DEPARTMENT OF DIGITAL SYSTEMS

NCSR DEMOKRITOS
INSTITUTE OF INFORMATICS AND
TELECOMMUNICATIONS

Generation of synthetic nano-material surfaces through machine learning / deep learning methods

by

Dimitrios Delikonstantis

Submitted

in partial fulfilment of the requirements for the degree of

Master of Artificial Intelligence

at the

UNIVERSITY OF PIRAEUS

July 2022

Author . . . . . .

Dimitrios Delikonstantis
II-MSc “Artificial Intelligence”
July, 2022

Certified by. . . . . .

Georgios Giannakopoulos
NCSR DEMOKRITOS Researcher
Thesis Supervisor

Certified by. . . . . .

Theodoros Giannakopoulos
NCSR DEMOKRITOS Researcher
Member of Examination Committee

Certified by. . . . . .

Georgios Petasis
NCSR DEMOKRITOS Researcher
Member of Examination Committee

Acknowledgments

I would like to express my deepest gratitude to Dr. Georgios Giannakopoulos, Dr. Vassilios Konstantoudis and Elena Stai for their huge support on this project. This endeavour would not have been possible without their guidance and vast knowledge.

Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of University of Piraeus and Inst. of Informatics and Telecom. of NCSR “Demokritos”.

Generation of synthetic nano-material surfaces through machine learning / deep learning methods

by

Dimitrios Delikonstantis

Submitted to the II-MSc "Artificial Intelligence" on
July, 2022,
in partial fulfillment of the
requirements for the MSc degree

Abstract
Nano-technology is a branch of science that engages with engineering and technology performed on a nano-scale level. Nano-material rough surfaces are characterized by structural diversity and complexity. Nano-materials exhibit unique properties, such as friction, contact deformation and wettability, which is correlated to their surface roughness. Investigating the relationship between surface roughness and properties can lead to better nano-material designs and even to the discovery of new nano-materials with unique properties.
In this research, we study the domain field of nano-roughness and nano-material surfaces as well as machine learning methods, with a focus on deep learning, in order to generate realistic synthetic nano-surface images, given specific surface roughness parameters. To this end we examine how prior domain knowledge can empower a model to provide realistic, synthetic surface images. We build upon previous work which was implemented by Vasileios Sioros in his research "Generating Realistic Nanorough Surfaces Using an N-Gram-Graph Augmented Deep Convolutional Generative Adversarial Network"[1]. We contribute to the aforementioned work by introducing a new loss component to the network, reflecting the heights and frequency spectrum of nano-surface images. Furthermore, we propose a novel network architecture in an effort to reduce the checkerboard artifact observed in generated nano-surface images. Additionally, we optimize multi-component losses so that they equally contribute to the network's learning process. We evaluate generated nano-surface images with quantitative measures. Also, a qualitative evaluation is carried out by a domain expert.
In conclusion, our results are substantially improved compared to previous work's results.

The source code is available at https://github.com/ddelikonstantis/RoughML.

Thesis Supervisor: Georgios Giannakopoulos
Title: NCSR DEMOKRITOS Researcher (Special Functional Scientist)

Contents

Acknowledgments **3**

1 Introduction **11**

2 Background and related work **13**
2.1 **Basic terms and concepts** 13
2.1.1 **Surface nano-roughness** 13
2.1.2 **Artificial Intelligence** 19
2.2 **Related work** 35

3 Proposed method **41**
3.1 **Problem definition** 41
3.2 **Our approach** 42
3.2.1 **Network overview** 42
3.2.2 **Network architecture** 42
3.2.3 **Similarity content** 48
3.2.4 **Training** 51

4 Experimental results **55**
4.1 **Experimental setup** 55
4.1.1 **Dataset** 55
4.1.2 **Experimental results on network initial weights** 58
4.1.3 **Experimental results on N-Gram Graph parameters** . . . 62
4.1.4 **Experimental results on higher surface image dimension** 69
4.2 **Results and discussion** 71
4.2.1 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 2, \xi_y = 2, \alpha = 0.5}$ 72
4.2.2 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 2, \xi_y = 2, \alpha = 1.0}$ 74
4.2.3 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 4, \xi_y = 4, \alpha = 0.5}$ 76
4.2.4 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 4, \xi_y = 4, \alpha = 1.0}$ 78
4.2.5 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 8, \xi_y = 8, \alpha = 0.5}$ 80
4.2.6 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 8, \xi_y = 8, \alpha = 1.0}$ 82
4.3 Discussion 83

5 Conclusions **85**

List of Figures

1 Root Mean Square Roughness R_q[4] 15
2 Skewness R_{sk} and the magnitude distribution curve 16
3 Kurtosis R_{ku} 17
4 An Artificial Neural Network 20
5 Sigmoid activation function 27
6 Hyperbolic Tangent (Tanh) activation function 28
7 Rectified Linear Unit (ReLU) activation function 29
8 LeakyReLU activation function 29
9 Maxout[28] activation function 30
10 Generative Adversarial Nets[29] training progression. The blue line represents the discriminative distribution, black line is the real data distribution and green line is the generative distribution. Diagram (a) shows early stages of training, (b) and (c) present intermediate and advanced training stages respectively and (d) describes an optimal solution. 31
11 DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.[7] 33
12 Deep Convolutional Generative Adversarial Network model 42
13 Generator architecture 44
14 Discriminator architecture 45
15 Generator-produced nano-rough surface images for initial weight values with $mean = 0, stdev = 0.02$. 59
16 Real dataset nano-rough surface images for initial weight values with $mean = 0, stdev = 0.02$. 60
17 Graph plots for initial weight values with $mean = 0, stdev = 0.02$. 60
18 Generator-produced nano-rough surface images for initial weight values with $mean = 0, stdev = 0.2$. 61
19 Real dataset nano-rough surface images for initial weight values with $mean = 0, stdev = 0.2$. 61
20 Graph plots for initial weight values with $mean = 0, stdev = 0.2$. 62
21 Generator-produced nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 0.5$. 63
22 Real dataset nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 0.5$. 64
23 Graph plots for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 0.5$. 64

24 Generator-produced nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 1.00$. 65
25 Real dataset nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 1.00$. 65
26 Graph plots for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 1.00$. 66
27 Generator-produced nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 0.5$. 66
28 Real dataset nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 0.5$. 67
29 Graph plots for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 0.5$. 67
30 Generator-produced nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 1.00$. 68
31 Real dataset nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 1.00$. 68
32 Graph plots for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 1.00$. 69
33 Generator-produced nano-rough surface images for dimension $d = 256$. . 70
34 Real dataset nano-rough surface images for dimension $d = 256$. 70
35 Graph plots for dimension $d = 256$. 71
36 Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 0.5$. . 72
37 Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 0.5$. 72
38 Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 0.5$. 73
39 Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 1.0$. . 74
40 Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 1.0$. 74
41 Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 1.0$. 75
42 Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 0.5$. . 76
43 Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 0.5$. 76
44 Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 0.5$. 77
45 Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 1.0$. . 78
46 Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 1.0$. 78

47 Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 1.0$. 79
48 Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 0.5$. . 80
49 Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 0.5$. 80
50 Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 0.5$. 81
51 Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 1.0$. . 82
52 Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 1.0$. 82
53 Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 1.0$. 83

1 Introduction

Artificial intelligence[2] assists in making processes more efficient across several industries. Machine Learning methods are used in combination with nanotechnology in industrial manufacturing, monitoring and computing processes. The combination of AI and nanotechnology produces successful results in emerging applications. These applications include analyzing large data sets, designing and discovering new nano-materials, and developing more efficient hardware to manage computationally demanding machine learning algorithms.

Nano-technology is a branch of science that engages with engineering and technology performed on a nano-scale level which is about 1 to 100 nano-meters. Nano-materials exhibit unique properties, such as friction, contact deformation and wettability, which is correlated to their surface roughness. Investigating the relationship between surface roughness and properties can lead to better nano-material designs and even to the discovery of new nano-materials with unique properties.

In this research, we study the domain field of nano-roughness and nano-material surfaces[3][4] as well as machine learning methods[5][6], with a focus on deep learning[7], in order to generate realistic synthetic nano-surface images, given specific surface roughness parameters. To this end we examine how prior domain knowledge can empower a model to provide realistic, synthetic surface images. We evaluate generated nano-surface images with quantitative measures. Also, a qualitative evaluation is carried out by a domain expert.

We build upon previous work which was implemented by Vasileios Sioros in his research ”Generating Realistic Nanorough Surfaces Using an N-Gram-Graph Augmented Deep Convolutional Generative Adversarial Network.”[1] Our contribution in the aforementioned work is summarized as follows.

- Introduce a new loss component to the network, reflecting the heights and frequency specturm of nano-surface images.
- Propose a novel architecture in an effort to reduce the checkerboard artifact observed in generator-produced nano-surface images.
- Optimize multi-components losses for equal contribution to the Generator’s learning ability.
- Extend the network functionality so that it produces higher surface image dimensions.

- Add early stopping criterion when training the network.
- Support incremental training of the network through checkpoints.

The rest of the text is organized as follows. Section 2 introduces the theoretical concepts that are related with our research as well as particular related work implemented by other scientists in this field. Section 3 introduces our proposed method and approach to the application. Section 4 presents our experiments and achieved results. Finally, section 5 refers to the conclusions and future work.

2 Background and related work

In this section we cover the theoretical concepts of surface roughness and the corresponding parameters we utilized for generating nano-rough surfaces. We continue further with Artificial Intelligence topics with a focus on Deep Learning. We introduce various aspects of Neural Networks while expressing their relationship to our research. Finally, we present related work in the field and state what we borrow from it or do differently.

2.1 Basic terms and concepts

We begin with the field of nano-technology and its relationship to surface roughness. Furthermore, we describe the roughness parameters we took into consideration when generating nano-rough surfaces.

2.1.1 Surface nano-roughness

Nano-technology is a branch of science that engages with engineering and technology performed on a nano-scale level which is about 1 to 100 nano-meters[8]. It involves the ability to understand and control matter on an atomic scale where unique phenomena occur. At this scale, materials begin to exhibit unique properties that affect physical, chemical, and biological behavior. By manipulating atoms and their properties scientists are able to create novel applications such as materials with higher strength, lighter weight, increased control of light spectrum, and greater chemical reactivity as well as new structures and devices with unique properties.
Nano-technology includes various fields of science such as surface science[9], organic chemistry, molecular biology, semiconductor physics, energy storage, engineering, micro-fabrication[10] and molecular engineering. The related research and applications ranges from extensions of conventional device physics to totally new approaches based upon molecular self-assembly, from developing new materials with dimensions on the nano-scale level to direct control of matter on the atomic scale.

Nano-metrology[11] quantifies the dimensions of nano-materials and devices. It has a crucial role in order to produce nano-materials and devices with a high degree of accuracy and reliability in nano-manufacturing. Nano-materials are characterized by morphology, composition, structure, physical and chemical properties. The measurement of length or size, force, mass, electrical and other properties is included in the field of nano-metrology. When a nano-material is manufactured, its structural

characterization is needed before it is used. Image analysis is widely used in the area of metrology for determining if an object is manufactured within specified tolerances.

When processing a material, roughness is produced[12]. It is identified by peaks and valleys of different size and space between them. It is a result of machine processing during manufacturing. Deterministic surface textures[13] can be researched by simple methods and functions. But for most processed surfaces, the textures are random, either isotropic or anisotropic, and either Gaussian or non-Gaussian. These are called stochastic surfaces. Isotropic surfaces exhibit the same properties in all directions whereas anisotropic surfaces exhibit different properties in different directions. Gaussian surfaces have height values that follow a normal distribution whereas non-Gaussian surfaces have height values that do not conform to a normal distribution. Stochastic surfaces can be characterized using statistical parameters taking into account all samples of the surface profile. On the other hand, deterministic surfaces have structures or features that must be characterized individually and then averaged over all features.
Surface interaction[14] is the contact between the material and the morphology of the surface. The nature of a material and its industrial production process define the shape of its surface. At a nano-scale level, most surfaces are observed to be rough. Roughness is characterized by coarseness of different size and spacing. Roughness is a component of surface texture.
Surface roughness evaluation is crucial when it comes to controlling certain properties such as friction, heat and electric current conduction. Surface geometry[4] can be characterized by a vast number of parameters as it is extremely complex. Surface parameters describe characteristics of the surface profile. Surface profile is the measurement of the maximum peak-to-valley depth of the surface. Surface parameters are categorized in the following groups:

- Amplitude parameters
  - Root-Mean-Square Roughness (R_q)[4]

$$R_q = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2} \tag{2.1}$$

    where;
    n is the number of samples along the assessment length (μm) of the surface profile,
    y_i is the height of the profile for sample i.

R_q represents the standard deviation of the distribution of surface heights. It is a crucial parameter to describe the surface roughness by statistical methods. The Root-Mean-Square mean line is the line that divides the profile line so that the sum of squares of the fluctuations of the profile height is equal to zero.

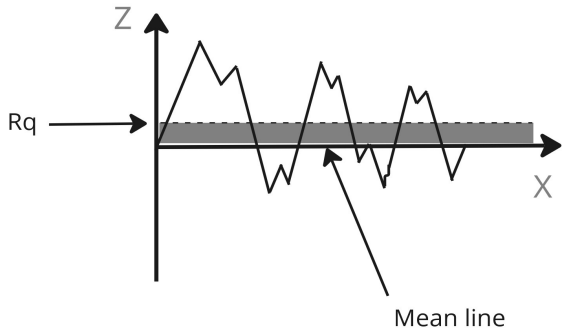

Figure 1: Root Mean Square Roughness R_q[4]

– Skewness (R_{sk})[4]

$$R_{sk} = \frac{1}{NR_q^3}(\sum_{i=1}^{N} Y_i^3) \tag{2.2}$$

where;
N is the number of samples along the assessment length (μm) of the surface profile,
R_q is the Root-Mean-Square Roughness parameter,
Y_i is the height of the profile for sample i.

Skewness is the profile magnitude probability density function over the measured length of the surface. It is the third central moment and it calculates the symmetry of the profile according to the mean line. Skewness is easily affected by high peaks or deep valleys. An equal number of peaks and valleys results in a symmetrical height distribution and therefore has zero skewness. In figure 2, it is clear that profiles with no peaks have negative skewness whereas profiles with high peaks have positive skewness. This parameter provides extra information as it can differentiate between two profiles that have the same R_q but with dissimilar forms.

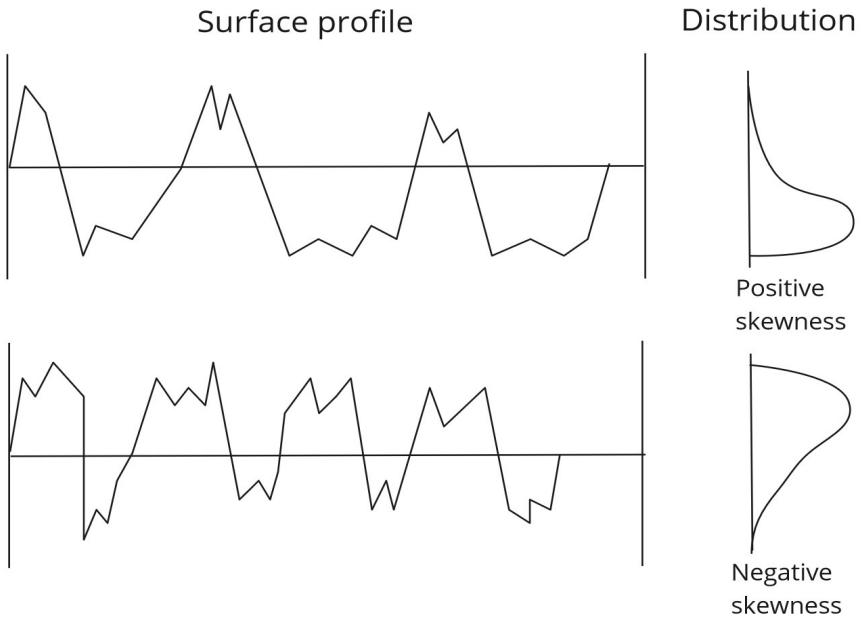

Figure 2: Skewness R_{sk} and the magnitude distribution curve

– Kurtosis (R_{ku})[4]

$$R_{ku} = \frac{1}{NR_q^4}(\sum_{i=1}^{N} Y_i^4) \tag{2.3}$$

where;
N is the number of samples along the assessment length (μm) of the surface profile,
R_q is the Root-Mean-Square Roughness parameter,
Y_i is the height of the profile for sample i.

Kurtosis coefficient is the fourth central moment of profile magnitude probability density function over the assessment length. It represents the harshness of the probability density of the profile. Platykurtoic distributions have negative kurtosis whereas leptokurtoic distributions have positive kurtosis. In figure 3, we can clearly see that the platykurtoic distribution curve has respectively few high peaks and low valleys when $R_{ku} < 3$. On the other hand, the leptokurtoic distribution curve has relatively many high peaks and low valleys when $R_{ku} > 3$.

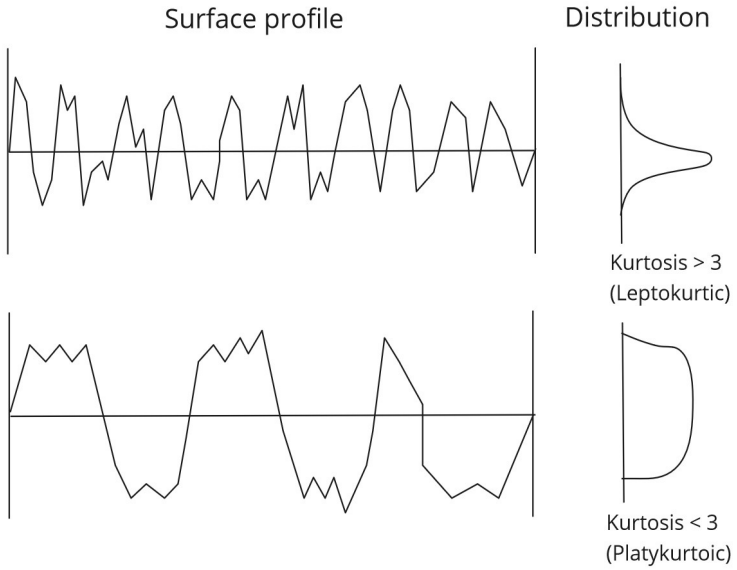

Figure 3: Kurtosis R_{ku}

– Auto-Correlation Function (ACF) and Correlation lengths (ξ_x, ξ_y)[4]

$$ACF(\delta_x) = \frac{1}{N-1}\sum_{i=1}^{N} y_i y_{i+1} \tag{2.4}$$

where;
N is the number of samples along the assessment length (μm) of the surface profile,
y_i is the height of the profile at point number i.

The Auto-Correlation Function (ACF) provides information about the distribution of peaks and valleys across the surface. It estimates a degree of similarity for surface heights. In practice, it is a quantitative measure between a laterally shifted and an unshifted version of the profile. It gives crucial information about the association between the wavelength and the amplitude properties of the surface.
Correlation lengths (ξ_x, ξ_y) describe the correlation characteristics of the ACF and is the length (in X and Y axis) of a nano-rough surface where the ACF drops to a specific value 10% ($ACF = 0.1$) of the unshifted version of the profile. Infinite correlation length values result in a perfectly periodic

wavelength whereas zero correlation lengths result in a completely random waveform. Points on the surface profile that are separated by more than a correlation length are considered as uncorrelated.

- Hybrid parameters
  - Roughness exponent (α)

$$E\left[\frac{R(n)}{S(n)}\right] = Cn^H \quad \text{as n} \longrightarrow \infty \tag{2.5}$$

where;
$R(n)$ is the range of the first n cumulative surface sample deviations from the mean,
$S(n)$ is the surface profile (sum) of the first n standard deviations,
$E[x]$ is the expected surface sample value,
n is the number of samples along the assessment length (μm) of the surface profile,
C is a constant.

The Roughness exponent (α)[15] measures the long-term memory of the surface profile. It relates to the self-similarity and local smoothness of the surface profile. It takes values between 0 and 1. Based on the value of α we can classify the surface profile into one of the three categories:

* $\alpha < 0.5$
  A mean-reverting surface profile with negative auto-correlation. Values close to 0 indicate a locally spiked surface profile.
* $\alpha = 0.5$
  Indicates a completely uncorrelated surface profile.
* $\alpha > 0.5$
  A trending surface profile with positive auto-correlation. Values close to 1 indicate a locally smoothed surface profile.

The aforementioned section outlined core concepts of nanotechnology and nano-metrology followed by a detailed reference of the surface nano-roughness parameters that are utilized in our research. In the following section the field of Artificial intelligence is analyzed and how it is used in our work.

2.1.2 Artificial Intelligence

Artificial Intelligence[16] is intelligence manifested by machines. On the other hand, natural intelligence is displayed by animals and humans. Artificial Intelligence research has been described as the field of study of intelligent agents, which denotes a system that becomes aware of its environment and takes actions that maximize its chance of attaining its objectives.
AI applications comprise of advanced web search engines, recommendation systems, comprehending human speech, self-driving cars, automated decision-making and competing at the highest level in strategic games.
Sub-fields of AI research include reasoning, knowledge representation, planning, machine learning[6][1][5], natural language processing[17], perception and combinatorial optimization.
For the purpose of our research, we use machine/deep learning which are sub-fields of Artificial Intelligence.

Machine Learning[18] involves the use of algorithms that learn and methods that leverage data to improve performance on some set of tasks. Supervised Machine Learning algorithms build a model based on a labeled set of data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so[19]. Unsupervised Machine Learning algorithms are given data with no target outputs and try to find a relationship or a pattern. Machine Learning is used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is impractical or unfeasible to develop conventional algorithms to perform those tasks.
A main objective of a learner algorithm is to generalize from the training data. Generalization is the ability of a learning machine to perform accurately on new, unseen examples of the given learning dataset, called training dataset. The training dataset comes from some generally unknown probability distribution and the learner algorithm has to build a general model about this data space that enables it to produce sufficiently accurate predictions in new cases.
Machine learning is divided into three broad categories.

- Supervised learning
  The model is trained with example inputs and their desired outputs and the goal is to learn a general rule or structure that maps inputs to outputs.

- Unsupervised learning
  No target outputs are provided to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be used for discovering hidden patterns in data.

- Reinforcement learning
  A model interacts with a dynamic environment in which it must perform a specific task. As it explores its problem space, the model is given feedback that is analogous to rewards, which it tries to maximize.[20]

Deep learning[21] is a branch of machine learning and typically describes methods based on artificial neural networks. Generally, artificial neural networks try to mimic the behavioural process of the human brain by utilizing a vast quantity of data. The conception of neural networks started as a model of how neurons work in the brain. In practice, it is an attempt in model learning that results from a combination of data inputs, weights and bias. These elements function together to accurately recognize, classify and describe objects within the data inputs.

