EVALUATION OF PROCEDURALLY GENERATED TERRAINS VIA ARTIFICIAL AND CONVOLUTIONAL NEURAL NETWORKS

GANI RAHMON

JULY 2018

EVALUATION OF PROCEDURALLY GENERATED TERRAINS VIA ARTIFICIAL AND CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF

NATURAL AND APPLIED SCIENCES OF

IZMIR UNIVERSITY OF ECONOMICS

BY

GANI RAHMON

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

JULY 2018

M.S. THESIS EXAMINATION RESULT FORM

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Abbas Kenan ÇİFTÇİ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Süleyman KONDAKÇI
Head of Department

We have read the thesis entitled **"Evaluation of Procedurally Generated Terrains via Artificial and Convolutional Neural Networks"** completed by **GANI RAHMON** under supervision of **Asst. Prof. Kaya OĞUZ** and **Asst. Prof. Mehmet TÜRKAN** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Mehmet TÜRKAN
Co-Supervisor

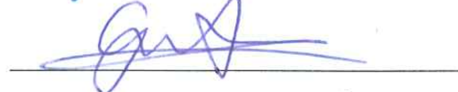
Asst. Prof. Kaya OĞUZ
Supervisor

Examining Committee Members

Date: 04.07.2018

Prof. Dr. Aybars UĞUR
Dept. of Computer Engineering, EGE

Asst. Prof. Gazihan ALANKUŞ
Dept. of Mechatronics Engineering, İUE



Asst. Prof. Kaya OĞUZ
Dept. of Computer Engineering, İUE

ABSTRACT

EVALUATION OF PROCEDURALLY GENERATED TERRAINS VIA ARTIFICIAL AND CONVOLUTIONAL NEURAL NETWORKS

GANI RAHMON
M.S. in Computer Engineering
Graduate School of Natural and Applied Sciences
Supervisor: Asst. Prof. Kaya OĞUZ
Co-Supervisor: Asst. Prof. Mehmet TÜRKAN
July 2018

An important factor to keep players interested in a gaming environment is the game content. However, the exponential increase of both gamer population and the production cost over the last decade caused the game content to meet new scalability challenges. To minimize costs related to the creation of content, procedural content generation techniques are used, which automates game content generation.

The significant part of the content for many games is a terrain. An interesting terrain will help to keep the player inside the game. There are many techniques used to generate terrain procedurally, but the main drawback of those techniques is that it is hard to control the generation process. Because of lack of control over generation process, it is hard to get the desired result that the user requires.

In this study, the procedurally generated terrains are evaluated using artificial and convolutional neural networks to meet the user requirements. In order to give a good evaluation result artificial and convolutional neural networks are trained using the real-world map data. Real-world map data are classified into three classes, such as sealevel, lowland, and mountain. The Diamond-Square Algorithm and Perlin Noise are used to procedurally generate terrains. The procedurally generated terrains are then evaluated using the ANN and CNN models until the desired class type is generated according to the user requirements.

Keywords: Procedural Content Generation, Procedural Terrain Generation, Neural Network, Convolutional Neural Network, TensorFlow.

ÖZ

YAPAY VE KONVOLÜSYONLU SİNİR AĞLARI YOLUYLA PROSEDÜREL ÜRETİLEN ARAZİLERİN DEĞERLENDİRİLMESİ

GANI RAHMON
Bilgisayar Mühendisliği, Yüksek Lisans
Fen Bilimleri Enstitüsü
Tez Danışmanı: Dr. Öğr. Üyesi Kaya OĞUZ
İkinci Tez Danışmanı: Dr. Öğr. Üyesi Mehmet TÜRKAN
Temmuz 2018

Oyun içeriği, oyuncuların oyun ortamlarında yer almasında önemli bir faktördür. Bununla birlikte, hem oyuncu nüfusunun hem de son on yılda üretim maliyetlerinin katlanarak artması nedeniyle yeni ölçeklenebilirlik zorluklarıyla karşılaşmaktadır. İçerik oluşturma ile ilgili maliyetleri en aza indirmek için oyun içerik üretimini otomatikleştiren prosedürel içerik oluşturma teknikleri kullanılacaktır.

Arazi, birçok oyun için içeriğin önemli bir parçasıdır ve ilginç bir arazi oyuncunun oyun içinde kalmasını sağlar. Araziyi prosedürel olarak üretmek için kullanılan birçok teknik vardır, ancak bu tekniklerin temel dezavantajı, üretim sürecini kontrol etmenin zor olmasıdır. Üretim süreci üzerinde kontrol eksikliği nedeniyle, kullanıcının istediği sonucu elde etmek zordur.

Bu çalışmada, prosedürel olarak üretilen araziler, kullanıcı gereksinimlerini karşılamak için yapay ve evrişimli sinir ağları kullanılarak değerlendirilmiştir. İyi bir değerlendirme sonucu elde etmek için yapay ve evrişimli sinir ağları gerçek dünya haritası verileri kullanılarak eğitilmiştir. Gerçek dünya haritası verileri deniz seviyesi, ova ve dağ gibi üç sınıfa ayrılır. Elmas-Kare Algoritması ve Perlin Gürültüsü, prosedürel olarak araziler oluşturmak için kullanılır. Prosedürle üretilen alanlar, kullanıcı gereksinimlerine göre istenen sınıf tipi üretilinceye kadar, ANN ve CNN modelleri kullanılarak değerlendirilmektedir.

Anahtar Kelimeler: Prosedürel İçerik Üretimi, Prosedürel Arazi Üretimi, Sinir Ağı, Konvolüsyonel Sinir Ağı, TensorFlow.

ACKNOWLEDGEMENT

I want to thank Asst. Prof. Kaya OĞUZ and Asst. Prof. Mehmet TÜRKAN for their help and support during my thesis study. With the help of them, I learned many new topics and I am very glad to have them as my instructors. I had a great time with them.

I am also thankful to my family, especially to my father and mother for their support during my master's study. It was hard for my mother, but I am very proud that she stayed strong all the time.

TABLE OF CONTENTS

Front Matter i

Abstract . . . iii

Öz . . . iv

Acknowledgement . . . v

Table of Contents . . . ix

List of Tables . . . x

1 Introduction **1**

2 Background **4**

2.1 Procedural Content Generation (PCG) . . . 4

2.2 Reasons to Use PCG . . . 5

2.3 Hierarchical Structure of PCG . . . 6

2.3.1 Game Space: Indoor and Outdoor Maps . . . 7

2.4 Games using PCG . . . 7

2.5 Procedural Terrain Generation (PTG) . . . 8

2.6 Terrain Representation 8

2.7 Terrain Generation 11

2.7.1 Fractal-based Generation 11

2.7.2 Physics-based Generation 18

2.7.3 Texture-based Generation 19

2.7.4 Comparison of Terrain Generation Techniques 20

2.8 Related Works 21

3 Methods **24**

3.1 Processing Real Terrain Images 24

3.1.1 Choice of Data Structure 24

3.1.2 Real Terrain Images Source 26

3.1.3 Classification of Real Terrain Data 28

3.1.4 Descriptive Statistics 28

3.1.5 Other Statistics 31

3.2 Artificial Neural Network (ANN) 33

3.2.1 Types of ANN 34

3.2.2 Activation Functions 35

3.2.3 Backpropagation 37

3.3 Convolutional Neural Network (CNN) 40

3.3.1 Understanding Convolution 40

3.3.2 Feature Map . . . 41
3.3.3 Introducing Non-Linearity (ReLU) . . . 42
3.3.4 Layers Used to Build CNNs . . . 43
3.3.5 Summing Up . . . 45
3.4 TensorFlow (TF) . . . 47
3.4.1 Parallelism in TensorFlow . . . 47
3.4.2 Architecture of a TensorFlow . . . 48
3.4.3 Image Representation as Tensors . . . 48
3.4.4 Tensors in TF . . . 49
3.4.5 Summing Up . . . 49
3.5 Procedural Terrain Generation Techniques . . . 50
3.5.1 Diamond-Square Algorithm . . . 50
3.5.2 Perlin Noise Algorithm . . . 50

4 Implementation **51**
4.1 Feature Extraction . . . 51
4.1.1 Statistical Data . . . 53
4.1.2 Slope Data . . . 53
4.1.3 Partitioning . . . 54
4.1.4 Histograms . . . 56
4.1.5 Results . . . 57

4.2 ANN Implementation 58
4.2.1 Different Feature Combinations for ANN 58
4.2.2 ANN Setup 58
4.3 CNN Implementation 62
4.3.1 Architecture of the CNN 62
4.3.2 Augmenting Data 63
4.3.3 Reading Inputs 64
4.3.4 CNN Layers 64
4.3.5 Predictions 66
4.3.6 Optimization and Accuracy 67
4.4 PTG Techniques Implementation 67
4.4.1 Diamond-Square Algorithm 68
4.4.2 Perlin Noise 69
4.4.3 Feature Extraction of Procedurally Generated Terrains . . 71
4.4.4 Evaluation of Procedurally Generated Terrains 73

5 Results & Discussion 74

5.1 Feature Combination for ANN 74
5.2 Convolutional Neural Network Test 80
5.3 Evaluation of Procedurally Generated Terrains 82

6 Conclusion & Future Work 86

LIST OF TABLES

2.1 Comparison between the noise functions [13]. 17

2.2 Comparison of terrain generation techniques [6]. 21

4.1 Number of features in each category. 57

4.2 Different combinations of the features. 58

4.3 Ranges for each class. 71

5.1 Feature combination network results using 10 hidden neurons. . . . 75

5.2 Feature combination network results using 20 hidden neurons. . . . 76

5.3 Feature combination network results using 30 hidden neurons. . . . 76

5.4 Feature combination network results using 40 hidden neurons. . . . 77

5.5 Results of the 10-folds cross-validation 78

5.6 CNN test image samples results 81

5.7 Sealevel generation using Diamond-Square Algorithm. 82

5.8 Lowland generation using Diamond-Square Algorithm. 84

5.9 Mountain generation using Perlin Noise 85

LIST OF FIGURES

2.1 Hierarchical structure of the taxonomy of PCG [3]. 6

2.2 2D height-map image (left). 3D render of height-map image (right). 9

2.3 TIN model representation. (a) Top-down. (b) Perspective view [6]. 10

2.4 Poisson Faulting on different iterations [6]. 12

2.5 General process of Diamond-Square Algorithm [8]. 14

3.1 Overall Structure and Flow of the Algorithm. 25

3.2 Height map of city Izmir. 26

3.3 Layout of terrain.party [20]. 27

3.4 A positive (left) and negative (right) skewed distribution [22]. . . . 29

3.5 A positive (left) and negative (right) kurtosis example [22]. 30

3.6 Histogram example of a height map. Generated using R. 32

3.7 First derivative with respect to x and y. 33

3.8 Single neuron computation [23]. 34

3.9 Types of neural network [24]. 35

3.10 Single hidden layer neural net backpropagation [25]. 37

3.11 Neural Network Learning Process Review. 39

3.12 Convolution operation process [26]. 41

3.13 Obtaining rectified feature map [26]. 42

3.14 A simple CNN Architecture. 45

4.1 Feature Extraction Algorithm. 52

4.2 Circular Ruggedness Flowchart. 55

4.3 Histogram with range between 0 - 9000 meters. 56

4.4 10-folds cross-validation example over the data samples. 61

4.5 K-fold cross-validation flowchart. 62

4.6 Architecture of our CNN. 63

4.7 Flowchart of Diamond-Square Algorithm. 68

4.8 Diamond-square height map example. 69

4.9 Perlin noise height map example. 70

5.1 Confusion Plot. 79

5.2 ROC Plot. 79

5.3 Color region representation. 83

5.4 2D height-map image (left). 3D render of height-map image (right). 83

5.5 2D height-map image (left). 3D render of height-map image (right). 84

5.6 2D height-map image (left). 3D render of height-map image (right). 85

Chapter 1

Introduction

Nowadays, computer games take an important place in our lives. Daily, millions of people throughout the world play computer games. They are amused with games such as League of Legends, StarCraft, FarmVille, and Minecraft. Game content from three-dimensional items up to difficult puzzles presents a significant amusement role for them.

According to the report named "Two-Thirds of American Households Regularly Play Video Games" done in 2017 by the US Entertainment Software Association (ESA) reveals that American families playing video games regularly include 65% of the population and 72% of those game players age are older than 18 years old. Moreover, the average of the game player age is 35 years old. Yet, the report "Essential Facts About the Computer and Video Game Industry" done in 2017 reports that American families having a device which is used to play video games indicates 67% of the population [1].

