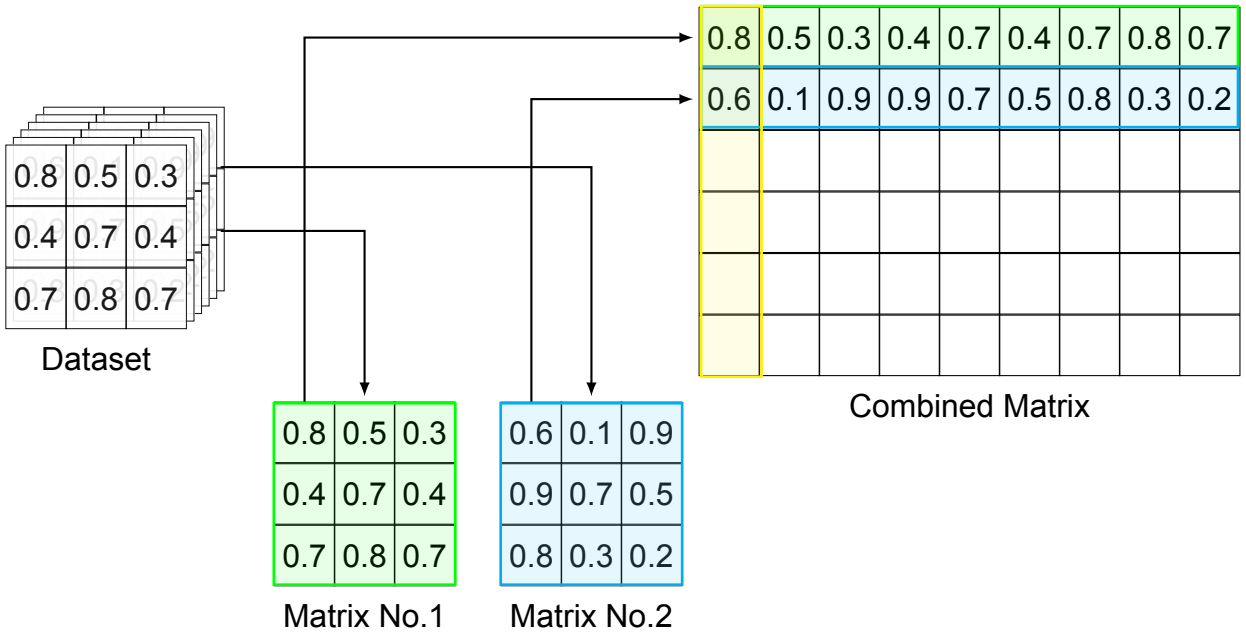


Figure 3.3: Nanorough Surface Quantization. The multiple 2D matrices making up the training data set are transformed to 1D matrices by unrolling them in row-major order. The 1D matrices are then compressed into a single 2D matrix where every row corresponds to an individual 1D matrix. Binning is then carried out separately on every column of this 2D matrix, resulting to different bin edges per column.

While training our models, quantization is carried out on batches of artificially generated nanorough surfaces. The batches are again transformed, as previously described, into a 2D matrix. Binning is now carried out using the initially calculated bin edges.

Quantization serves as a preprocessing step to every graph-based content similarity metric.

3.2.3 Content Similarity Metrics

We developed various methods of measuring how similar two nanorough surfaces are, which utilize the n-gram graph model and its variants to represent nanorough surfaces as graphs. To be more specific, we developed the **N-Gram Graph (NGG)**, the **2D Array Graph (A2D)** and the **Hierarchical Proximity Graph (HPS)** content similarity metrics. In all cases, we used co-occurrences of symbols as the edges' weight factor. Additionally, we designed a non-graph-based content similarity metric referred to as **Fourier & Histogram Space (FHS)** content similarity, which will be used to evaluate the realisticity of the artificially generated nanorough surfaces.

In the following sections it is assumed that, the data serving as input to the similarity metric at hand, has already been quantized.

3.2.3.1 The N-Gram Graph Content Similarity Metric (NGG)

The NGG content similarity metrics follows the **MeMoG** approach presented in section 2.5.2.1. More specifically, an n-gram graph is created for each and every one of the nanorough surfaces making up the training data set. Afterwards, the individual graphs are all merged into a single representative graph. Having created the representative graph, the similarity of a nanorough surface with regard to the training data set (also referred to as *appropriateness*) is calculated by means of 2.55.

Given that the n-gram graph is designed to work on one-dimensional data the two-dimensional matrices, representing nanorough surfaces, must be first reshaped into one-dimensional vectors. The transformation is done in row-major order and is referred to as *Flattening*.

The construction of an n-gram graph, given an $N \times N$ input matrix $\mathbb{M}$, consists of:

1. Flattening matrix $\mathbb{M}$ into vector $\mathbb{V}$
2. Generating the n-grams corresponding to $\mathbb{V}$, using *Algorithm 2*
3. Passing a sliding window over the resulting n-gram array, creating edges connecting every pair of n-grams within the sliding window.

A more thorough presentation of the procedure is given in *Algorithm 4*. Bare in mind that, we adopt the non-symmetric edge weighting approach described in section 2.5.

Algorithm 4 Constructing an n-gram Graph given an $N \times N$ input matrix $\mathbb{M}$

$\mathbb{V} \leftarrow \mathsf{Flatten}(\mathbb{M})$
$\mathbb{SS}^n \leftarrow \mathsf{ExtractNGrams}(\mathbb{M})$
$G \leftarrow \emptyset$
for $w \in [0, \cdots, \lfloor \frac{\mathsf{LEN}(SS^n)}{\mathbb{W}} \rfloor]$ **do**
  for $ngram_y \in \{ngram_{w \cdot \mathbb{W}}, \cdots, ngram_{(w+1) \cdot \mathbb{W}}\}$ **do**
    for $ngram_x \in \{ngram_{w \cdot \mathbb{W}}, \cdots, ngram_{(w+1) \cdot \mathbb{W}}\}$ **do**
      if $ngram_y \neq ngram_x$ **then**
        $G \leftarrow G \bigcup_+ \{(ngram_y, ngram_x)\}$

where $\bigcup_+$ stands for the *Edge Update* operator, which either introduces a new edge, with an associated weight of 1, to the graph or increases the weight of a pre-existing edge.

More specifically, given that an edge already exists, any consecutive occurrence of the same vertex pair within the predefined n-gram window contributes equally to the total edge weight. ExtractNGrams stands for the process carried out by *Algorithm 2*. Flatten is described by *Algorithm 5*.

Algorithm 5 Flattening a $N \times M$ input matrix $\mathbb{M}$

$\mathbb{V} \leftarrow [0, \cdots, 0]^{(N \times M, 1)}$
for $y \in [0, \cdots, N]$ **do**
    for $x \in [0, \cdots, M]$ **do**
        $\mathbb{V}[y \cdot M + x] = \mathbb{M}[y, x]$

The individual n-gram graphs are merged using the Update operator presented previously in section *Section 2.5.2.1*. We are employing a dynamic learning rate, as described in the aforementioned section, so that the representative graph's edges are assigned weights averaging the weights of all the individual graphs that have contributed to it.

3.2.3.2 The Two-Dimensional Array Graph Content Similarity Metric (A2G)

The NGG and A2D content similarity metrics differ only with respect to the construction of the per nanorough surface individual n-gram graphs.

In contrast to the NGG content similarity metric, A2D is able to processes two-dimensional data. More specifically, given a two-dimensional matrix corresponding to a nanorough surface, graph construction in the context of the A2D content similarity consists of:

1. Initializing an empty graph
2. Sliding a two-dimensional window of predetermined size over the given matrix and processing individual sub-matrices of it.
3. Adding edges to the graph, connecting all possible pairwise combinations of symbols for each and every one of those sub-matrices.

This process is described by *Algorithm 6*:

Algorithm 6 Constructing a 2D Array Graph given an $N \times N$ input matrix $\mathbb{M}$

$G \leftarrow \emptyset$
for $y \in [0, \cdots, N]$ **do**
    for $x \in [0, \cdots, N]$ **do**
        $vertex_{y,x} = \mathbb{M}[y, x]$
        $neighborhood \leftarrow \emptyset$
        $neighbor_y^{min} = \text{Clamp}_N(y - \lfloor \frac{\mathbb{W}}{2} \rfloor)$
        $neighbor_y^{max} = \text{Clamp}_N(y + \lfloor \frac{\mathbb{W}}{2} \rfloor)$
        for $neighbor_y \in [neighbor_y^{min}, \cdots, neighbor_y^{max}]$ **do**
            $neighbor_x^{min} = \text{Clamp}_N(x - \lfloor \frac{\mathbb{W}}{2} \rfloor)$
            $neighbor_x^{max} = \text{Clamp}_N(x + \lfloor \frac{\mathbb{W}}{2} \rfloor)$
            if $neighbor_y \neq y$ or $neighbor_x \neq x$ **then**
                $vertex_{neighbor_y, neighbor_x} = \mathbb{M}[neighbor_y, neighbor_x]$
                $G \leftarrow G \bigcup_+ \{(vertex_{y,x}, vertex_{neighbor_y, neighbor_x})\}$

where operator Clamp_N constraints the input in the real value range $[0, N]$ and is given by the following equation:

$$\text{Clamp}_N(v) = \max(0, \min(v, N)) \tag{3.6}$$

3.2.3.3 The Hierarchical Proximity Graph Content Similarity Metric (HPS)

As the name suggests, this similarity metric utilizes the **Hierarchical Proximity Graph** model to represent nanorough surfaces. In contrast to NGG and A2D, HPS does not use a representative graph, but rather maintains a collection of distinct hierarchical proximity graphs corresponding to individual nanorough surfaces. Moreover, the appropriateness of a nanorough surface is calculated using 2.60.

As described in section 2.5.2.2, HPGs utilize a per-level index to keep track of the symbols that serve as vertices on the per-level graphs that make up the HPG. This index is a fuzzy key-value like storage, referred to as **Graph Index**, wherein graphs serve as keys to symbol values. The Graph Index is responsible for mapping from a graph to a symbol.

The mapping is not an exact one and graphs that are *"close enough"* are treated as identical. Two graphs are considered identical based on how their similarity value compares to the hyper-parameters, **minimum** and **maximum merging margin** ($\mu_{\text{min}}, \mu_{\text{max}}$).

Given a graph G, performing a **Graph Index** look-up is described by *Algorithm 7*

Algorithm 7 Perform a Graph Index look-up given a graph G

$S \leftarrow -1$
for $i \in [0, \cdots, \text{LEN}(GraphIndex)]$ **do**
    $similarity_{(G, GraphIndex[i])} = \text{NVS}(G, G_i)$
    if $similarity \geq \mu_{\text{max}}$ **then**
        $S \leftarrow i$
        return
    else
        if $similarity \geq \mu_{\text{min}}$ **then**
            $GraphIndex[i] \leftarrow GraphIndex[i] \bigcup \{G\}$
            $COUNT[i] = COUNT[i] + 1$
            $S \leftarrow i$
            return
        else
            if $1 - similarity \gtrapprox 0$ **then**
                $G \leftarrow (G \cap GraphIndex[i])'$
if $S < 0$ **then**
    $S \leftarrow \text{LEN}(GraphIndex)]$
    $GraphIndex[S] = G$
    $COUNT[S] = 1$
    return

As you can see, for every existing key-graph we calculate its similarity to the query-graph at hand. There are three scenarios:

1. If the similarity of the graph at hand with an existing graph/key is greater than the maximum merging margin, then simply return the corresponding integer symbol.
2. If the similarity of the graph at hand with an existing graph/key is not greater than the maximum merging margin but is greater than the minimum merging margin then merge the graphs together and again return the corresponding integer symbol.
3. In case none of the previous statements is true, remove any edges included in the key-graph from the query-graph and proceed with the rest of the graph-keys. If, even after processing the entirety of the graph index, there is no match then simply add a new entry corresponding to the supplied graph and return a newly created symbol indicated by the graph index size.

Having described the inner workings of the **Graph Index**, we shall now provide a more thorough explanation of the HPG construction procedure, described in section 2.5.2.2. Given an $N \times N$ input matrix $\mathbb{M}$ constructing an HPG is carried out by *Algorithm 8*

Algorithm 8 Construct a **Hierarchical Proximity Graph** G given an $N \times N$ input matrix $\mathbb{M}$. $\mathbb{W}$ denotes the original window size, whilst $\mathbb{W}^*$ denotes the current level window size

$PerLevelData[0] \leftarrow \mathbb{M}$
$PerLevelGraphs[0] \leftarrow \text{GraphOf}(\mathbb{M})$
for $level \in [1, NumberOfLevels]$ **do**
    $\mathbb{W}^* \leftarrow \mathbb{W} \cdot level$
    for $y \in [0, N]$ **do**
        for $x \in [0, N]$ **do**
            $\mathbb{M}^* \leftarrow \text{Submatrix}^{\mathbb{W}^*}_{(y,x)}(\mathbb{M})$
            s-neighborhood $\leftarrow \text{GraphOf}(\mathbb{M}^*)$
            $symbol \leftarrow \text{GraphLookUp}(\text{s-neighborhood})$
            $PerLevelData[level][y, x] \leftarrow symbol$
    $PerLevelGraphs[level] \leftarrow \text{GraphOf}(PerLevelData[level])$

where GraphOf denotes the per-level graph / s-neighborhood construction procedure. We used the **2D Array Graph** approach, described in *Algorithm 6*, to represent the s-neighborhoods, any n-gram graph variant method could be used instead. GraphLookUp refers to the process described by *Algorithm 7*. Finally, $\text{Submatrix}^{\mathbb{W}^*}_{(y,x)}$ refers to the process of extracting an $N \times N$ sub-matrix of $\mathbb{M}$, centered around coordinates (y, x) and is described by *Algorithm 9*.