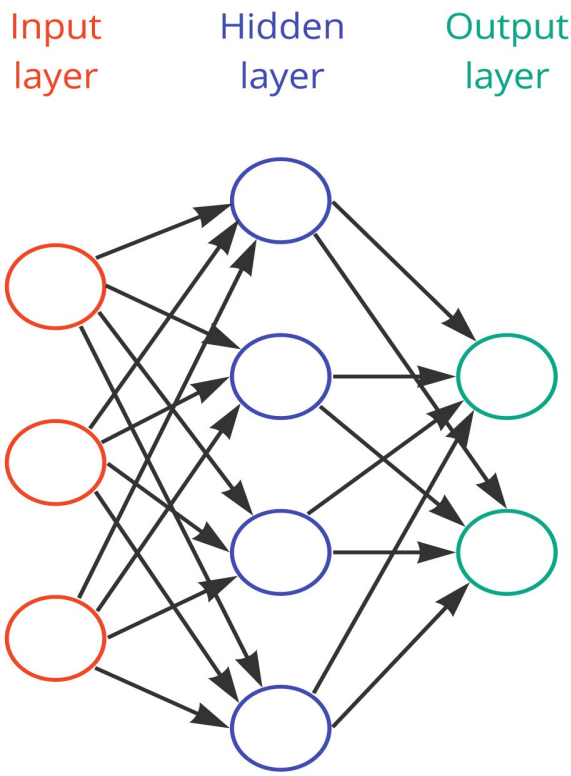

Figure 4: An Artificial Neural Network

Deep Neural Networks[22] are composed from multiple layers of interconnected nodes, building upon the output of previous layers to process and make a prediction or classification. This process of computations within the network is called forward

propagation. The input and output layers of a deep neural network are called visible layers. The input layer is where the deep learning model takes the data for processing, and the output layer is where the final prediction or classification takes place. There is another operation called back-propagation that uses algorithms to compute mistakes in predictions and then reconfigures the weights and biases of the model by going backwards through the layers in an attempt to train the model.
Together, forward propagation and back-propagation allow a neural network to make predictions and minimize its errors. Over time, the algorithm converges and steadily becomes more accurate. In practice, deep learning algorithms are extremely complicated, and there are many different types of neural networks to address specific problems. The most common neural network types[22] are:

- Convolutional Neural Networks, used primarily in computer vision and image classification applications.
- Recurrent Neural Networks are typically used in natural language processing and speech recognition applications.
- Long Short-Term Memory networks, are mainly used in sentiment analysis, language modeling, speech recognition, and video analysis.
- Generative Adversarial Networks, is a deep learning framework model that finds use in image-to-image translation, text-to-image synthesis, realistic image generation, super resolution, video prediction and $3D$ object generation.

For our work we focus on a framework that combines Convolutional Neural Networks and Generative Adversarial Networks. More specifically, we use a model that is called Deep Convolutional Generative Adversarial Networks.

In Deep Learning, a Convolutional Neural Network represents a class of Artificial Neural Networks and is widely used in image and video recognition, image segmentation, image classification, medical image analysis and natural language processing. CNNs are a type of artificial neural networks that use a mathematical function called convolution in place of general matrix multiplication in at least one of their layers. They are specifically configured to handle pixel data. A CNN comprises of an input layer, hidden layers and an output layer. In feedforward Neural Networks, middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a CNN, the hidden layers incorporate layers that perform convolutions. Usually, this includes a layer that does a multiplication of the convolution filter with the layer's input matrix. As the convolution filter slides along the input matrix for the layer, the convolution generates

a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers. In a CNN, the input is a tensor with shape: (Number of inputs) $\times$ (Input height) $\times$ (Input width) $\times$ (input channels). Channel represents the total number of colors within the image. For Red-Green-Blue images channel takes the value of 3, for gray-scale images channel takes the value of 1. After going through a convolutional layer, the image becomes abstracted to a feature map with shape: (Number of inputs) $\times$ (Feature map height) $\times$ (Feature map width) $\times$ (Feature map channels). Convolutional layers convolve the input and move its result to the next layer. Each convolutional neuron processes data only for its specific field. Convolutional Neural Networks are ideal for image and video data because spatial relations between separate features are taken into account during convolution. CNNs may include pooling layers in combination with convolutional layers. Pooling layers reduce the dimensions of the feature maps by merging the output of neurons at one layer into a single neuron in the next layer. Thus, it reduces the number of parameters to learn and the amount of computation performed within the network. There are two popular types of pooling: max pooling and average pooling. Max pooling uses the maximum value in the feature map while average pooling takes the average value. Convolutional Neural Networks make use of hyperparameters. Those are specific parameter settings that are used to control the learning process.

- Kernel size
  The kernel is the number of pixels processed together. It is represented as the kernel's dimensions $Height \times Width$.

- Padding
  Padding is the addition of a number of pixels on the border of an image. This is done so that the border pixels are not diminished in the output because they would get involved only in one instance during the convolution. The padding value is usually one less than the kernel dimension. For a 2×2 kernel the padding would be 1 extra pixel on the border of the image.

- Stride
  The stride is the number of pixels that the kernel moves on each convolution. For a 2×2 stride the kernel moves 2 pixels in height and width for each corresponding convolution.

- Pooling
  Pooling is down-sampling the convolution output and therefore reducing computational cost at the cost of information loss.

Back-propagation[23] is an algorithm for supervised learning of artificial neural networks using gradient descent. Provided an artificial neural network and an error function, the algorithm calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptrons to multi-layer feed-forward neural networks. The delta rule is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network.[24]
The "backwards" part of the name is derived from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately.
Back-propagation is analogous to calculating the delta rule for a multi-layer feed-forward network. Thus, like the delta rule, back-propagation requires three things.

1. A dataset consisting of input-output pairs $(\overrightarrow{x_i}, \overrightarrow{y_i})$, where $\overrightarrow{x_i}$ is the input and $\overrightarrow{y_i}$ is the desired output of the network for input $\overrightarrow{x_i}$. The set of input-output pairs of size N is denoted $X = \{(\overrightarrow{x_i}, \overrightarrow{y_i}), ..., (\overrightarrow{x_N}, \overrightarrow{y_N})\}$.

2. A feed-forward neural network, whose parameters are collectively denoted θ. In back-propagation, the parameters of primary interest are w_{ij}^k, the weight between node j in layer l_k and node i in layer l_{k-1}, and b_i^k, the bias for node i in layer l_k. There are no connections between nodes in the same layer and layers are fully connected.

3. An error function, $E(X, \theta)$, which defines the error between the desired output $\overrightarrow{y_i}$ and the calculated output $\overrightarrow{\hat{y_i}}$ of the neural network on input $\overrightarrow{x_i}$ for a set of input-output pairs $(\overrightarrow{x_i}, \overrightarrow{y_i}) \in X$ and a particular value of the parameters θ.

Training a neural network with gradient descent requires the calculation of the gradient of the error function $E(X, \theta)$ with respect to the weights w_{ij}^k and biases b_i^k. Then, according to the learning rate α, each iteration of gradient descent updates the weights and biases (collectively denoted θ) according to:

$$\theta^{t+1} = \theta^t - a\frac{\partial E(X, \theta^t)}{\partial \theta} \tag{2.6}$$

where;
θ^t denotes the parameters of the neural network at iteration t in gradient descent.

The error function in original back-propagation is the mean squared error.

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{2.7}$$

where,
y_i is the target value for input-output pair $(\overrightarrow{x_i}, y_i)$ (Artificial Neural Network with one output y_i),
$\hat{y}_i$ is the computed output of the network on input $\overrightarrow{x_i}$.
Back-propagation attempts to minimize the error function $E(X, \theta)$ with respect to the neural network's weights θ.

Neural Networks make use of optimization algorithms. An optimization algorithm detects the value of the parameters that minimize the error when mapping inputs to outputs. These optimization algorithms have an effect on the accuracy of the model. Furthermore, they affect how fast the model trains.
Practically, for every epoch the model trains, the values of the parameters, or the weights as they are usually called, need to be modified in order for the loss function to minimize. An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the model. The optimization algorithm changes the weights and the learning rate of the neural network. Learning rate is a parameter that gives the model a scale of how much model weights should be updated. A high learning rate means more drastic changes in the weights, whereas a low learning rate means smaller weight changes. An optimizer decreases the overall loss and increases the accuracy of the model. This is a complex task since a deep learning model usually consists of millions of parameters. Therefore, choosing a suitable optimization algorithm for an application might be difficult.
The most well known optimizers are:

- Gradient Descent[25]
  The simplest machine learning optimizer is the gradient descent. Gradient descent optimizes the objective function by descending in a direction given by the negative gradients of the parameters in the final iteration. Respectively, it utilizes a constant learning rate value which controls how large or small a stride to make towards the negative gradient. Gradient descent uses all samples in each iteration to change a particular parameter. It is described by the following formula.

  $$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta) \tag{2.8}$$

  where;
  θ represents the parameters,
  $J(\theta)$ is a cost function,

η is the learning rate

- Stochastic Gradient Descent[25]
  Another way of optimizing is to use one sample at a time in each iteration to change a parameter. This method is called Stochastic Gradient Descent (SGD). By iterating every sample in every epoch though can be slow and also create noisy jumps which may fall far away from the minimum cost. An extension of the SGD is the Stochastic Gradient Descent with momentum. It performs better than the SGD in regard to the noisy jumps. Stochastic Gradient descent with momentum accelerates the descent where gradients steadily point to one direction and decelerates when gradients show change.

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.9}$$

  where;
  θ represents the parameters,
  $J(\theta)$ is a cost function,
  η is the learning rate,
  $x^{(i)}$, $y^{(i)}$ is a training sample and its label correspondingly

- Mini-Batch Gradient Descent[25]
  A more efficient way of optimizing is to use batches of samples from the dataset that gives a local estimate of the direction that gives the optimal parameter values. This is called Mini-Batch Stochastic Gradient Descent. Gradient descent and Stochastic Gradient descent require a default learning rate value. Choosing a right learning rate can prove difficult since a high value may result to sub-optimal generalization and a low value may result in slow learning.
  There are many alterations of the gradient descent. A noteworthy one is the Newton's method which takes second order derivatives of the cost function. It is useful for quadratic functions but prohibitive to compute in practice.

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta; x^{(i;i+n)}; y^{(i;i+n)}) \tag{2.10}$$

  where;
  θ represents the parameters,
  $J(\theta)$ is a cost function,
  η is the learning rate,
  $x^{(i)}$, $y^{(i)}$ is a training sample and its label correspondingly
  n is a mini-batch of n training samples

- Adagrad[25]
  Adaptive Gradient algorithm also known as AdaGrad shows good results on large scale learning systems. It uses mostly first order information but also relies

on some properties of second order information. The formula is described as follows.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{2.11}$$

where;
θ_t represents the update for every parameter at a time step t,
η is the learning rate,
G_t is a diagonal matrix which contains the sum of the squares of the past gradients with respect to all parameters θ.
Whereas there is a manual adjust of the global learning rate, each feature has a dynamic learning rate. Large gradients have smaller learning rates while small gradients have large learning rates. This reduces the noise created from hugely different samples when calculating the cost derivative. AdaGrad can be sensitive to the gradient initial conditions. If the initial gradients are large then the learning rate will be low. Furthermore, due to the accumulation of the squared gradients in the denominator, the learning rate will continue to decrease eventually reaching to zero and stopping the training completely.

- AdaDelta
  Initially described by Matthiew Zeiler[26], AdaDelta is an adaptive learning rate method for gradient descent. It is a stochastic technique that actively changes over time and is a lot faster than Stochastic Gradient Descent (SGD). There is no need to neither manually adjust nor set an initial value to the learning rate in this method. There is indication that it is robust to noisy gradient of sample input and works well with a large variety of model architectures and applications. AdaDelta is an extension of the Adagrad optimizer that attempts to solve its radically diminishing learning rates. In order to do this, AdaDelta computes the average of the past n (window size) gradients and averages out. It does this by using exponentially weighted averages.

- Adam
  Adaptive Moment Estimation (Adam) is an extension to Stochastic Gradient Descent and is widely used in computer vision and natural language processing. It was presented by Diederik Kingma et al[27]. Adam combines the advantages of two other optimizers: Adagrad and RMSprop.
  Instead of adapting the parameter learning rates based on the average first moment as in RMSprop, Adam also makes use of the average of the second moments of the gradients which is the variance. In more detail, it calculates an exponential average of the gradient and the squared gradient, and the β_1 and β_2 parameters control the decay rates of these averages.
  Adam consists of the parameters α which is the learning rate, β_1 which is the

exponential decay for the first moment parameters and β_2 which is the exponential decay for the second moment parameters.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon}\hat{m}_t \tag{2.12}$$

where;
θ_t represents the update for every parameter at a time step t,
η is the learning rate,
m_t and υ_t are estimates of the first moment (mean) and the second moment (variance) of the gradients respectively.

In artificial neural networks, the activation function of a node defines the output of that node given an input. More specifically, it decides whether a neuron should be activated or not. The role of the activation function is to derive output from a set of input values provided to a node or a layer. The purpose of an activation function is to add non-linearity to the neural network. The activation functions we utilized for our work are the following.

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

This activation function takes any real value as input and outputs in the range of 0 to 1. The larger the input, the closer the output value will be to 1, whereas the smaller the input, the closer the output will be to 0. It is commonly used for models where the output needs to be predicted as a probability.

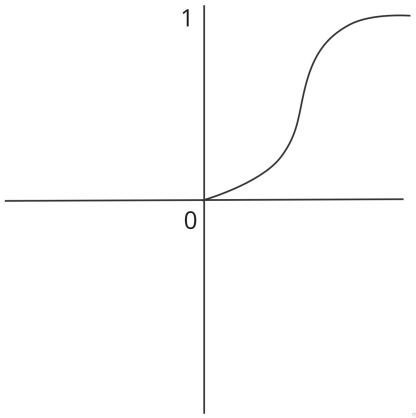

Figure 5: Sigmoid activation function

- Hyperbolic Tangent (Tanh)

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.14}$$

Tanh activation function is very similar to Sigmoid with the difference that the output range is from -1 to 1. The larger the input, the closer the output value will be to 1, whereas the smaller the input, the closer the output will be to -1. It helps in centering the data and makes learning for the next layer much easier.

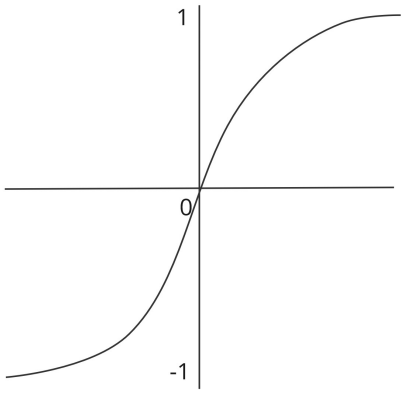

Figure 6: Hyperbolic Tangent (Tanh) activation function

- Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x) \tag{2.15}$$

ReLU activation function takes values from 0 to 1. The neurons will only be deactivated if the output of the linear transformation is less than 0 otherwise they are activated. It is computationally more efficient than Sigmoid and Tanh. Also, it accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear property.

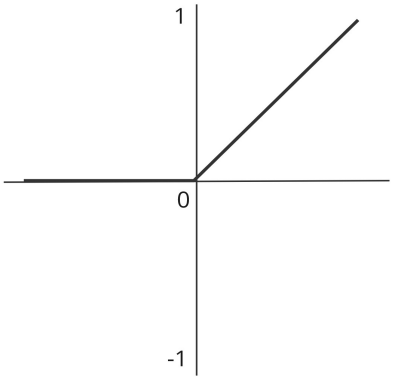

Figure 7: Rectified Linear Unit (ReLU) activation function

- LeakyReLU

$$f(x) = \max(0.1x, x) \tag{2.16}$$

Leaky ReLU is an improved version of ReLU function. It has a small positive slope in the negative area. The advantage of Leaky ReLU is that it enables backpropagation, even for negative input values.

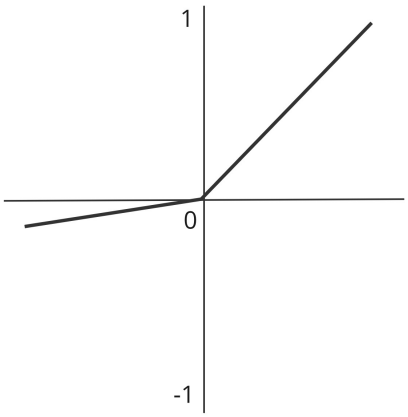

Figure 8: LeakyReLU activation function

- Maxout

$$f(x) = \max(w_1^T x + b_1, w_2^T x + b_2) \tag{2.17}$$

The Maxout unit is a generalization of the ReLU and the Leaky ReLU activation functions. Both ReLU and Leaky ReLU are special cases of Maxout. The main drawback of Maxout is that it is computationally expensive as it doubles the number of parameters for each neuron.

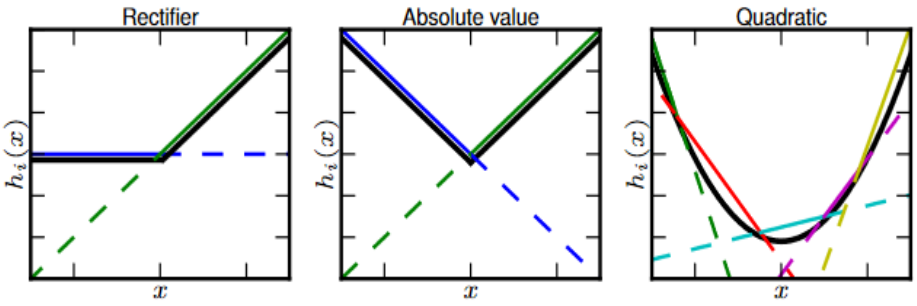

Figure 9: Maxout[28] activation function

A relatively recent breakthrough in generative modelling is the introduction of Generative Adversarial Networks by Ian Goodfellow et al.[29] Generative Adversarial Networks present a new technique for approximating a generative model through an adversarial procedure. This procedure involves two models, a Generative model G and a Discriminative model D. Those models are trained concurrently for a predefined number of epochs and the task of each is different. The generative model G tries to map the training data distribution and the discriminative model D estimates the probability that the sample derived from the training data or the generative model G. The training task for G is to maximize the probability of D making a mistake on its prediction.
This technique represents a mini-max two-player game. A useful outcome of this game is the state where model G apprehends the training data distribution and model D cannot determine where the sample came from. That means that model D predicts randomly on the sample origination. The authors specify, that in the case of deep learning the training procedure occurs with back-propagation without the need of Markov chains.
G and D play the following two-player mini-max game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{2.18}$$

In practice though, equation 2.18 does not provide enough gradient for G to learn well. During early training, when G is inadequate, D rejects data quite easily because it is very obvious that they are different from the training data. In this case,

$log(1 - D(G(z)))$ diminishes. To counter this issue, the authors change the objective function. Rather than training G to minimize $log(1 - D(G(z)))$ they train G to maximize $logD(G(z))$. This objective function provides the same fixed point of the dynamics of G and D but also much more effective gradients for G early in training.

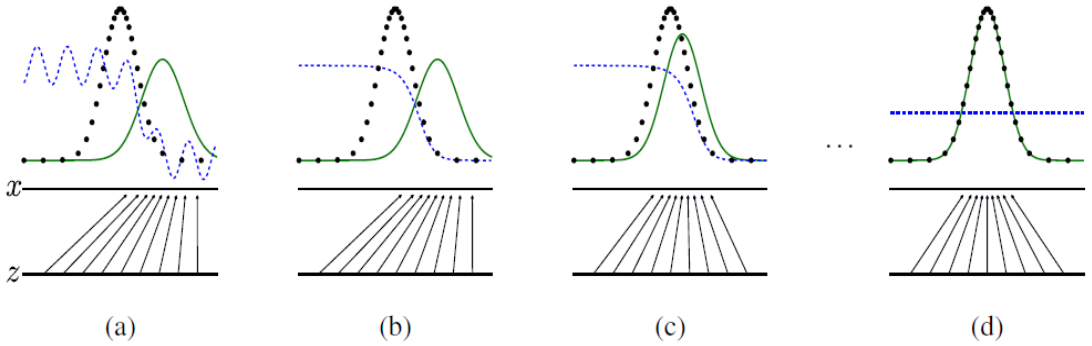

Figure 10: Generative Adversarial Nets[29] training progression. The blue line represents the discriminative distribution, black line is the real data distribution and green line is the generative distribution. Diagram (a) shows early stages of training, (b) and (c) present intermediate and advanced training stages respectively and (d) describes an optimal solution.

Figure 10 shows the progression of the GAN framework training. GANs are synchronously trained for an iterative number of epochs. During training the discriminative distribution (blue line) distinguishes between samples from real data distribution (black line) from those of the generative distribution (green line). The line z is the domain from which z is sampled. The line x is the domain of the real data. Progressively, as training goes on the system converges and as a result the generative distribution has eventually mapped the real distribution and the discriminative model is unable to differentiate between the two.
The authors of the paper[29] propose the following algorithm 1 for training a generative adversarial network. They make use of mini-batch stochastic gradient descent. A mini-batch is a fixed number of training samples that is less than the actual dataset. So, in each iteration, the network is trained on a different group of samples until all samples of the dataset are used in an epoch. In algorithm 1 the number of training iterations is the number of epochs and k steps is the number of mini-batch iterations.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

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

$$\bigtriangledown_{\theta_d} \frac{1}{m} \sum_{\iota=1}^{m} [\, logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$$

6: **end for**
7: Sample minibatch of m noise samples $\{z^{(1)}, ...z^{(m)}\}$ from noise prior $p_g(z)$.
8: Update the generator by descending its stochastic gradient:

$$\bigtriangledown_{\theta_g} \frac{1}{m} \sum_{\iota=1}^{m} log(1 - D(G(z^{(i)})))$$

9: **end for**

The gradient-based updates can use any standard gradient-based learning rule. The authors of the paper[29] used momentum for their experiments. Global optimality is achieved when $p_g = p_{data}$. The authors trained the framework on a range of datasets including MNIST, the Toronto Face Database (TFD), and CIFAR-10. The generator uses both Rectified Linear Unit and Sigmoid activation functions, while the discriminator uses maxout activation functions.
Finally, the authors note that this framework comes both with advantages and disadvantages. The main disadvantages are that there is no explicit representation of $p_g(x)$ and that both models need to be trained synchronously to achieve good results. The advantages are that no Markov chains or inference learning is needed.

In the aforementioned paragraphs we presented a deep learning method which is based on an adversarial process between two neural networks. For the means of our work, we implement algorithm 1 for the purpose of training our model.

Deep Convolutional Generative Adversarial Networks are a direct extension of Generative Adversarial Networks. First introduced by Alec Radford et al.[7], DCGANs are a powerful competitor for unsupervised learning. The authors of the paper

propose an architecture that uses explicitly convolutional and convolutional-transposed layers in the discriminator and generator, respectively. They also provide a set of architectural guidelines that provide stability during training. Those guidelines are summarized below.

- Pooling layers should be replaced by strided convolutions for the discriminator and fractionally-strided transposed-convolutions for the generator.
- Batch-normalization in both the generator and the discriminator except generator output layer and discriminator input layer.
- Fully connected hidden layers should be removed for deeper architectures.
- Rectified Linear Unit7 activation functions for all layers of the generator except the output layer which uses Hyperbolic Tangent6.
- LeakyReLU8 activation function for the discriminator for all layers.