To keep a game player entertained while playing a computer game the quality of the game content should match the demands of the player community. In other words, an important factor to keep players interested in a gaming environment is game content. However, the exponential increase of both gamer population and the production cost over the last decade caused the game content to meet new

scalability challenges. The construction of a difficult game is not easy. It is a time-consuming work affecting the project costs and budget. To minimize costs related to the creation of content, Procedural Content Generation (PCG) techniques are used, which automates game content generation. The principal idea behind PCG is that game content is created by the use of computers performing a correctly implemented algorithm or procedure and not manually with the help of human designers.

The significant part of the content for many games is the terrain. An interesting terrain will help to keep the player immersed in a video game. Terrains can be generated in different ways, such as using design tools or generate them procedurally. Creating a good quality terrain using design tools requires too much effort and experience from the design artist. However, generating terrains procedurally will save a lot of time, but it has its own drawbacks.

There are many techniques used to generate terrain procedurally, but the main drawback of these techniques is that it is hard to control the generation process. Since procedural terrain generation techniques use pseudo-random number generators to simulate randomness that is observed in nature, each time the different terrain will be generated when these techniques are executed. Because of this randomness and lack of user control, it is hard to get the desired result that the user requires.

In recent years, the heuristic approaches became a popular topic for the researchers and many papers are published every year related to this topic. Since the heuristic approaches become popular, new methods using these approaches are defined for procedural terrain generation, which gives more control to the user on the generation process of the terrain.

Moreover, another field of computer science which mimics the human brain function, known as Artificial Neural Network (ANN), is now a trending topic for many applications. The human brain can recognize the human faces, road signs and objects easily, but it is a difficult process for the computers. Because of it, the Artificial Neural Network (ANN) is applied to interpret the images, road signs,

and human faces. The new type of neural network, known as Convolutional Neural Network (CNN) become popular in recent years, which has a great influence in an image recognition.

Having such great tools is there any possibility to get the realism that we are accustomed to as an outcome of procedurally generated terrain.

The results of a procedurally generated terrain cannot be defined as good or bad; although they may contain certain parameters and features, they may not give us the realism that we are accustomed to. Therefore, the main purpose of this study is to evaluate a procedurally generated terrain with artificial and convolutional neural network so that the produced terrain is close to the desired one. In order to give a good evaluation result artificial and convolutional neural networks need to be trained using the real-world map data. The key requirements to promote this purpose is as follows:

- Finding the real-world maps data having a height-map structure.
- Analyzing the existing techniques for Procedural Terrain Generation and selecting the ones suitable for this study.
- Processing the height-map images derived from the real-world data and extracting features from them.
- Training the Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) for evaluation of a procedurally generated terrains.

The main contribution of this study is the introduction of a new approach for getting the desired output terrain from a procedural terrain generation techniques by evaluating the outcomes with artificial and convolutional neural networks since the user has a lack of control over the generation process.

The structure of the study is represented by the following chapters. In Chapter 2 procedural content and terrain generation is discussed. Chapter 3 provides methods used in this study. Implementation of methods is described in Chapter 4. Chapter 5 indicates the results and discussions. The conclusion of the thesis and suggestions for the future work are presented in Chapter 6.

Chapter 2

Background

This chapter provides a number of points that assist as essential background material to understand the procedural terrain generation methods. First, the procedural content generation is explained. Afterwards, the procedural terrain generation is discussed and common representations of the terrain data are defined. Then, a variety of methods for producing terrain is discussed and this is followed by describing important procedural terrain generation techniques. Finally, the discussion of similar works concludes this chapter.

2.1 Procedural Content Generation (PCG)

In order to understand PCG, the key term should be well understood. The key term is "content" and the content can be identified as an object, character, maps and etc. in video games. The well-defined algorithm or a procedure, which are used to generate something, is defined by the terms "procedural" and "generation". The PCG indicates an algorithmic creation of game content such as maps, characters, or weapons with limited or no human participation [2].

As mentioned before the principal idea behind PCG is that a game content is created by the use of computers performing a correctly implemented algorithm

or procedure and not manually with the help of human designers. In order to have control of the design process, the human designers should be able to modify the parameters of the method to influence the final product.

2.2 Reasons to Use PCG

There are a number of different reasons for using PCG in video games. Perhaps the most obvious reasons are to reduce the content generation time and to simplify the work of a human designer.

In order to make a good quality game that will stand out in the application market, hundreds of people consisting of designers, audio engineer, and programmers need to work on it. Many of them are needed to work on creating game content. However, with the usage of PCG in games, small companies, which can't afford many people to work for them, with a small group of six or seven game developers are able to create content-rich games that can rival with large companies in the market.

Allowing entirely new kinds of games is another reason to use PCG. There is no reason why the games need to end if there is a software which is able to create game content at the rate the game is played.

Customization of the game content to the demands and tastes of the game player is another reason for using PCG in video games. The neural network model can be applied to understand the player's preferences in the game and to generate player-adaptive contents in the game that will maximize the enjoyment of the player playing the game.

Finally, many procedural algorithms randomly generate content and sometimes the new content can be created which a human might not think of.

2.3 Hierarchical Structure of PCG

M. Hendrikx et al.[3] proposes a hierarchical structure which contains the different possible varieties of content that can be generated procedurally in which indicates several layers, where lower level composed by what he calls Game Bits, that consists of basic features such as vegetation and textures, which may or may not be used by upper layers to form elements that result in final form. Figure 2.1 illustrates this proposed taxonomy.

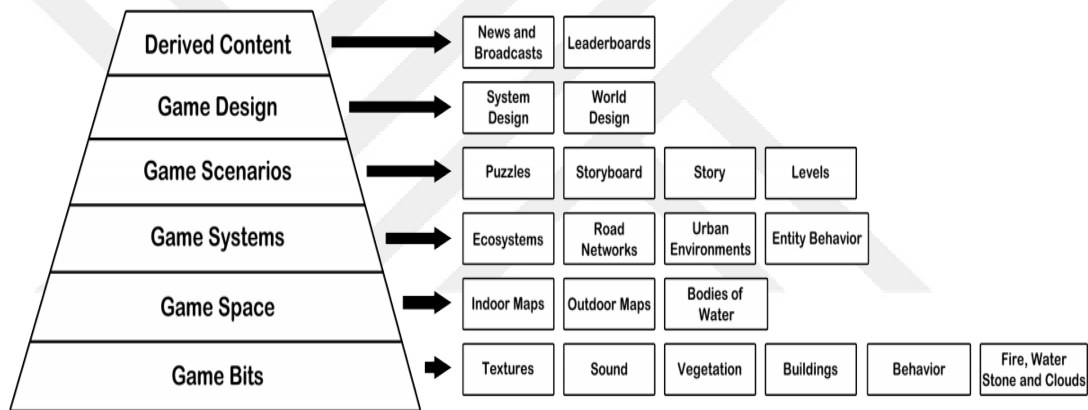


Figure 2.1: Hierarchical structure of the taxonomy of PCG [3].

The fundamental factors of game content, that basically don't occupy user if regarded separately, are known as game bits. The place where a game is played is known as the game space. Moreover, the game space is crowded with game bits between which player travel. To make games convincing and attractive the game systems are used. The game scenarios basically explain the process and form in which game stories discover. The design of a game consists of content like goals, such as what is player trying to succeed, and rules, such as what can be done in the game. A derived content can help the player to dive deep into the game environment, such as many players record their game skills for inspection outside or inside of the game.

2.3.1 Game Space: Indoor and Outdoor Maps

Indoor and outdoor maps are not similar. It is very important to understand the difference between them, because later in this study we will dive deep into outdoor maps generation. The composition and a corresponding location of an indoor area divided into the rooms is a description of indoor maps. Different layers can be connected using stairs, rooms are connected using corridors, and all can be merged to make dungeons. Caves are a different kind of indoor maps.

Outdoor maps are descriptions of the structure and elevation of the outdoor terrain. Many games which have outdoor maps have also an indoor map. The change among indoor and outdoor maps is done separately, because of the major technological diversity in the illustration and rendering among them. For instance, the popular game World of Warcraft has extensive outdoor maps and various indoor maps and the passage among those maps are done with the help of specific entrance regions and teleportation gates. Many successful games on the market like World of Super Mario, perform a great use of outdoor maps and some other games use a mix of indoor and outdoor maps.

2.4 Games using PCG

Procedural content has been used in games from the early eighties. One of the first games that use procedural content generation techniques is the game Rogue, a dungeon-crawling game where the player controls an adventurer through dungeons, which are generated dynamically by an algorithm of procedural generation every time the new game starts. The Diablo series was inspired by the Rogue. Placement of items and monsters and generation of maps are the procedural content generation used in a video game Diablo. A turn-based strategy game Civilization IV provides a novel game-play experience using a random map generation. The generation of the complete world and its content is done using PCG procedures in a very popular game Minecraft. A roguelike-platform game Spelunky uses a PCG to automatically generate different of game levels [2].

Other games that use PCG for games techniques is Left4Dead. According to the computer-analyzed anxiety level of the player, the scenarios are produced by forming enemy fights dynamically in Left4Dead.

2.5 Procedural Terrain Generation (PTG)

The terrain is the most fundamental feature of many games, which the player needs to explore. An interesting terrain will help to keep the player immersed in a game. Producing a game terrain for a player to discover needs a significant amount of resources and time which may lead to increase in a cost of development. PTG can help to reduce much of the work with the help of using pseudo-random algorithms.

Terrain and texture can be generated with the help of procedures that tends to create noise. The outcome of the created noise may look like random, but it is the outcome of a pseudo-random set of operations known as a noise function. The noise which is created with the help of these operations can be useful because it is not totally random instead it has a structure.

2.6 Terrain Representation

An interesting terrain should be used to keep player excited to discover different areas and spend more time playing. Because of it, an important decision needs to be taken to represent a terrain. Selecting data structure to represent terrain will influence the number of accessible tools in generating terrains. Also, limitation in some kinds of terrain features is possible with different representation.

The terrain representation that is most widely used nowadays is probably the height-map because it is the simplest representation of terrain and has a two-dimensional grid-based data structure. Because of their uniform grid-based nature, height-maps are easy to use. To express mathematically, a height-map

is a scalar function of two variables, where x and y are coordinate points for determining the location of an elevation value h as shown in the equation below [16].

$$h = f(x, y) \tag{2.1}$$

Figure 2.2 demonstrated an example of height-map described as a two-dimensional image on the left and the matching three-dimensional rendering on the right. The intensity of pixels describes the elevation data of the terrain, which is saved as a grayscale image with a single channel. Moreover, the intensity of the elevation data in a grayscale image can be interpreted as black being the lowest and white being the highest point. The grayscale image gives an advantage to height-maps when processing an image in order to compress, adjust or analyze terrain models. Such as implementing a Gaussian filter to make the rocky terrain look smoother.

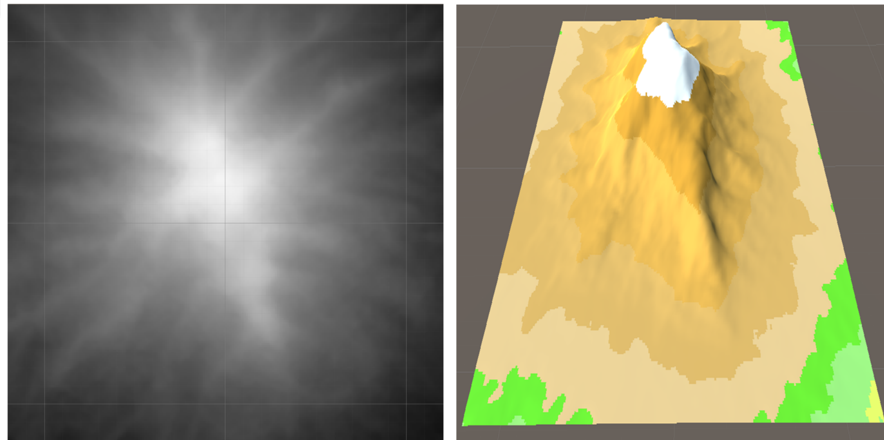


Figure 2.2: 2D height-map image (left). 3D render of height-map image (right).

Height-maps can be encoded by applying a variable number of bits. The format will identify the number of bits used in an height-map image. Such as single channel with 8-bit will allow only 256 potential height values and representing a very complicated terrain will be harder with those number of bits. Most Digital Elevation Model (DEM), which refers to a height-map digital form, files are saved by applying 16-bit images, resulting in 65,536 height values. Another advantage of height fields is that many portions of the real-world terrain data are available in DEM form.