Algorithm 9 Extracting a sub-matrix $\mathbb{M}^*$ of size $\mathbb{W}^* \times \mathbb{W}^*$ centered around coordinates (y, x)

$x^*_{\text{min}} = \text{Clamp}_N(x - \lfloor \frac{\mathbb{W}^*}{2} \rfloor)$
$x^*_{\text{max}} = \text{Clamp}_N(x + \lfloor \frac{\mathbb{W}^*}{2} \rfloor)$
$y^*_{\text{min}} = \text{Clamp}_N(y - \lfloor \frac{\mathbb{W}^*}{2} \rfloor)$
$y^*_{\text{max}} = \text{Clamp}_N(y + \lfloor \frac{\mathbb{W}^*}{2} \rfloor)$
for $y^* in [y^*_{\text{min}}, \cdots, y^*_{\text{max}}]$ **do**
    for $x^* in [x^*_{\text{min}}, \cdots, x^*_{\text{max}}]$ **do**
        $\mathbb{M}^*[y^* - y, x^* - x] = \mathbb{M}[y^*, x^*]$

3.2.3.4 The Fourier & Histogram Space Content Similarity Metric (FHS)

As mentioned before, FHS is the only non-graph-based content similarity metric. While training, it requires calculating and storing the *2D Fast Fourier Transform (2D FFT)* and histogram of height values corresponding to each nanorough surface belonging to the training data set. On evaluation time, calculating the appropriateness of a given nanorough surface $\mathbb{M}$ can be broken down into the following tasks (we denote the training data set as $\mathbb{T}$):

1. Calculating the 2D FFT of the provided nanorough surface.
2. Calculating the histogram of height values corresponding to the provided nanorough surface.
3. Calculating the *Root Mean Square Deviation (RMSD)*, with regards to the mean 2D FFT (RMSD_F) and histogram (RMSD_H), between the provided nanorough surface and a subset of the training data set (by default the training data set in its entirety).
4. Calculating the expression

$$\frac{1}{1+\frac{\mathrm{RMSD_H}(\mathbb{M},\mathbb{T})+\mathrm{RMSD_F}(\mathbb{M},\mathbb{T})}{2}} \tag{3.7}$$

   which corresponds to the appropriateness of the provided nanorough surface.

The whole procedure is described by *Algorithm 10*.

Algorithm 10 Calculating the FHS similarity of a nanorough surface $\mathbb{M}$, assuming an n-large collection FH of pairs of matrices and vectors corresponding to the 2D FFT and Histogram of the nanorough surfaces making up the training data set

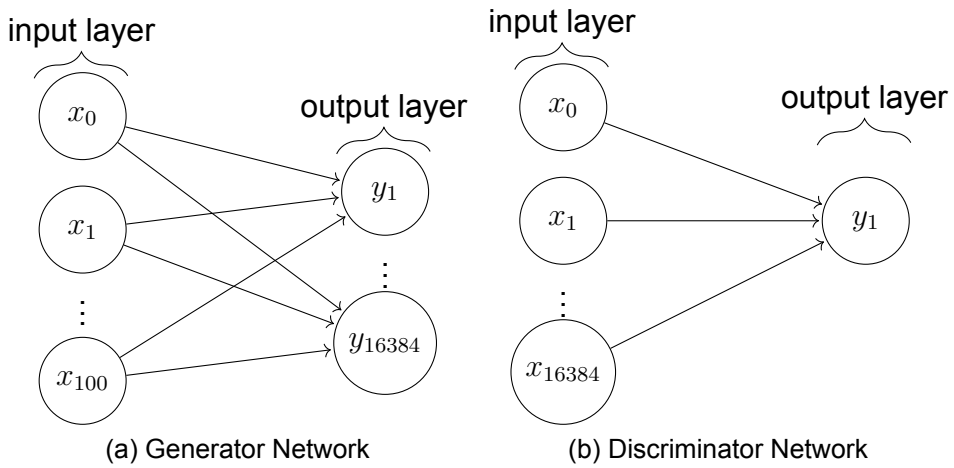
$H_{\mathbb{M}} \leftarrow \mathrm{Histogram}(\mathbb{M})$
$F_{\mathbb{M}} \leftarrow \mathrm{FF2D}(\mathbb{M})$
$loss_{total} \leftarrow 0$
for $Histogram, Fourier \in FH$ **do**
    $MeanSquareHistogramError \leftarrow \overline{(Histogram - H_{\mathbb{M}})^2}$
    $MeanSquareFourierError \leftarrow \overline{(Histogram - H_{\mathbb{M}})^2}$
    $loss = loss + \frac{\sqrt{MeanSquareHistogramError}}{\mathrm{n}\times 2}$
    $loss = loss + \frac{\sqrt{MeanSquareFourierError}}{\mathrm{n}\times 2}$
return $\frac{1}{1+loss_{total}}$

3.2.4 Frameworks

Two vastly different approaches to modeling nanorough surfaces were developed. We developed a **Single-Layer Perceptron GAN (SLPGAN)**, which is going to serve as our baseline and a DCGAN aiming at improving on the performance of our baseline model.

Both models expect a matrix (also referred to as *batch of latent vectors*) of size $BatchSize\times 100\times 1\times 1$ drawn from a Gaussian distribution. The models were trained on data sets containing nanorough surface representations of size $128\times 128\times 1$. The process can be generalized to different latent space vector and nanorough surface sizes.

3.2.4.1 Single-Layer Perceptron GAN (SLPGAN)



(a) Generator Network (b) Discriminator Network

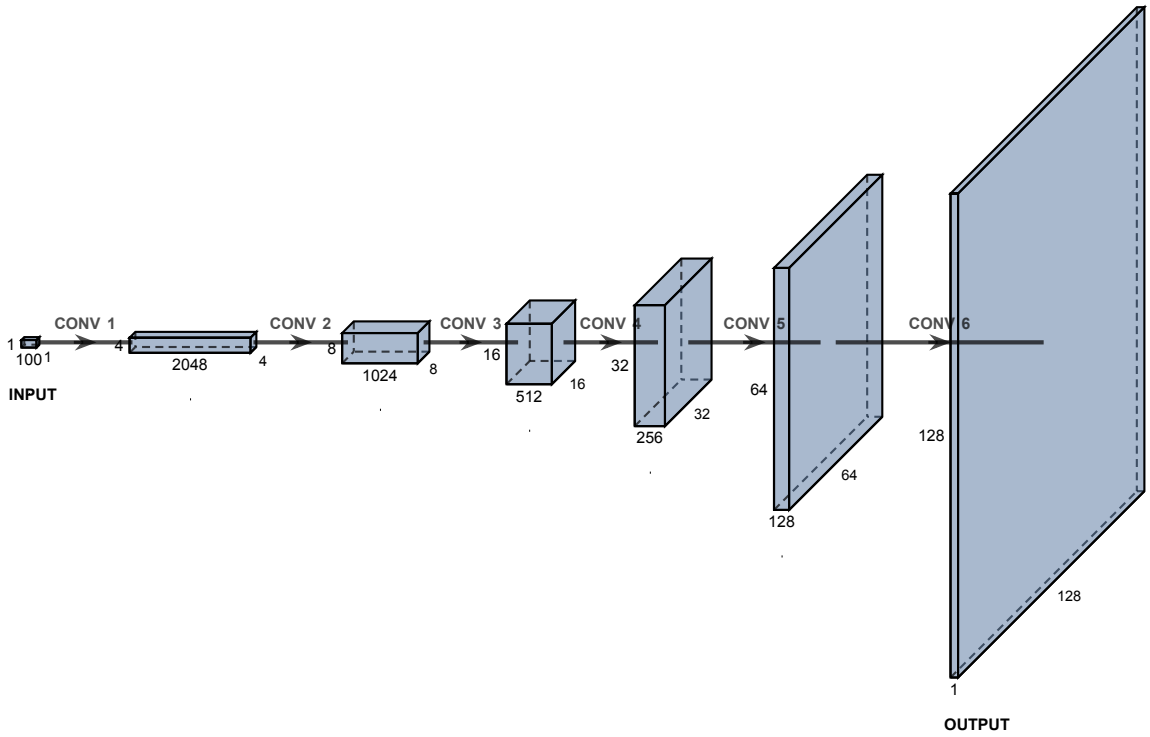
Figure 3.4: The SLPGAN framework. This visualization was produced using code adapted from *David Stutz*'s work [3].

The SLPGAN consists of two Single-Layer Perceptron networks that serve as the generator-discriminator pair.

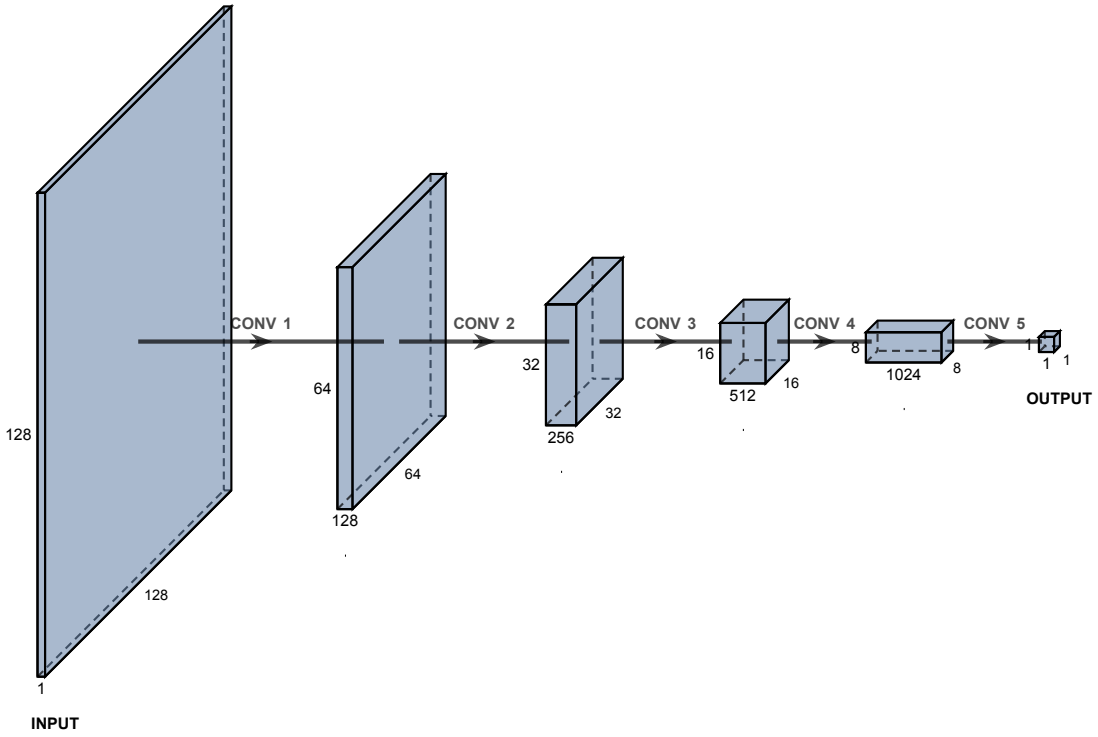
The generator consists of a single feed-forward layer with 100 input units and 16384 (128×128) output units. No activation function is used by the generator. Every batch processed by the SLP Generator must be initially *transformed* so that the feed-forward layer can process it. The output is also reshaped, so that it matches the nanorough surface matrix representation. For example, given a matrix of size $BatchSize \times 100 \times 1 \times 1$ i.e. a batch of $BatchSize$ $100 \times 1 \times 1$ latent vectors, the batch is transformed into a matrix of size $BatchSize \times 100$. It is then processed by the feed-forward layer which outputs a matrix of size $BatchSize \times 16384 \times 1$, which is finally reshaped into a matrix of size $BatchSize \times 1 \times 128 \times 128$.

The discriminator consists of a single feed-forward layer with 16384 input nodes and a single output node and is paired with a *Sigmoid* activation function. Again the input is required to be transformed, and more specifically *flattened*, on model entry. Given an input matrix of size $BatchSize \times 1 \times 128 \times 128$, the matrix is firstly transformed into a two-dimensional matrix of size $BatchSize \times 16384$, which is then processed by the feed-forward layer and passed through the activation function. The resulting $BatchSize \times 1$ matrix contains the scores corresponding to the nanorough surfaces at hand.

3.2.4.2 Deep Convolutional GAN (DCGAN)



(a) Generator Network



(b) Discriminator Network

Figure 3.5: The DCGAN framework. This visualization was produced using software adapted from *PlotNeuralNet* [37]

Our DCGAN implementation is heavily based on the work of *Alec Radford et al.* [2] and shares the same basic principles. The changes made to the architecture mainly concern the difference in input dimensions. More specifically, the original implementation of

DCGAN supports 64×64 images, while in our case the supported size was increased to 128×128 units.

The generator is comprised of 6 deconvolutional layers. A deconvolutional layer is identical to a standard convolutional layer except that it is mainly used for up-sampling data instead. Every deconvolutional layer is paired with a batch normalization layer and a ReLU activation function, with the exception of the last layer. All deconvolutional layers employ a kernel size of 4×4, a stride of 2, and padding of 1 with the exception of the initial layer, which uses a stride of 1 and no padding.