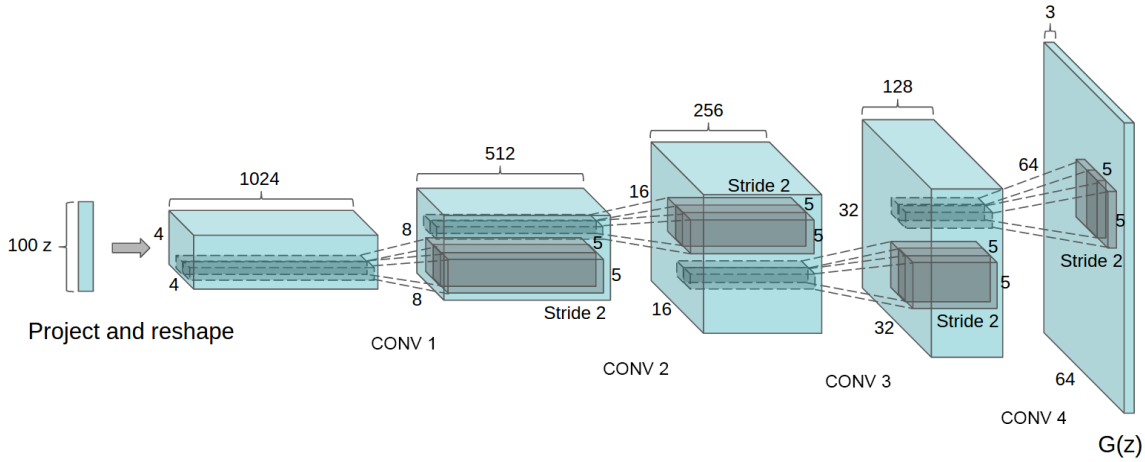

Figure 11: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.[7]

The authors of the paper trained DCGANs on three datasets; Large-scale Scene Understanding (LSUN), Imagenet-1k and a Faces dataset. The training was implemented with mini-batch Stochastic Gradient Descent with a batch size of 128. All weights were initialized from a zero-centered normal distribution with standard deviation of 0.02. Adaptive Moment Estimation was used with suggested

hyper-parameter values. Learning rate was suggested to $\alpha = 0.0002$ and momentum $\beta_1 = 0.5$. LeakyReLU negative slope is suggested to 0.2.
The results are claimed to be satisfactory and suggest that the DCGAN framework and specifically the generative model estimates the data distribution remarkably. This suggestion is amplified by the researched architectural guidelines and the training stability.

For the means of our research we implement a Deep Convolutional Generative Adversarial Network in order to generate realistic synthetic nano-rough surface images. Our DCGAN model implements algorithm 1 and borrows architectural guidelines suggested by Alec Radford et al.[7]
More specifically, we use strided convolutions for the discriminator and fractionally-strided transposed-convolutions for the generator as suggested in the paper. Furthermore, we use Adam with hyper-parameter values $\alpha = 0.0002$ and momentum $\beta_1 = 0.5$ as it helps in stability during training and is suggested by the authors. Additionally, we use batch-normalization in all layers in both models except the output layer of the generator and the input layer of the discriminator as suggested. On the other hand, we differentiate our model architecture compared to the authors by not using Tanh6 in the generator output layer. Instead, we do not use an activation function in the output layer.
A more analytical configuration of our proposed DCGAN model is discussed in section 3.

Our DCGAN model incorporates a set of similarity metrics during training. In the following paragraphs we introduce the concept of these similarity metrics and the reason we incorporated them in our model.

In the field of Natural Language Processing, n-grams are sequences of n items derived from a text or a corpus. The items can be letters, syllables or words according to the application. An n-gram of size 1 is referred to as a unigram, size 2 is a bigram, size 3 is a trigram.
G. Giannakopoulos[17] proposed the n-gram graph model. It is is a graph-based, statistical method of representing a text or a corpus. The n-gram graph is a graph

$$G = \{V^G, E^G, L, W\} \tag{2.19}$$

where;
V^G is the set of vertices,
E^G is the set of edges,
L is a function assigning a label to each vertex and to each edge,
W is a function assigning a weight to every edge.

The graph has n-grams as its vertices $v^G \in V^G$ and the edges $e^G \in E^G$ connecting the n-grams indicate the proximity of the corresponding vertex n-grams.
We make use of N-Gram Graphs to assist our algorithm to assimilate the local co-occurences of surface peaks from the real surface images to the generator-produced ones.

The Fast Fourier Transformation decomposes an image into sines and cosines of varying amplitudes and phases, which reveals repeating patterns within the image. In our work, we use Fast Fourier Transformation to assist our algorithm to assimilate the frequency spectrum similarity of the real surface images into the generator-produced ones.

2.2 Related work

This section refers to the related work that has been done by other scientists in this domain. More specifically, the following paragraphs introduce concepts such as the effects of surface roughness on wetting behaviour, Single Image Super Resolution with GANs, nanorough surface generation and nanostructure design with GANs.
The research of Antonis Stellas et al.[6] assesses if there is a link between surface morphology and wettability on a nanoscale level with the use of artificial intelligence. The nanorough surfaces are created with an algorithm that uses the inverse Fourier transform which produces surfaces with Gaussian or non-Gaussian height distributions, morphologically based on the roughness parameters RMS (R_q), skewness (R_{sk}), kurtosis (R_{ku}) and correlation lengths (ξ_x, ξ_y). The Wenzel model is assumed in order to predict the true (active) area of the surface. According to the Wenzel model, the roughness ratio r parameter is defined as the active surface area divided by the projected surface area. Different types of areas are produced depending on the aforementioned surface roughness parameters and are linked to the roughness ratio r in a database. This database is split into train and test datasets. Then deep neural networks are trained on the train dataset and predict on the test dataset. The model is evaluated and inspected on the effect the surface roughness parameters have on surface wettability.
The morphology and geometry of the nano-surface is crucial and describes its interaction with matter and elements. A few properties that are related with the manufacturing of the nano-surface are RMS (R_q), correlation length (ξ) and height parameters. Assume there is a function that would link roughness ration r and contact angle (Wenzel model) with roughness parameters (RMS, correlation length and height parameters) would help the production parameter selection process. The authors of the paper use AI techniques to connect the geometry and morphology of

nano-surfaces with their characteristics and usability.
In order to produce nanorough surfaces the authors distinguish the surfaces between two types: Gaussian and non-Gaussian. Gaussian surfaces are generated with the RMS of the height distribution and the correlation lengths (ξ_x, ξ_y). Non-Gaussian surfaces are generated as follows:

1. A random two-dimensional non-Gaussian noise via the Johnson transformation system giving as input parameters: mean, RMS (R_q), skewness (R_{sk}), kurtosis (R_{ku}).
2. If the distribution parameters don't converge, then the Pearson transformation system is used.
3. After that, the skewness and kurtosis values are determined so that they satisfy accurately the setup. If not the surface generation is repeated.
4. At last, the surface begins to correlate by reforming the height sequence in the X and Y directions imitating a known Gaussian surface with correlation lengths ξ_x and ξ_y.

Three different machine learning models are used to predict any correlation between the input and output properties:

1. Linear regression
2. Random forests
3. Neural networks

The training data consists of 3000 nano-surfaces. The evaluation metric RMSRE (Root Mean Square Relative Error) is used to assess how well the models behave. Random forests and neural networks performed better than linear regression with $RMSRE = 4\%$ and $RMSRE = 2\%$ respectively.
The neural network model indicates that RMS (R_q) value of the height distribution has the highest correlation with wettability. Correlation lengths ξ_x and ξ_y present lower importance in wettability while skewness and kurtosis show the least importance.
In our research we incorporate the algorithm presented by Antonis Stellas et al.[5] for creating our training dataset of nanorough surfaces.

In their research, Christian Ledig et al.[30] study the domain of Single Image Super Resolution (SISR). This domain aims to acquire a High Resolution (HR) image from its Low Resolution (LR) equivalent one. It has been an active area of interest with a lot of great progress and still is. But there is one problem that persists despite the progress that has been achieved. That being the realistic visualization of the details of

the image in high up-scaling resolutions. Recent studies have focused on minimizing the MSE (Mean Squared Error) which results to high frequency details missing in the output.
The authors present the SRGAN model which is a generative adversarial network (GAN) for image super resolution (SR). It is the first framework that is able to reshape realistic images for 4x up-scaling factors. The loss function of this framework contains an adversarial cost and a content loss. The adversarial loss consists of a discriminator which tries to distinguish original images from up-scaled ones. On the other hand, the content loss focuses on visual perception than pixel space similarity. A deep residual network reconstructs the details of down-sampled images. A mean-opinion-score (MOS) indicates that up-scaling with SRGAN is very effective.
Single Image Super Resolution is a critical task and usually the details in reshaped images is missing. Usually, the MMSE (Minimum Mean Squared Error) is used as a loss function between the original and the rebuilt image. The issue with MMSE is that it fails to re-scale the visual details of the image in the notion that a human would perceive them as artificial. This is because MSE minimizes the euclidean distance on the training data pixel-wise. The authors propose a SRGAN framework where it uses a ResNet with skip-connection and also uses more optimization techniques than MMSE. Furthermore, it combines a high-level feature mapping of the VGG architecture with a discriminator to solve the perceptually visual content that is missing from the HR methods in previous studies.
Linear predictions were the first to tackle the SISR problem. While those predictions were fast they would fail to capture the visual details in the output image. More advanced methods would focus on low and high image resolution pair mapping to establish a relationship in the training data. More recently, convolutional neural networks have presented astonishing results. Lately, GANs lay out much more visually perceptive images to the extent that is comparable to the human eye perception.
The authors propose a deep ResNet architecture by utilizing the idea of GANs to create a perceptual loss function for photo-realistic SISR:

1. Set up a ResNet for image SR (Super Resolution) with up-scaling factor 4x optimized for MSE.
2. Propose SRGAN with a new perceptual loss, which is based on feature maps of the VGG network.
3. Evaluate with a Mean Opinion Score (MOS) test on images from three public datasets.

During training, the low resolution images are obtained by applying a gaussian filter on the high resolution images followed by a downsampling procedure with factor r. After that, the generator is trained to map a high resolution image from its low

resolution equivalent one. Additionally, the discriminator tries to classify between the high resolution images from the real images. It is noteworthy that the authors follow the architectural guidelines summarized by Radford et al[7].
In his research, Vassilis Sioros[1] introduces two models for generating synthetic nanorough surfaces that inherit the morphological, geometrical and roughness attributes of the real ones. The real nanorough surfaces are produced by the algorithm presented by Antonios Stellas[5] and incrorporate the Root Mean Square Roughness (R_q), Skewness (R_{sk}), Kurtosis (R_{ku}), Correlation length (ξ_x, ξ_y) and Hurst exponent (α) parameters.
A Single-Layer Perceptron Generative Adversarial Network (SLPGAN) model that consists of two Single-Layer perceptron networks; one for the generative modeling and one for the discriminative and a DCGAN model. Binary Cross entropy is used as a loss function paired with N-gram graphs[17].
The SLPGAN proves to be quite unstable during training therefore the generator cannot establish the data distribution whereas the DCGAN model presents sufficient results on various datasets.
As for the model architecture, the author follows the guidelines presented in the work of Alec Radford et al.[7].

In their work J. Eastwood et al.[31] they present a method of creating realistic, synthetic surface texture data with a progressively growing generative adversarial network (PPGAN). The model is trained on two datasets that consist of surface texture data that have followed a different manufacturing process and measurement method. A PPGAN model is an extension of the GAN architecture. A PPGAN begins with a vastly downsampled version of the training data and after a predefined number of epochs additional transpose-convolution and convolution layers are appended to the generator and discriminator accordingly, that increase the resolution of the image. The authors begin with a downsampled dataset of 4×4 pixels and increase the resolution to 512×512 pixels.
Then, they compare statistically the distributions between real and produced surfaces by considering two parameters; S_q which is the the RMS height deviation from the mean profile area and S_z which is the maximum height. The distributions produced by the generator show good agreement with the training data.

In their research Jin-Woong Lee et al.[32] propose various models; a DCGAN, a Cycle-consistent GAN and a conditional GAN-based image to image translation (Pix2Pix) model to generate realistic, virtual micrographs; optical microscopy (OM) and scanning electron microscopy (SEM) images of steel surfaces and wire-shaped electrodes for use in Li ion batteries.
The authors gathered the labeled datasets from Hyundai steel Co. industry. More

specifically, they gathered 19216 steel micrographs (9216 OM micrographs and 10000 SEM micrographs) with a size of 256×256 pixels. The training images were magnified variously from $\times 1000$ to $\times 5000$.
The authors state that the quality of the generated micrographs was so good that they could not be differentiated from real microstructure images. The similarity between the real and produced images was measured with KL-divergence which showed below 0.1. The cycle-GAN performed better than DCGAN in terms of image quality. While successfully producing sufficient results with unlabeled image data to realize synthetic images, the authors could not obtain labeled micrograph datasets with their corresponding properties so that they build a property-to-microstructure generation model.

In their research S. Ringdahl et al.[33] study the correlation between surface roughness and ice adhesion strength using molecular dynamic simulations and machine learning. Surface nanoroughness affects the ice composition next to the substrate and therefore deeply affects the local ice adhesion strength. An algorithm for creating random rough surfaces was developed for the needs of the research. The molecular dynamic simulations showed that for rougher surfaces, ice adhesion strength was remarkably lower than smoother surfaces.
Two separate machine learning SVM (Support Vector Machines) algorithms were trained with the results of the molecular dynamic simulations; regression and classification correspondingly. The regression model utilizes the user input data about the substrate and gives back a quantitative prediction of the ice adhesion strength. The SVM presented promising prediction results, which can enable further research in icephobic surface design applications.
In their research O. F. Ogoke et al.[34] study the porosity created in parts produced with Additive Manufacturing. Porosity negatively influences the produced components and their properties during manufacturing.
They propose a GAN framework with Mallat Scattering Transform-based autocorrelation methods to produce synthetic nanorough surfaces. The generated surfaces are then compared to the real surfaces and their porority distributions are evaluated. During evaluation, the generated surfaces are compared to the real data distribution, and are shown to match the univariate distributions of the individual metrics, as well as the bivariate distributions that describe the correlations between the individual pore properties.

Previous work in this domain encountered specific limitations. More specifically, they lack a quantitative method of evaluating the produced images. Most methods rely on visual evaluation of the generated images. Furthermore, instability was encountered when training DCGAN models.

In our work, we use deep learning to generate realistic synthetic nano-surface images, to support nanometrology and nanotechnology processes. To this end we plan to examine how prior domain knowledge can empower a model to provide realistic, synthetic surface images, possibly at different scales and dimensions.
We introduce a unique DCGAN architecture which learns from a set of novel similarity metrics such as N-gram graphs, height histogram and fourier transformation.
This research can lead to new nanomaterial and nanostructure designs and find routes to produce new materials with unique properties.

3 Proposed method

This section introduces our proposed method for generating synthetic nano-rough surfaces. The text in this section is organized as follows.
We define our problem with an objective function that we wish to solve. Then, we propose our network architecture and the limitations we encountered. After that, we present the similarity content we incorporated into our models. Finally, we present our algorithmic procedure for training the network. Also, throughout the section we emphasize our contributions to this research.

3.1 Problem definition

Given a set of real nano-rough surfaces S, we want to be able to learn a generator function G which can produce synthetic surfaces S' that hold properties identical to the real ones.
Based on the work of Vasileios Sioros in *Generating realistic nano-rough surfaces via a Generative Adversarial Network*[1], a nano-rough surface can be characterized as a $2D$ matrix that contains the height values of each point sample of the surface plane.

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

Each nanorough surface is characterized by a set of properties. Those are, the Root-Mean-Square Roughness (R_q), Skewness (R_{sk}), Kurtosis (R_{ku}), Correlation lengths (ξ_x, ξ_y) and Roughness exponent (α). We refer to this set of properties I as the configuration of the surface.

$$I = (R_q, R_{sk}, R_{ku}, \xi_x, \xi_y, \alpha) \tag{3.2}$$

Our purpose is to produce nano-rough surfaces with specific I without previous knowledge of the I configuration of the dataset, but only from nano-rough surface samples. We seek to learn an objective function that maps a set of real nano-rough surfaces with a specific I configuration to a set of generator-produced nano-rough surfaces.

$$F = P(S^{(I)}) \rightarrow P(S'^{(I)}) \tag{3.3}$$

where,
$P(S^{(I)})$ is the powerset of the set of real nano-rough surfaces with configuration I,
$P(S'^{(I)})$ is the powerset of the set of generator-produced nano-rough surfaces with configuration I.

3.2 Our approach

3.2.1 Network overview

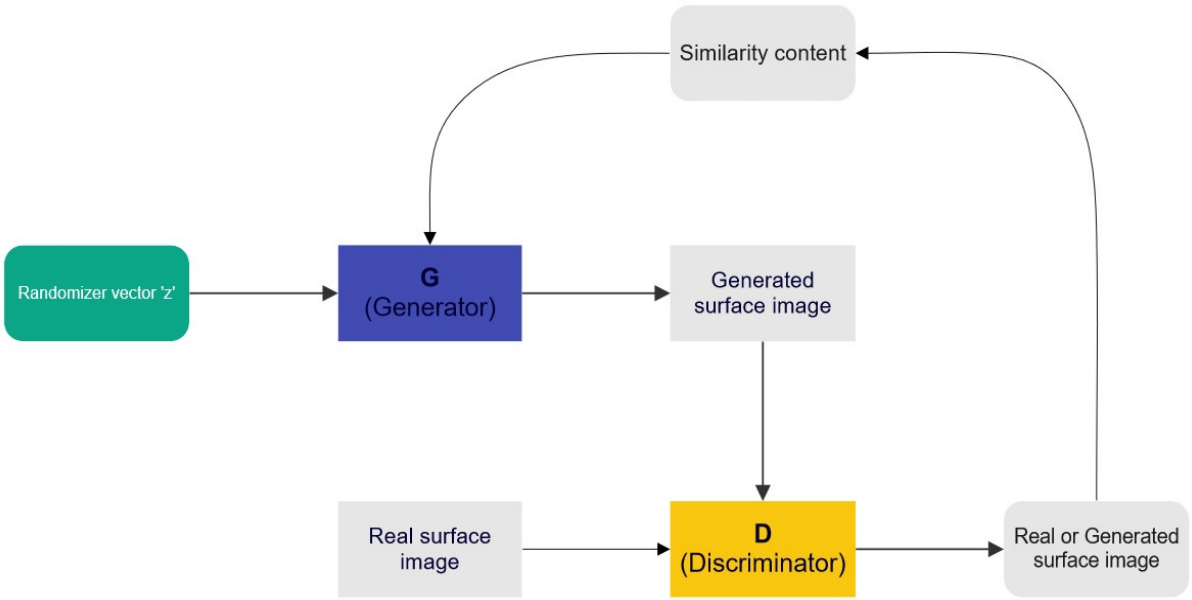

Figure 12: Deep Convolutional Generative Adversarial Network model

Our system is comprised of a Deep Convolutional Generative Adversarial Network. The generator consists of a transposed-convolutional neural network and the discriminator of a convolutional neural network, respectively. The network is trained on a set of nano-rough surfaces that is in accordance with a specific property configuration I.
Our implementation can be found in the following repository https://github.com/ddelikonstantis/RoughML.

3.2.2 Network architecture

Our network comprises of a Generative and a Discriminative model respectively. The Generator as shown in figure 13 consists of 6 transposed-convolutional layers for up-sampling the data from an input randomized vector z. Up-sampling refers to a process that increases the spatial resolution as a means to generate an output that is equal to the image dimension d. The randomized vector z is of size $z = 128$ and is drawn randomly from a normal distribution with $mean = 0$ and $variance = 1$. The Discriminator as shown in figure 14 consists of 6 Convolutional layers for

down-sampling the data. Down-sampling refers to a process that reduces the spatial resolution of a surface image while keeping the same two-dimensional representation. All of the Generator's transposed-convolutional layers are paired with batch-normalization, except from the output layer. Batch-normalization helps the model to train faster and decreases the importance of chosen initial weight values. All layers include Rectified Linear Units, except from the output layer.
All of the Discriminator's convolutional layers are paired with batch-normalization, except from the input and output layer. The Discriminator's convolutional layers are paired with Leaky Rectified Linear Unit functions, except from the output layer, which uses Sigmoid.
The training dataset is loaded via a dataloader into our model in minibatches. The minibatch consists of $minibatch = 64$ nano-rough surface samples. Throughout experimentation we found out that bigger mini-batch sizes result in slower training. In this research, we focus only on grayscale images therefore we use one image channel $c = 1$. The network architecture is configured to process surface images with dimension $d = 128$.

The Generator gets as an input a randomized tensor z that has the following shape.

$$minibatch \times c \times 1 \times 1 \tag{3.4}$$

where,
$minibatch$ is the minibatch size,
c is the image channel

After the up-sampling process the Generator outputs a tensor that has the following shape.

$$minibatch \times c \times d \times d \tag{3.5}$$

where,
$minibatch$ is the minibatch size,
c is the image channel,
d is the dimension of the nano-rough surface image

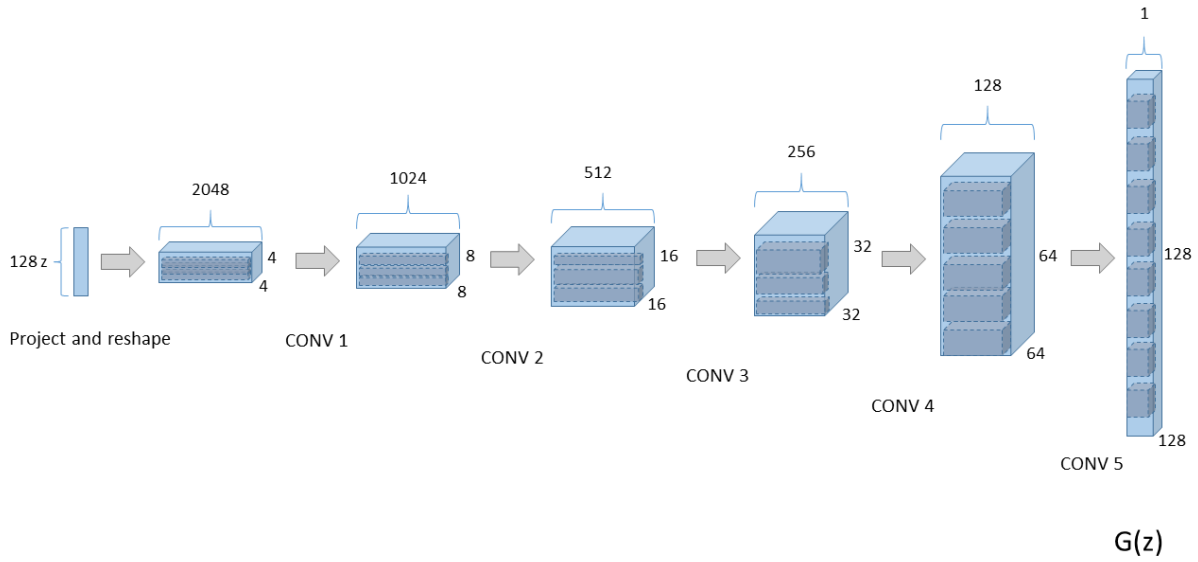

Figure 13: Generator architecture

The Discriminator gets as an input a tensor which is either the Generator's output or surface images from the real dataset. Both have the following shape.

$$minibatch \times c \times d \times d \tag{3.6}$$

where,
$minibatch$ is the minibatch size,
c is the image channel,
d is the dimension of the nano-rough surface image

After the down-sampling process the Discriminator outputs a one-dimensional tensor that includes the target output probabilities and has the following shape.

$$[minibatch] \tag{3.7}$$

where,
$minibatch$ is the minibatch size

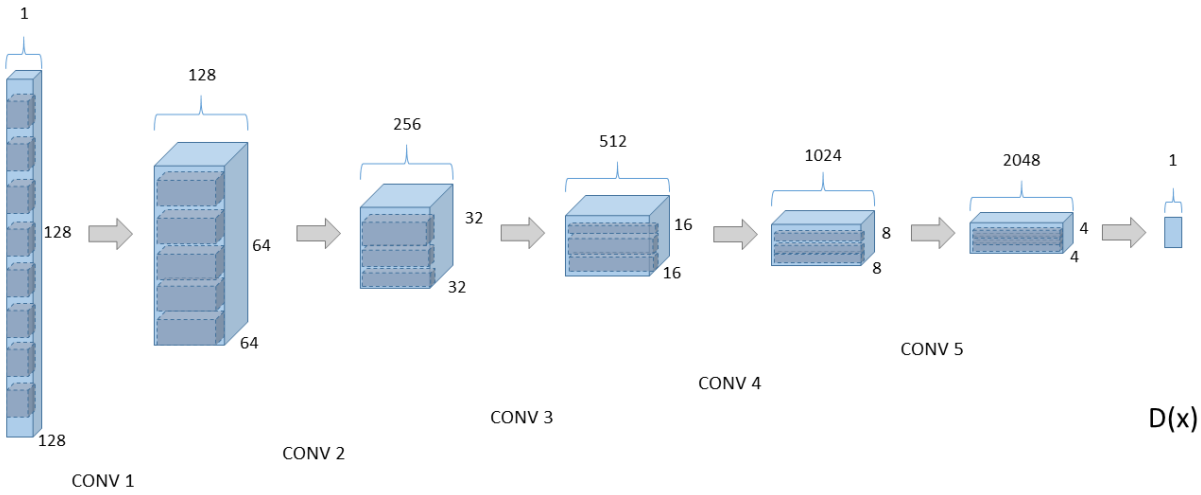

Figure 14: Discriminator architecture

A crucial contribution in this work is that we introduce multiple different network architectures. There are two reasons for the introduction of different network architectures. These reasons are summarized below.