Another alternative for representing terrain is the voxel grid. Voxels describe a value on a regular grid in a 3D space. Moreover, voxels are able to save data like opacity and color. The main advantage that voxels have is that they are capable of generating terrain with overhangs and caves. However, they have also disadvantages and the main disadvantage is that voxel grids have a large memory and storage overhead. One of the popular examples of a voxel-based environment is a game Minecraft.

The paper presented by Gerrit Greff [4] use the voxels to for terrain representation. Gerrit defines "an interactive terrain design system, which enables the user to generate localized, specific terrain features, as well as generalized global characteristics."

The common style of managing terrain is to describe its surface being a random mesh of two-dimensional primitives embedded in the three-dimensional space. A triangular irregular network (TIN) is a kind of mesh structure in which terrain is formed from a collection of coupled, different sized triangles. To produce a proper representation of terrain, the vertices of the triangles are accurately determined, usually with a Delaunay triangulation algorithm [5]. TINs are capable of capturing 3D structures such as caves and maintain a level-of-detail (LOD) system. For instance, less detailed areas represented with few larger triangle, while higher density areas are represented with few larger triangles. The storage overhead for TINs is small, because of the LOD system. The Figure 2.3 taken from the paperwork of Justin Crause [6] demonstrates an example of TIN model.

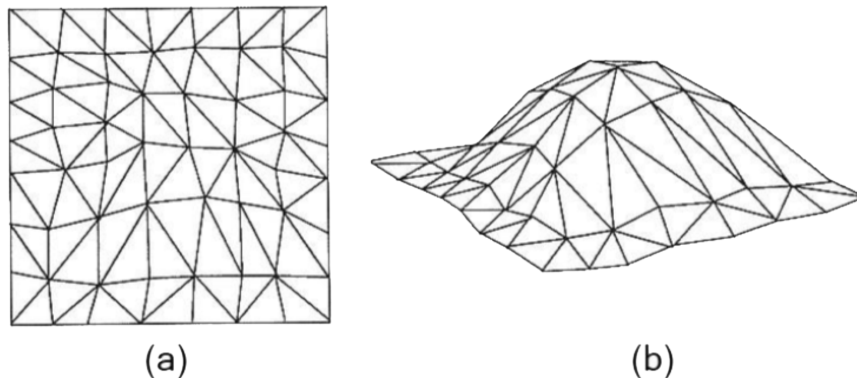


Figure 2.3: TIN model representation. (a) Top-down. (b) Perspective view [6].

One of the advantages of using this method is that many computer modeling and animation tools maintain this model. However, due to their non-uniform structure, the process of automatically generating using procedural methods is not clear.

Height-maps are the widely used terrain representation and supported by common terrain generation packages, such as Terragen. Also, for height-map images implementing image processing procedures are easy. Moreover, one of the main reason to use height-map in our study is that real landscape data produced from the satellites is stored in this format.

2.7 Terrain Generation

The procedural terrain generation methods can be classified into three categories: Fractal-based, Physics-based, and Texture-based. Shortly, to simulate the characteristics of a real terrain fractal surface is produced using a stochastic procedure. The erosion procedures are used to the terrain surface in order to increase the level of realism of a terrain and these procedures are known as physical simulations. Lastly, texture-based techniques get procedures from texture synthesis and basically replicate data of a source image to create a novel terrain. There are some programs, such as Terragen, which are using procedural methods for generating terrains quickly.

2.7.1 Fractal-based Generation

Benoit Mandelbrot was the first to use the term fractal in his book named "The Fractal Geometry of Nature" in 1982. Mandelbrot noticed that natural forms often include self-similar patterns and zoomed regions are statistically alike to the original form. This leads him to the introduction of a fractal geometry [6].

Two key properties represent fractals: self-similarity and chaotic. Dividing a fractal into a miniature variant of itself is referred as self-similarity. Due to the fractals infinite complexity, they are described as chaotic.

The term 'fractal-based' is used because not all techniques are really fractal. Methods that create terrain which displays self-similar patterns despite the fact that the algorithm is not mathematically fractal are classified with this term. The method of presenting those self-similar patterns are defined as Fractional Brownian motion (fBm). A series of iterations of a stochastic algorithm is involved in order to achieve fBm.

The earliest form of fractal terrain generation is known as Poisson Faulting. This method includes making use of a sequence of Gaussian random displacements or faults to a plane. To explain more simply, a line is selected throughout the plane and one aspect replaced with an arbitrary height value and this value is decreased after every fault to keep away from sudden height adjustments within the very last ensuring terrain. Example of the faulting procedure, which are captured at different synthesis states are demonstrated in the Figure 2.4.

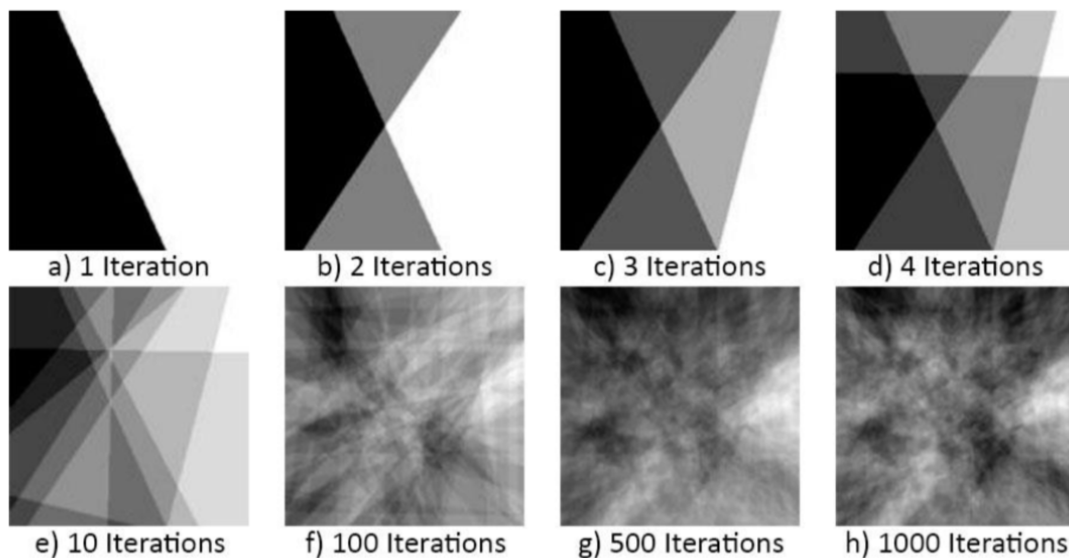


Figure 2.4: Poisson Faulting on different iterations [6].

2.7.1.1 Noise Function

Noise algorithm is a fundamental part of many procedurally generated terrain algorithms. Noise appearance may completely look random and distinctive, but it is usually the results of a pseudo-random collection of methods referred to as a noise function.

One of the important notation is that a lot of noise functions aren't naturally fractal. Some algorithms like Perlin Noise and Simplex Noise are alternatively used in conjunction with fBm to generate fractal pictures. To achieve fBm, several octaves of noise is generated, each of them is produced with an increased frequency and decreased amplitude and afterward, these octaves are summed together. However, the results are more detailed noise. There are many different algorithms for noise generation and the popular one will be discussed in the next section.

2.7.1.2 Noise Generation Techniques

There are many different techniques to generate noise and the following techniques are the most popular ones:

- Diamond-Square Algorithm
- Value Noise
- Perlin Noise
- Simplex Noise
- Worley Noise

2.7.1.2.1 Diamond-Square Algorithm

One of the methods to generate a height-map is a diamond-square algorithm. This method was originally proposed by Fournier, Fussell, and Carpenter at 1982

[7] and it is an improvement to the midpoint displacement algorithm. The Figure 2.5 demonstrates the general process of the diamond-square algorithm.

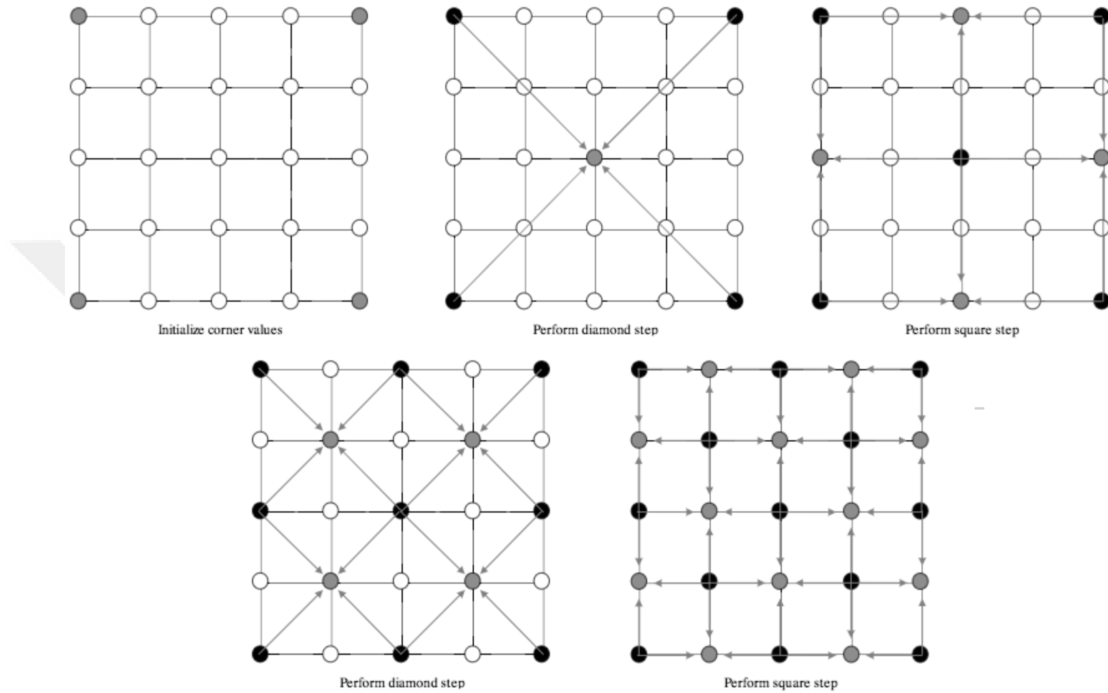


Figure 2.5: General process of Diamond-Square Algorithm [8].

Basically, the diamond-square algorithm starts with a 2D square array and initial values of the four corners points are set. Then the diamond and square steps are done in a loop with a condition that all values of the array are set. In the diamond step, the center point of a square in the array is found by taking the average of the four points located in the corners and adding a random value. This process is done for every square in the array. In the square step, the center point of a diamond in the array is found by taking the average of the four points located in the corners and adding a random value. This process is done for every diamond in the array.

The size of a random value should be decreased with every iteration. When the square steps are performed sometimes the edge points of the array will have only three adjacent values initialized instead of four. To handle such kind of a situation there are many different methods, but the simplest one is to take the average of those three adjacent values.

2.7.1.2.2 Value Noise

Value Noise is not naturally fractal like Midpoint Displacement. The basic characteristic of the fractal image is self-similarity, where zooming an image will give the same result as the large one, but Value Noise does not have this property. As mentioned earlier a function called Fractional Brownian Motion (fBm) is used in conjunction with Value Noise.

One of the advantages of the Value Noise is that it is not difficult to implement. An arbitrary grid is set over the plane. A random value is given for every grid-factor. The cost is an interpolation among the nearest grid-factors for each pixel in-between those grid-factors, such as in 1D, among the two nearest grid-factors and in 2D, among the closest four grid-factors. In order to locate the nearest grid-factors, the location variables should be rounded up and down. While this is trivial, the interpolation function should be selected carefully [9].

Cosine, cubic and linear are three common interpolation functions. Linear interpolation, known as LERP, not only used in the noise production. It offers quick results, however, creates a rough line. Cosine interpolation is barely slower but offers rounded outcomes. Cubic interpolation is extraordinarily slow, however, offers best outcomes. Choosing between them is mostly depend on the application. Linear is needed for real-time noise map generation. However, if time is not the main factor, usage of cubic interpolation will give good results [9].

2.7.1.2.3 Perlin Noise

Perlin Noise was developed by Ken Perlin in 1983 and was formally described in 1985 on a paper named “An Image Synthesizer” [10]. It is a very effective algorithm that is used often in PCG. Perlin noise can be used for any type of rough, wave-like texture or material in game development, such as fire effects, clouds, water and procedural terrain generation.

Perlin Noise can be described for any dimensions, but most commonly used in 2 or 3-dimensional function. Implementation of Perlin Noise normally involves 3

steps, which are defining a grid with random gradient vector, computation of the dot product among the distance-gradient vectors and linearly interpolate among those values.