As previously mentioned, the latent space vectors, that the model expects as input, are transformed into two-dimensional matrices corresponding to nanorough surfaces, via a series of strided two-dimensional deconvolutional layers.

The generator model does not utilize any activation function on the final layer, as constraining the model's output to a certain range is not desirable in this scenario.

The discriminator consists of 5 strided convolutional layers. The first layer is paired with a LeakyReLU activation function, the 3 following layers are additionally paired with a batch normalization layer each, while the last layer is only paired with a Sigmoid activation function. Every layer utilizes a kernel size of 4×4, a stride of 2, and a padding of 1, with the exception of the last layer, which utilizes a kernel size of 8×8, a stride of 1, and no padding.

The Sigmoid activation function is used to produce the scores of the nanorough surfaces at hand. Strided convolution is used, in favor of pooling, to down-sample the initial input as it lets the network learn its own pooling function. Batch normalization along with the $\mathrm{LeakyReLU}$ activation function promote healthy gradient flow through the discriminator, which in the context of the GAN framework, is critical for the learning process of the discriminator, as well as the generator model.

3.2.4.3 Training

When it comes to training our models, we used a slightly modified version of the GAN training algorithm presented in [1] and described by *Algorithm 1*. In our version, we alter the generator's loss based on the the output of one of the graph-based content similarity metrics. More specifically, on every batch iteration, the output of the generator, which corresponds to a batch of artificial nanorough surfaces, is passed through the chosen graph-based content similarity and the appropriateness of the provided nanorough surfaces is calculated. These values, one for every sample in the batch are used to calculate the per-batch average appropriateness of artificial data. This process is carried out for every single batch making up the training data set and the values corresponding to single artificial batches are used to calculate the running average of the per-batch average appropriateness of artificial data. Hence, the per batch iteration generator loss $GeneratorLoss_n$ is:

$$GeneratorLoss_n = \frac{\mathrm{BCE}(\mathbb{D}(\mathbb{G}(Noise)), Labels)}{\epsilon + \sum_{i=0}^{n} \frac{\mathrm{CS}(\mathbb{G}(Noise)))}{N}} \tag{3.8}$$

where $\mathbb{G}$ and $\mathbb{D}$ correspond to the generator and discriminator networks respectively and

BCE denotes the **Binary Cross Entropy** Loss and is defined as:

$$\mathrm{BCE}(x, y) = -\frac{1}{N}\sum_{n=0}^{N}[y_n \cdot \log(x_n) + (1\boxtimes y_n) \cdot \log(1\boxtimes x_n)] \tag{3.9}$$

x and y are N sized vectors corresponding to the predicted and real label values respectively. **BCE** is commonly used when training a binary classifier, such as our discriminator model and serves as a means of determining how accurate a model really is.

$\mathbb{D}(\mathbb{G}(Noise))$ results in the creation of a $BatchSize \times 1$ vector containing values in the range $[0, 1]$ corresponding to the scores of the artificial nanorough surfaces generated by the generator ($\mathbb{G}(Noise)$). When training the generator the $Labels$ or y is a vector of size $BatchSize \times 1$ filled with ones and $\mathrm{BCE}(\mathbb{D}(\mathbb{G}(Noise)), Labels)$ corresponds to the standard GAN generator loss calculation.

The denominator is calculated by dividing the accumulated content similarity values (CS) of all so far processed batches by the total number of batches (N). Finally, a padding value of $0 \leq \epsilon \leq 1$ is added to the denominator. For our purposes, a value of 0.5 was chosen. Notice that similarity values close to 0 result in greater loss values, while similarity values close to 1 result in smaller loss values, compared to ones returned by the standard GAN training procedure. The intuition behind this approach is that the introduction of the content similarity metric will serve as an implicit learning rate scheduling mechanism, providing the generator with valuable feedback as to the quality of the generated nanorough surfaces and guiding throughout the training process. Initially, we were merely adding the content loss i.e. $1 - \mathrm{CS}(\mathbb{G}(Noise)))$, but we quickly noticed that this approach would have little to no effect to the value of the generator loss. This occurs due to BCE taking any possible value in $\mathbb{R}^+$, while CS only able to take values in the range $[0, 1]$.

We opted to train the discriminator on a certain batch only once ($k = 1$) as this was the cheapest option computationally-wise and is the approach followed by the authors of the original paper.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

We are going to be training each model for 100 epochs, using *Backpropagation* and the *Adam* optimization algorithm with a learning rate of 2×10^{-4}, $\beta_1 = 0.5, \beta_2 = 0.999$ and a batch size of 32. The data are going to be reshuffled on every epoch.

We are going to be evaluating our framework based on the per-epoch discriminator output and the loss of both the generator and discriminator models per epoch. More specifically, we are going to be monitoring the following expression

$$\mathsf{GeneratorLoss} \times (\mathsf{DiscriminatorLoss} + 1) \tag{4.1}$$

Expression 4.1 corresponds to the generator's *discriminator-relative performance* per epoch. This measure is going to be indicative of how well the generator performs with respect to how efficient of a discriminator, it is required to face. This metric is essential, as a generator model that succeeds in tricking a discriminator model, only because of the discriminator model's poor performance, should not be considered well-behaved.

In addition to the discriminator's output per epoch, we are going to be examining the expression following expression

$$|D(\mathsf{Real}) - 0.5| + |D(\mathsf{Fake}) - 0.5| \tag{4.2}$$

where $D(\mathsf{Real})$ and $D(\mathsf{Fake})$ correspond to the per-epoch mean output of the discriminator, when it comes to real and artificially generated nanorough surfaces, respectively. Ideally, this amount would converge to 0 implying that the discriminator is unable to distinguish real from artificially generated nanorough surfaces.

We are also going to be examining the mean FHS content similarity of the nanorough surfaces produced by the generator model. This is going to provide us with significant insight with regards to how realistic the resulting nanorough surfaces can be considered and whether or not the generator is trained successfully.

We are going to be training our models on a data set consisting of 1000 128×128 nanorough surfaces generated using a configuration of *RMS* $= 3$, *Skewness* $= 0$, *Kurtosis* $= 3$, *Correlation Lengths* $= (\xi_y, \xi_x) = (8, 8)$ and *Smoothing Factor* $= \alpha = 1$.

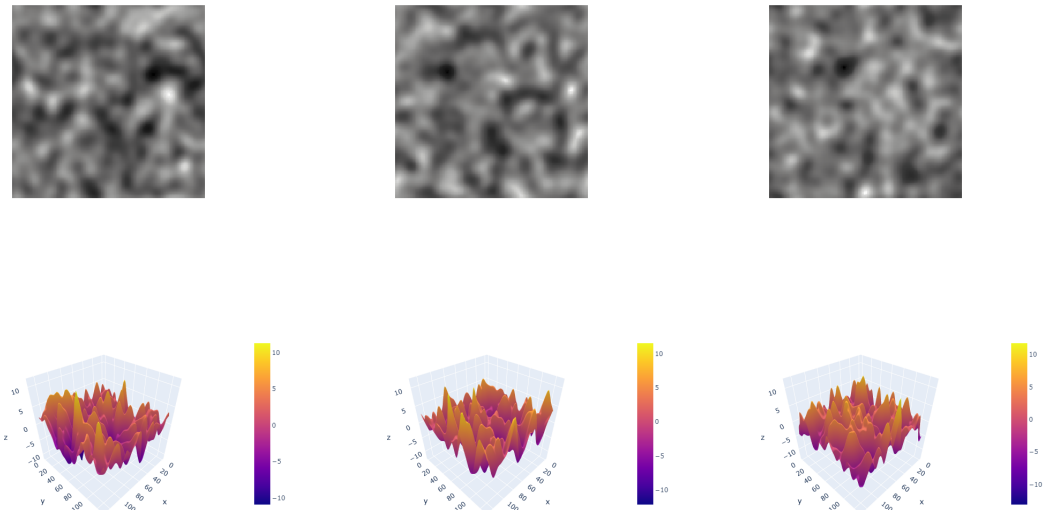


Figure 4.1: Real nanorough surface samples ($\xi_y = 8, \xi_x = 8, \alpha = 1$)

The grayscale and 3D surface representation of a few nanorough surface samples belonging to this data set are shown in *Figure 4.1*.

The experiments were carried out in a workstation equipped with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 Mhz with 6 Core(s) and 12 Logical Processor(s), 16,0 GB of RAM, an NVIDIA GeForce GTX 1660 Ti GPU with an additional 6GB of VRAM and 24 processing units.

4.1.1 DCGAN Weight Initialization

According to the original paper by *Alec Radford et al.* [2], all weights of the DCGAN architecture are supposed to be initialized from a zero-centered Normal distribution with a standard deviation of 0.02.

This initialization scheme proved prohibitive to the training of our models, as it resulted in consistently high generator loss values and the generator completely diverging. This is showcased in *Figure 4.2*

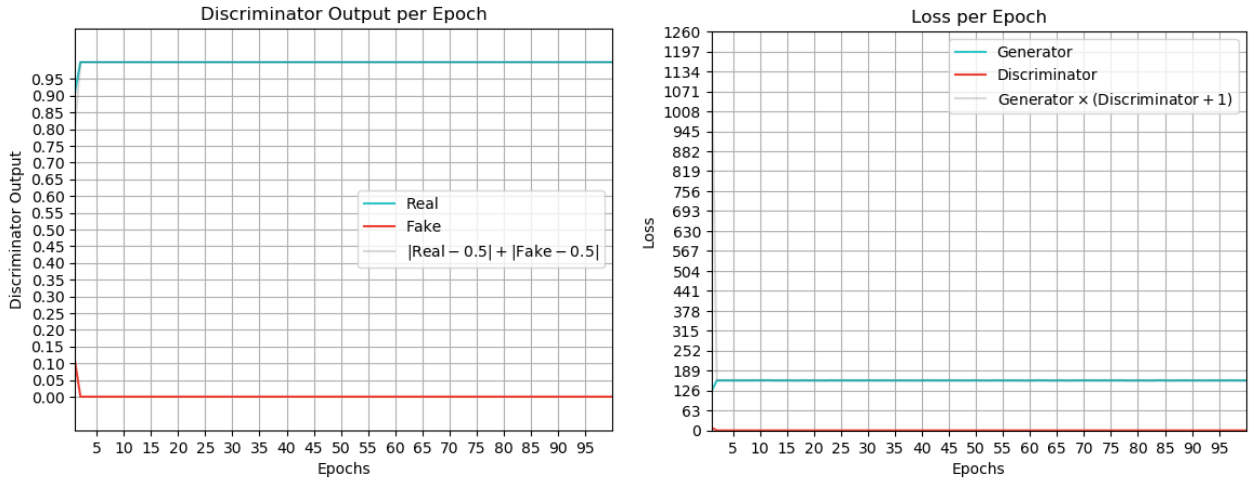


Figure 4.2: Training DCGAN with the conventional weight initialization scheme

We therefore decided that, we will not be utilizing this initialization scheme in the rest of our experiments.

4.1.2 Evaluating the scalability of the Content Similarity Metrics

We will now be comparing the various graph-based content similarity metrics, that we developed with regards to their scalability.

We will be training every content similarity metric, with different numbers of nanorough surfaces of different dimensions. We will be using 2, 4, 6, 8 or 10 nanorough surfaces of dimensions 2×2, 4×4, 8×8, 16×16 and 32×32. Each content similarity metric is going to be trained on every possible pairwise combination of the aforementioned values.

4.1.2.1 Scalability of the N-Gram Graph Content Similarity Metric

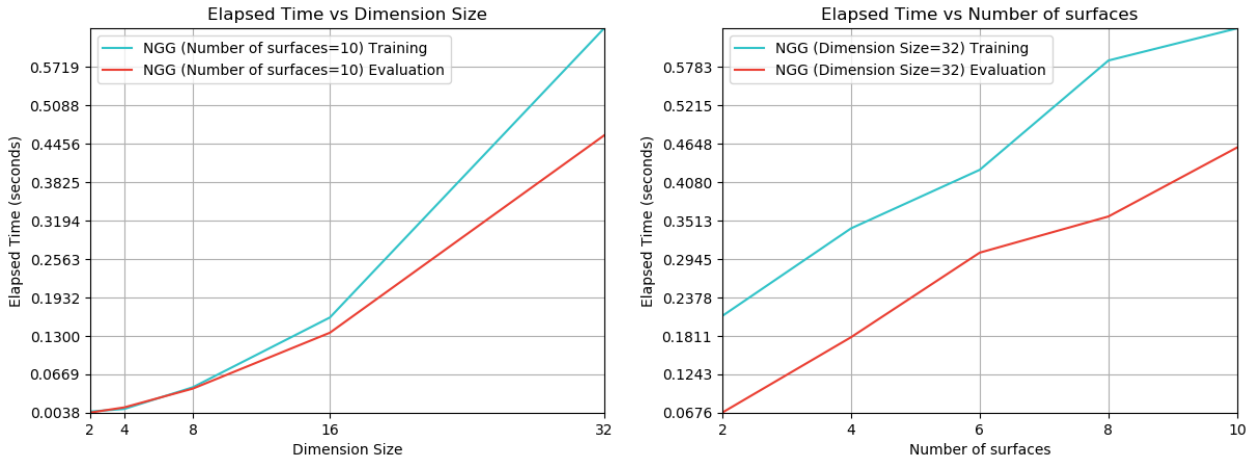


Figure 4.3: NGG scalability

NGG's computational cost with regards to both training and inference increases logarithmically with respect to the dimension size. In fact, every time the dimension of the nanorough surfaces is squared, the training/inference time increases by a factor of ≈ 4. NGG scales linearly, when it comes to both training and inference, with respect to the number of surfaces. This behavior is showcased in *Figure 4.3*.