1. Checkerboard artifact
2. Higher surface image dimension d

During early stages of our research we observed a characteristic squared grid lined pattern in our generator-produced surface images across both dimensions. We refer to this as checkerboard artifact. After study, we found out that this issue is quite known in related studies[35]. Therefore, we began to investigate methods on how to reduce it and decided to experiment with the network architecture. We contributed to this research by introducing two different network architectures, each with a different combination of $kernel, stride, padding$ sizes within each layer of the network models. Our proposed network architectures modify only the aforementioned hyperparameters of the models. We present the different network architectures as follows.

- **First proposed network architecture for the checkerboard artifact**

- Generator model
    1. Transposed-convolution layer - kernel size=4, Stride=1, Padding=0, batch-normalization, ReLU
    2. Transposed-convolution layer - kernel size=3, Stride=3, Padding=1, batch-normalization, ReLU
    3. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
    4. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
    5. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
    6. Transposed-convolution layer - kernel size=2, Stride=2, Padding=2
- Discriminator model
    1. Transposed-convolution layer - kernel size=2, Stride=2, Padding=2, LeakyReLU
    2. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
    3. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
    4. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
    5. Transposed-convolution layer - kernel size=3, Stride=3, Padding=1, batch-normalization, LeakyReLU
    6. Transposed-convolution layer - kernel size=4, Stride=1, Padding=0, Sigmoid

- **Second proposed network architecture for the checkerboard artifact**
    - Generator model
        1. Transposed-convolution layer - kernel size=4, Stride=1, Padding=0, batch-normalization, ReLU
        2. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
        3. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
        4. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU

5. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
6. Transposed-convolution layer - kernel size=4, Stride=4, Padding=4

– Discriminator model

1. Transposed-convolution layer - kernel size=4, Stride=4, Padding=4, LeakyReLU
2. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
3. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
4. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
5. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
6. Transposed-convolution layer - kernel size=4, Stride=1, Padding=0, Sigmoid

We decided to proceed with the **second proposed network architecture** for our experiments regarding the checkerboard artifact.

Another contribution to this work is that we introduced a third network architecture to produce higher surface image dimension from $d = 128$ to $d = 256$.

- **Third proposed network architecture for higher surface image dimension from d=128 to d=256**

  – Generator model

  1. Transposed-convolution layer - kernel size=4, Stride=1, Padding=0, batch-normalization, ReLU
  2. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
  3. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
  4. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, ReLU
  5. Transposed-convolution layer - kernel size=4, Stride=4, Padding=3, batch-normalization, ReLU

6. Transposed-convolution layer - kernel size=4, Stride=4, Padding=4

– Discriminator model

1. Transposed-convolution layer - kernel size=4, Stride=4, Padding=4, LeakyReLU
2. Transposed-convolution layer - kernel size=4, Stride=4, Padding=3, batch-normalization, LeakyReLU
3. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
4. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
5. Transposed-convolution layer - kernel size=2, Stride=2, Padding=1, batch-normalization, LeakyReLU
6. Transposed-convolution layer - kernel size=4, Stride=1, Padding=0, Sigmoid

It is worthy to note that we encountered computational power limitations regarding the proposed network architecture for higher surface image dimension due to the vast number of parameters of the network.

3.2.3 Similarity content

Our network utilizes three different similarity metrics. We reasoned it is important to make use of diverse similarity content in an effort to describe multiple aspects of the nano-rough surface. We summarize the similarity content as follows.

- **Binary Cross-Entropy / Log**
  Predicts a probability for a nano-rough surface image to belong to a certain class. In our case, it classifies between real surface images and generator-produced ones.

- **N-Gram Graphs (ArrayGraph2D variant)** [17]
  A graph representation of nano-rough surfaces that compares the local co-occurences of surface peaks between real nano-rough surfaces and generator-produced ones.

- **Height Histogram and Fourier**
  A Fourier representation of nano-rough surface images for comparing the frequency content between real surface images and generator-produced ones.

Binary Cross-Entropy / Log
Binary Cross-Entropy is a loss function that is used in binary classification tasks. It calculates the loss of a sample by computing the following average.

$$BinaryCrossEntropy = -\frac{1}{output\_size} \sum_{i=1}^{output\_size} y_i * log\hat{y}_i + (1 - y_i) * log(1 - \hat{y}_i) \quad (3.8)$$

where $\hat{y}_i$ is the i^{th} scalar value in the model output,
y_i is the corresponding target value,
$output\_size$ is the number of scalar values in the model output.

This is equivalent to the average result of the Categorical Cross-Entropy loss function applied to many independent classification applications, each problem having only two possible classes with target probabilities y_i and $(1-y_i)$. We apply Binary Cross-Entropy content loss to both models, the Generator and Disriminator respectively. By minimizing Binary Cross-Entropy similarity content the Discriminator learns to distinguish between real and generator-produced surfaces images, whereas the Generator learns to produce surface images similar to the real ones.

Preprocessing
A nano-rough surface is composed from a $2D$ matrix that includes floating point numbers that correspond to its height values. These values may have a non-standard distribution due to outliers and multi-modal distribution. In an effort to improve the network performance we transform the surface values to a different standard distribution.
For the N-Gram Graph content a prepossessing method called Discretization takes place that bins continuous floating point data into intervals with the quantile grouping method. Discretization creates 5 bins where each bin contains an equal number of values.
For the Height Histogram and Fourier content preprocessing occurs with histogram analysis. We use histogram to transform the surface values to a normal distribution.

N-Gram Graphs (ArrayGraph2D)
Before training, the N-Gram Graph content of the training dataset is calculated for every nano-rough surface.
During inference, the N-Gram Graph content is calculated for each generator-produced surface .
Then, the generator-produced graph is compared to the training dataset's graph.
We use the ArrayGraph2D (AG2D) variant of the N-Gram Graphs. Algorithm 2 describes how N-Gram Graph content processes a nano-rough surface matrix.

Algorithm 2 Creating a 2D array graph from a matrix M with dimensions $N \times N$.

$G \leftarrow \emptyset$
for $y \in [0, ..., N]$ **do**
  for $x \in [0, ..., N]$ **do**
    $vertex_{y,x} = M[y, x]$
    $neighbourhood \leftarrow \emptyset$
    $neighbour_y^{min} = Clamp_N(y - [\frac{w}{2}])$
    $neighbour_y^{max} = Clamp_N(y + [\frac{w}{2}])$
    for $neighbour_y \in [neighbour_y^{min}, ..., neighbour_y^{max}]$**do**
      $neighbour_x^{min} = Clamp_N(x - [\frac{w}{2}])$
      $neighbour_x^{max} = Clamp_N(x + [\frac{w}{2}])$
      if $neighbour_y \neq y$ **or** $neighbour_x \neq x$ **then**
        $vertex_{neighbour_y, neighbour_x} = M[neighbour_y, neighbour_x]$
        $G \leftarrow G \cup \{(vertex_{y,x}, vertex_{neighbour_y, neighbour_x})\}$

where $Clamp_N$ contains the floating point range value $[0, N]$ and is calculated by the following operation.

$$clamp(v) = \max(0, \min(v, N - 1) \tag{3.9}$$

Height Histogram and Fourier

Another crucial contribution in this work is that we introduced the Height Histogram and Fourier similarity metric to the learning procedure.
Height Histogram and Fourier calculates the $2D$ Fast Fourier Transform and the height histogram of a given nano-rough surface.
Before training, the height histogram values as well as the FFT are calculated for every nano-rough surface contained in the training dataset and are stored.
The number of bins required for the histogram height values is calculated using the following formula.

$$bins = \max(10, 10^{(\min(log(s^2))-3)}) \tag{3.10}$$

where s is the image dimension.

This dynamic selection of bins ensures a correlative behaviour between the image dimension and the bin size.
During inference, the Height Histogram and Fourier similarity between each generator-produced surface and the training dataset's surfaces is computed.
Lets assume a produced nano-rough surface M and the training nano-rough surface dataset T. The procedure is presented in algorithm 3.

Algorithm 3 Height Histogram and Fourier similarity content between a generator-produced nano-rough surface M and training dataset surfaces T.

$H_M = Histogram(M)$
$F_M = FFT2D(M)$
$diff = 0$
$Hist\_diff = 0$
$Fourier\_diff = 0$
for $Histogram, Fourier \in FH_T$ **do**
    $Hist\_diff += \sqrt{(\overline{Histogram_T - H_M})^2}$
    $Fourier\_diff += \sqrt{(\overline{Fourier_T - F_M})^2}$
$Hist\_diff = Normalized(Hist\_diff)$
$Fourier\_diff = Normalized(Fourier\_diff)$
$diff = Weighted(Hist\_diff + Fourier\_diff)$
$loss = 1 - (\frac{1}{1+diff})$
return $loss$

3.2.4 Training

Training is split into two main parts. Part 1 updates the Discriminative and part 2 updates the Generative model respectively.

1. In the first training part, we want to maximize the probability of the Discriminator so that it correctly classifies a given surface image, as real or generator-produced.

$$log(D(x)) + log(1-D(G(z))) \tag{3.11}$$

   where $log(D(x))$ is the Discriminator output on real surface images, $log(1-D(G(z)))$ is the Discriminator output on generator-produced surface images.

   Since we implement training in mini-batches, we calculate this in two steps.

   In the first step, we construct a mini-batch of real surface images from the training dataset and do a forward pass through the Discriminator. After that, we calculate the Binary Cross-Entropy loss $log(D(x))$ on real surface images with the *real* label 1. Then, we normalize the Binary Cross-Entropy loss output. One of our contributions in this work is that we normalize all losses. In the final paragraph of this section we will present our reasoning behind this modification. Additionally, we calculate the gradients in a backward pass.

In the second step, we construct a mini-batch of generator-produced surface images and do a forward pass of this mini-batch through the Discriminator. After that, we calculate the Binary Cross-Entropy loss $log(1-D(G(z)))$ on generator-produced surface images with the *generator-produced* label 0. We normalize the Binary Cross-Entropy loss output as in the first step and then calculate the gradients with a backward pass. The gradients are accumulated with previous gradients calculated in the first step.

Finally, we compute the loss of the Discriminator as a weighted sum over the normalized individual losses from the real and generator-produced mini-batches. Then, we call a step of the Discriminator's optimizer to update the parameters.

2. During the second training part we train the Generator. We want the Generator to minimize $log(1-D(G(z)))$ in an effort to generate better surface images. But, this was shown in previous studies[36] to not provide sufficient gradients, especially early in the learning process. We instead maximize $log(D(G(z)))$ as suggested by said studies.

   As a first step, we classify the Generator output with the Discriminator. Then we extract each individual similarity content for the generator-produced mini-batch as shown below.

   - The Binary Cross-Entropy difference is computed for the generator-produced batch by using real labels as generator-produced labels.
   - The N-Gram Graph content is calculated for every surface in the generator-produced minibatch. Before training the N-Gram Graph content is calculated for every surface in the training dataset and is stored. During inference the Normalized Value Similarity between the generator-produced and the training dataset is computed. The Normalized Value Similarity is the N-Gram Graph loss for the current minibatch.
   - The Height Histogram and Fourier content is calculated for every surface in generator-produced minibatch. Before training the Height Histogram and Fourier content is calculated for every surface in the training dataset and is stored. During inference the difference between each generator-produced surface content with every training dataset surface content is accumulated. This happens recursively for every generator-produced surface. The accumulated difference for all generator-produced surfaces is the Height Histogram and Fourier loss for the current minibatch.

A crucial contribution to this work is that we normalize each one of the independent losses. Additionally, we weight them so that all three contribute equally to Generator's learning. We came to this conclusion after noticing that the loss with the highest value often hindered the rest from performing. Since our intent for the generator-produced surface images is to assimilate diverse content equally we implemented this functionality.
Therefore, the overall Generator loss is computed as a sum of the individual normalized and weighted component losses.

Finally, we compute the Generator's gradients in a backward pass, and update the Generator's parameters.

Algorithm 4 presents the entire training process of our model. All losses are accumulated within each minibatch and at the end of the batch they are divided by the total number of minibatch iterations as a means to get average losses for each epoch.

Algorithm 4 Deep Convolutional Generative Adversarial Network training algorithm. In the algorithm, BCE refers to Binary Cross Entropy, NGG refers to N-Gram Graphs and HHF refers to Height Histogram and Fourier similarity contents respectively. Furthermore, x refers to the real training minibatch.

for $epoch$ in $total\_epochs$
  for x in $dataloader$
    $netD.zero\_grad()$
    $label = 1$
    $output = netD(x)$
    $errD\_real = Normalized(BCE(output, label))$
    $errD\_real.backward()$
    $Real\_training = output.mean.item()$
    $z = randomized\ vector$
    $Fake\_batch = netG(z)$
    $label = 0$
    $output = netD(Fake\_batch)$
    $errD\_fake = Normalized(BCE(output, label))$
    $errD\_fake.backward()$
    $Fake\_training = output.mean.item()$
    $errD\_total = Weighted(errD\_real + errD\_fake)$
    $optimizerD.step()$
    $netG.zero\_grad()$
    $label = 1$
    $output = netD(Fake\_batch)$
    $errG\_BCE = Weighted(Normalized(BCE(output, label)))$
    $errG\_NGG = Weighted(Normalized(NGG(Fake\_batch, Training\_dataset)))$
    $errG\_HHF = Weighted(Normalized(HHF(Fake\_batch, Training\_dataset)))$
    $errG\_total = errG\_BCE + errG\_NGG + errG\_HHF$
    $errG\_total.backward()$
    $Fake\_test = output.mean.item()$
    $optimizerG.step()$
    $lossG += errG\_total$
    $lossD += errD\_total$
    $lossG\_BCE += errG\_BCE$
    $lossG\_NGG += errG\_NGG$
    $lossG\_HHF += errG\_HHF$
    $Real\_training\_output += Real\_training$
    $Fake\_training\_output += Fake\_training$
    $Fake\_test\_output += Fake\_test$
  $lossG /= len(dataloader)$
  $lossD /= len(dataloader)$
  $lossG\_BCE /= len(dataloader)$
  $lossG\_NGG /= len(dataloader)$
  $lossG\_HHF /= len(dataloader)$
  $Real\_training\_output /= len(dataloader)$
  $Fake\_training\_output /= len(dataloader)$
  $Fake\_test\_output /= len(dataloader)$

4 Experimental results

In order to perform accurate experimentation, a plan of action was established in an effort to present reliable and valid results. This involved a specific set of questions that was decided to be investigated as a means to obtain substantial information about the network performance and capabilities. Often a question would result in a new one throughout the course of the research. The course plan initially comprised of the following questions.

- Can we improve the network's learning ability?
- Can we reduce the checkerboard artifacts observed in surface images?
- Can we extend the network for higher image dimension (d)?
- What effect do the N-Gram Graph parameters have on the overall learning progression of the network as well as the individual progression of the N-Gram Graph?
- What kind of effect do the chosen initial weights have on our network performance and results?

4.1 Experimental setup

4.1.1 Dataset

For our experimentation we make use of nano-rough surfaces that are produced from an algorithm proposed by A. Stellas et al[5]. The nano-rough surface dataset consists of $2D$ square matrices with surface height values. The height values define the nano-rough surface profile. Positive height values describe peaks in the surface and negative height values represent valleys. Each chosen value of the roughness parameters affects the morphology of the produced surface. For theoretical concepts on nano-roughness refer to section 2.1.1. We summarize the dataset parameters as follows.

- Number of surfaces (s)
  Represents the total number of nano-rough surface samples used for our training dataset.
- Dimension of surfaces (d)
  Represents the size of the $2D$ square matrix that defines the dimension of the nano-rough surface. Resulting surfaces are square matrices with dimensions $d \times d$.

- Root-Mean-Square Roughness (R_q)
  Represents the Root-Mean-Square Roughness (R_q) value of the surface.
- Skewness (R_{sk})
  Represents the skewness (R_{sk}) value of the surface.
- Kurtosis (R_{ku})
  Represents the kurtosis (R_{ku}) value of the surface.
- Correlation lengths (ξ_x, ξ_y)
  Represent the correlation lengths (ξ_x, ξ_y) value of the surface in 2 dimensions.
- Roughness exponent (α)
  Represents the Roughness exponent (α) value of the surface.

Each parameter affects a certain aspect of the morphology of the surface. Therefore, choosing different parameter values produce a surface with unique morphology.

The algorithm for producing nano-rough surfaces presented by A. Stellas et al.[5] comprises of the following steps.

1. White noise generation $Z(x, y)$ with N discrete points in a square grid with length rL. The white noise height distribution has $mean = 0$ and $variance = R_q$. R_q value is user input.
2. Gaussian filter generation.
   - For isotropic surfaces: $F_{iso}(x, y) = exp(-(\frac{2x^2+2y^2}{\xi_x^2}))$
   - For anisotropic surfaces: $F_{aniso}(x, y) = exp(-(\frac{2x^2}{\xi_x^2} + \frac{2y^2}{\xi_y^2}))$
3. In the next step, the surface becomes correlated by calculating either Z_{iso} or Z_{aniso} (depending on the selected surface type) with correlation lengths ξ_x and ξ_y.
   - For isotropic surfaces: $Z_{iso} = \frac{2}{\sqrt{\pi}} \frac{rL}{N\sqrt{\xi_x}} \cdot IFFT(FFT(Z) \cdot FFT(F_{iso}))$
   - For anisotropic surfaces:
     $Z_{aniso} = \frac{2}{\sqrt{\pi}} \frac{rL}{N\sqrt{\xi_x \xi_y}} \cdot IFFT(FFT(Z_{WG}) \cdot FFT(F_{aniso}))$

   where,
   $FFT(Z)$ is the Fast Fourier Transformation of the white noise in step 1,
   $FFT(F_{iso})$ and $FFT(F_{aniso})$ is the Fast Fourier Transformation of the Gaussian filter in step 2,
   $IFFT$ is the Inverse Fast Fourier Transformation

4. Finally, the roughness parameters R_q, ξ_x, ξ_y are calculated for the surface. Parameter R_q is calculated by using the formula in step 3, (Z_{iso}) or (Z_{aniso}) while correlation lengths ξ_x, ξ_y require initially the calculation of the auto-correlation function ACF.

   - $ACF(r_x) = \frac{1}{R_q^2(l-r_x)} \sum_1^{l-r_x} (y(x) - \langle y \rangle)(y(x+r_x) - \langle y \rangle)$

   From the ACF formula, two points are taken into consideration:
   $ACF(x, y = 0) = \frac{1}{e}$ and $ACF(x = 0, y) = \frac{1}{e}$.
   For such small values the ACF is considered to advance exponentially.
   Therefore ξ_x and ξ_y parameters result from:

   - $ACF(x, y = 0) = \frac{1}{e} \iff x = \xi_x$
   - $ACF(x = 0, y) = \frac{1}{e} \iff y = \xi_y$

Due to the computationally demanding nature of neural networks experimentation was performed in two separate hardware setups. One served for development and another for experiments. The setup for development consisted of an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, 16.0 GB of RAM, an NVIDIA GeForce GTX 1060 GPU with max-Q Design. This setup served as a means to develop the software and test small-scale experiments. As for the experimentation setup, it consisted of an AMD Ryzen Threadripper Pro 3955WX CPU @ 3.9GHz, 128 GB of RAM and an NVIDIA RTX A6000 GPU. Software was developed with Python version 3.7.1 on Visual Studio Code.

The PyTorch framework was used for software development of the Deep Convolutional Generative Adversarial Network. PyTorch is an open source machine learning framework based on the Torch library. Furthermore, we made use of libraries such as NumPy, SciPy, math, scikit-learn, pandas, OpenCV, Matplotlib, Pillow, itertools and PyINSECT.

In general, deep learning training processes are stochastic in nature. This randomness is often acceptable and indeed desirable. However, in order to have clear answers and be confident in our results we incorporated repeatable randomness in our experiments. We configured PyTorch to incorporate repeated randomness through random number generator seeding, so that multiple calls, given the same inputs, will produce the same result.

We used the Adaptive moment estimation algorithm for optimizing our network with learning rate $lr = 0.0002$, $\beta_1 = 0.5$, $\beta_2 = 0.999$. We set the parameter $bias = False$ for

each convolutional and transposed-convolutional layer of the network. Furthermore, we set the LeakyReLU $negative\_slope = 0.2$ where applied in the convolutional layers of the Discriminator. Additionally, we set ReLU and LeakyReLU inplace operation $inplace = True$.

Experimental results are evaluated by the following measures.

- Qualitative evaluation by domain experts.
- Quantitative measures evaluation through graph plots. Throughout section 4 we evaluate quantitative results based upon three graph plots.

    1. Generator and Discriminator losses per epoch
    Presents the average and normalized Generator loss per epoch as a weighted sum of the individually average and normalized Binary Cross-Entropy, N-Gram Graph, Height Histogram and Fourier losses. Also, the average and normalized Discriminator loss per epoch as a weighted sum of the individually normalized Binary Cross-Entropy loss over the real training minibatch and the generator-produced tranining minibatch.
    2. Binary Cross-Entropy, N-Gram Graph, Height Histogram and Fourier losses per epoch
    Presents the average and normalized Binary Cross-Entropy, N-Gram Graph, Height Histogram and Fourier losses per epoch.
    3. Discriminator output per epoch
    Presents the average Discriminator output per epoch on real training minibatch, generator-produced training minibatch and generator-produced test minibatch.