The complexity of Perlin Noise is $O(2^n)$ for n representing dimensions because, for every assessment of the noise function, at every node of the including grid cell the dot product of the gradient vectors and placement have to be evaluated.

2.7.1.2.4 Simplex Noise

Simplex Noise was developed by Ken Perlin in 2001 [11]. The main reason of Simplex Noise is to overcome the weaknesses of Perlin Noise.

One of the main advantages of a Simplex Noise over Perlin Noise is that it has a lower computational cost and needs little multiplications. The complexity of Simplex Noise is $O(n^2)$ for n representing the dimensions.

Using hypercubes as the grid is the main weak point of a Perlin noise. In two dimension it will be square and in three dimensions it will become a cube. Moreover, for every used pixel this equates to 2^n corners for each in a given dimension.

Simplex noise solves the main weakness of the Perlin noise by using simplices rather than hypercubes. The form with a little corner in a dimension is known as simplex. Such as a triangle in 2D, a pyramid in 3D, and a shape with n+1 corners in n dimensions.

Implementation of Simplex Noise normally involves 4 steps:

- Coordinate skewing
- Simplicial subdivision
- Gradient choice
- Kernel summation

2.7.1.2.5 Worley Noise

Worley Noise is a noise function introduced in 1996 by Steven Worley [12]. It is basically used to generate procedural textures, such as the texture of stone or water. However, it is not very useful for terrain generation when used alone.

The simple concept of a Worley Noise is to take random factors in the surface after for each factor in surface take the distance to the nth-closest point as some kind of color data. To be more precise:

- Scatter random points onto a surface
- Noise $F_n(x)$ is a distance to nth-closest point to x

2.7.1.3 Comparison of Noise Generation Techniques

The Table 2.1, taken from the paper of Thomas J. Rose and Anastasios G. Bakaoukas [13] demonstrates the comparison between the noise functions mentioned in a previous section.

Table 2.1: Comparison between the noise functions [13].

Algorithm	**Speed**	**Quality**	**Memory Requirements**
Diamond-Square Algorithm	Very Fast	Moderate	High
Value Noise	Slow - Fast*	Low - Moderate*	Very Low
Perlin Noise	Moderate	High	Low
Simplex Noise	Moderate**	Very High	Low
Worley Noise	Variable	Unique	Variable
* Depends on what interpolation function is used.			
**Scales better into the higher dimensions than Perlin Noise			

The comparison is done based on a standard: speed, memory requirement, and quality. The speed suggests what number of photos may be generated in one second on the test hardware. Quality is mostly subjective and depends on the application.

Objective evaluation is needed to determine the quality of the procedurally generated terrains and this is the main focus of our study.

In conclusion, it can be understood from the table above that if speed is essential and memory is not an issue than Diamond-Square Algorithm is the best option to choose. However, the best algorithm among all is Simplex Noise, but it is difficult to understand and debug. In many of the occasions, the classic Perlin Noise will probably be satisfactory.

The main reason for choosing Perlin Noise instead of Simplex Noise in our study is that Simplex Noise is difficult to understand and debug. Since 2D and 3D is a point of interest in our study, Perlin and Simplex Noise have almost the same performance in these dimensions.

2.7.2 Physics-based Generation

The noise itself can generate exciting terrains, however, erosion can assist by adding an extra layer of realism to the generated terrains. Physics-based generation techniques are used to enhance the realism of procedurally generated terrain by simulating physical influences that happen in nature such as erosion. The most popular erosion techniques are thermal and hydraulic erosions.

2.7.2.1 Thermal Erosion

Thermal erosion is a simulation where the soil falls to a lower area if the angle is too sharp. This operation continues in a loop until the condition, which is reaching the maximum angle of balance for the material, is meet. Thermal erosion is one of the simplest erosion techniques to model. Moreover, it runs efficiently and works quickly. By changing the type of the neighborhood the running time can be improved for this algorithm. The Von Neumann neighborhood, the Moore neighborhood, and the rotated Von Neumann neighborhood are three standard neighborhood types. While increasing in speed, the rotated Von Neumann neighborhood offers good outcomes. However, the Moore neighborhood is the slowest

but gives the best outcomes [9].The drainage patterns simulation cannot be done using thermal erosion. However, this can be performed using hydraulic erosion.

2.7.2.2 Hydraulic Erosion

Hydraulic erosion is a simulation which stores water on the points of the terrain and let it move down into basins, eroding the surface on its way. Hydraulic erosion is very slow but gives good quality outcomes and needs a lot of memory. While the thermal erosion work with the source image, hydraulic erosion needs a sediment and water table, which results in the use of the memory three times more than of thermal erosion.

At first in the algorithm, a water and sediment tables are set up. A water table is an array of the image size that keeps the water amount in every pixel and sediment table is used to follow the sediment amount in the water. There are four parts that are done for every pixel [9]:

1. Rainfall: add how much rain falls per iteration to each pixel
2. Erosion: move the amount of eroded soil from the base image to sediment
3. Movement: if possible move water downhill
4. Evaporation: remove the percentage of the evaporated water from the pixel

The above steps are done for each pixel in each iteration. Because of it this technique gives good quality outcomes but tends to work slowly. The resulting image looks apparently that it has been eroded.

2.7.3 Texture-based Generation

Texture-based generation strategies obtain methods from the sector of texture synthesis. In computer graphics especially for procedural texture generation, texture synthesis is a widely used approach. Textures can be structured or stochastic,

such as stochastic textures comprise little structure, being near to random noise while structured textures defined as owning a repetitive, ordinary pattern.

There are two most important techniques for texture synthesis, which are patch-based and pixel-based. In the pixel-based technique, the texture is generated pixel by pixel, where the next pixels value is defined by its local neighborhood. The disadvantage of the pixel-based technique is that they have a tendency to mislay their global formation. This drawback is eliminated in patch-based techniques. Duplicating and sewing section of pixels of origin to the result is the main technique of a patch-based strategy. These methods suitably for realistic texture-based terrain generation, because they maintain global structure and patterns.

Raffe et al. [14] present an evolutionary algorithm to help in the creation of three-dimensional terrain with the help of selecting miniature height-map patches that were derived from sample maps for the in-game terrain generation. To satisfy the user's expectations novel patch-based terrain model is improved, which improves control over the development process. In this work the advantages of an interactive two-level parent selection operation are defined and also how to seamlessly join patches of terrain together is demonstrated.

Turk et al. [15] offer a new patch-based system for terrain synthesis which uses digital elevation model files and generates difficult outcomes. The operation begins with a user's sketch and DEM example file. The method gets patches form the sample data that suit the characteristics observed in the user's sketch.

2.7.4 Comparison of Terrain Generation Techniques

The Table 2.2, which is taken from the paper of Justin Crause [6] demonstrates the comparison between the terrain generation techniques mentioned in a previous section.

To conclude, fractal-based methods are able to run fast on current CPUs and the generation process is not fully controlled. The parameters that are set

Table 2.2: Comparison of terrain generation techniques [6].

	Speed	**User-Control**	**Realism**	**Main Limitations**
Fractal-based	Very fast	Low-High	Low	- Absence of natural erosion - Non-intuitive control parameters - Pseudo-random output terrain
Physics-based	Thermal: Fast Hydraulic: Slow	Low	Thermal: Medium Hydraulic: High	- Complex to implement - Requires a base terrain - Minimal user control
Texture-based	Slow	Medium	High	- Limited user control - Output dependant on number of input terrains (exemplars)

do not impact the outcome directly, which results in a lack of user control over the generation process of a terrain. Physics simulation may be used to increase realism to a base terrain with the help of combining actual weather influences. However, this technique frequently relies on the base terrain quality, because it also suffers from the low level of user control. Texture-based techniques make terrain generation more realistic by obtaining methods from the sector of texture synthesis and use the real terrains as their data source. When the generation process is controlled thru sketches the user gains more control. Using the GPU the runtime of these methods is not quite long.

In our study we are dealing with the fractal-based methods, especially with Diamond-Square Algorithm and Perlin Noise, to generate realistic terrains.

2.8 Related Works

When reviewing literature for similar studies, several papers were found that were similar to aspects of our study in some ways.

The study "*Terrainosaurus: Realistic Terrain Synthesis Using Genetic Algorithms*" by Ryan L.Saunders [16] present Terrainosaurus "a new design-by-example method for synthesizing terrain height-maps." The user outlines the design of the landscape by drawing out easy areas with the help of CAD-style interface and defines the preferred terrain properties of every area by giving height

map example displaying these properties. The height map provided as an example will basically come from real-world GIS (Geographic Information System) data sources. A genetic algorithm is used to mix together pieces of elevation data from an example height-maps in a visually attainable way to produce a height-map similar to the user's design at a different level of detail. The main benefit of the recommended approach is that unlimited variety of reasonable realistic terrain can be generated with the little help of user effort and expertise.

Terrainosaurus is similar to our approach in a way that we both use the GIS data sources to extract properties of the realistic terrain in order to come up with the realistic terrain as an outcome.Also, a sample mean, standard deviation, skewness and kurtosis are used to measure the similarity of two terrains in this study, but we are using these statistics to extract features for our neural network model.

Another study "*Fast, Realistic Terrain Synthesis*" by Justin Crause [6] propose "a patch-based terrain synthesis system that utilizes a user sketch to control the location of desired terrain features, such as ridges and valleys." Digital elevation models of real terrains are applied as example terrain, from which candidate patches of information are obtained and rivaled in opposition to user's shape. The final terrain is generated by seamlessly merging the best candidates. However, usage of real terrains results in highly appearing realistic terrains.

This method has some similarities to ours in a way that we both use real terrains to have a realistic terrain as an outcome.

The study "*Example-Based Realistic Terrain Generation*" by Li et al. [17] propose "a new approach to terrain generation based on terrain examples." They provide a semiautomatic terrain generation technique using a four process genetic algorithm method in order to create a different kind of terrain samples by applying just user inputs. A rough sketch of terrain silhouette map is specified by the users then terrain samples are retrieved based on support vector machine (SVM) form the terrain dataset and to complete the terrain silhouette map the areas are cut from terrain examples.

The main commonality of this approach with our method is that we both use terrain datasets to have a realistic terrain as an outcome.

Digne et al. [18] in study "*Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks*" present "an example-based authoring pipeline that uses a set of terrain synthesizers applied to particular tasks and every time terrain synthesizer is a Conditional Generative Adversarial Network (CGAN) trained via using real-world terrains and their drawn complements." Artists create a coarse sketch of the main terrain features, such as rivers and valleys then an algorithm synthesizes a terrain automatically suitable to an artist's sketch applying the learned features of the training samples. This framework gives a high level of realism and presents simple terrain authoring for the least design price.

The main similarity of this approach with our study is that we both use real-world terrains to train the Neural Network. This study uses CGAN neural network to synthesize a terrain, but we are using a neural network in our study to evaluate a terrain.

The study "*Terrain Generation Using Genetic Algorithms*" by Ong et al. [19] present "a procedure using genetic algorithms to generate 3D terrain datasets." They use a two-pass genetic algorithm procedure to generate different types of terrain applying only intuitive user inputs. They permit a user to define an unfinished sketch of the terrain region limits, and the genetic algorithm is used to improve these limits. Then they pair this with a database of given terrain data to create an artificial terrain, that is going to be optimized by applying a second genetic algorithm.

The main similarity of this approach with our study is that they are using mean, variance, minimum, maximum, and slope of the sample elevations for measurements of the likelihood of their generated region with the source example. But we are using mean, variance and slope of a real-world terrain to extract features for our neural network model.

Chapter 3

Methods

In the Figure 3.1 the overall structure and the flow of the algorithm is demonstrated. Each stage is covered in more details in subsequent sections.

3.1 Processing Real Terrain Images

The image contains information, such as color and by processing or analyzing an image it can be converted to a mathematical representation. For instance, an image has a width and height and the information contained in it can be represented by a two-dimensional matrix. The features can be extracted by processing an image.