4.1.2.2 Scalability of the Two-Dimensional Array Graph Content Similarity Metric

A2G showcases a similar behavior to NGG. In fact, the training and evaluation computational cost with respect to dimension size again increases logarithmically, whilst it increases linearly with respect to the number of surfaces.

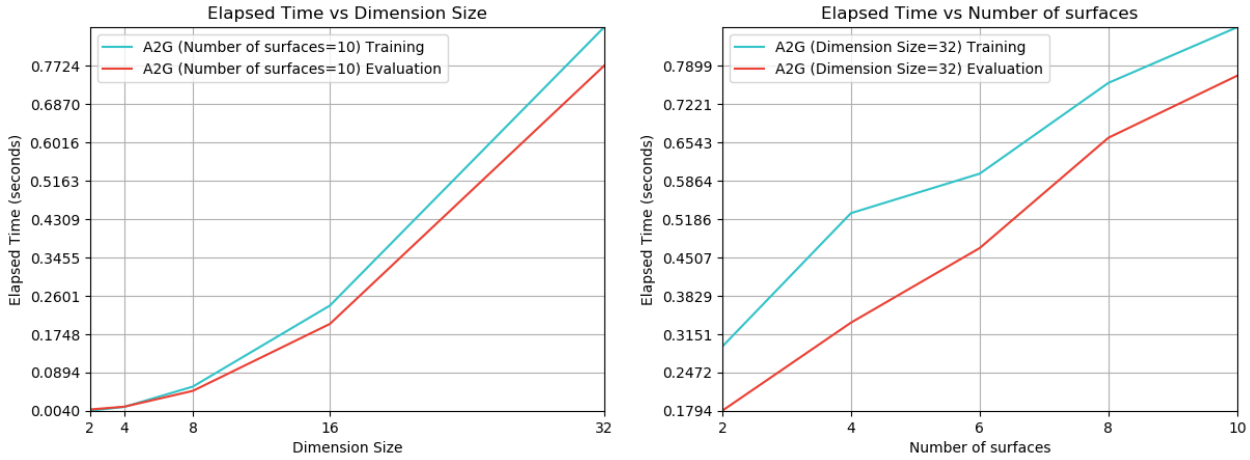


Figure 4.4: A2G scalability

4.1.2.3 Scalability of the Hierarchical Proximity Graph Content Similarity Metric

Figure 4.5 indicates that HPS scales exponentially with regards to the nanorough surface's dimension size and linearly with regards to the number of nanorough surfaces required to be processed. *Figure 4.6* further supports this.

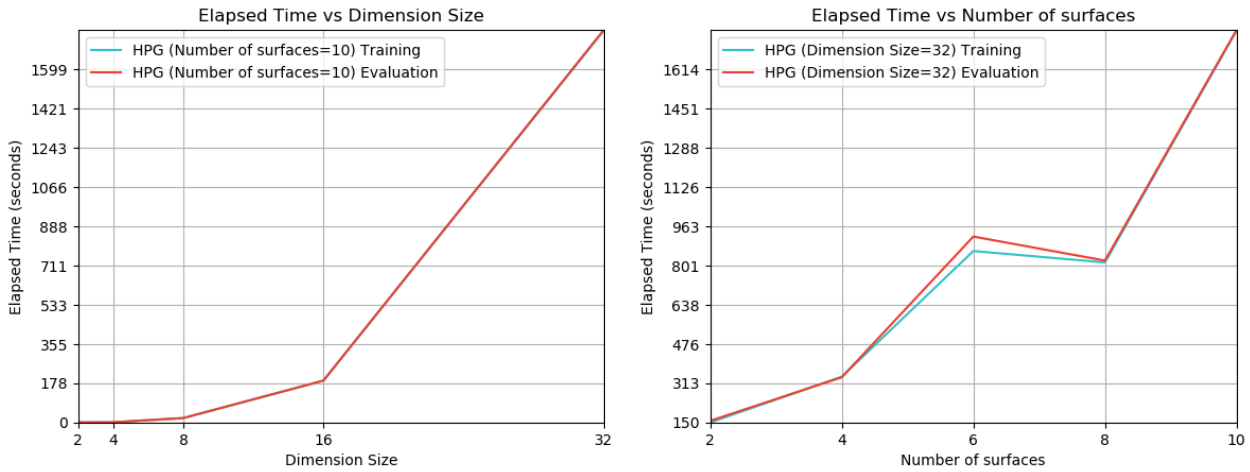


Figure 4.5: HPS scalability

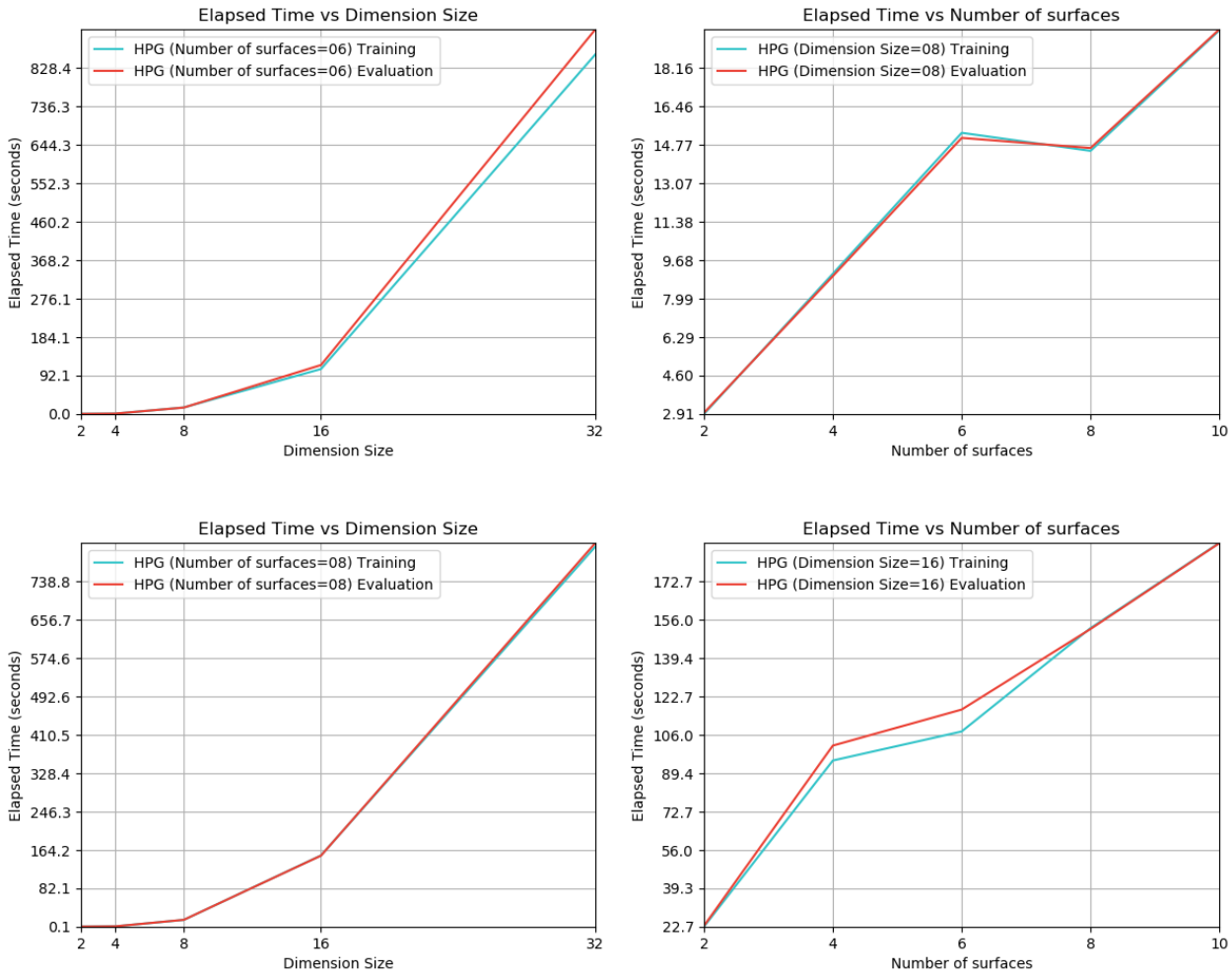


Figure 4.6: HPS scalability (Additional cases)

NGG proved to be the computationally cheapest of them all. A2G is a bit more costly computationally-wise when compared to NGG, but not by a great margin. A2G's per sample (nanorough surface) graph construction from a 2D matrix is a lot more complex when compared to the NGG's vector (flattened matrix) graph construction. HPS is the most costly of them all and by a great margin. Throughout, the HPG graph construction process multiple 2D array graphs are created representing the different neighborhoods making up the nanorough surface. HPGs also utilize multiple levels of graphs, as well as a graph index. As a result, the greater computational cost is to be expected. This is the reason why we will not be utilizing the HPS content similarity metric throughout the rest of our experiments.

4.1.3 Evaluating the FHS Content Similarity Metric

The FHS Content Similarity measure had to undergo thorough evaluation, so that we were able to determine how good of a nanorough surface similarity metric it really is. This was required, as we will be using FHS to evaluate the realisticity of the artificially generated nanorough surfaces.

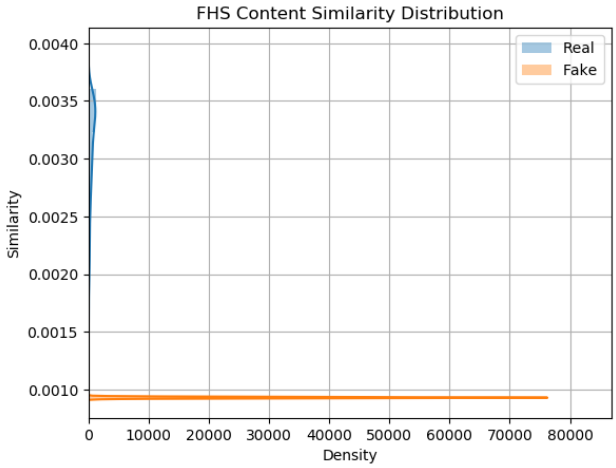


Figure 4.7: Comparing the FHS values of real and artificial nanorough surface samples ($\xi_y = 8, \xi_x = 8, \alpha = 1$)

The nanorough surfaces used in the evaluation of FHS originate from a nanorough dataset corresponding to a configuration of $\xi_y = 8, \xi_x = 8, \alpha = 1$, while the artificially generated nanorough surfaces were created by populating a two-dimensional matrix with values sampled from a Gaussian distribution and then scaled by the RMS characterizing the real nanorough surfaces.

Figure 4.7 showcases that FHS is able to distinguish nanorough surfaces originating in the real data distribution from artificial ones.

4.2 Results and Discussion

4.2.1 SLPGAN paired with A2G

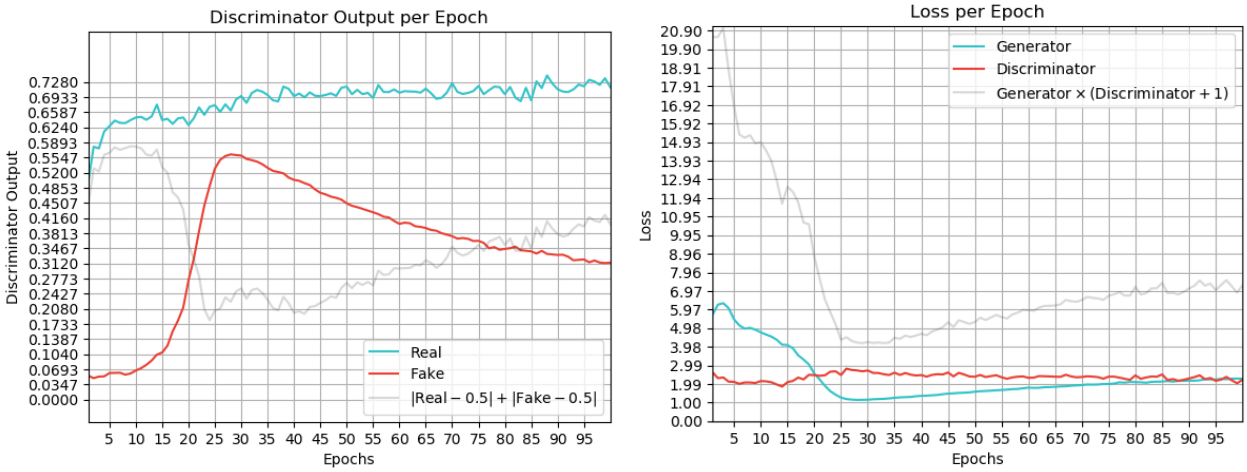


Figure 4.8: Training SLPGAN

As indicated by *Figure 4.8*, the SLP discriminator fails to consistently identify samples originating from the generator's data distribution. It is apparent that, especially during the first few training epochs, the generator is able to trick the discriminator into assigning

higher than expected scores to artificially generated nanorough surfaces. Consequently, the generator is not provided with appropriate feedback and is improperly trained. After the 25-30 training epoch mark, the discriminator seems to finally be able to distinguish real from fake data.