4.1.2 Experimental results on network initial weights

Weight initialization is an important aspect of a neural network model. The nodes in the neural network are composed of parameters (weights) used to calculate a weighted sum of the inputs. The optimization algorithm requires a starting point in the space of possible weight values from which to begin the optimization process. Weight initialization refers to the process of setting the weights of the neural network to small random values that define the starting point for the optimization of the model. According to Alec Radford et al.[7], the initial weights of the DCGAN network should be randomly initialized from a normal distribution with mean=0, stdev=0.02.
We wondered what kind of effect, if any, the chosen initial weight values would have on our network performance and results. Therefore, we conducted a preliminary

experiment on the same dataset with different initial weights applied on both models, one with values $\boldsymbol{mean = 0, stdev = 0.02}$, and another with $\boldsymbol{mean = 0, stdev = 0.2}$. We gave the optimization algorithm a much larger initial space of possible weight values to begin with. This way, if any difference in results or performance, would become apparent. We trained our network with $\boldsymbol{s = 300}$ nano-rough surface samples for $\boldsymbol{50}$ epochs in both cases.

1. Random initial weight values from a normal distribution with $\boldsymbol{mean = 0, stdev = 0.02}$.

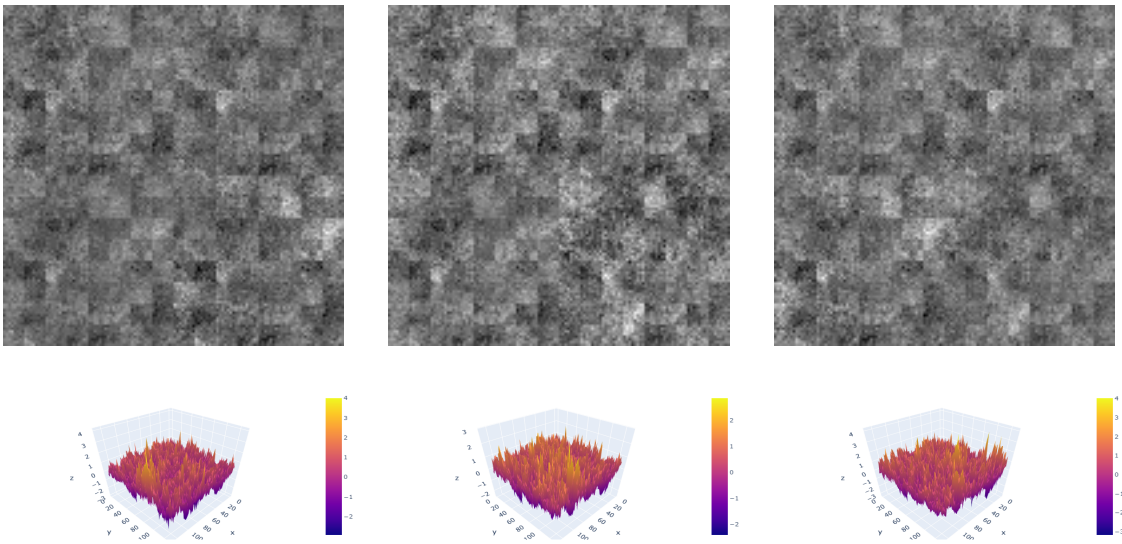

Figure 15: Generator-produced nano-rough surface images for initial weight values with $mean = 0, stdev = 0.02$.

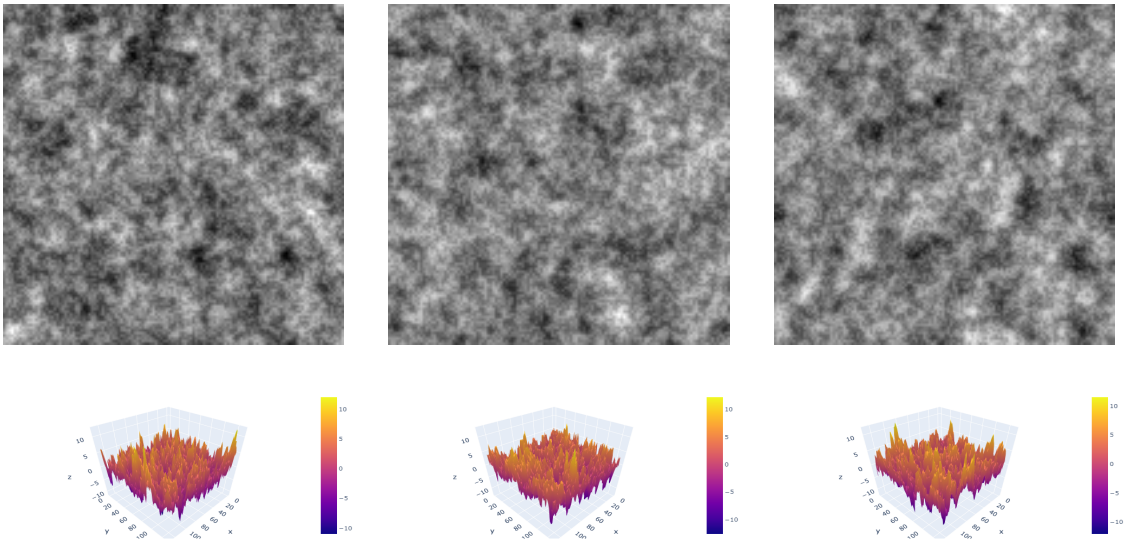

Figure 16: Real dataset nano-rough surface images for initial weight values with $mean = 0, stdev = 0.02$.

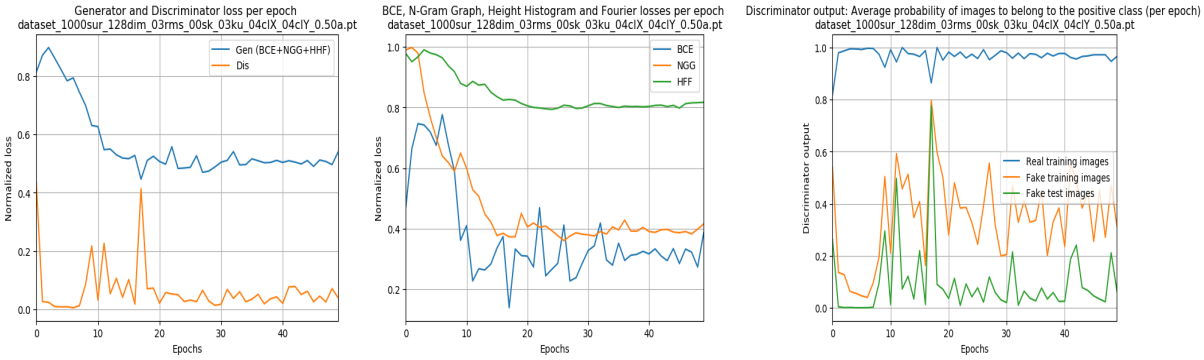

Figure 17: Graph plots for initial weight values with $mean = 0, stdev = 0.02$.

2. Random initial weight values from a normal distribution with $\boldsymbol{mean = 0, stdev = 0.2}$.

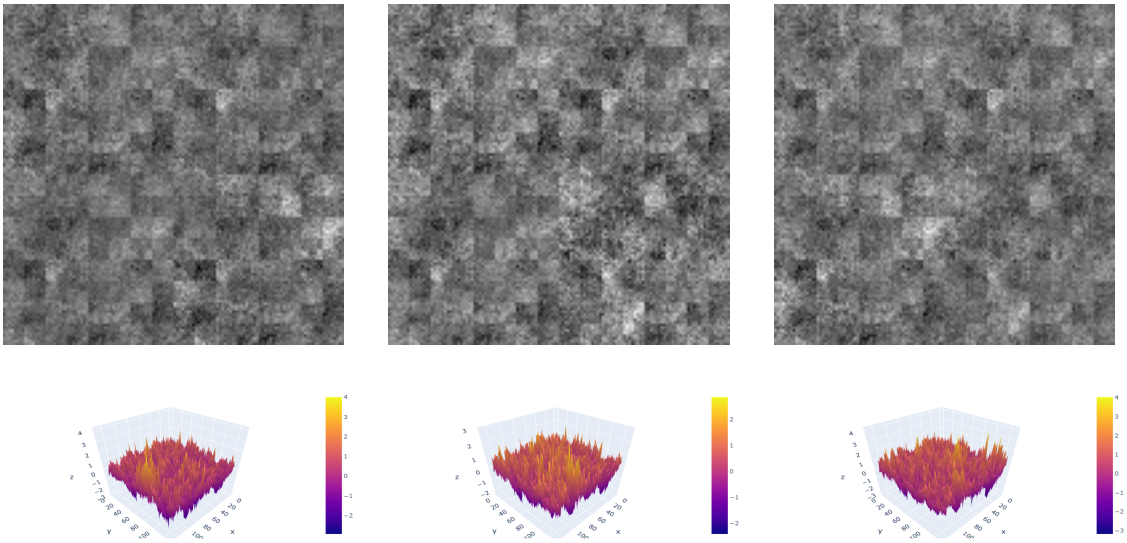

Figure 18: Generator-produced nano-rough surface images for initial weight values with $mean = 0, stdev = 0.2$.

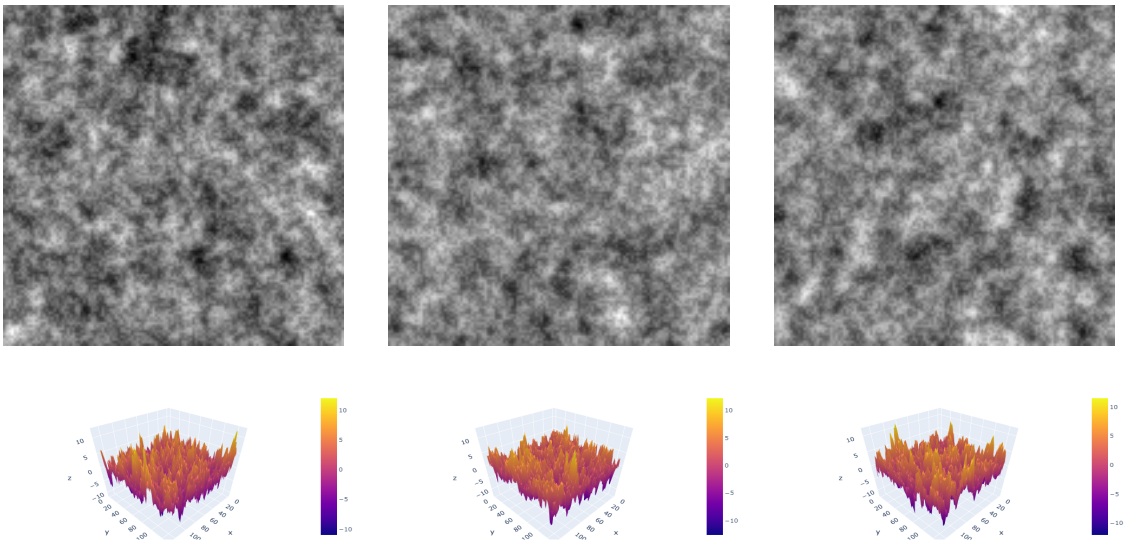

Figure 19: Real dataset nano-rough surface images for initial weight values with $mean = 0, stdev = 0.2$.

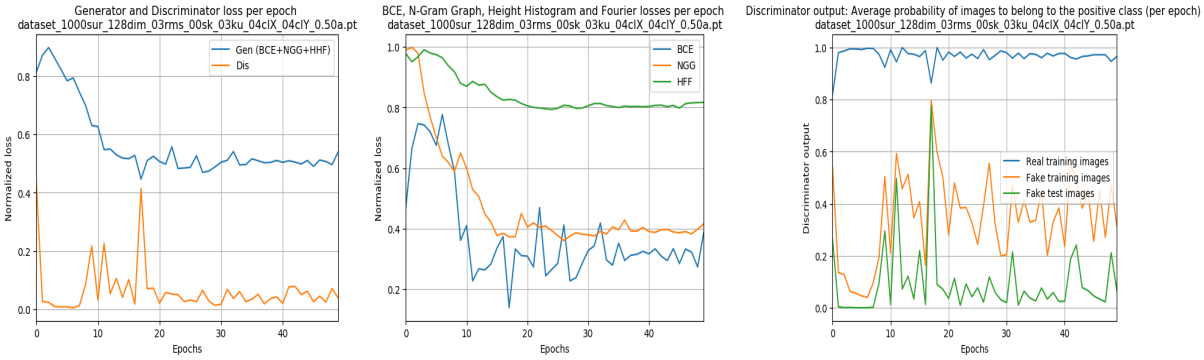

Figure 20: Graph plots for initial weight values with $mean = 0, stdev = 0.2$.

Early findings show that, initial weights appear to have an insignificant effect on learning. Furthermore, we observe that during training loss plots and Discriminator output behave almost the same, especially in early stages.
However, the experiment was implemented with insufficient number of surface samples $s = 300$ for a limited number of epochs $epochs = 50$.
We can't make any conclusive statements regarding the initial weights effect on learning and we consider that this subject requires an extended experiment with more surface samples for a prolonged number of epochs.
For the means of our research we decided for the initial weight values to be randomly initialized from a normal distribution with $mean = 0, stdev = 0.02$.

4.1.3 Experimental results on N-Gram Graph parameters

As a next step, we wondered what effect do the N-Gram Graph parameters have on the overall learning progression of the network. N-gram graphs content has three main parameters that affect similarity.

- ***N***
  Represents the N-gram size. For example, $N = 1$ refers to unigrams, $N = 2$ refers to bigrams and so on. In our case N-grams refer to the discretization transformed bins within the surface matrices.

- ***Window_size***
  Represents the number of non-symmetrical neighbouring N-grams that are taken into account for calculating the weights of the edges of the graph.

- ***Stride***
  Is the step s from previous to the next N-gram.

We tested with $\boldsymbol{s = 300}$ training samples for **50** epochs. To get an extended idea, we decided to experiment with two different sets of N-Gram Graph parameters $N = 3, window\_size = 3, stride = 1$ and $N = 8, window\_size = 8, stride = 1$ on two datasets with different Roughness exponent $\alpha = 0.5$ and $\alpha = 1.0$. We summarize the experiment cases below.

1. $\boldsymbol{N = 3, window\_size = 3}, stride = 1$
   - $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 0.5}$
   - $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 1.0}$
2. $\boldsymbol{N = 8, window\_size = 8}, stride = 1$
   - $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 0.5}$
   - $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 1.0}$

The reason for different Roughness exponent (α) cases is because we want to observe the N-Gram Graph parameters for surface profiles with different self-similarity and local-smoothness.

1. $\boldsymbol{N = 3, window\_size = 3}, stride = 1$
   - $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 0.5}$

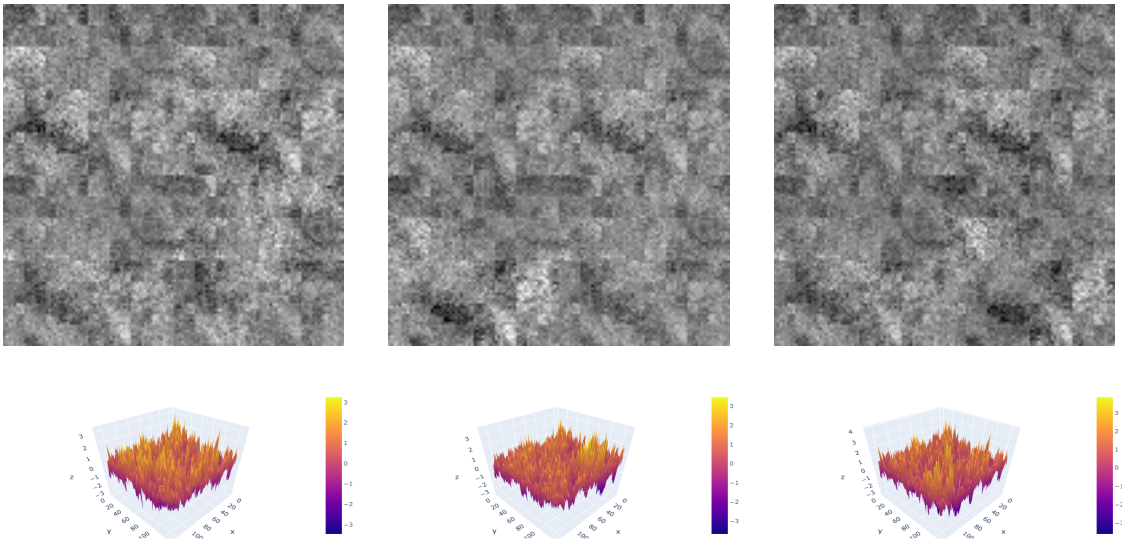

Figure 21: Generator-produced nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 0.5$.

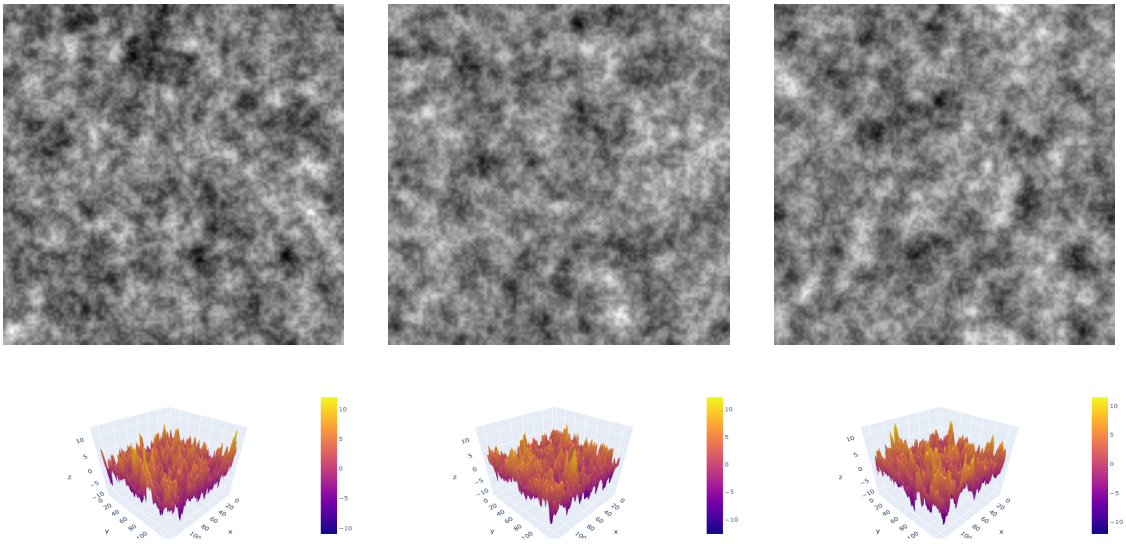

Figure 22: Real dataset nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 0.5$.

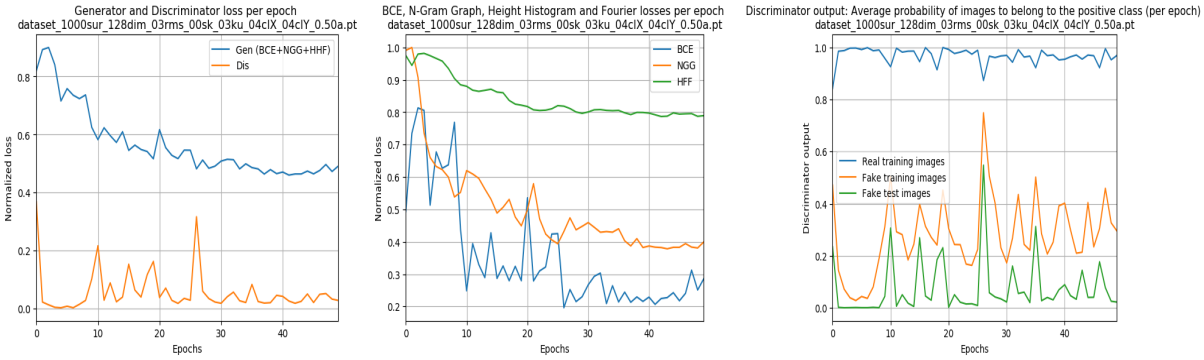

Figure 23: Graph plots for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 0.5$.

- $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 1.0}$

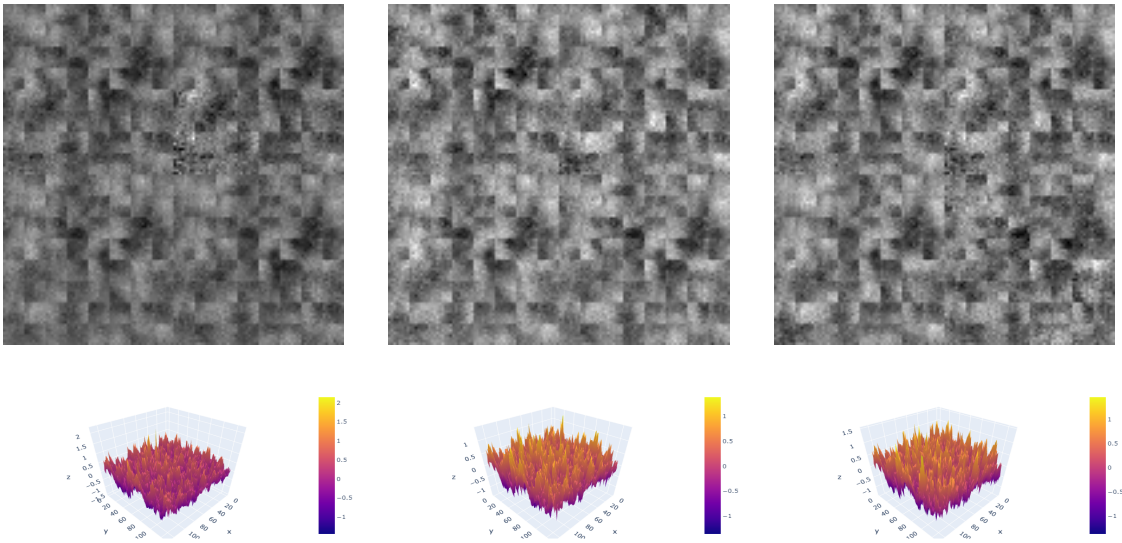

Figure 24: Generator-produced nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 1.00$.