3.1.1 Choice of Data Structure

The terrain is the critical part of our algorithm, so the representation of it is very important. Two facts were taken into account when choosing the data structure to represent terrain. The first one is that what type of data structure is used for terrain in most current, real-time applications and the second one is that the real elevation terrain data are presented in which form of data structure. After

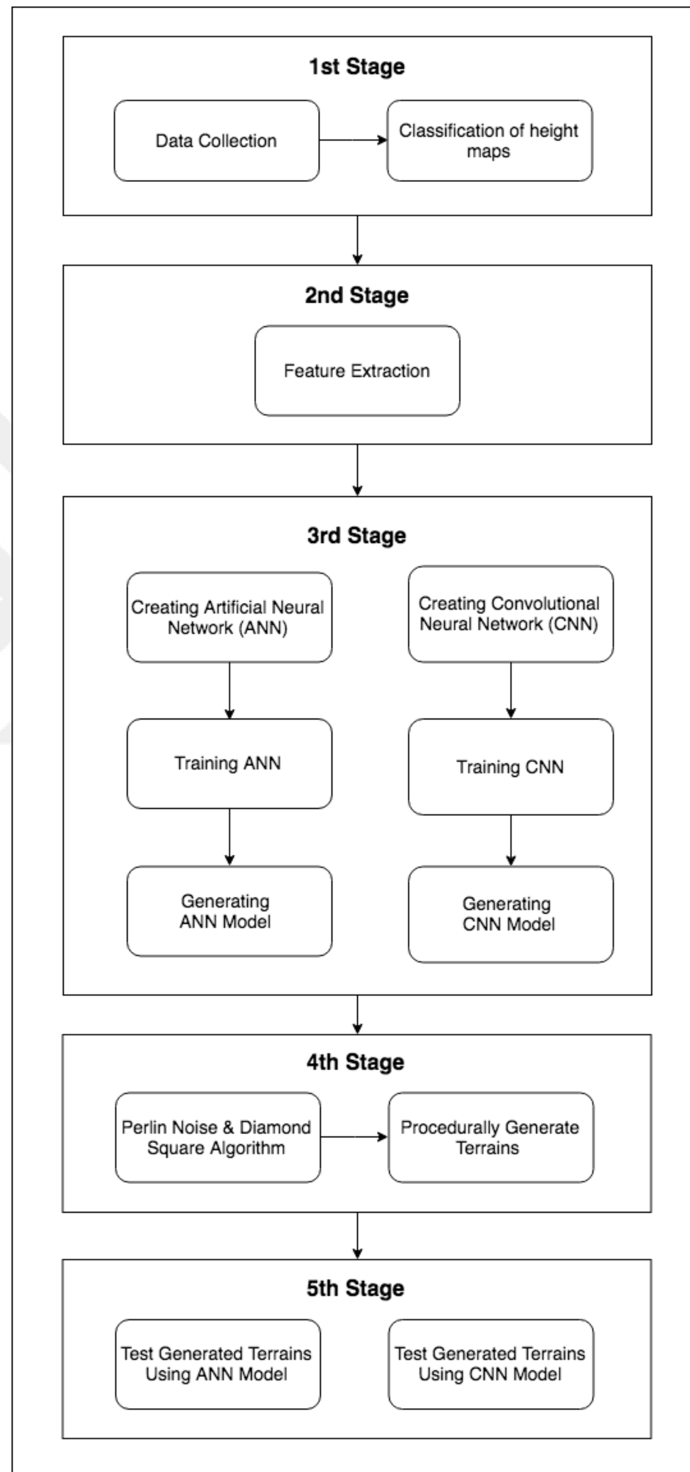


Figure 3.1: Overall Structure and Flow of the Algorithm.

doing background research we decided to choose the height-map as terrain data structure. The Figure 3.2 demonstrates a simple example of a height-map.

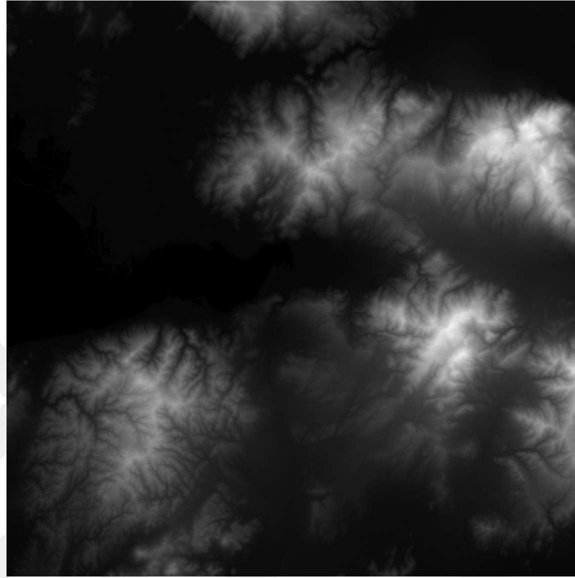


Figure 3.2: Height map of city Izmir.

3.1.2 Real Terrain Images Source

In order to understand the level of realism of generated terrains, we need to compare them with real-world terrains. There are many agents providing the Digital Elevation Model (DEM) of the places, but many of them have some restrictions. Some provide only 5-6 areas and the user is limited to the data which is provided by them. The free choice is eliminated in such cases.

The real-world terrain images are acquired from the online resource `http://terrain.party` which provides real-world terrain images. It provides a free choice over the world's map. The range of choice in the area is represented with a square of 8×8 km and 60×60 km.

Basically terrain.party provides height maps for the Cities: Skylines map editor. It produces a height map of 16-bit PNG's covering 1081×1081 pixels. The Figure 3.3 demonstrates the layout of the terrain.party.

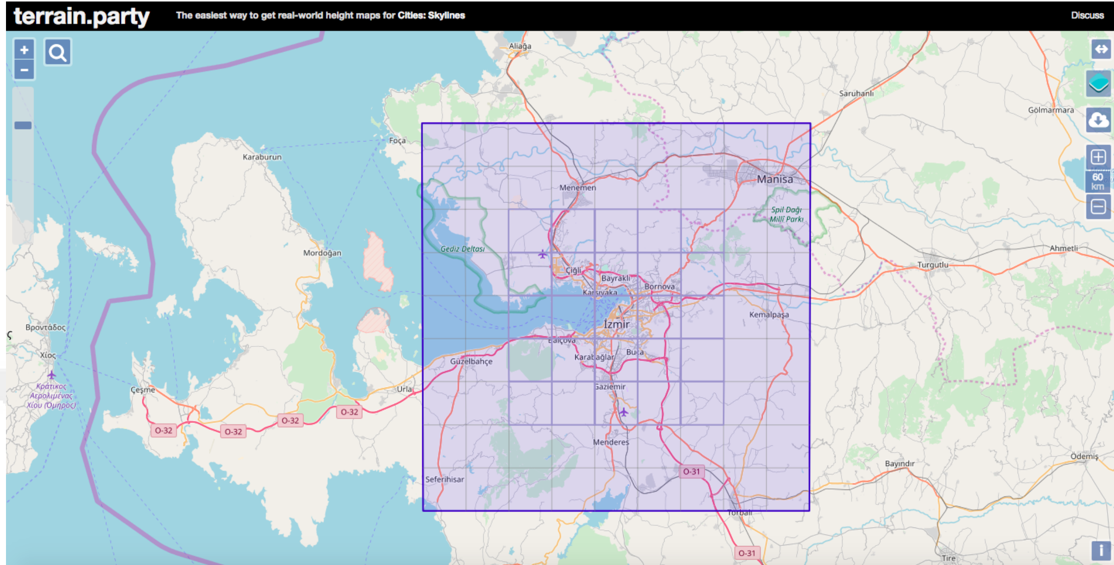


Figure 3.3: Layout of terrain.party [20].

The selected area of the map can be downloaded as ZIP file and the following data will be available inside these ZIP file [20]:

- **ASTER 30m**: a recent public survey of elevation on Earth having high coverage and high 30m resolution, however, occasional gaps are created in the data, because of the confusion by high concentrations of clouds and mountains.
- **SRTM3 v4.1**: is the result of significant work from CGIAR-CSI and has a 30m resolution in the US and 90m resolution elsewhere.
- **SRTM30+**: a 900m resolution dataset which is used to follow the general contours of the land without having every single contour.
- **Merged**: it a mix of ASTER 30m, USGC NED 10m, and SRTM30+, which provides a good global elevation data (ASTER) and better US elevation data (USGS NED) with hole0filling and bathymetry from SRTM30+.

The specification is not needed, because terrain.party will provide all of the data mentioned above inside a ZIP file and also a README file explaining all this as appropriate.

From all of those available data merged height-map should be selected because it is a mix of other data in a way that tries to be sensible and also README file suggest to look for this one as a height map.

3.1.3 Classification of Real Terrain Data

In our study we are classifying terrain data into three classes:

1. **Sealevel**: this class contains terrain height maps which are basically taken from the lakes and seas of the real world map.
2. **Lowland**: this class contains terrain height maps which are taken from the areas such as airports and cities where the surface is somehow flat.
3. **Mountain**: this class contains terrain height maps which are taken from the well known real-world mountains such as Nanga Parbat and Everest.

3.1.4 Descriptive Statistics

The descriptive statistics explain the data in a way that make sense. In our algorithm, measured statistics are used as an input parameter for ANN in order to train the neural net to distinguish between the terrain classification. Basically, the descriptive statistics used in this study are mean, standard deviation, skewness and kurtosis.

The mean is one of the best-known statistics and the average value found in a sample. The equation below is used to calculate the mean value of a sample.

$$\bar{x} = \frac{\sum x_i}{n} \tag{3.1}$$

The mean is the first-order measure of a statistical distribution. In addition, mean is a value about which the distribution is focused.

The standard deviation is a very useful statistic. It reveals how tightly data are grouped about the mean. The equation below is used to calculate the standard deviation value of a sample.

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}} \tag{3.2}$$

The standard deviation is a second-order measure of a statistical distribution defining the range in which the distribution propagates.

The skewness is the third statistic, which is a measurement of the symmetry of a distribution. It describes how much a distribution differs from a normal distribution, either to the right or to the left. It can be negative, positive or zero. When a skewness value is zero it means that the distribution is symmetrical around the mean. However, if the skewness value is positive or negative then this means that the distribution is either shifted to the right or left of the mean (Figure 3.4).

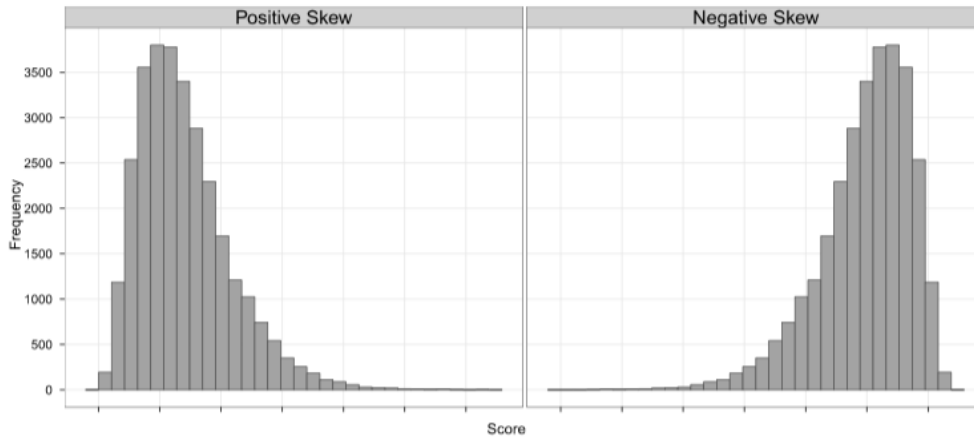


Figure 3.4: A positive (left) and negative (right) skewed distribution [22].

The equation [21] below demonstrates how to find the skewness value for a univariate data.

$$g_1 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^3 / N}{s^3} \tag{3.3}$$

where s is the standard deviation and x is the mean and N is the number

of samples. The above equation for skewness is known as the Fisher-Pearson coefficient of skewness [21].

If the data is collected to the left a positive skew occurs and if the data is collected to the right a negative skew occurs.

The kurtosis is the fourth statistic, which measures if the dataset is heavy-tailed or light-tailed as opposed to a normal distribution. When a kurtosis value is zero it means that the distribution is normal. However, if the value of kurtosis is high it indicates that the data set is heavy-tailed, also known as leptokurtic, and if the value of kurtosis is low it indicates that the data set is light-tailed, also known as platykurtic (Figure 3.5).