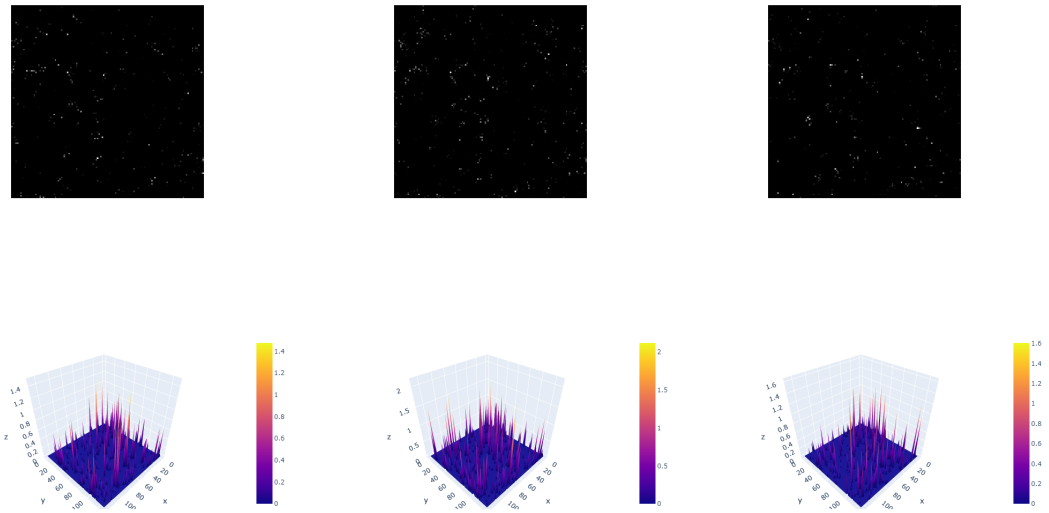


Figure 4.9: Nanorough surface samples generated by SLPGAN

Sadly, the SLPGAN framework suffers from the **Mode Collapse** problem, meaning that the generator learns to produce only a small set of outputs over and over again. This becomes obvious when comparing nanorough surfaces drawn from the real data distribution with nanorough surfaces artificially generated by the framework at hand (*Figure 4.1* and *Figure 4.9*).

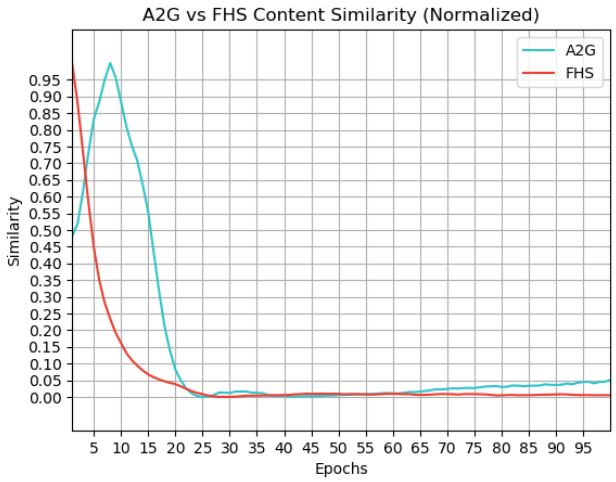


Figure 4.10: A2G & FHS similarity scores in the case of SLPGAN ($\xi_y = 8, \xi_x = 8, \alpha = 1$)

In *Figure 4.10* we can also see that both A2G and FHS stay close to 0, meaning that the SLPGAN is unable to generate realistic nanorough surfaces.

4.2.2 DCGAN

When using the DCGAN framework, even in the absence of any content similarity metric, the discriminator is able to train almost to optimality at every epoch, thus promoting the healthy training of the generator, which is consequently capable of better fitting the data compared to the SLPGAN case.

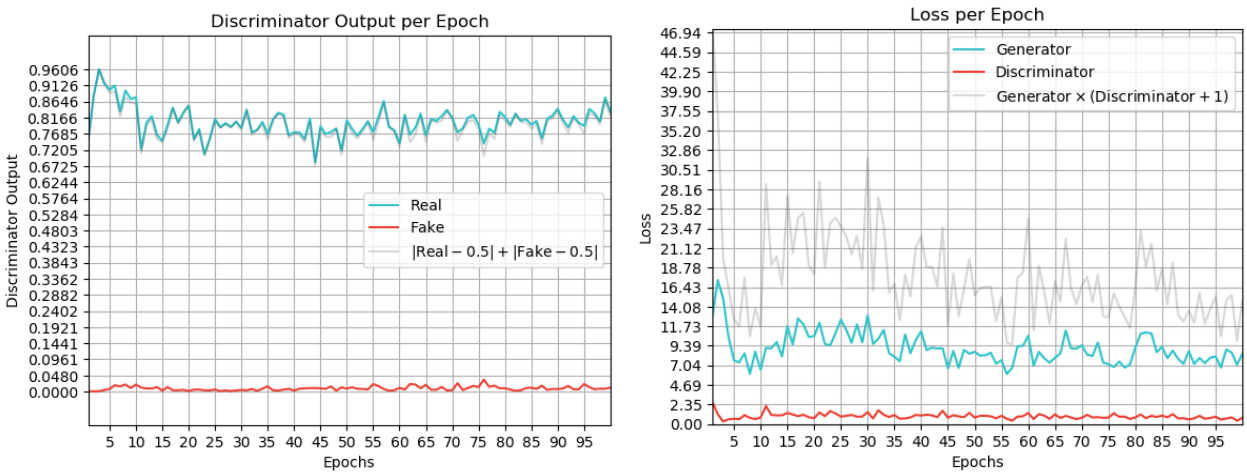


Figure 4.11: Training DCGAN

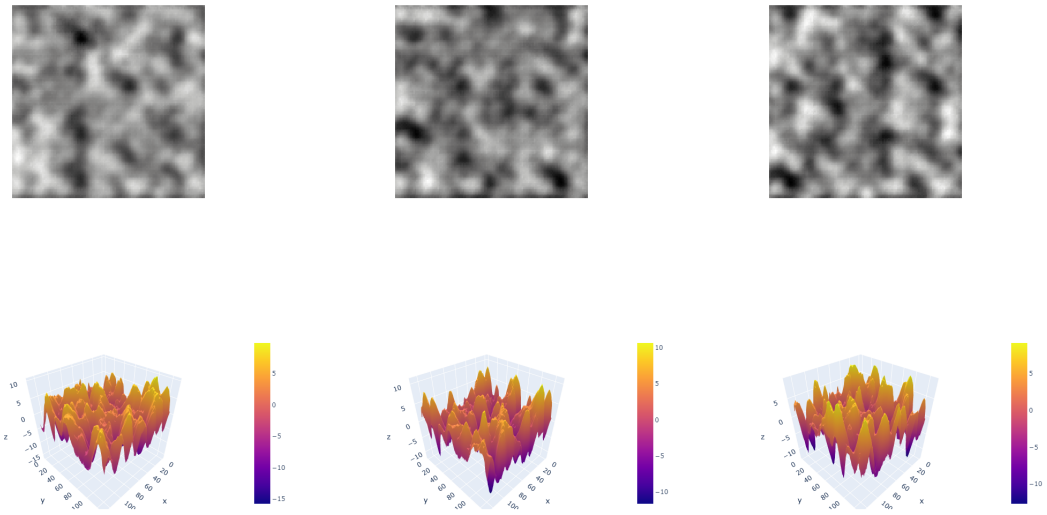


Figure 4.12: Nanorough surface samples generated by DCGAN

All issues aside, DCGAN is able to generate sufficiently realistic nanorough surfaces with regards to the topology of the original surfaces and the stochasticity characterizing these microstructures. One noticeable downside of this approach is the introduction of a substantial amount of noise to the resulting nanorough surface. The nanorough surfaces drawn from the real data distribution do not showcase such rampant changes in height values. This becomes apparent when comparing figures *Figure 4.1* and *Figure 4.12*.

4.2.3 DCGAN paired with NGG

DCGAN paired with the NGG Content Similarity is yet again able to produce sufficiently realistic nanorough.

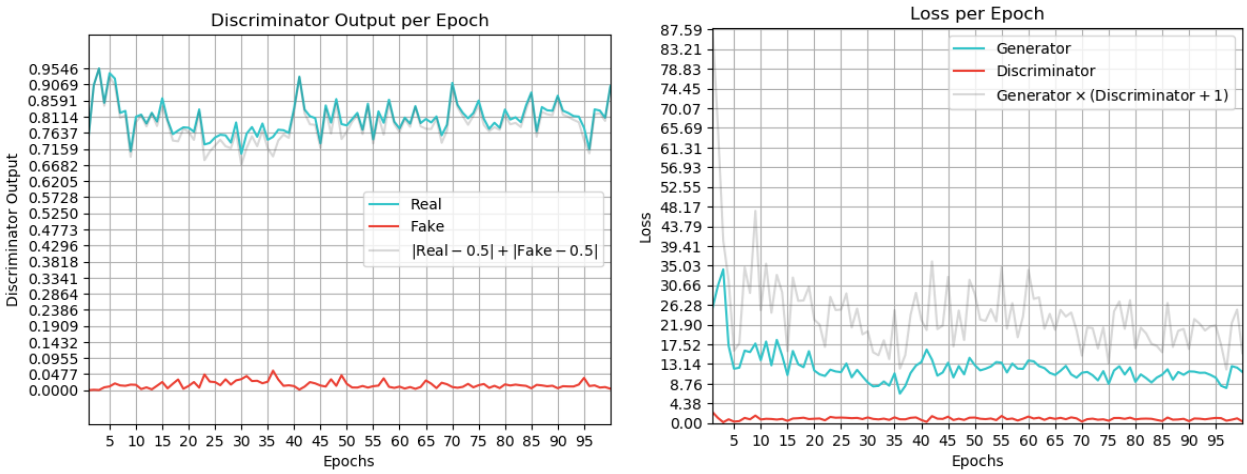


Figure 4.13: Training DCGAN paired with NGG

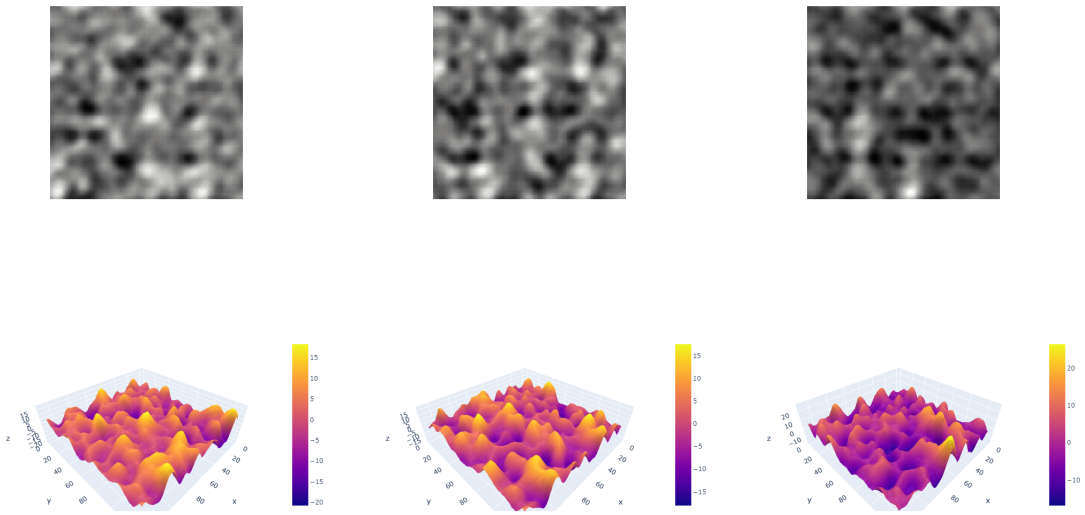


Figure 4.14: Nanorough surface samples generated by DCGAN+NGG

We notice from *Figure 4.14* that in this case a different type of noise is introduced. In contrast to 4.2.2, there are no rampant changes in the height values, instead the topology of the generated nanorough surfaces is characterized by similarly high mountains and valleys. This could be a result of the NGG content similarity metric failing to discriminate between topologies of different steepness.

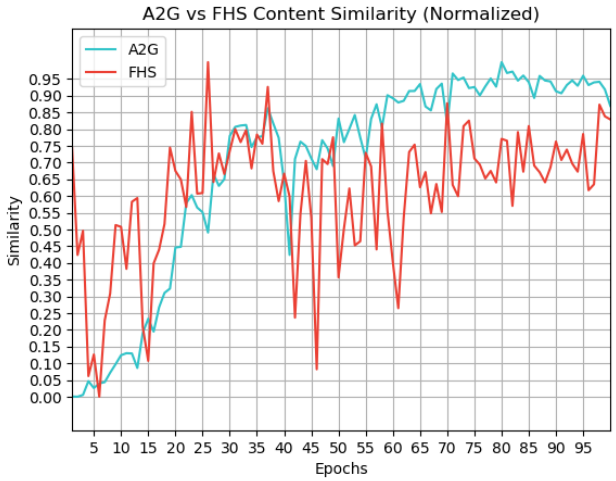


Figure 4.15: A2G & FHS similarity scores in the case of DCGAN+NGG ($\xi_y = 8, \xi_x = 8, \alpha = 1$)

Figure 4.15 showcases that both A2G and FHS reach a plateau after the 80^{th} epoch, indicating that the framework has reached its expressive limits.

4.2.4 DCGAN paired with A2G

As in 4.2.2, the discriminator showcases near-optimal performance when it comes to distinguishing real from artificially generated samples, throughout the training process, thus allowing the generator to properly train, by providing him with appropriate feedback.