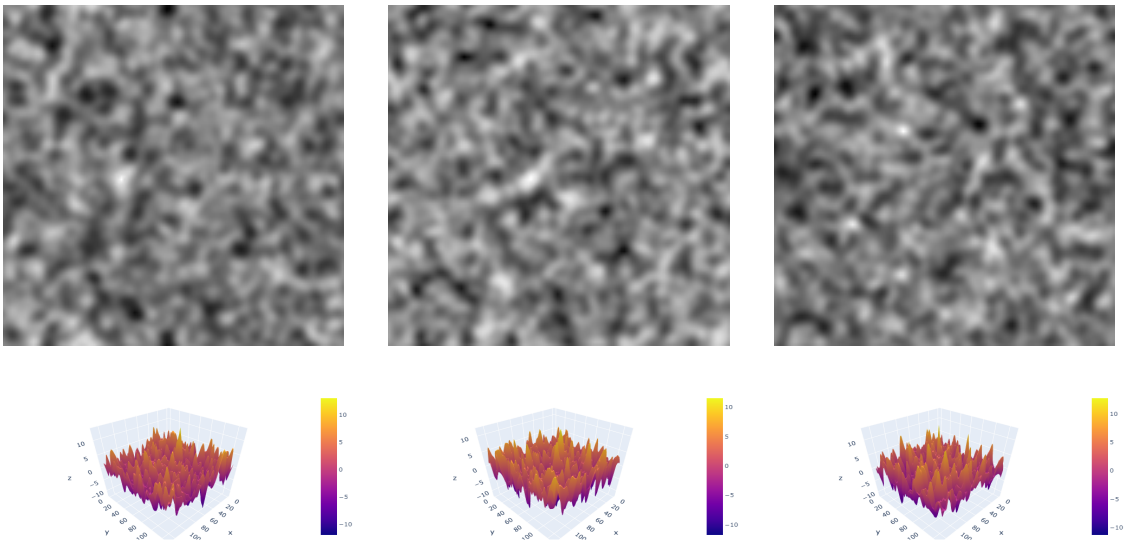

Figure 25: Real dataset nano-rough surface images for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 1.00$.

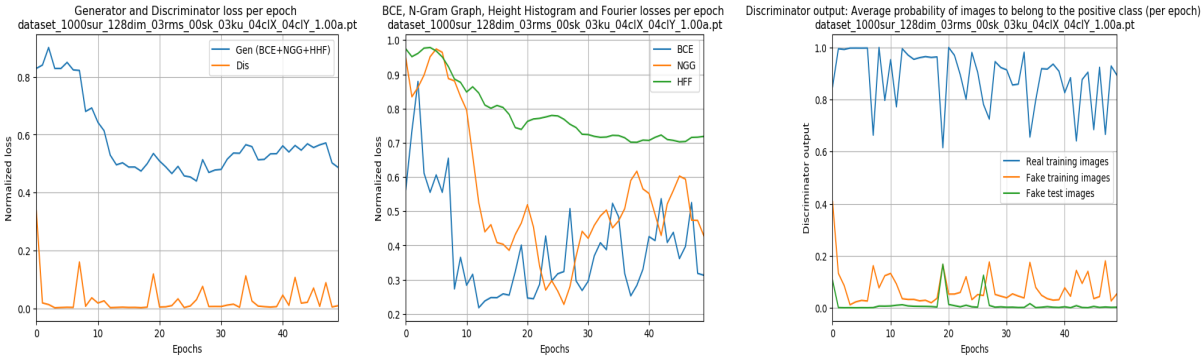

Figure 26: Graph plots for $N = 3, window\_size = 3, stride = 1$ and $\alpha = 1.00$.

2. $\boldsymbol{N = 8, window\_size = 8}, stride = 1$

   - $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 0.5}$

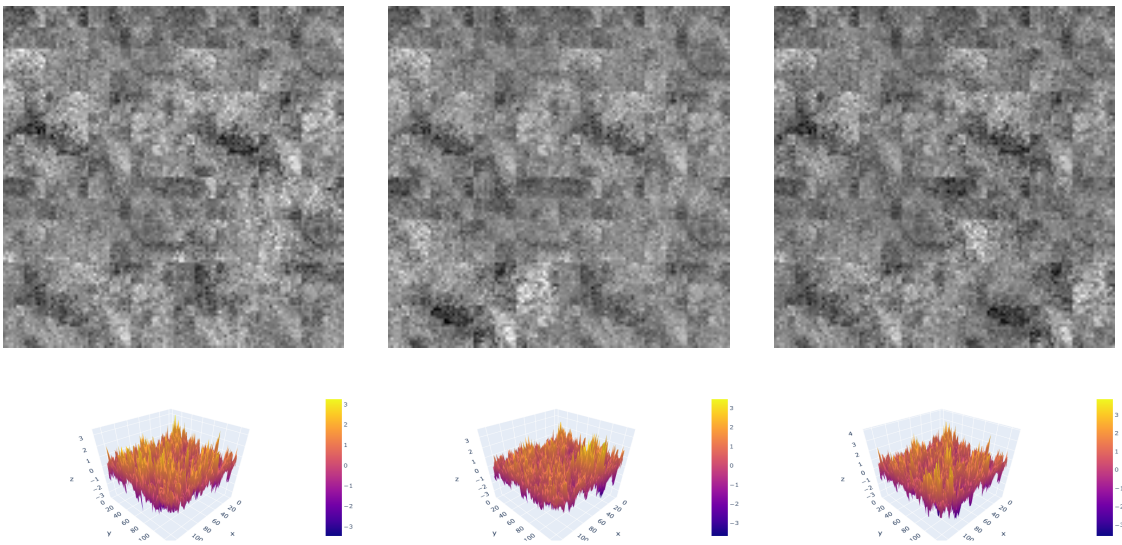

Figure 27: Generator-produced nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 0.5$.

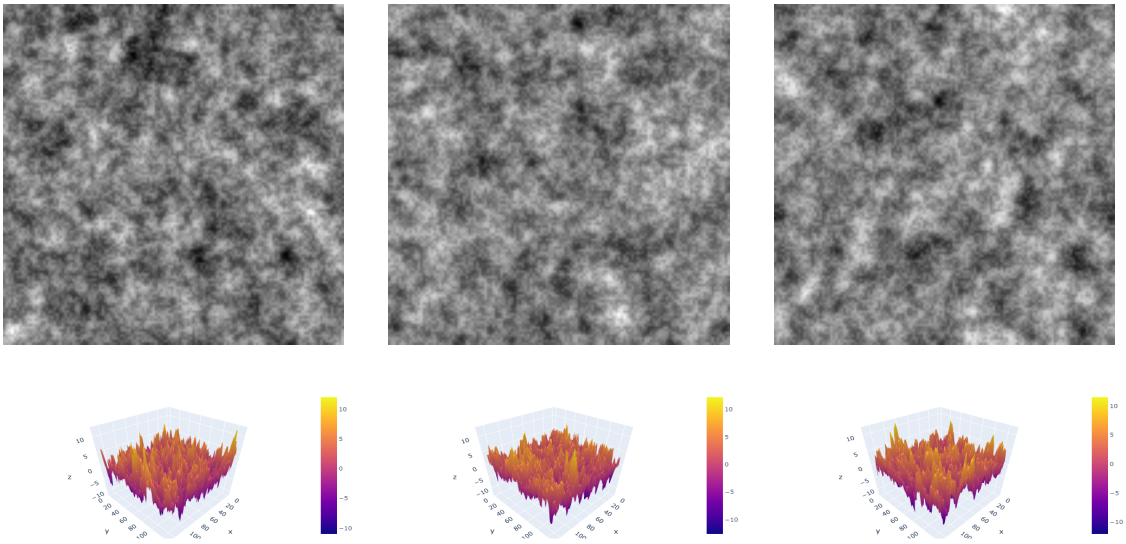

Figure 28: Real dataset nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 0.5$.

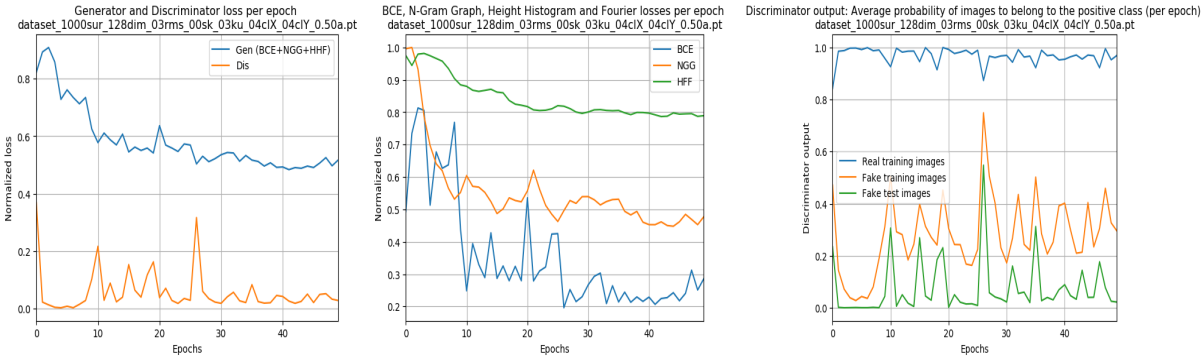

Figure 29: Graph plots for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 0.5$.

- $s = 300, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \boldsymbol{\alpha = 1.0}$

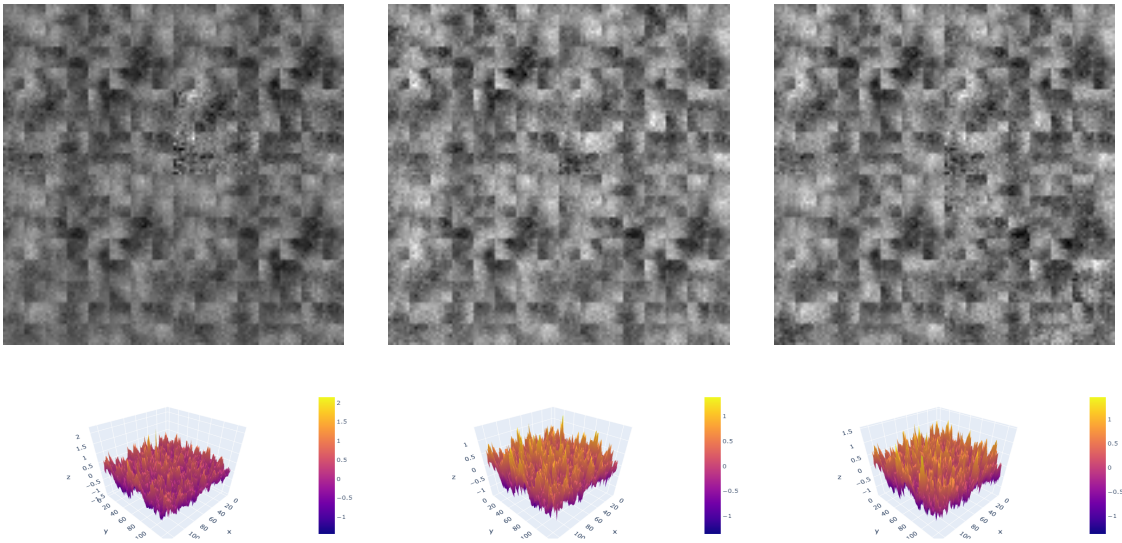

Figure 30: Generator-produced nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 1.00$.

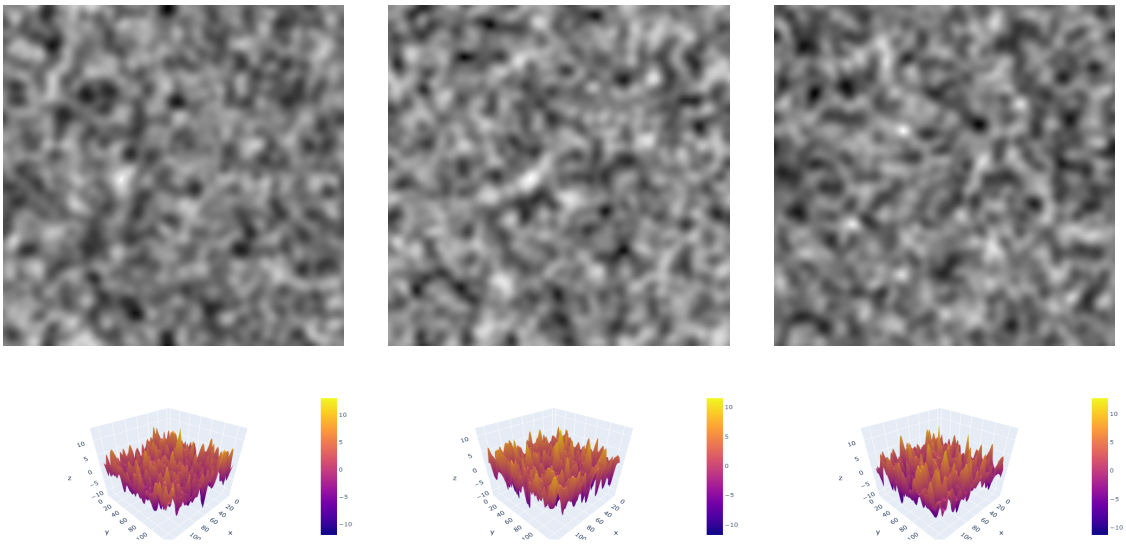

Figure 31: Real dataset nano-rough surface images for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 1.00$.

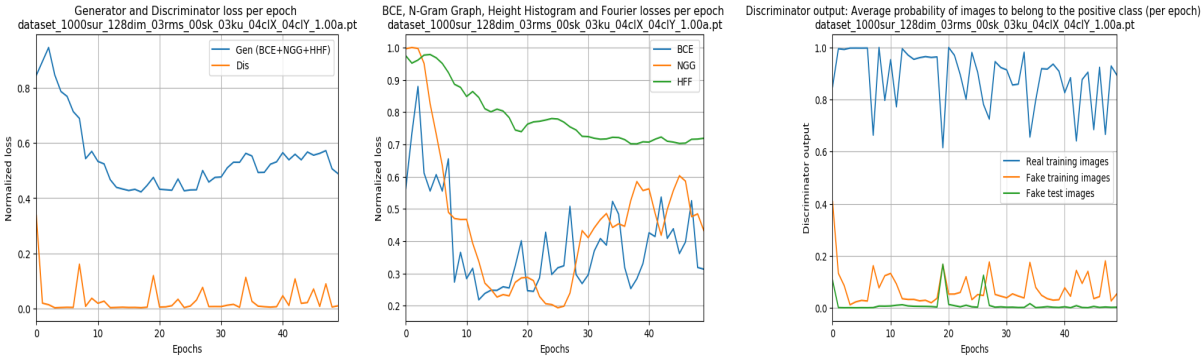

Figure 32: Graph plots for $N = 8, window\_size = 8, stride = 1$ and $\alpha = 1.00$.

After observing the plots, we notice a slightly better performance of the N-gram graph and Height Histogram and Fourier similarity metrics on both cases for $\alpha = 1.0$. As for the $\boldsymbol{N}, \boldsymbol{window\_size}$ parameters there is inconclusive evidence that suggest that one performs better than the other. Experimentation with more surface samples and for more epochs might yield more conclusive evidence.

4.1.4 Experimental results on higher surface image dimension

As previously mentioned in section 3.2.2 a crucial contribution of our work is that we extended the network architecture to process surface images with dimension $\boldsymbol{d = 256}$. As a next step we tested our model with nano-rough surface images with size $\boldsymbol{d = 256}$. However, due to limitations in computational and memory power required for the vast number of parameters we were able to train our network only with $\boldsymbol{s = 50}$ surface samples for 10 epochs. Therefore, we note that this test serves only as a preliminary experiment.

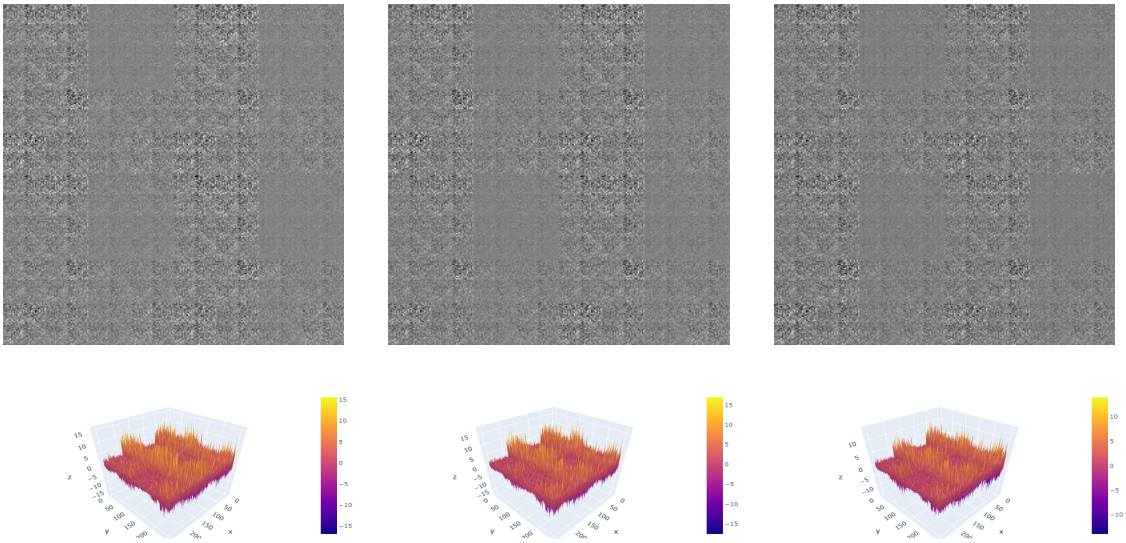

Figure 33: Generator-produced nano-rough surface images for dimension $d = 256$.

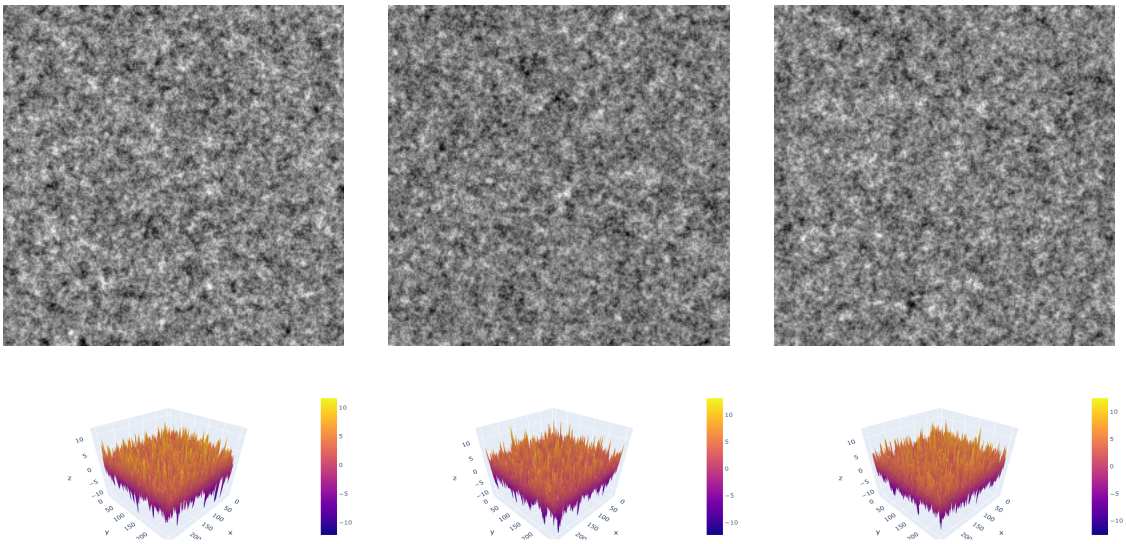

Figure 34: Real dataset nano-rough surface images for dimension $d = 256$.

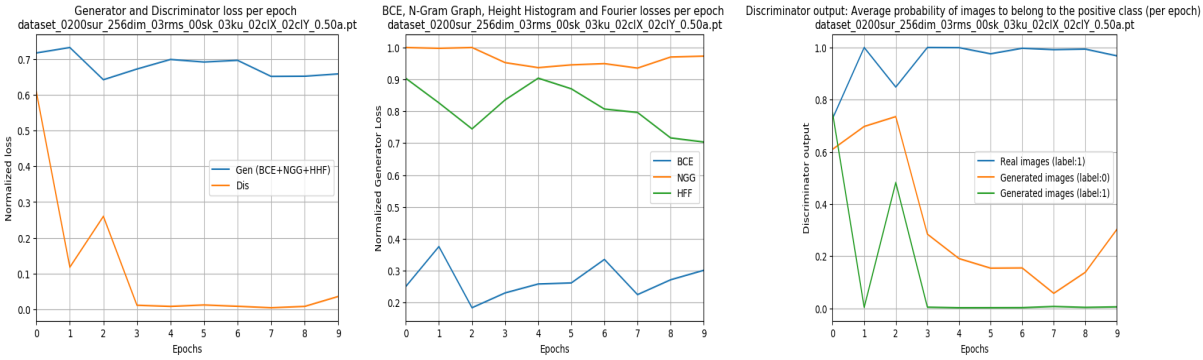

Figure 35: Graph plots for dimension $d = 256$.

Early findings, show that there is potential in processing surface images with dimension $d = 256$. However, we encountered computational limitations due to the vast number of parameters of the network. We were able to train our network only with $\boldsymbol{s = 50}$ surface samples for **10** epochs.
Therefore, we can't make conclusive statements as the obtained results are insufficient. We emphasize that this experiment requires further experimentation. It would be interesting to repeat the experiment on a high-performance computing system with $s = 1000$ samples for 100 epochs and examine the obtained results.

4.2 Results and discussion

Throughout this section we present our achieved results on 6 different nano-rough surface datasets. Each dataset has a different combination of roughness parameters. We summarize the cases as follows.

1. Dataset with $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 2, \xi_y = 2, \alpha = 0.5}$
2. Dataset with $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 2, \xi_y = 2, \alpha = 1.0}$
3. Dataset with $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 4, \xi_y = 4, \alpha = 0.5}$
4. Dataset with $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 4, \xi_y = 4, \alpha = 1.0}$
5. Dataset with $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 8, \xi_y = 8, \alpha = 0.5}$
6. Dataset with $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 8, \xi_y = 8, \alpha = 1.0}$

4.2.1 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 2, \xi_y = 2, \alpha = 0.5}$

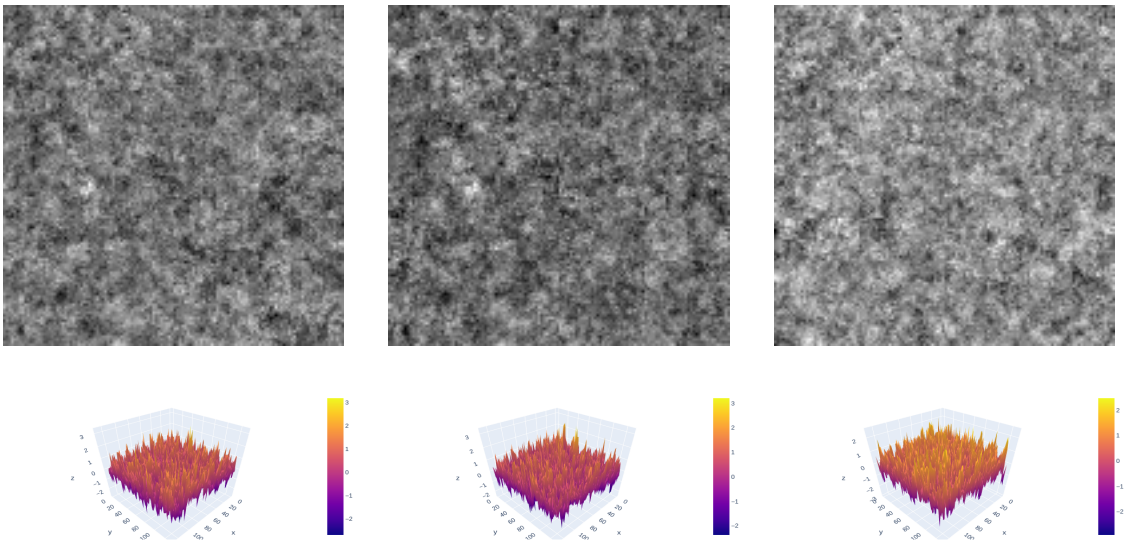

Figure 36: Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 0.5$.