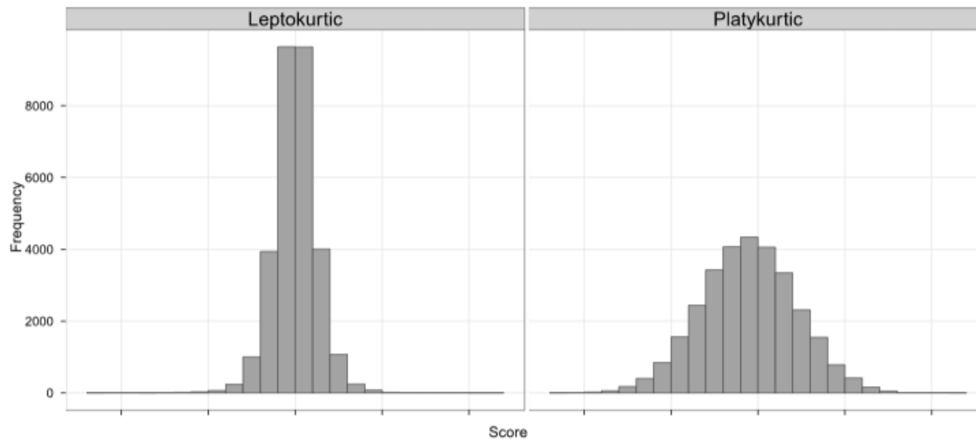


Figure 3.5: A positive (left) and negative (right) kurtosis example [22].

The equation [21] below demonstrates how to find a kurtosis value for a univariate data.

$$kurtosis = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^4 / N}{s^4} \tag{3.4}$$

where s is the standard deviation and x is the mean and N is the number of samples.

3.1.5 Other Statistics

Besides the four statistics mentioned in the previous section, three more statistical data are used as an input parameter in this study, which are slope, histogram, ruggedness.

The slope is very important when we want to find a line between two points, which tells how steep the line is, such as the higher the number, the steeper the line. Shortly, a slope is a value that defines both the direction and the steepness of the line and denoted by letter m. It is a measure of a rate of how slow or fast changes are taking place. The equation below demonstrates how to calculate the slope between two points.

$$Slope = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.5}$$

In our study, we need an equation to calculate slope between two points in a 3-dimensional space because height maps are in form of grayscale images and they have width, height and elevation data as luminosity. The equation below demonstrates how to calculate the slope of two points in 3D space.

$$Slope = \frac{rise}{run} = \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}} \tag{3.6}$$

First of all the distance is calculated between the two points, which results to run. The height will be the rise. Then height is divided to the distance, which is similar to dividing rise with a run, to get the slope between two points in a 3D space. In addition, Δx is equal to $\Delta x = x_b - x_a$, Δy is equal to $\Delta y = y_b - y_a$ and Δz is equal to $\Delta z = z_b - z_a$, between two points having (x_a, y_a, z_a) and (x_b, y_b, z_b) as their coordinates.

A histogram is a plot of statistical information that uses rectangles to underline frequency distribution of a set of data allowing for the inspection of data for its underlying distribution, such as normal distribution, outliers, kurtosis, and

skewness. The Figure 3.6 shows the histogram of a mountain region height map.

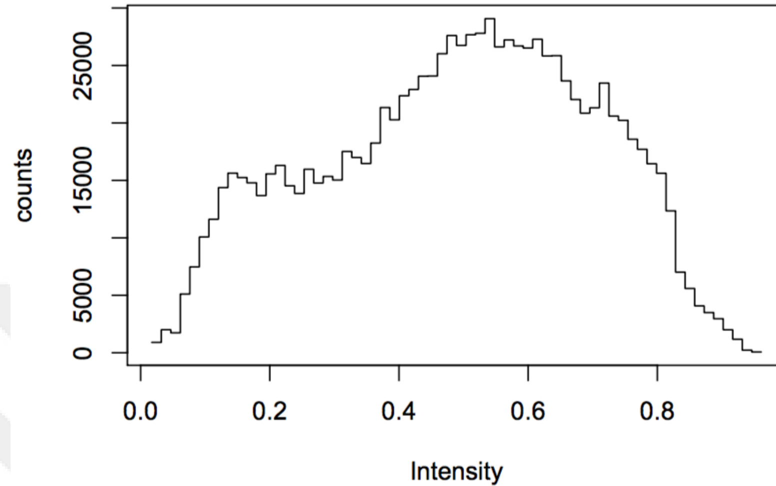


Figure 3.6: Histogram example of a height map. Generated using R.

In order to find the ruggedness or edge of an image, the first derivative is taken. In an image, edges are produced by different factors, such as surface color discontinuity, surface normal discontinuity, depth discontinuity, and illumination discontinuity. To characterize an edge is a place where a rapid change in the image intensity function. To detect those changes the first derivative is taken and the extreme values of derivative will correspond to the edges. In order to differentiate a digital image, we are going to take a discrete derivative. The Equation 3.7 demonstrates this process.

$$\frac{\partial f}{\partial x} \approx F[x+1, y] - F[x, y] \tag{3.7}$$

This equation can also be taken with respect to y and will give different results than with respect to x. The Figure 3.7 demonstrates those differences.

Another way to find the ruggedness of an image is by selecting points by traversing in a circular movement. In this method, points are taken from different places, rather than from a neighborhood like in previous section method. By doing so the points, which are not close to each other are taken and then the equation from the previous section is used to understand the ruggedness of an image.

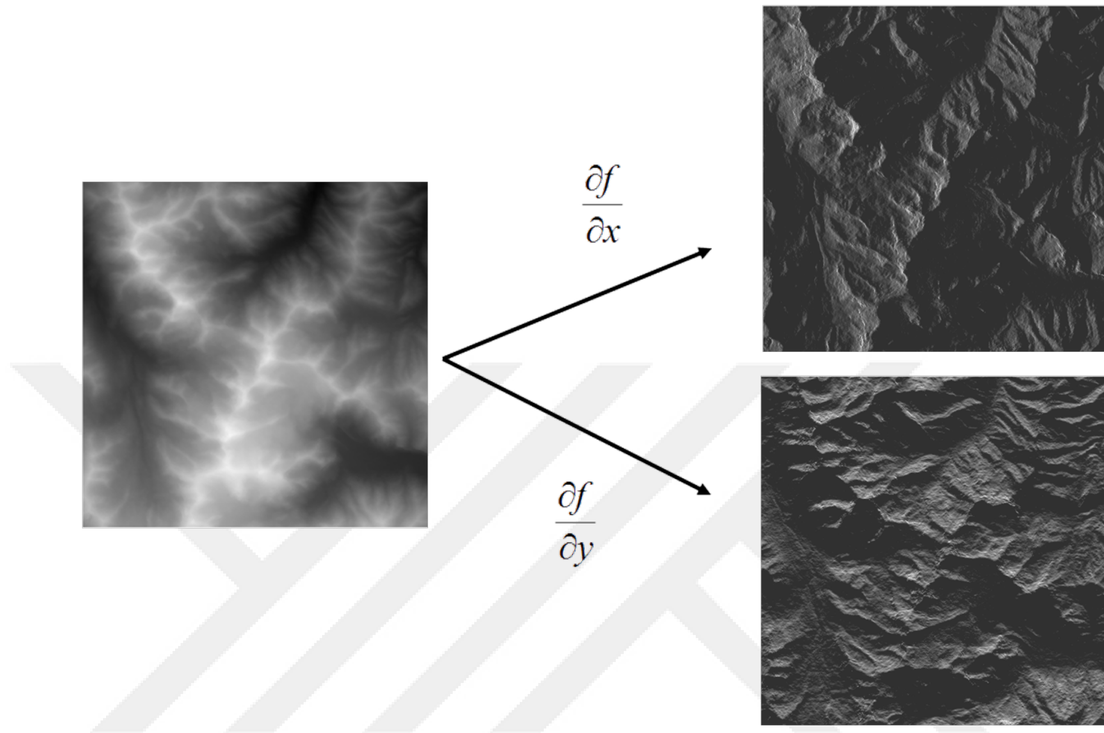


Figure 3.7: First derivative with respect to x and y.

The equation below is used to find the coordinate points while traversing in a circular movement on an image, where r is the radius and α is an angle.

$$x = r\cos(\alpha), \quad y = r\sin(\alpha) \tag{3.8}$$

3.2 Artificial Neural Network (ANN)

Our brain easily makes interpretation of what we see. It would not take any effort for us to tell the difference between the cat and dog or understand another person's emotion. However, those are without a doubt hard issues to solve with a computer, or machine.

An artificial neural network (ANN) is a computational model that is inspired by the human brain. It consists of an interconnected network of simple processing units which could learn from experience with the aid of enhancing its connections.

Structures of interconnected neurons that transfer information among each other are the common representation of ANNs. Synapses are simply weighted values and artificial neurons are connected through them. In order to make a neural net adaptive to inputs and able to learn, the connections have numeric weights.

The equation below demonstrates how artificial neurons compute their outputs.

$$y = f(w^\top x + b) \tag{3.9}$$

where f is a nonlinear function, w is the connection weights, x is the input, and b is the bias term.

The Figure 3.8 shows how to compute the output for a single neuron, where output of neuron is $Y = f(w1.x1 + w2.x2 + b)$

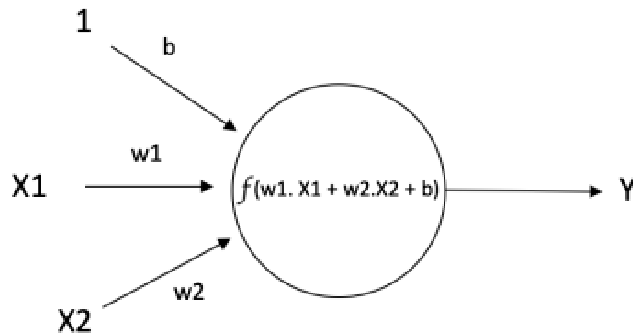


Figure 3.8: Single neuron computation [23].

3.2.1 Types of ANN

There are two types of neural networks. The first one is a simple neural network, also known as multilayer perceptron, and deep neural network. Figure 3.9 demonstrates these types of neural networks.

Input data is given to the input layer and passes after processing to the output layer to give a result. Moreover, the main processing and calculation is a

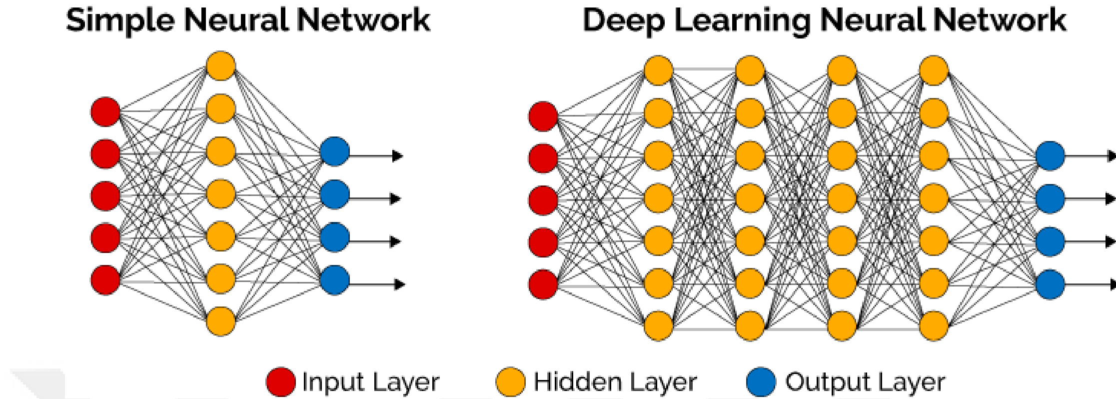


Figure 3.9: Types of neural network [24].

responsibility of the hidden layers. By increasing the number of hidden layers, we move from a shallow neural network to a deep neural network. Deep neural networks are capable of significantly more complex behavior than their shallow counterparts.

3.2.2 Activation Functions

The inputs are processed in a neuron using an activation function. The activation functions main purpose is to introduce nonlinearity into the output of the neuron, because many real-world data is nonlinear. The networks behavior depends on the selection of the activation function. There are many activation functions, but three of them will be discussed briefly.

3.2.2.1 Step Functions

The first function designed for machine learning algorithms was the step function. It basically consists of a simple threshold function that changes the Y value from 0 to 1. The step function has been historically used for classification problems, such as logistic regression with two classes.

The step function actually work as a limiter. Each input that goes into this function might be applied to receive either assigned a value of 0 or 1. Because of

that, it is simple to see how it can be helpful in classification problems.

3.2.2.2 Sigmoid Functions

The sigmoid functions are very beneficial in the sense that they squash their given inputs into a bounded interval. They are called sigmoid functions due to their shape in the Cartesian plane, which looks like an S shape. They can be very useful in combination with other functions, such as step function. The most popular one of the sigmoid functions, which can be found in an application, are hyperbolic tangent and logistic functions.