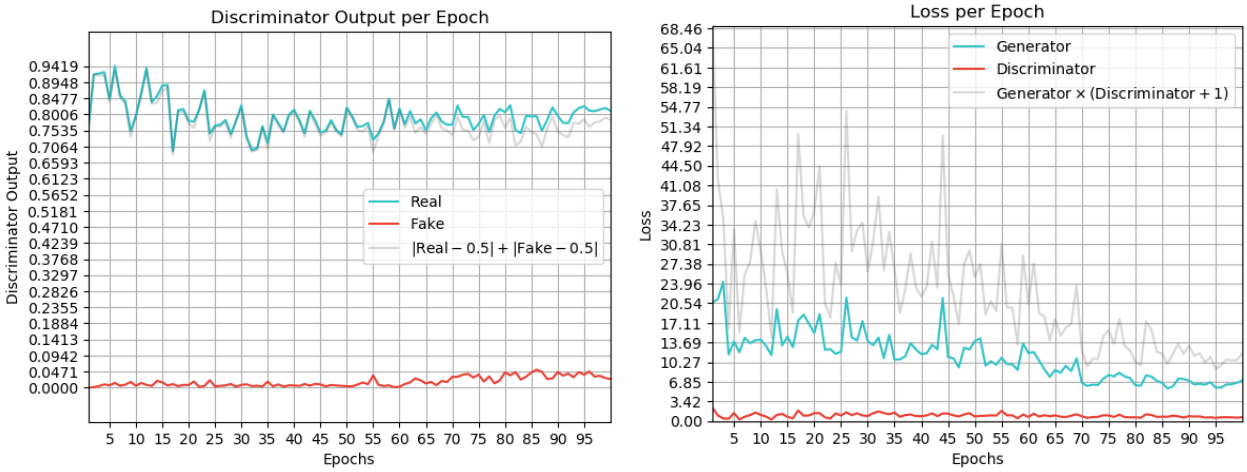


Figure 4.16: Training DCGAN paired with A2G

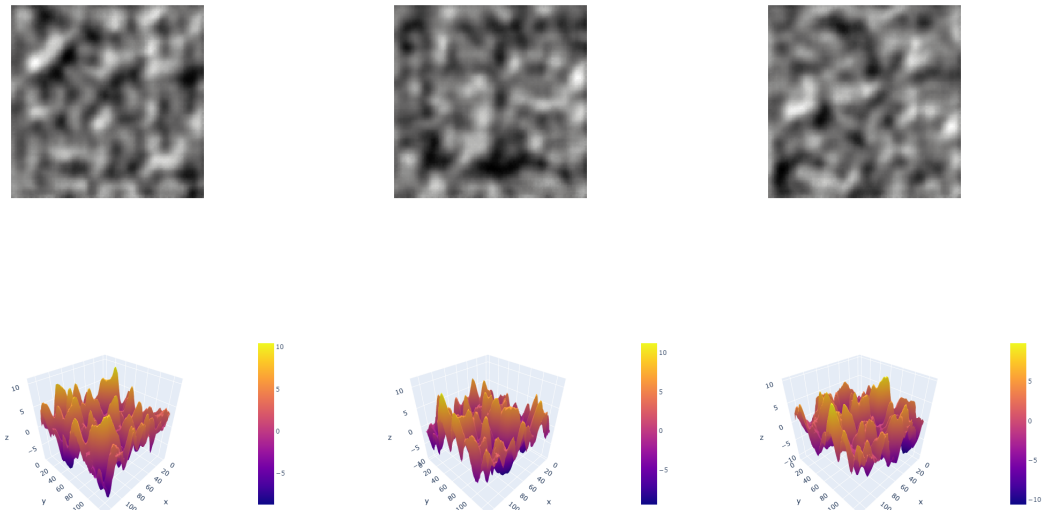


Figure 4.17: Nanorough surface samples generated by DCGAN+A2G

We can immediately tell two main advantages of this method over the conventional GAN training procedure. First of all, the discriminator tends to miscategorize sufficiently more samples in this case. Given that only the generator's training procedure is altered, this behavior indicates that the generator model produces even more realistic nanorough surfaces and is therefore capable of more frequently tricking the discriminator. Secondly, the per-epoch generator loss follows a clearly descending trajectory, which indicates that the training procedure is a lot more stable. As previously stated, the intuition behind this is that the augmented generator loss calculated by 3.8, serves as an implicit learning rate scheduling mechanism.

This approach showcases the best results so far with regards to mimicking the topology and the stochasticity of the real nanorough surface samples. In contrast to, utilizing no content similarity or NGG, this approach does not introduce rampant changes in height values nor does it result in a topology consisting of similarly high mountains and valleys.

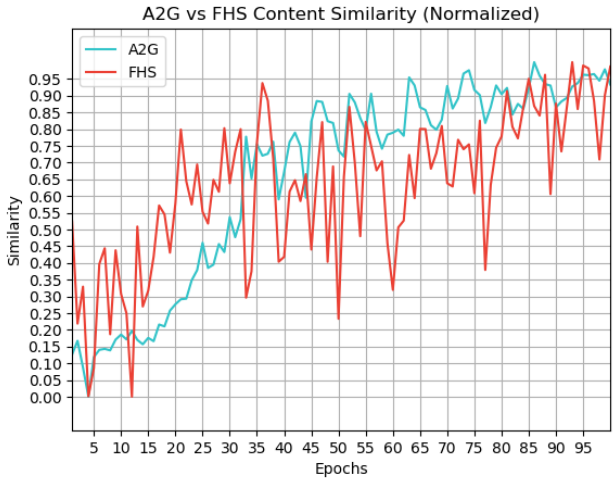


Figure 4.18: A2G & FHS similarity scores in the case of DCGAN+A2G ($\xi_y = 8, \xi_x = 8, \alpha = 1$)

It is clear from *Figure 4.18* that, both A2G and FHS content similarity increases as the

training procedure progresses. This is a strong indicator that the generator model is able to produce realistic nanorough surfaces.

4.2.5 Evaluating the statistical significance of our results

We decided to evaluate the statistical significance of our results using the **Wilcoxon signed-rank test** approach. The Wilcoxon signed-rank test is a non-parametric version of the paired t-test and, given a set of matched samples x and y, tests whether the distribution of the differences $x - y$ is symmetric about zero.

In our case, we are interested in comparing DCGAN paired with no content similarity metric, DCGAN paired with NGG, and DCGAN paired with A2G.

First of all, we trained the FHS evaluation content similarity metric on the $\xi_y = 8, \xi_x = 8, \alpha = 1$ data set. We then generated 30 nanorough surface samples using each method. These samples were generated by providing each generator with a fixed input drawn from a standard normal distribution. Having generated our 3 different nanorough surface populations, we collected the FHS values corresponding to each population. The distributions of these values are shown in *Figure 4.19*.

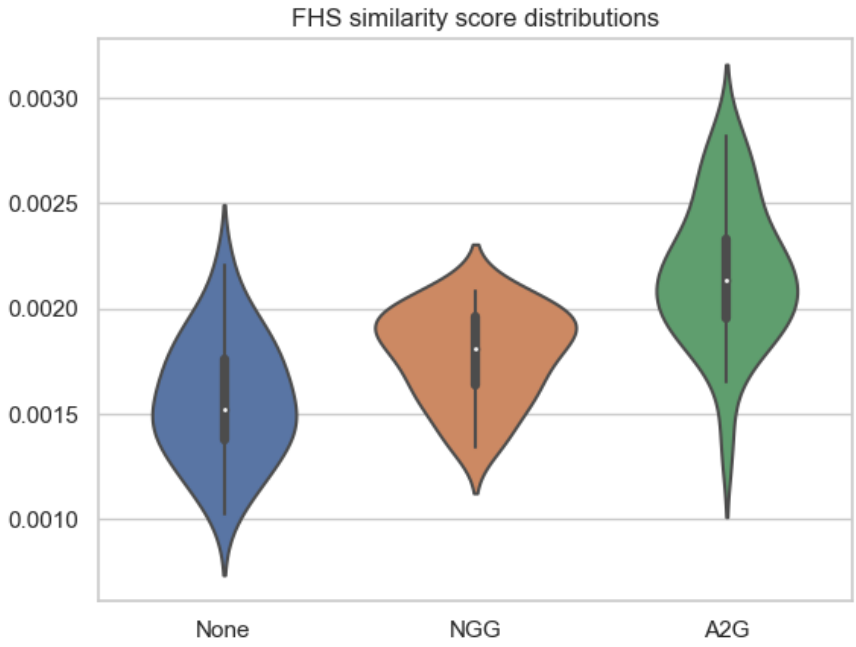


Figure 4.19: The FHS score populations used in the context of the Wilcoxon signed-rank tests

Having collected the FHS similarity scores corresponding to the 3 different populations of nanorough surfaces, we perform 3 **two-tailed** Wilcoxon signed-rank tests, with our null hypothesis being that the distribution of the differences $x - y$ is symmetric about zero.

Samples	W^+	p-value
A2G vs NGG	5.500000e+01	2.613431e-04
A2G vs None	1.000000e+00	1.920921e-06
NGG vs None	9.700000e+01	5.319684e-03

Table 4.1: The results of the two-tailed Wilcoxon signed-rank test with regards to different sample combinations. W^+ stands for the sum of the ranks of the differences above zero

The results are showcased in *Table 4.1*. We reject the null hypothesis, in every single case, at a confidence level of $< 1\%$, concluding that there is a **significant** difference in FHS similarity between each pair of nanorough surface groups.

We now need to determine how the 3 approaches rank with respect to FHS similarity. We perform 3 additional **one-tailed** Wilcoxon signed-rank tests, where we again compare the 3 approaches, the only difference being that now the null hypothesis is that the median is negative against the alternative that it is positive.

Samples	W^+	p-value
A2G vs NGG	4.100000e+02	1.306715e-04
A2G vs None	4.640000e+02	9.604606e-07
NGG vs None	3.680000e+02	2.659842e-03

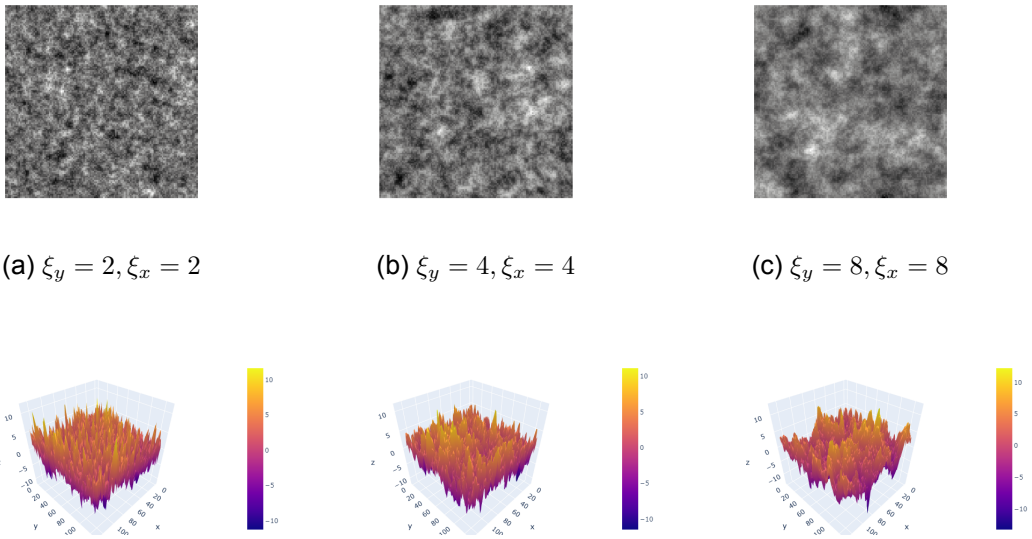
Table 4.2: The results of the one-tailed Wilcoxon signed-rank test with regards to different sample combinations. W^+ stands for the sum of the ranks of the differences above zero

The results of the one-tailed tests are shown in *Table 4.2*. We reject the null hypothesis that the median of the differences is negative in every single case. This implies that in every single case the first approach outperforms the second one. It is now apparent that there is a clear hierarchy with regards to FHS appropriateness and consequently with regards to the realisticity of the generated nanorough surfaces:

$$\mathrm{DCGAN} + \mathrm{A2G} > \mathrm{DCGAN} + \mathrm{NGG} > \mathrm{DCGAN}$$

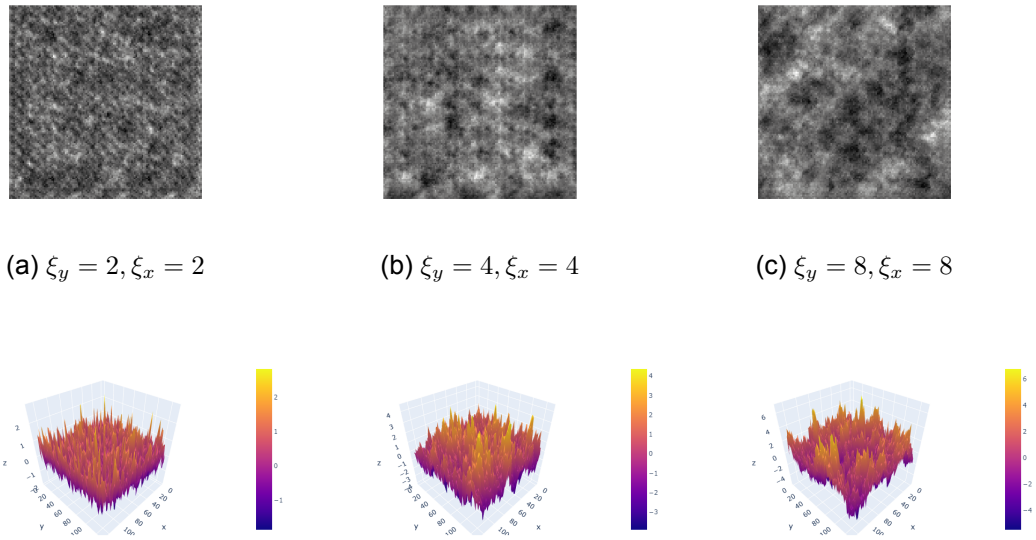
4.2.6 Training DCGAN paired with A2G on additional data sets

Having determined that DCGAN paired with the A2G content similarity metric showcases the most promising results, we tested this architecture on 5 additional data sets. These data sets correspond to different combinations of correlation lengths and alpha and should provide us with greater insight, as to how different parameters affect the performance of our framework. All data sets consist of 1000 nanorough surfaces and each nanorough surface is 128×128 pixels large.