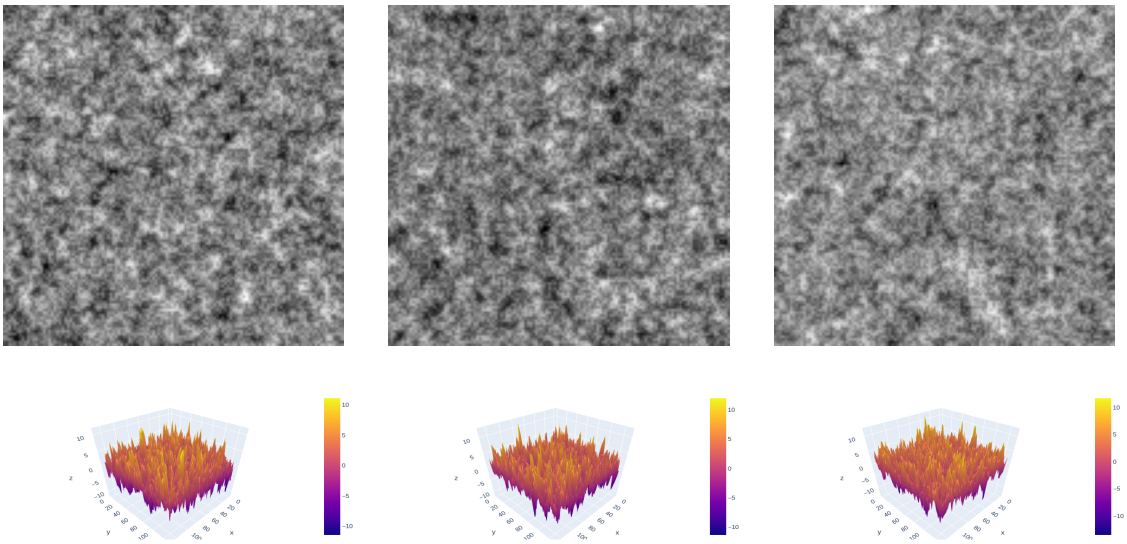

Figure 37: Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 0.5$.

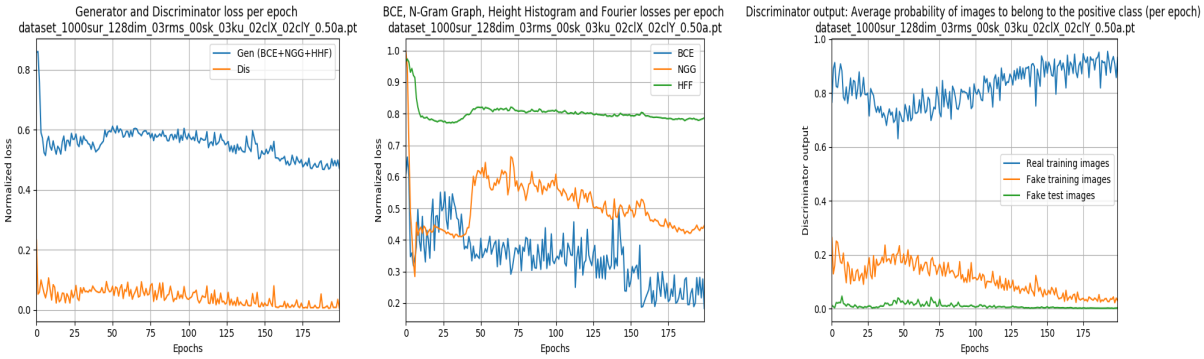

Figure 38: Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 0.5$.

After training our network with $s = 1000$ surface samples for 200 epochs the generator-produced surface images in figure 36 present substantial similarity to the real ones in figure 37. According to the domain expert, the frequency content as well as the local co-occurences of surface peaks of the real surface images is assimilated in the generator-produced ones. This occurs also in the next cases we review in following sections.

Taking a closer look at the Generator and Discriminator loss plot in figure 38 we observe that the discriminative model is trained better than the generative. This is also observable in the Discriminator output plot where it gradually trains better on both classes after epoch 50. In practice, this is something we noticed happening quite often throughout the course of our experimentation.

Height Histogram and Fourier similarity under-performs compared to the other similarity metrics. It reaches its rather optimal solution around epoch 30 and doesn't seem to offer much as the epochs progress. On the other hand, Binary Cross-Entropy and N-Gram Graph similarities present substantial progression after epochs 25 and 50 respectively.

Furthermore, it is noteworthy, that the checkerboard artifact is mildly observable in the generator-produced images. In their study, Andrew Aitken et al.[35] state that one of the reasons for the checkerboard artifact seems be the result of the transposed convolution overlap during the up-sampling process. Transposed-convolution overlap is when the kernel size is not fractionally-divided by the stride size. But even a fractionally-divided kernel by the stride can still lead to checkerboard artifacts according to the study. In our network architecture we incorporated a kernel that is fractionally-divided by the stride in every layer of the network for both models. We introduced this specific network architecture in section 3.2.2 as a means to reduce the checkerboard artifact.

4.2.2 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 2, \xi_y = 2, \alpha = 1.0}$

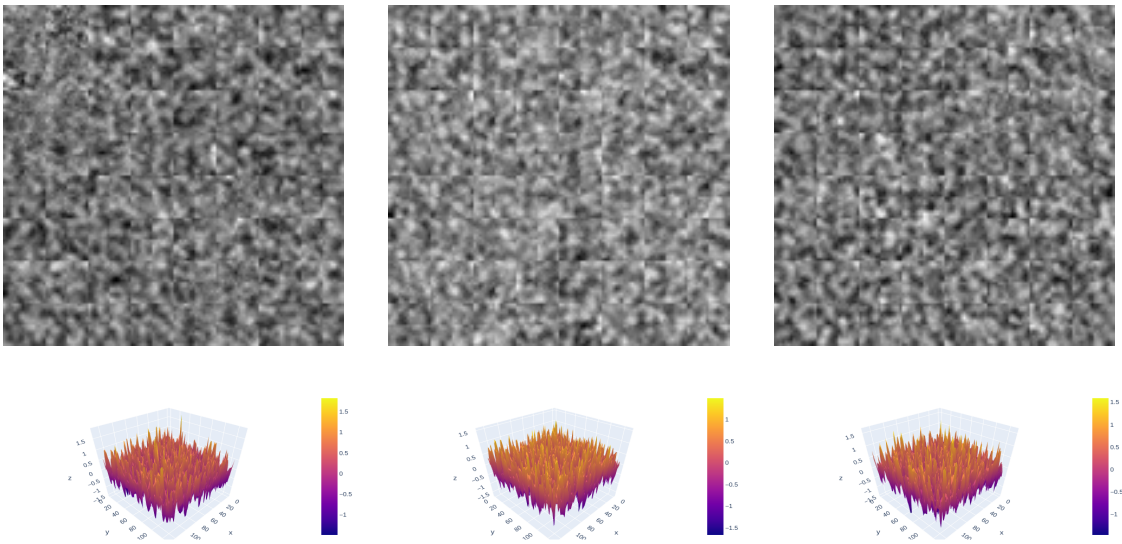

Figure 39: Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 1.0$.

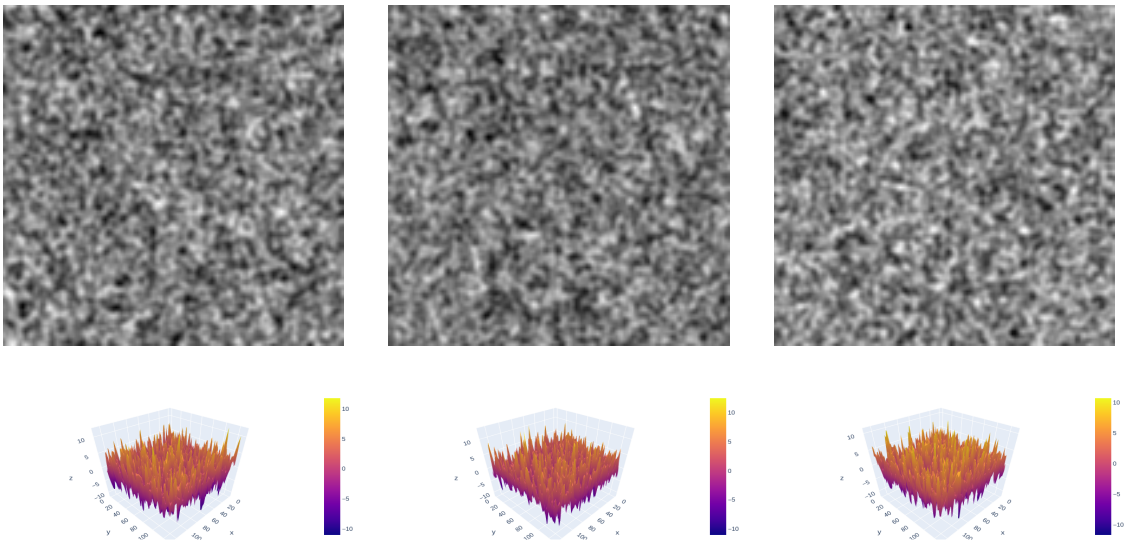

Figure 40: Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 1.0$.

Figure 41: Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 2, \xi_y = 2, \alpha = 1.0$.

For a higher Roughness exponent ($\alpha = 1.0$) the checkerboard artifact becomes visible as squared grid lines in figure 39. According to the domain expert, this is directly correlated to the higher Roughness exponent (α) value due to the fact that the surface profile is more locally-smoothed, therefore exposing the checkerboard artifact irregularity futher. In the previous case 4.2.1 for Roughness exponent ($\alpha = 0.5$) the checkerboard artifact was able to rather *hide* in the more locally-spiked surface profile according to the domain expert.
As for the similarity content, Binary Cross-Entropy reaches *optimality* at epoch 40 and does not contribute much after that. Notably, N-Gram Graph presents a substantial improvement in performance regarding to the previous case 4.2.1 but tends to gradually perform worse after epoch 60.
In a similar manner as previous case 4.2.1, Height Histogram and Fourier seems to under-perform compared to the other similarity metrics.

4.2.3 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 4, \xi_y = 4, \alpha = 0.5}$

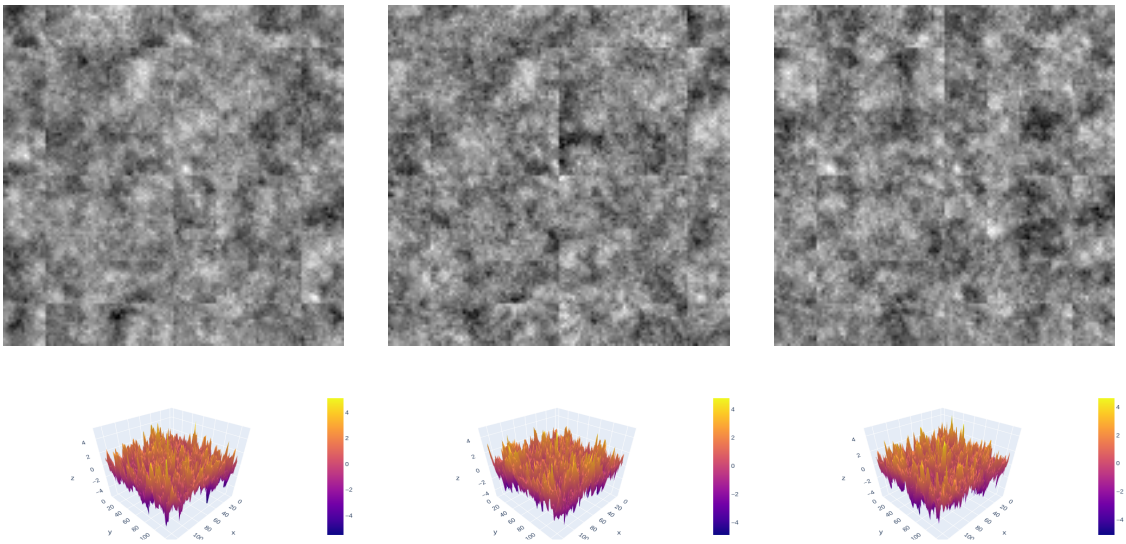

Figure 42: Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 0.5$.

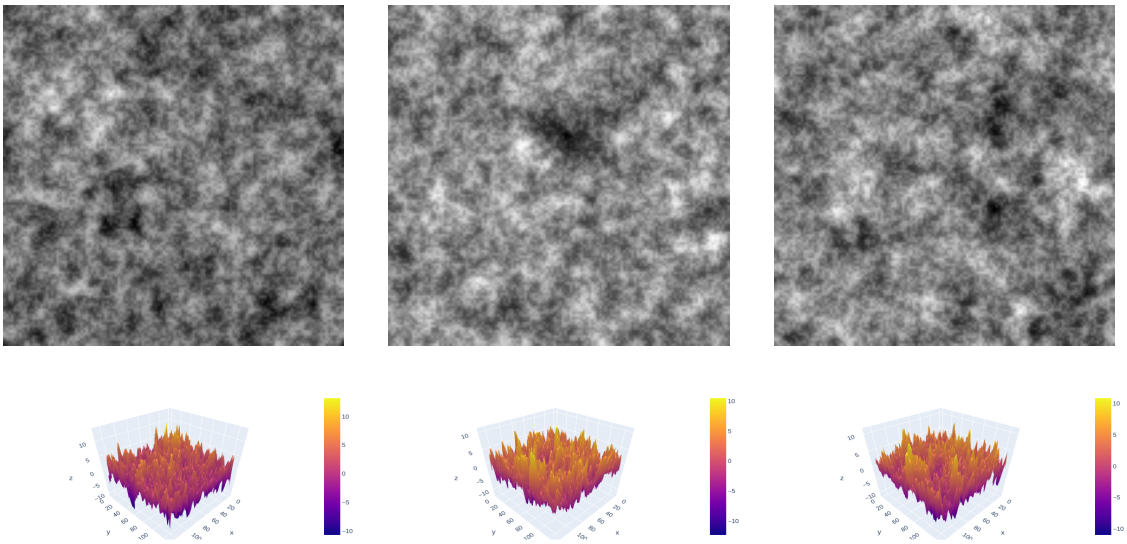

Figure 43: Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 0.5$.

Figure 44: Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 0.5$.

In this case, the checkerboard artifact appears to be less acute when compared to generated results for case 4.2.2 but more acute when compared to generated results for case 4.2.1. Furthermore, It appears to have slightly larger square magnitude in the grid when compared to previous cases. According to the domain expert this is correlated to the bigger $\xi_x = 4, \xi_y = 4$ values.
Additionally, Height Histogram and Fourier similarity presents a tremendously better performance when compared to cases 4.2.1 and 4.2.2. On the contrary, Binary Cross-Entropy performs relatively worse when compared to previous cases. Notably, there is a sudden and abrupt increase in performance occurring in epoch 50. N-Gram Graph performs in a similar manner when compared to previous cases.
The discriminative model performs worse in this experiment when observing the loss in figure 44 compared to previous cases in figures 38 and 41 respectively.

4.2.4 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 4, \xi_y = 4, \alpha = 1.0}$

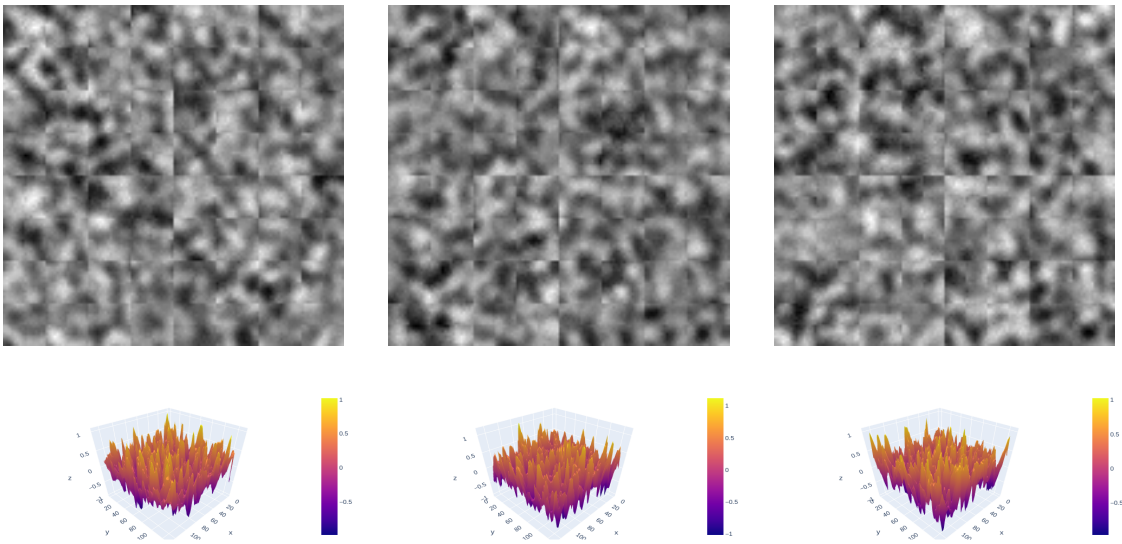

Figure 45: Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 1.0$.

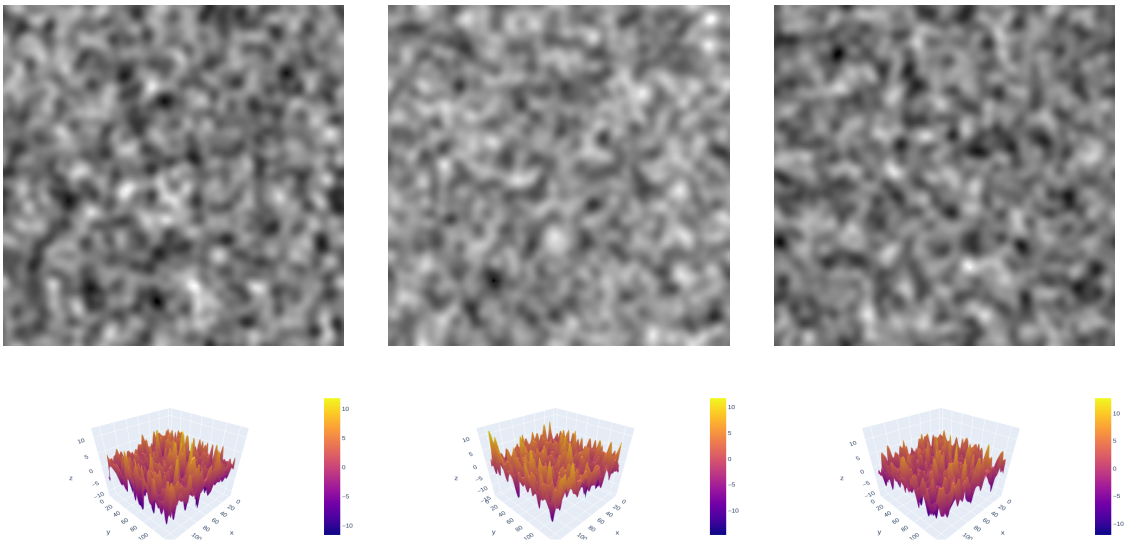

Figure 46: Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 1.0$.

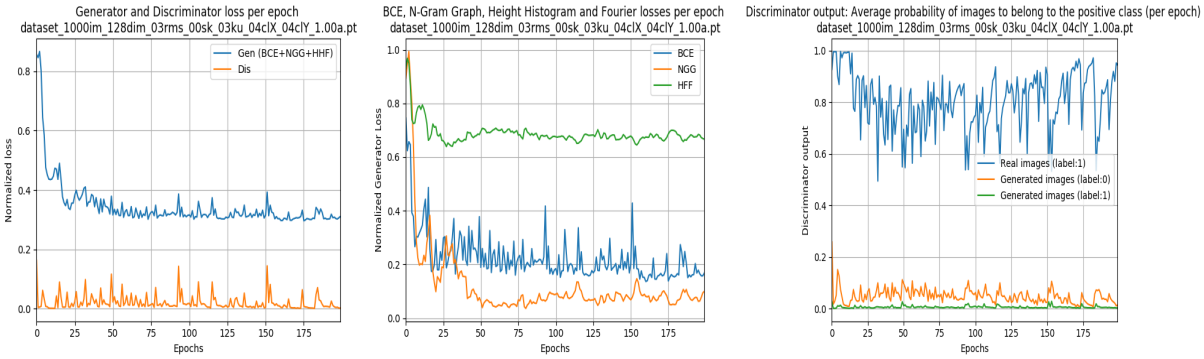

Figure 47: Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 4, \xi_y = 4, \alpha = 1.0$.

For correlation lengths $\xi_x = 4, \xi_y = 4$ and Roughness exponent $\alpha = 1.0$, the checkerboard artifact is acutely visible when observing the generated images in figure 45. As previously stated, there is a correlation to the checkerboard artifact harshness with higher α according to the domain expert.
As for the similarity metrics, the contribution of the N-Gram Graph content to the Generator in figure 47 is exceptional and outperforms the other similarity metrics. Also, it the best performance we have observed so far compared to itself in previous cases. Height Histogram and Fourier reaches optimality in epoch 25 and does not seem to offer more to learning after that. Binary Cross-Entropy also performs better in this case when compared to previous cases.
As for the discriminative model, it does perform well on the real training dataset something we can observe in the discriminator output plot in figure 47.

4.2.5 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 8, \xi_y = 8, \alpha = 0.5}$

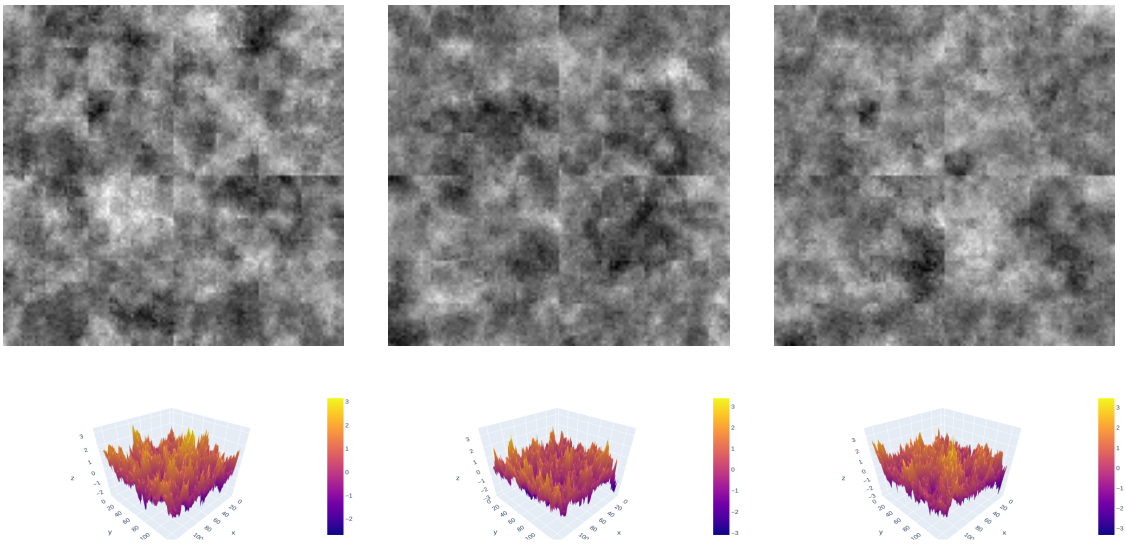

Figure 48: Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 0.5$.