3.2.2.2.1 Logistic Function (Sigmoid)

As the name mentions, the logistic function is generally applied in logistic regression. The equation below shows how it is defined and the result will give a sigmoid over the (0,1) interval,

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}}. \tag{3.10}$$

3.2.2.2.2 Hyperbolic Tangent Function

The hyperbolic tangent function is based on the tangent function. The hyperbolic tangent, also known as TanH, is defined as the equation below.

$$f(x) = \frac{2}{1 + \mathrm{e}^{-2x}} - 1 \tag{3.11}$$

It generates a sigmoid over the (-1,1) interval. The hyperbolic tangent is used in many application and is probably the most used function of the sigmoid family.

3.2.3 Backpropagation

Backpropagation is used to understand if the neural network is correctly predicting in each epoch. The backpropagation is one of the easiest and most common techniques used for supervised training of multilayer neural network. In supervised training, the network output is compared with a target output and the network error is computed depending on the difference between them. Backpropagation is just a gradient descent approach [25] to decrease the total squared error of the output measured by the network.

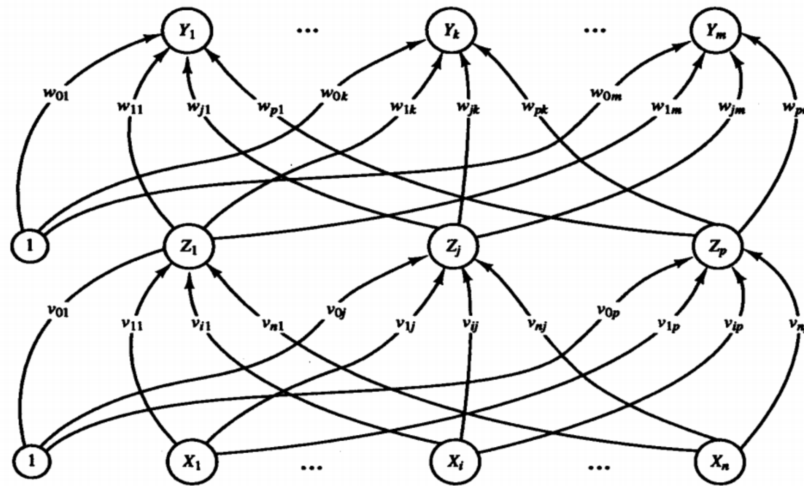


Figure 3.10: Single hidden layer neural net backpropagation [25].

Figure 3.10 above demonstrates a multilayer neural network with a single hidden layer and the direction of the data flow is feed-forward. The hidden units and output units may also have biases. However, during the backpropagation, error signal is sent in the opposite direction.

Three steps are required to train the neural network by backpropagation:

- Feed-forward or forward-propagation
- Calculation and backpropagation of the compared error
- Weights arrangement

In the feed-forward, the weights are initialized randomly and the data is sent to the hidden units. Each hidden unit then makes computations using its activation function and sends the results to the next layer, which is output units. Finally, each output unit makes a computation using its activation function to make a prediction of the network for the given input model. This step is also known as forward-propagation, because the flow is in the forward direction.

When the feed-forward step is executed the predicted results will be available. Loss function is then measured, which is the comparison of the predicted results with its target results, determining the compared error for that model. For classification problems, two types of loss functions are used the most, which are the mean square error (MSE) and cross-entropy error. However, for classification problems, the cross-entropy error often does better than MSE.

MSE is calculated using the below equation,

$$E_{total} = \sum \frac{1}{2}(target - output)^2. \tag{3.12}$$

Cross-entropy error is calculated using the below equation, where t is the target and y is the output.

$$E = -\sum_{i=1}^{nout}(t_i log(y_i) + (1 - t_i) log(1 - y_i)) \tag{3.13}$$

When the total error is calculated the backpropagation step will begin. The main purpose of the backpropagation is to decrease the error of the network for each output by adjusting the weights in the network and to make the output predicted by a network to be closer to the target output.

In the backpropagation, how much a difference in a weight influences the total error is analyzed. The derivative of the functions is taken in order to find how the change in a weight affecting the total error. For example, in order to calculate the change in w_3 affecting the total error the following equation is used, which is

also known as the gradient with respect to w_3.

$$\frac{\partial E_{total}}{\partial w_3} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_3} \tag{3.14}$$

After calculating how the change in a weight affects the total error, the new weight is computed and updated using the equation below, where η represents the learning rate,

$$w_3 = w_3 - \eta \times \frac{\partial E_{total}}{\partial w_3}. \tag{3.15}$$

To make the weight adjustments very slowly and smoothly the very small constant is presented known as learning rate.

The whole process will iterate until the convergence. It may need many iterations in order to learn because the weights are adjusted with little delta step at a time.

Figure 3.11 below reviews the learning process of neural networks.

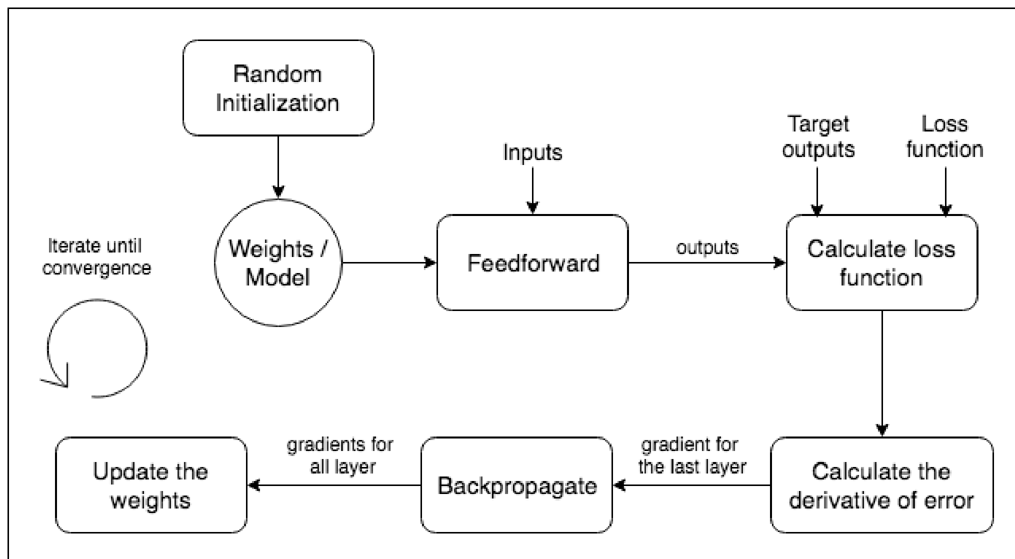


Figure 3.11: Neural Network Learning Process Review.

3.3 Convolutional Neural Network (CNN)

Convolutional Neural Networks, abbreviated as CNN's or ConvNets, are deep neural networks which are very useful in image classification and recognition. Also, they might be used commonly to categorize photos, such as naming the objects in the photo, group them by similarity, an example can be an image search, and carry out object identification inside scenes. CNNs are algorithms which do pretty good job in identifying human faces, street signs, tumors and lots of different factors of visual data. Another application field of ConvNets is audio when the audio is represented visually as a spectogram. The influence of ConvNets in an image recognition is one of the main reasons for the increase of study of deep learning in the recent years around the world. ConvNets are powering predominant advances in computer vision, which has an application for self-driving cars and robots.

3.3.1 Understanding Convolution

From Latin, "convolver", "to convolve" meaning to roll together. An integral measure of how much two functions overlay when one passes over the other is a mathematical explanation of the convolution.

CNN's derive their call from the "convolution" operation. Extracting features from the input photo is the main objective of the convolution in a CNN case. Convolution maintains the spatial relationship between pixels with the aid of learning picture capabilities using small squares of input data.

An image can be expressed as a matrix holding pixel values. The RGB image will have three channels and three matrices for each channel. Grayscale will have the only single channel and a single matrix for this channel. For example, an image with a 5×5 matrix of pixel values are only 1 and 0 and another matrix with 3×3 with 1 and 0 values. The convolution of the 5×5 image and 3×3 matrix will give an output, which is known as feature map. The whole process is demonstrated in Figure 3.12 below.

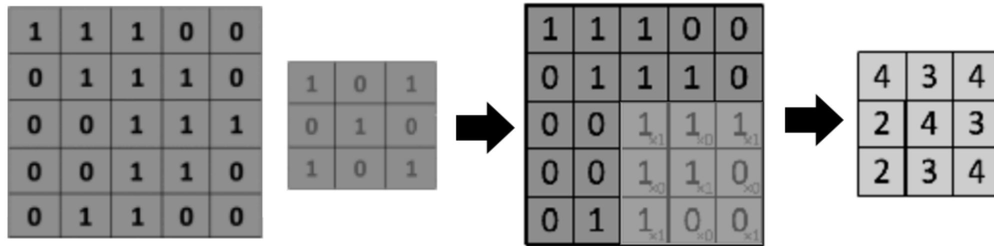


Figure 3.12: Convolution operation process [26].

The final result is computed by sliding the 3×3 matrix over the 5×5 image by 1 pixel, known as "stride", and for each position. Element-wise multiplication among the two matrices are carried out and the multiplication results are added to get the final result which forms a single element of the output matrix.

In ConvNet terminology, the 3×3 matrix is known as filter or kernel and the resultant matrix generated by sliding the kernel over the original image and calculating the dot product is known as Convolved Feature or Feature Map. The main purpose of filters is to detect features form the original image depending on the filter characteristics.

3.3.2 Feature Map

Basically, a ConvNet using a training session determines the values of the filters, but there is a need of specification of some parameters such as filter size, number of filters and etc before the training session. The number of image features depends on the number of filters and having more filters will result in more features and the better neural network will become at recognizing patterns in images.

There are three parameters which control the size of feature map:

1. **Depth**: represents a number of filters used in the convolution process, every filter tends to learn something different in the input image.

2. **Stride**: number of a pixel value that is used to slide the filter. In order to slide pixel value by one, the stride should be one. Smaller feature maps are created by increasing the stride number.

3. **Zero-padding**: fills the input matrix with zero values around the border. The main advantage provided with this is the size control of the feature maps. Wide convolution term is used when zero-padding is used and narrow convolution term is used when zero-padding is not used.

3.3.3 Introducing Non-Linearity (ReLU)

ReLU is a non-linear procedure and the name is a shortage of Rectified Linear Unit. Basically, after each convolution operation, the ReLU is used. The main function of the ReLU is to change all negative values appearing in the feature map to zero and it is an element-wise procedure used for each pixel. Convolution is a linear operation and we convert the result to non-linear with the help of ReLU function. Introducing non-linearity in the CNN is the main purpose of the ReLU.

To understand the ReLU operation more clearly, Figure 3.13 is given. The feature map is obtained from the original image and the ReLU operation is applied to this feature map.

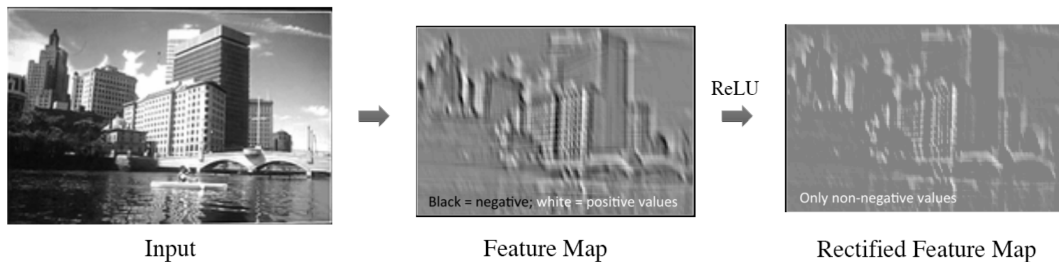


Figure 3.13: Obtaining rectified feature map [26].

Other nonlinear functions like sigmoid or tanh may be used as well in place of ReLU. However, ReLU has better performance in many cases.

3.3.4 Layers Used to Build CNNs

A simple CNN is build using several layers. Those layers using differentiable operation convert one volume of activations to other. To construct a CNN architecture three most important type of layers are used, which are stacked to form an overall CNN architecture. The layers are convolutional layer, pooling layer, and fully connected layer

3.3.4.1 Convolutional Layer

Most of the hard computational calculations are done in the convolutional layer and it is the fundament of a ConvNet. In the convolutional layer, the convolution operation discussed in Section 3.3.1 is performed. Since neurons perform the convolution operation to input, they are known as convolutional neurons. One of the important parameters in a convolutional neuron is the filter size.