(a) $\xi_y = 2, \xi_x = 2$ (b) $\xi_y = 4, \xi_x = 4$ (c) $\xi_y = 8, \xi_x = 8$

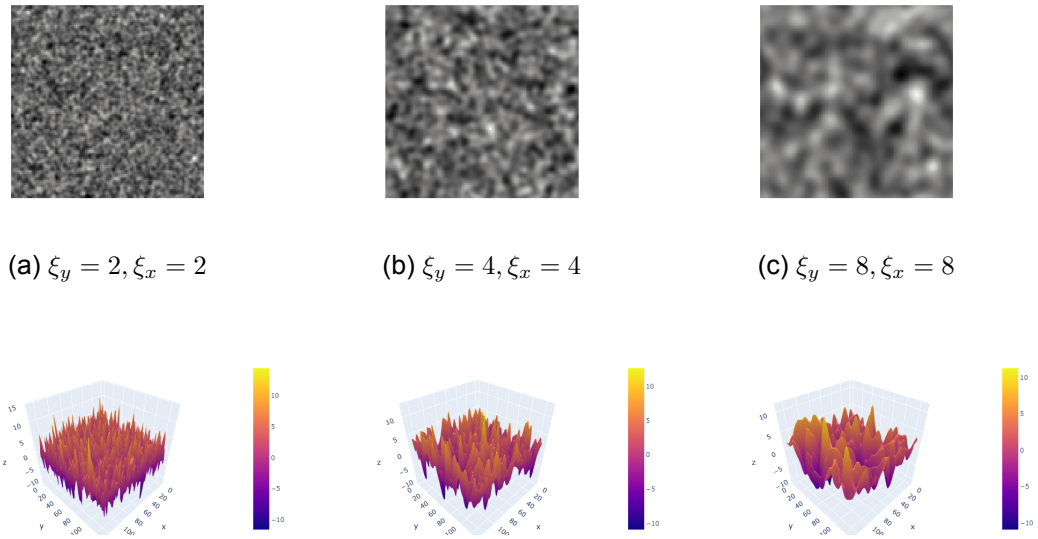
Figure 4.20: Real nanorough surface samples ($\alpha = 0.5$)



(a) $\xi_y = 2, \xi_x = 2$ (b) $\xi_y = 4, \xi_x = 4$ (c) $\xi_y = 8, \xi_x = 8$

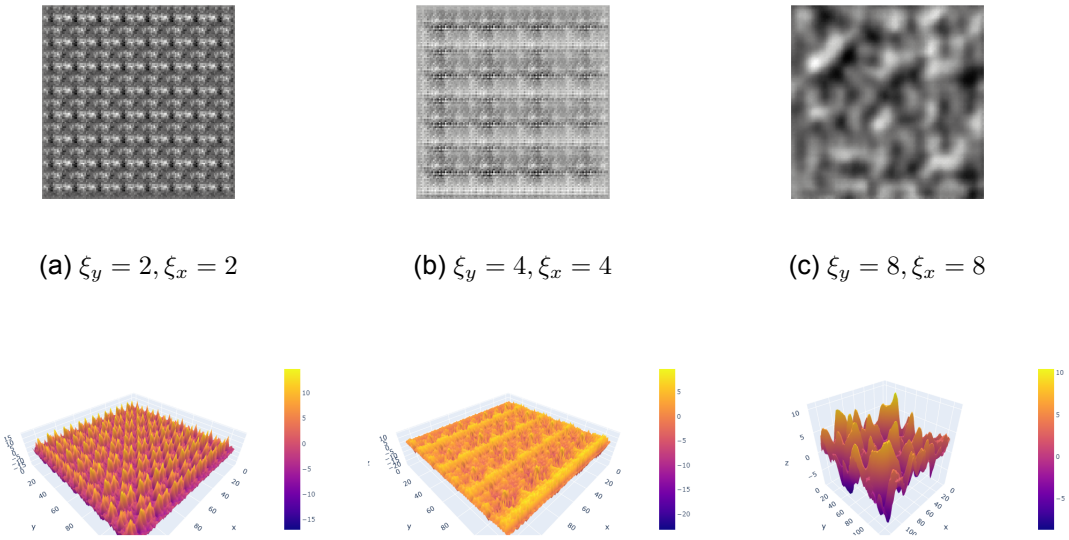
Figure 4.21: Nanorough surface samples generated by DCGAN+A2G ($\alpha = 0.5$)

DCGAN paired with the A2G content similarity metric is able to produce realistic nanorough surfaces, for data sets corresponding to an $\alpha = 0.5$. The nanorough surfaces appear noisy, but the amount of noise introduced is relatively insignificant. This problem could be mitigated by fine-tuning parameters such as the learning rate, the batch size, the optimization algorithm initial decay rates etc. but we are not going to be exploring this in this work. This behavior is showcased in *Figure 4.20* and *Figure 4.21*.



(a) $\xi_y = 2, \xi_x = 2$ (b) $\xi_y = 4, \xi_x = 4$ (c) $\xi_y = 8, \xi_x = 8$

Figure 4.22: Real nanorough surface samples ($\alpha = 1$)



(a) $\xi_y = 2, \xi_x = 2$ (b) $\xi_y = 4, \xi_x = 4$ (c) $\xi_y = 8, \xi_x = 8$

Figure 4.23: Nanorough surface samples generated by DCGAN+A2G ($\alpha = 1$)

As we can see from *Figure 4.22* and *Figure 4.23*, our framework fails to correctly model nanorough surfaces corresponding to $\alpha = 1$, regardless of the correlation lengths ξ_y, ξ_x, with the exception of $\xi_y = 8$ and $\xi_x = 8$. The smoothing quality, that higher values of α introduce, in combination with the small correlation length result in highly stochastic nanorough surfaces. The fact that, in this case, our training data so closely resemble random noise could be the reason why our framework fails to successfully model them.

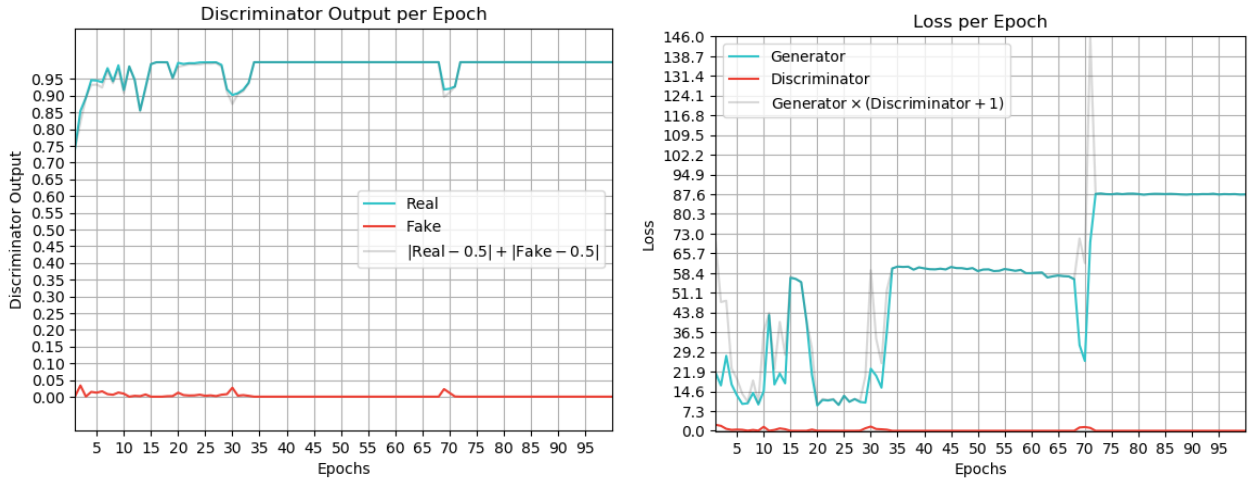


Figure 4.24: Training DCGAN paired with A2G ($\xi_y = 4, \xi_x = 4, \alpha = 1$)

A configuration of $\xi_y = 4$ and $\xi_x = 4$ corresponds to our framework's so far worst performance. As indicated by *Figure 4.24* the framework fails to fit the training data and the training procedure effectively halts after a few epochs. We decided to examine if utilizing no content similarity metric would result in a better performance.

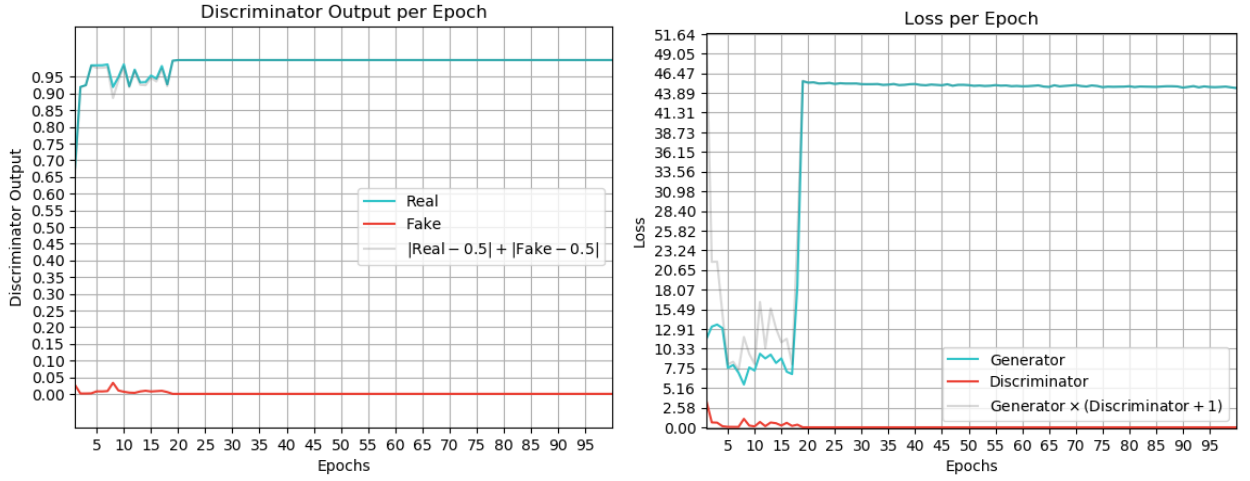
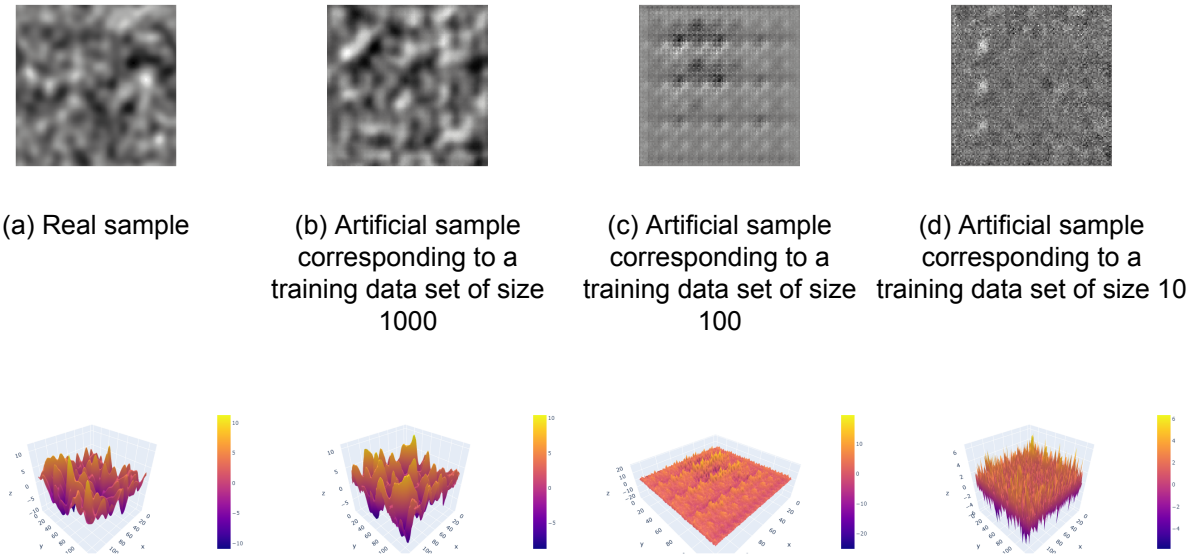


Figure 4.25: Training DCGAN ($\xi_y = 4, \xi_x = 4, \alpha = 1$)

Figure 4.25 demonstrates that DCGAN with no content similarity metric was unable to effectively fit the data as well. In fact, DCGAN paired with A2G was able to escape the local minimum, where the training procedure had previously halted, and carry on for a few more epochs.