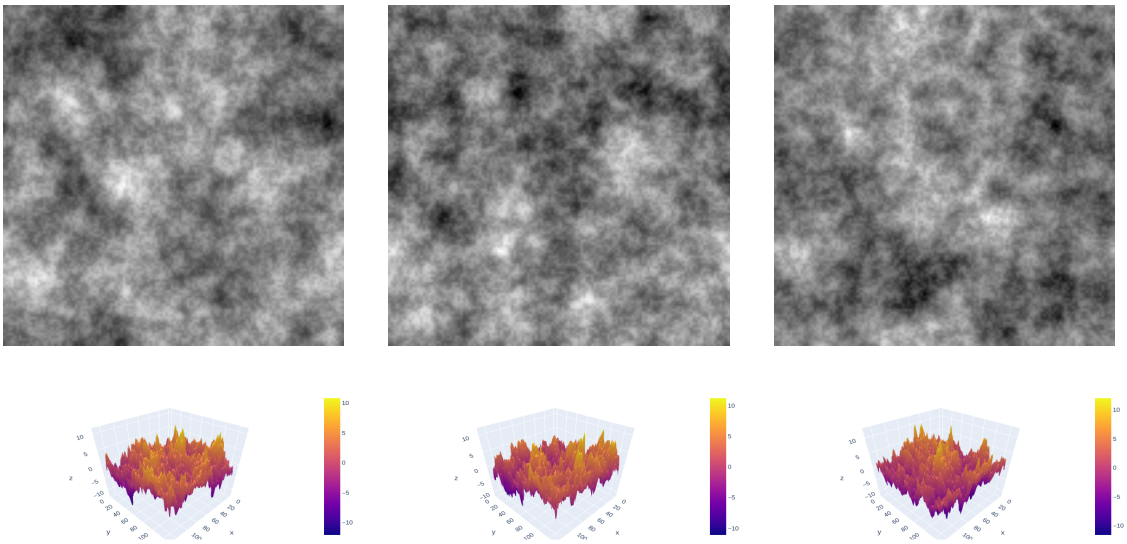

Figure 49: Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 0.5$.

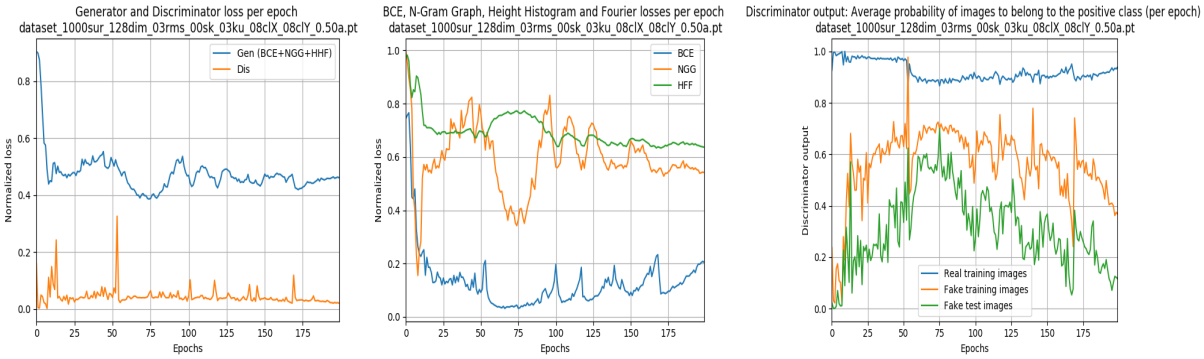

Figure 50: Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 0.5$.

After we observe the generated results in figure 48 we see that the checkerboard artifact still persists. But, it has milder granularity than cases 4.2.2 and 4.2.4 and harsher granularity than cases 4.2.1 and 4.2.3 respectively. The magnitude of the squares in the grid is larger due to the higher correlation lengths $\xi_x = 8, \xi_y = 8$, according to the domain expert.
In this case, we observe a higher adversarial process in the network when compared to previous cases. This statement occurs from the the fact that the Discriminator presents higher uncertainty on the generator-produced test minibatch, which provides margin for the Generator to perform better. If we observe the Discriminator output plot in figure 50 we see that there is gradual uncertainty on the generator-produced minibatch beginning at epoch ≈ 25, ending at epoch ≈ 175 and peaking in epoch 75. After epoch ≈ 175 the Discriminator progressively classifies better.
As for the similarity metrics, Height Histogram and Fourier presents a very slow and gradual progress with a characteristic mild relapse in epochs $\approx 55 - 95$. N-Gram Graph also presents a gradual progress during training with a characteristic, steep rise in performance in epochs $\approx 55 - 95$. In a similar manner, Binary Cross-Entropy gradually progresses during training, with a characteristic, steep rise in performance in epochs $\approx 55 - 95$. This characteristic and sudden increase in the Generator performance for epochs $\approx 55 - 95$ is supplemental to the Discriminator's higher uncertainty for the generator-produced test minibatch for epochs $\approx 55 - 95$ in figure 50.

4.2.6 Results for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \boldsymbol{\xi_x = 8, \xi_y = 8, \alpha = 1.0}$

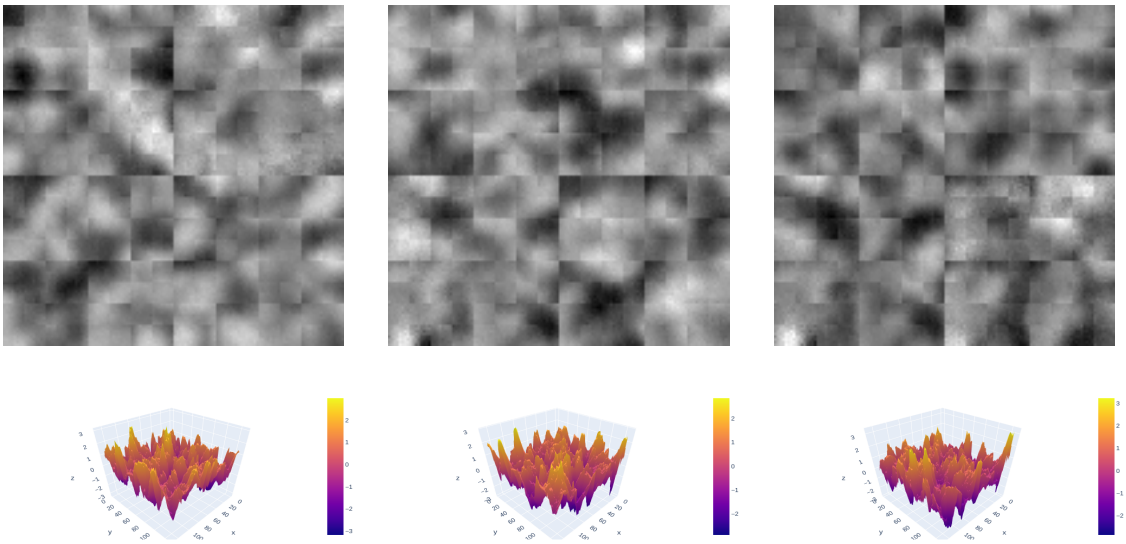

Figure 51: Generator-produced nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 1.0$.

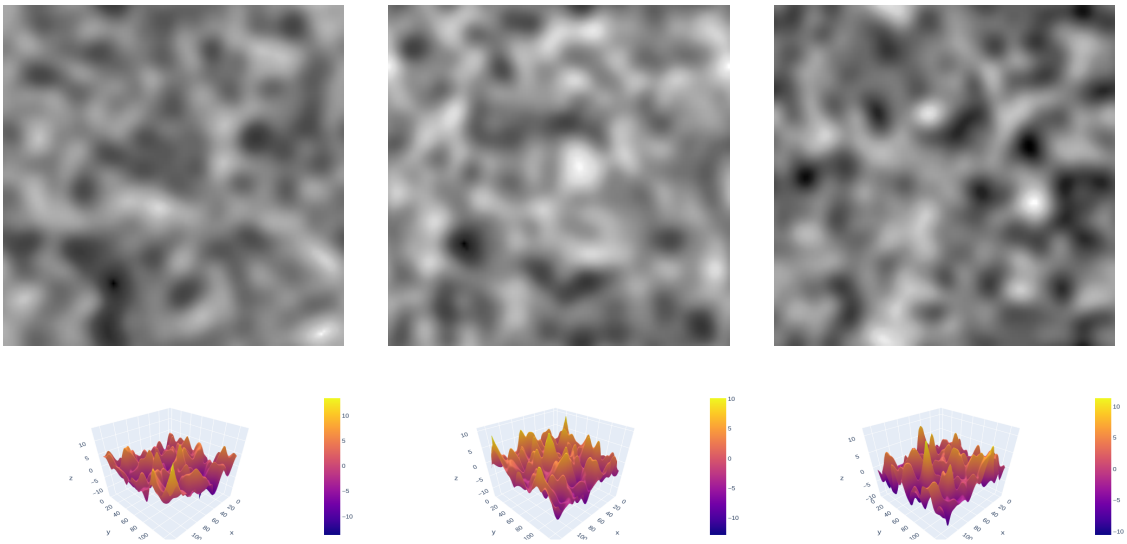

Figure 52: Real training nano-rough surface images for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 1.0$.

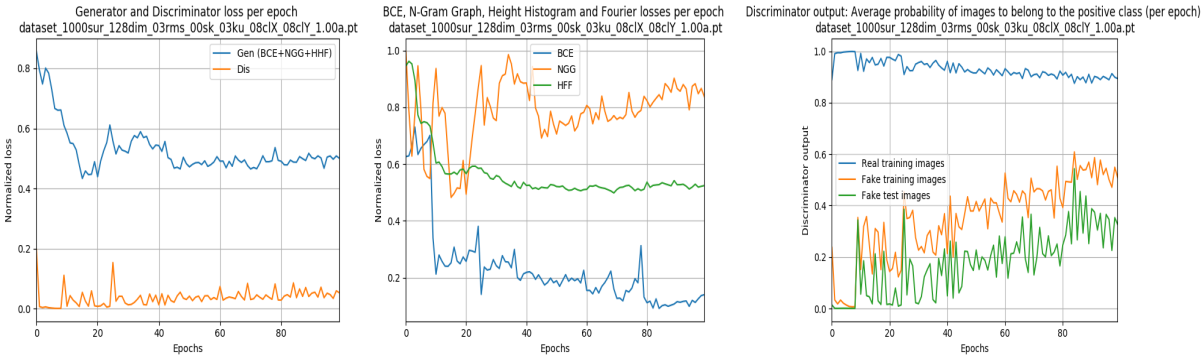

Figure 53: Graph plots for dataset with properties $s = 1000, d = 128, R_q = 3, R_{sk} = 0, R_{ku} = 3, \xi_x = 8, \xi_y = 8, \alpha = 1.0$.

In this case, we observe a harsh checkerboard artifact for the generated images in figure 51. It appears to be the harshest of the cases we have reviewed so far. The magnitude of the squares in the grid is similar to case 4.2.5 for correlation lengths $\xi_x = 8, \xi_y = 8$.
Again in this case, we observe a higher adversarial process in the network when compared to previous cases. But it is different than case 4.2.5. In the Discriminator output plot 53 we observe a gradual convergence of the generator-produced test minibatch to the desired value of 0.5. In case 4.2.5 the Discriminator output in plot 50 gradually diverges after ≈ 175 from the desired value of 0.5.
As for the similarity metrics, Binary Cross-Entropy outperforms the other ones. N-Gram Graph peaks at ≈ 20 and after that it gradually contributes less on learning. Height Histogram and Fourier performs in a gradual and slow manner.

4.3 Discussion

Analysis of the results lead to substantial conclusions.
A preliminary experiment was implemented with different initial network weights. However, due to the computationally demanding nature of the network, the experiment was implemented with insufficient number of surface samples $s = 300$ for a limited number of epochs $epochs = 50$. Therefore, we came to the conclusion that we have insufficient evidence to make a statement regarding initial network weights and we consider that the task requires extended experimentation.
Furthermore, we examined the effect of different N-gram graph parameters on learning as well as the overall progression of the N-gram graph loss itself. However, again we trained the network with insufficient number of surface samples $s = 300$ for a limited number of epochs $epochs = 50$. Further experimentation by training with sufficient surface samples for more epochs might yield more conclusive evidence

regarding effect of the N-Gram Graph parameters on the network. Though, we did notice a pattern that the N-Gram Graph performs better on datasets with Roughness exponent ($\alpha = 1.0$). This superior performance might be correlated to the fact that for Roughness exponent ($\alpha = 1.0$) we have a locally-smoothed surface profile.
Additionally, we implemented a preliminary test of our network on higher surface image dimension $d = 256$. However, we encountered computational power deficiency due to the vast number of parameters of the network. We were able to test only $s = 50$ surface samples for 10 epochs. We concluded that it would be of crucial interest to repeat the experiment on a high-performance computing environment with $s = 1000$ samples for 100 epochs and analyze the obtained results.
Moreover, we trained our network on datasets with various nano-rough surface parameter values and evaluated the results. We summarize the collectively obtained results.
We observed a checkerboard artifact in resulted generator-produced surface images that relates to the correlation lengths ξ_x, ξ_y and Roughness exponent α values. The higher the correlation lengths ξ_x, ξ_y the larger the magnitude of the squares of the checkerboard artifact in generator-produced surface images. Also, the higher the Roughness exponent α the harsher the checkerboard artifact irregularity.
According to the domain expert, the frequency content as well as local co-occurences of surface peaks is successfully assimilated from the real surface images to the generator-produced ones.
Furthermore, according to the domain expert's evaluation the generator-produced surface images in most cases are substantially improved in regard to the previous work we built upon[1]. Additionally, the checkerboard artifact is reduced in most cases.
This is the collective result of three crucial contributions in this work. We summarize those as follows.

1. Introduction of Height Histogram and Fourier similarity content to the learning procedure.

2. Multi-component loss optimization. Normalizing and weighting individual loss components contributed substantially to the performance of the network.

3. Introduction of a novel network architecture in an effort to reduce the checkerboard artifact.

Even though the checkerboard artifact is reduced, it still persisted in resulted surface images. We consider this a limitation in our work and we conclude that further investigation is required.

5 Conclusions

In this research, we studied how we can utilize Deep Convolutional Generative Adversarial Networks as a means to generate synthetic nano-rough surface images. We showed that we are able to model the structural morphology of nano-rough surfaces. Our contribution to this work is summarized as follows.

1. We introduced a new loss component to the Generator as a means to reflect the heights and frequency spectrum of nano-surface images. This substantially improved the similarity of the generator-produced nano-surface images with the real nano-surface images.

2. Furthermore, we proposed a novel architecture which resulted in the reduction of the checkerboard artifact observed in generator-produced nano-surface images.

3. Additionally, we optimized multi-component losses so that they equally contribute to the Generator's learning ability. The optimization considerably improved the performance of the network.

4. We extended the network functionality so that it produces higher surface image dimensions.

5. Finally, we contributed to technical implementations. We added an early stopping criterion and we extended the network to support incremental training through checkpoints. The technical implementations contribute to lessening the computational cost needed for training the network.

Throughout the course of the research there were a lot of questions and findings that came up. We consider it would be advantageous to investigate them in the future. We summarize these possible next steps as follows.

1. It would be crucial to investigate further the checkerboard artifact irregularity observed in generator-produced surface images.

2. It would be meaningful to incorporate explainability into the network. Since we generate synthetic data it is crucial to provide interpretable explanations that yield deeper insight on the adversarial network process.

3. It would be useful to investigate the use of a genetic algorithm for optimizing the network and compare the results with other optimization algorithms.

4. It would be crucial to research the discriminator weight update ratio during training and examine the impact on the adversarial process. It would be interesting to analyze the network performance when the discriminator weights are updated at a slower ratio than the generator's.
5. Further experimentation with different hyperparameter values might yield substantial information regarding the network performance.

References

[1] Vasilis Sioros, George Giannakopoulos, and Vassileios Constantoudis. "Generating Realistic Nanorough Surfaces Using an N-Gram-Graph Augmented Deep Convolutional Generative Adversarial Network". In: *Proceedings of the 12th Hellenic Conference on Artificial Intelligence*. SETN '22. Corfu, Greece: Association for Computing Machinery, 2022. ISBN: 9781450395977. DOI: 10.1145/3549737.3549794. URL: https://doi.org/10.1145/3549737.3549794.

[2] Liam Critchley. "Where Nanotechnology and Machine Learning Meet". In: (Apr. 2021). URL: https://www.mouser.com/blog/where-nanotechnology-machine-learning-meet.

[3] Athanasios Arapis. "Μέτρα Πολυπλοκότητας Νανοδομημένων Επιφανειών". MA thesis. National Technical University of Athens, 2020.

[4] E.S. Gadelmawla et al. "Roughness parameters". In: *Journal of Materials Processing Technology* 123.1 (2002), pp. 133–145. ISSN: 0924-0136. DOI: https://doi.org/10.1016/S0924-0136(02)00060-2. URL: https://www.sciencedirect.com/science/article/pii/S0924013602000602.

[5] Antonios Stellas. "Μηχανική μάθηση και Νανοτεχνολογία: Σύνδεση δομικών και λειτουργικών παραμέτρων νανοδομημένων επιφανειών με νανοτραχύτητα". MA thesis. National Technical University of Athens, 2018.

[6] Antonios Stellas, George Giannakopoulos, and Vassilios Constantoudis. "Hybridizing AI and Domain Knowledge in Nanotechnology: the Example of Surface Roughness Effects on Wetting Behavior". In: *SETN Workshops*. 2020.

[7] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. DOI: 10.48550/ARXIV.1511.06434. URL: https://arxiv.org/abs/1511.06434.

[8] Raj Bawa et al. "Protecting new ideas and inventions in nanomedicine with patents". In: *Nanomedicine: Nanotechnology, Biology and Medicine* 1.2 (2005), pp. 150–158. ISSN: 1549-9634. DOI: https://doi.org/10.1016/j.nano.2005.03.009. URL: https://www.sciencedirect.com/science/article/pii/S1549963405000675.

[9] Martin Prutton. *Introduction to surface physics*. Clarendon Press, Mar. 1994. ISBN: 9780198534761.

[10] David Lyon and Alfred Hubler. "Gap size dependence of the dielectric strength in nano vacuum gaps". In: *IEEE Transactions on Dielectrics and Electrical Insulation* 20.4 (2013), pp. 1467–1471. DOI: 10.1109/TDEI.2013.6571470.

[11] David J. Whitehouse. *Handbook of Surface and Nanometrology*. 1st ed. Taylor & Francis, Dec. 2002. ISBN: 9780429145544. DOI: https://doi.org/10.1201/9781420034196.

[12] D.C. Agrawal. *Introduction to Nanoscience and Nanomaterials*. World Scientific Publishing, 2013. ISBN: 9789814397971. URL: https://books.google.gr/books?id=Ntl2tgAACAAJ.

[13] ISO Central Secretary. *Geometrical product specifications (GPS) — Surface texture: Profile — Part 2: Terms, definitions and surface texture parameters*. en. Standard ISO 21920-2:2021. Geneva, CH: International Organization for Standardization, 2021. URL: https://www.iso.org/standard/72226.html.

[14] J. Paulo Davim. *Mechatronics and manufacturing engineering : research and development*. 1st ed. Woodhead Publishing, Jan. 2012. ISBN: 9780857095893.

[15] Xiyue Han and Alexander Schied. *The Hurst roughness exponent and its model-free estimation*. 2021. DOI: 10.48550/ARXIV.2111.10301. URL: https://arxiv.org/abs/2111.10301.

[16] Erik Brynjolfsson and Tom Mitchell. "What can machine learning do? Workforce implications". In: *Science* 358.6370 (2017), pp. 1530–1534. DOI: 10.1126/science.aap8062. URL: https://www.science.org/doi/abs/10.1126/science.aap8062.

[17] George Giannakopoulos. "Automatic Summarization from Multiple Documents". PhD thesis. University of the Aegean and NCSR Demokritos, June 2009.

[18] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2. URL: https://www.worldcat.org/oclc/61321007.

[19] John R. Koza et al. "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming". In: *Artificial Intelligence in Design '96*. Ed. by John S. Gero and Fay Sudweeks. Dordrecht: Springer Netherlands, 1996, pp. 151–170. ISBN: 978-94-009-0279-4. DOI: 10.1007/978-94-009-0279-4_9. URL: https://doi.org/10.1007/978-94-009-0279-4_9.

[20] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

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

[22] Yoshua Bengio. "Learning Deep Architectures for AI". In: 2.1 (Jan. 2009), pp. 1–127. ISSN: 1935-8237. DOI: 10.1561/2200000006. URL: https://doi.org/10.1561/2200000006.

[23] John McGonagle, George Shaikouski, and Christopher William. “Backpropagation”. In: (2023). URL: https://brilliant.org/wiki/backpropagation/.

[24] Ingrid Russell. “The delta rule”. In: *University of Hartford, West Hartford. Accessed* 5 (2012).

[25] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747. URL: https://arxiv.org/abs/1609.04747.

[26] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. DOI: 10.48550/ARXIV.1212.5701. URL: https://arxiv.org/abs/1212.5701.

[27] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[28] Ian J. Goodfellow et al. “Maxout Networks”. In: (2013). DOI: 10.48550/ARXIV.1302.4389. URL: https://arxiv.org/abs/1302.4389.

[29] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[30] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 105–114.

[31] Joe Eastwood et al. “Generation and categorisation of surface texture data using a modified progressively growing adversarial network”. In: *Precision Engineering* 74 (2022), pp. 1–11. ISSN: 0141-6359. DOI: https://doi.org/10.1016/j.precisioneng.2021.10.020. URL: https://www.sciencedirect.com/science/article/pii/S0141635921002658.

[32] Jin-Woong Lee et al. “Virtual microstructure design for steels using generative adversarial networks”. In: *Engineering Reports* 3.1 (2021), e12274. DOI: https://doi.org/10.1002/eng2.12274. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/eng2.12274. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/eng2.12274.

[33] Simen Ringdahl et al. “Machine Learning Based Prediction of Nanoscale Ice Adhesion on Rough Surfaces”. In: *Coatings* 11.1 (2021). ISSN: 2079-6412. DOI: 10.3390/coatings11010033. URL: https://www.mdpi.com/2079-6412/11/1/33.

[34] Francis Ogoke et al. *Deep-Learned Generators of Porosity Distributions Produced During Metal Additive Manufacturing*. 2022. DOI: 10.48550/ARXIV.2205.05794. URL: https://arxiv.org/abs/2205.05794.

[35] Andrew Aitken et al. *Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize*. 2017. DOI: 10.48550/ARXIV.1707.02937. URL: https://arxiv.org/abs/1707.02937.

[36] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. DOI: 10.48550/ARXIV.1606.03498. URL: https://arxiv.org/abs/1606.03498.