4.2.7 Determining the minimal amount of training data required

In order to determine the amount of training data required, so that our generator model is able to generate realistic enough nanorough surface samples, we performed 2 experiments with 10 and 100 training samples originating from the $\xi_y = 8, \xi_x = 8, \alpha = 1$ data set.



(a) Real sample

(b) Artificial sample corresponding to a training data set of size 1000

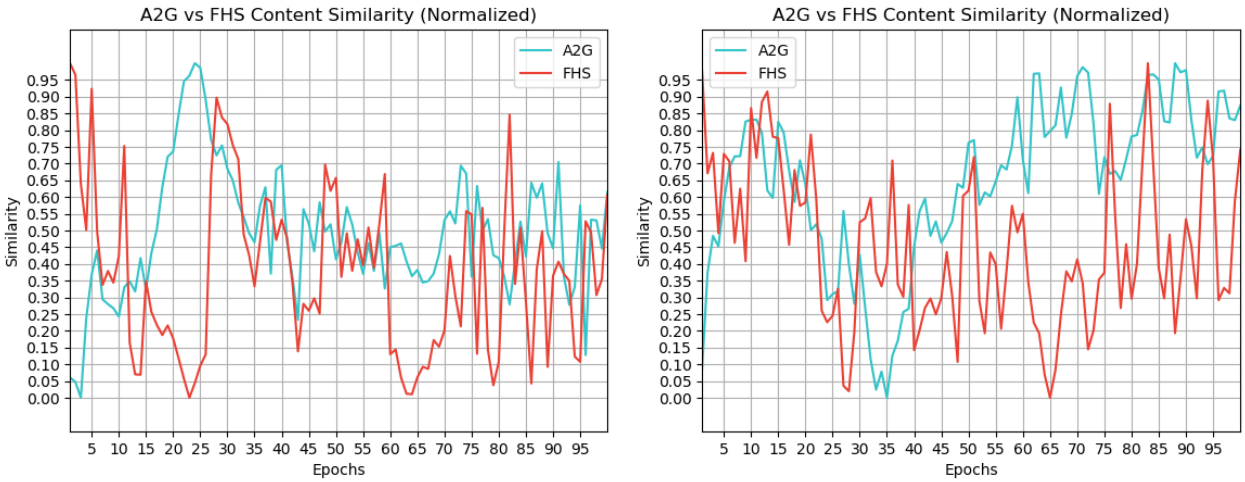
(c) Artificial sample corresponding to a training data set of size 100

(d) Artificial sample corresponding to a training data set of size 10

Figure 4.26: Comparing nanorough surface samples generated by DCGAN+A2G ($\alpha = 1$) with respect to the size of the training data set

It becomes immediately apparent from *Figure 4.26* that the amount of training samples required so that the generator model is capable of producing sufficiently realistic nanorough surfaces is close to 1000.

The generator having been trained on the data set consisting of 10 nanorough surfaces was unable to learn its characteristics and produced samples of pure noise. On the other hand, training the generator on 100 samples improved on the previous case, but again proved insufficient as the resulting nanorough surfaces appear flat and quite periodic. It is clear from *Figure 4.27*, that in the case of 10 training samples, the value of FHS showcases a decreasing tendency throughout the training process, while in the case of 100 training samples it does not.



(a) 10 training samples

(b) 100 training samples

Figure 4.27: A2G & FHS scores for different training data set sizes

4.2.8 Assessing the scalability of our framework

We will be examining the scalability of our framework with regard to nanorough surface generation. Given that our framework is aiming to provide an alternative solution to conventional Fourier-based methods, we will only be considering its performance on inference time.

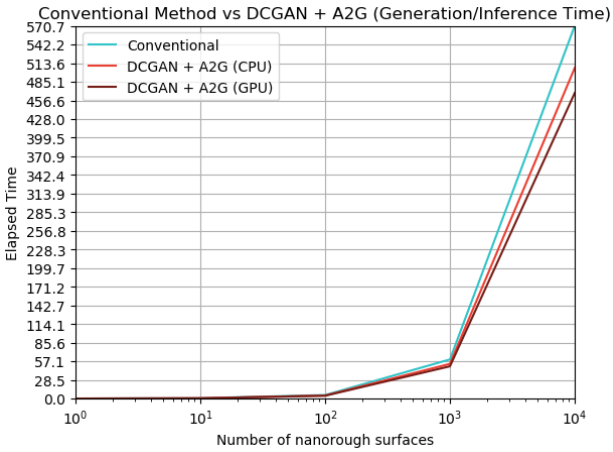


Figure 4.28: The generation cost as a function of the desired number of nanorough surfaces

We can clearly see from *Figure 4.28* that the required time to generate a given number of nanorough surfaces increases linearly to the number of nanorough surfaces in all 3 cases. We further observe that our approach outperforms that of *Antonios Stellas et al.* [16] with regards to required processing time. Utilizing the GPU provides us with even greater results.

5. CONCLUSIONS AND FUTURE WORK

In this work, we examined how GAN-based frameworks can be trained to generate realistic nanorough surfaces.

We developed 4 nanorough surface content similarity metrics, the **N-Gram Graph (NGG)**, **2D Array Graph (A2G)**, **Hierarchical Proximity Graph (HPS)**, and **Fourier & Histogram Space (FHS)** content similarity metrics. FHS assisted us in evaluating our framework, with regards to the realisticity of the generated nanorough surfaces, while the other 3 metrics were designed to be used during training, so that they provide additional feedback to the generator model and guiding it throughout the training process. HPS was ruled out, due to its prohibitive time requirements.

A **Single-Layer Perceptron GAN (SLPGAN)** served as our baseline. Due to its simplistic architecture, this model suffers from the *Mode Collapse* problem, which is quite common in the field of GANs, and was able to produce only a small subset of nanorough surfaces. More specifically, the SLPGAN generated nanorough surface with very small vertical fluctuations that closely resemble a flat plain.

We also developed a **Deep Convolutional GAN (DCGAN)**, which generally speaking was able to closely fit the training data and generate nanorough surfaces indistinguishable from real ones. DCGAN failed to model training data sets corresponding to configuration of small correlation lengths ($\xi_y = \xi_x \leq 4$) and high α ($\alpha = 1$) values, where the great stochasticity characterizing the data results in them closely resembling random noise. DCGAN performed the best on the $\xi_y = \xi_x \leq 8$, $\alpha = 1$ data set and worst on the $\xi_y = \xi_x \leq 4$, $\alpha = 1$ data sets. A number of nanorough surface training samples close to 1000 is minimally required, so that the generator is able to produce sufficiently realistic nanorough surfaces.

Lastly, we carried out multiple one and two-tailed Wilcoxon signed-rank tests, comparing the FHS similarity scores of DCGAN, DCGAN+NGG, and DCGAN+A2G, and confirmed that **DCGAN+A2G** outperforms the rest by a substantial margin.

There are a lot of different aspects of the problem that we weren't able to explore and an even greater number of questions that remain unanswered. Some topics that are of special interest and should be further investigated are:

1. Training our framework on nanorough surfaces of different correlation lengths, α values, kurtosis, skewness, etc. This is going to be telling of how well our framework is able to adapt to different nanorough surface parameter configurations.

2. Extending our CNN architecture so that it supports nanorough surfaces of sizes other than 128×128.

3. Having extended our CNN architecture, we could also elaborate on the scalability of our framework with regards to nanorough surface size. More specifically, we could train our framework on different nanorough surface sizes and evaluate its computational complexity on training as well as on inference/generation time.

4. Extending our framework, so that it is capable not only of generating realistic nanorough surfaces but also super-resolving them. By super-resolution we refer to the task of increasing the resolution of the nanorough surface, effectively enlarging it.

5. Evaluating our framework using other generative model-specific criteria like the **Fréchet Inception Distance** [38] or **Inception Score** [39].

6. A review of our findings can be carried out, wherein multiple experts in the domain of nanotechnology would be tasked with distinguishing real from synthetic data. This would provide us with valuable insight with regard to our model's expressive capabilities.

7. Given that our framework fails to adapt to nanorough surfaces with small correlation lengths and high smoothing factors, we would like to further investigate the reason why.

ABBREVIATIONS - ACRONYMS

2D	Two Dimensional
3D	Three Dimensional
A2G	Two-Dimensional Array Graph Content Similarity
ACF	Autocorrelation Function
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
BCE	Binary Cross Entropy
CNN	Convolutional Neural Network
FFT	Fast Fourier Transform
FHS	Fourier & Histogram Space Content Similarity
GAN	Generative Adversarial Network
HPG	Hierarchical Proximity Graph
HPS	Hierarchical Proximity Graph Content Similarity
MeMoG	Merged Model Graph
ML	Machine Learning
NGG	N-Gram Graph Content Similarity
NVS	Normalized Value Similarity
PDF	Probability Density Function
ReLU	Rectified Linear Unit
RMS	Root Mean Square
RMSD	Root Mean Square Deviation
RMSProp	Root Mean Square Propagation
SL	Supervised Learning
SLP	Single-Layer Perceptron
SS	Size Similarity
TanH	Hyperbolic Tangent
UL	Unsupervised Learning
VR	Value Ratio

VS	Value Similarity

BIBLIOGRAPHY

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

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

[3] D. Stutz, "Collection of latex resources and examples.." https://github.com/davidstutz/latex-resources, 2020.

[4] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119–130, 1988.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[7] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[8] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[9] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[10] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[12] G. Giannakopoulos and V. Karkaletsis, "N-gram graphs: Representing documents and document sets in summary system evaluation," *Theory and Applications of Categories*, 2009.

[13] G. Giannakopoulos, V. Karkaletsis, G. Vouros, and P. Stamatopoulos, "Summarization system evaluation revisited: N-gram graphs," *ACM Trans. Speech Lang. Process.*, vol. 5, pp. 1–39, 10 2008.

[14] A. Cecen, H. Dai, Y. C. Yabansu, S. R. Kalidindi, and L. Song, "Material structure-property linkages using three-dimensional convolutional neural networks," *Acta Materialia*, vol. 146, pp. 76–84, 2018.

[15] Z. Yang, Y. C. Yabansu, D. Jha, W. keng Liao, A. N. Choudhary, S. R. Kalidindi, and A. Agrawal, "Establishing structure-property localization linkages for elastic deformation of three-dimensional high contrast composites using deep learning approaches," *Acta Materialia*, vol. 166, pp. 335–345, 2019.

[16] V. C. Antonios Stellas, George Giannakopoulos, "Hybridizing ai and domain knowledge in nanotechnology: The example of surface roughness effects on wetting behavior," *Natural Sciences and AI Workshop, 11th Hellenic Conference on Artificial Intelligence (SETN)*, 2020.

[17] S. Noguchi and J. Inoue, "Stochastic characterization and reconstruction of material microstructures for establishment of process-structure-property linkage using the deep generative model," *Phys. Rev. E*, vol. 104, p. 025302, Aug 2021.

[18] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[19] Z. Yang, Y. C. Yabansu, R. Al-Bahrani, W. keng Liao, A. N. Choudhary, S. R. Kalidindi, and A. Agrawal, "Deep learning approaches for mining structure-property linkages in high contrast composites from simulation datasets," *Computational Materials Science*, vol. 151, pp. 278–287, 2018.

[20] R. Cang, Y. Xu, S. Chen, Y. Liu, Y. Jiao, and M. Yi Ren, "Microstructure Representation and Reconstruction of Heterogeneous Materials Via Deep Belief Network for Computational Material Design," *Journal of Mechanical Design*, vol. 139, 05 2017. 071404.

[21] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), p. 609–616, Association for Computing Machinery, 2009.

[22] D. Fokina, E. Muravleva, G. Ovchinnikov, and I. Oseledets, "Microstructure synthesis using style-based generative adversarial networks," *Physical Review E*, vol. 101, Apr 2020.

[23] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2019.

[24] L. Mosser, O. Dubrule, and M. Blunt, "Reconstruction of three-dimensional porous media using generative adversarial neural networks," *Physical Review E*, vol. 96, 04 2017.

[25] A. Gayon Lombardo, L. Mosser, N. Brandon, and S. Cooper, "Pores for thought: generative adversarial networks for stochastic reconstruction of 3d multi-phase electrode microstructures with periodic boundaries," *npj Computational Materials*, vol. 6, p. 82, 06 2020.

[26] A. Stellas, "Μηχανική μάθηση και Νανοτεχνολογία: Σύνδεση δομικών και λειτουργικών παραμέτρων νανοδομημένων επιφανειών με νανοτραχύτητα. (Greek) [machine learning and nanotechnology: Linking structural and functional parameters of nanorough surfaces]," Master's thesis, National Technical University of Athens, 2019.

[27] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[28] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–.

[29] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[31] G. Giannakopoulos, "A python implementation of the jinsect toolkit of n-gram graphs." https://github.com/ggianna/PyINSECT, 2021.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[33] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[34] P. T. Inc., "Collaborative data science." https://plot.ly, 2015.

[35] A. M. Mood, F. A. Graybill, and D. C. Boes, *Introduction to the Theory of Statistics*. New York City: McGraw Hill, 3 ed., 11 1973.

[36] R. J. Hyndman and Y. Fan, "Sample quantiles in statistical packages," *The American Statistician*, vol. 50, no. 4, pp. 361–365, 1996.

[37] H. Iqbal, "Harisiqbal88/plotneuralnet v1.0.0," Dec. 2018.

[38] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a nash equilibrium," *CoRR*, vol. abs/1706.08500, 2017.

[39] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016.