For instance, when convolutional layer applied to the image with a 7×7 matrix and 3 color channels, at each step, the 3x3x3 sized chunk is picked from an image and the convolution is calculated using the filter. The convolution operation results in a single output and the bias is added to this output. The convolution filter is slid over an image to calculate the output and the number of slides are called a stride, which was discussed in the previous sections. Connecting all outputs in two dimensions an output feature map is formed with the size of 3×3. Having two filters in one convolutional layer will result in an output of size $3{\times}3{\times}2$.

3.3.4.2 Pooling Layer

Usually after convolutional layer, pooling layer is applied to decrease the sample size. The main function of the pooling layer is to decrease the number of parameters and calculations in CNNs and additionally to manage over-fitting. Various types of the pooling exist, such as average, max, and sum. However, the max

pooling used in most applications. In max pooling, a filter with size $M \times M$ is taken and the max operation is applied to an image. When an input size is w1xh1xd1 and the filter size is $m \times m$ with stride S this will result in an output size of w2xh2xd2 as shown in the below equation.

$$w2 = (w1 - m)/S + 1, \quad h2 = (h1 - m)/S + 1, \quad d2 = d1 \tag{3.16}$$

The filter size of 2×2 with a stride of 2 is a common pooling layer parameters applied to many applications, which approximately decreases the size of input by half.

3.3.4.3 Fully Connected Layer

The fully connected layer is a traditional multilayer perceptron, which applies a softmax operation on the output layer. The name “Fully Connected” mention that all neurons from previous and next layers are connected to each other.

High-level features of the input photo is an outcome of the convolutional and pooling layers. The main goal of a fully connected layer is to use the outcome features for matching an input photo to different categories according to the training dataset.

Fully connected layer gives a result with probabilities summing up to one. To maintain the result the softmax is used as an activation function in the output layer of a fully connected layer. The main purpose of a softmax function is to crush the vector values between 0 and 1, which sums up to 1.

3.3.5 Summing Up

As mentioned in the previous sections, the fully connected layer behaves as a classifier while convolution layer and the pooling layer behave as a feature extractors from the input image.

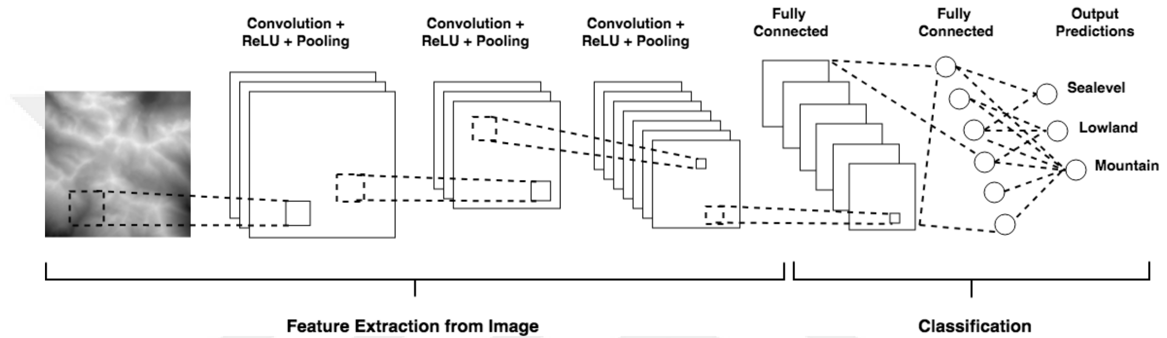


Figure 3.14: A simple CNN Architecture.

The step by step simple explanation is as follows:

- The original input image is examined for features using a filter that passes over it.
- Features maps are generated, one for each filter, as outputs and stacked over one another.
- The feature maps reduces using pooling.
- A new set of feature maps is created by passing filters over the first pooled stack.
- The second pooling is done, which reduces the second set of feature maps.
- A fully connected layer is applied that classifies output.

The general training procedure of the convolutional neural network can be summarized as follows:

- **Step 1**:Randomly initialize all parameters and filters values.
- **Step 2**: The neural net gets a training input image, process via the forward propagation step and discovers the output probabilities for every class.
- **Step 3**: Compute output layers total error. The equation of this computation is shown below.

$$TotalError = \sum \frac{1}{2}(target\ probability - output\ probability)^2 \tag{3.17}$$

- **Step 4**: Backpropagation is applied to measure the gradient of the failure to all weights. Moreover, to reduce the output error, gradient descent is used to arrange all parameter and weights values.
- **Step 5**: Repeat steps 2-4 with all dataset within the training set.

This is how the CNN is trained using the above steps, indicating that CNN is able to correctly classify photos of the training set by optimizing its weights and parameters.

The network will perform a forward propagation and give the result as a probability for every class when a new photo is inputted into the CNNs. If the training set is big enough and correctly constructed, the CNNs will correctly classify the new test images into categories.

The number of convolution and pooling layers could be replicated many times in a CNN. In addition, many best performing CNNs nowadays have tens of convolutional and pooling layers. Before having a pooling operation there can be many convolutions and ReLU operations in sequence.

3.4 TensorFlow (TF)

TensorFlow is a library developed by Google. TensorFlow was basically designed for jobs that need complex mathematical calculations, but it is a greatly versatile library. This is the main reason why TensorFlow was adjusted against the problem of deep neural nets and machine learning. Nowadays, TensorFlow holds the leading position on GitHub among all machine learning libraries. Google uses this library for implementing ML in almost all programs. As an example, Google voice search uses TensorFlow models.

The main component of TensorFlow is a computational graph because the structure of TensorFlow is based on the execution of a data flow graph and tensors which traverse among all the nodes through edges.

Two basic units of a data flow graph are as follows:

- A mathematical procedures are described as nodes.
- A multi-dimensional arrays, also known as a tensor, are described as edges.

The tensor can be described as an N-demensional vector mathematically. Indicating that it has an ability to describe an N-dimensional data set. New tensors are generated as a result of an operation in the graph and this process described as flow in a computational graph.

3.4.1 Parallelism in TensorFlow

One of the important property of TensorFlow is parallelism. In order to execute operations faster, TenforFlow provides users to use parallel computing devices. For parallel computing, the processes are scheduled automatically. There are two aspects of the distributed execution.

In the first one, it is a single system distributed execution where a single TF session creates a single worker and the worker is responsible for scheduling tasks

on different devices. In the second one, the number of workers increased and each is able to be on a separate or on same machines, every worker is responsible to operate its individual context.

3.4.2 Architecture of a TensorFlow

The architecture of TensorFlow is flexible and permits using calculation on more than one GPUs or CPUs. All this process can be done using a single API. To construct and execute computational graphs TensorFlow has a python interface, which is simple to use.

3.4.3 Image Representation as Tensors

The tensor structure helps us through giving the freedom to shape the dataset in the manner that is needed. It is very helpful when dealing with images, due to the nature of how information in images are encoded.

It is easy to understand that image has a width and height, because of it, it is logical to represent the information contained in the image with a two-dimensional matrix. However, images also have colors and another dimension is needed to store the color information and that is when tensors become very helpful.

Images are encoded into color channels, the image data is represented by each color intensity in a color channel at a given point. The most common type of image is RGB, which stand for red, green, blue. The intensity of each channel at each point with width and height can be represented by a matrix and ending up having three matrices and by combining them a tensor is formed.

3.4.4 Tensors in TF

TensorFlow holds data in tensors and tensor may be expressed in a variety of ways.

- **Constants**: defining tensor as a constant we will not be able to change it later on.
- **Variables**: variables can hold different values as opposed to constants. To be able to use variables in a computation graph it is necessary to initialize then before running the graph in a session.
- **Placeholders**: in order to feed data to a TF model from outside a model, the placeholders are used. Placeholders can be seen as holes in the model in which the data is passed. However, in order to pass the data into a model using placeholder the type of data along with its precision should be specified, such as float32 or int16.

3.4.5 Summing Up

TensorFlow is a very powerful library when making a convolutional neural network model. In addition, it is an open source library. It is very useful for our study because many mathematical computations are already defined inside the library for CNN and many documentation and tutorials are available online.

Many popular companies such as ebay, Uber, and Google use TensorFlow [27].

3.5 Procedural Terrain Generation Techniques

There are many techniques to procedurally generate terrains, many of them were discussed in Section 2.7.1.2 in Chapter 2. Two types of PTG techniques are used in our study, which are Perlin Noise and Diamond-Square Algorithm.

3.5.1 Diamond-Square Algorithm

The result of an improvement of a midpoint-displacement algorithm is a diamond-square algorithm and it is fractal based. The main advantage of this algorithm is speed. However, the disadvantage of this algorithm is the memory consumption. When speed is essential and plenty of memory is available this algorithm best fits to generate terrain procedurally. The detailed implementation will be discussed in the next chapter.

3.5.2 Perlin Noise Algorithm

Perlin noise algorithm is one of the widely used noise generation algorithms. It is not fractal, but in combination with fBm, it is capable of generating fractal terrains. In order to achieve fBm, we generate several octaves of noise, each of them is generated with a decreased amplitude and increased frequency, and these octaves are all summed together to produce the fractal terrain. The detailed implementation will be addressed in the next chapter.

Chapter 4

Implementation

In this chapter, an important implementation details of the algorithms used in this study will be discussed.

4.1 Feature Extraction

As mentioned in the previous chapter, the height-map was chosen as the terrain data structure. The height map has a two-dimensional grid-based data structure, where x and y are coordinate points determining the location of an elevation value h as demonstrated in the Equation 2.1 on Chapter 2.

The terrain.party web application was used to get real-world height maps. The height-maps provided from terrain.party is a height map of 16-bit PNGs covering 1081×1081 pixels. All height-maps are cropped to have a resolution of 1024×1024 in our study. All of this cropping is done by native image toolbox and the crop regions are selected randomly. As a result, 16-bit PNGs covering 1024×1024 pixels are obtained.

However, the original elevation range provided from terrain.party is for 1081×1081 pixels and after cropping the number of pixels are going to decrease and for this reason, an original elevation range is converted to a new cropped image range.

The process of conversion is simple and it is done by providing the old range and new range and then making conversion between those ranges. The equation below is used to make those conversions.

$$\begin{aligned} rangeOld &= maxOld - minOld \\ rangeNew &= maxNew - minNew \\ valueNew &= (((valueOld - minOld) * rangeNew)/rangeOld) + minNew \end{aligned} \tag{4.1}$$

The range conversion is done using R application. R is widely used language in statistical calculations and graphical analysis.

As mentioned in the previous chapter the terrain data is classified into three classes, such as sealevel, lowland, and mountain. Fifty height maps for each class for training purpose and fifteen height maps for each class for a testing purpose are obtained from terrain.party. The Figure 4.1 demonstrates the basic flow of the feature extraction algorithm.

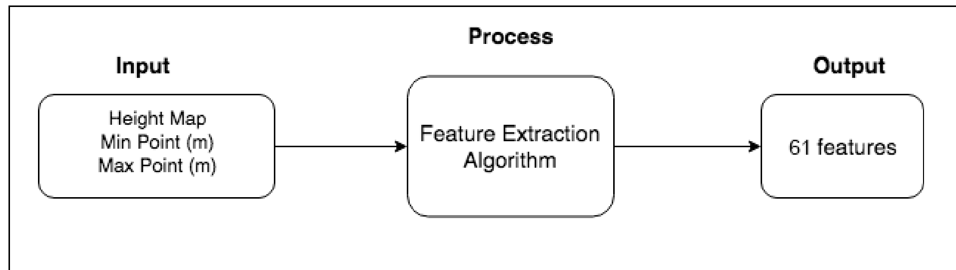


Figure 4.1: Feature Extraction Algorithm.

To extract features of the height map a C# script running on the Unity3D engine is used. Unity3D is a user-friendly and powerful cross-platform 3D engine development environment, which is easy to use.

The height map with the converted original elevation range for each class is set as an input parameter. Then in the process, each pixel of the height map is traversed and the elevation data is saved in a 2D array. After the elevation data is saved in a 2D array the feature extraction begins. In this study the features are classified into 5 categories:

- Statistical data
- Slope data
- Partitioning
  - The first derivative of parts (Zero Ruggedness)
  - The circular ruggedness of parts (Circular Ruggedness)
- Histograms

4.1.1 Statistical Data

In the statistical data, the basic statistics of a height map, such as mean, standard deviation, skewness, and kurtosis are calculated. All of them is calculated as meter unit.

The statistical data will provide the basic knowledge about the distribution of the elevation data of a height-map image and it can help to better understand the type of a height-map image.

4.1.2 Slope Data

In the slope data, six slopes from the height map are obtained. The first slope is the main slope of the height map, which is calculated by finding the minimum and maximum elevation points and if there are many same max and min points then the first one which is met is taken. In order to find the slope between those points, the slope in a three-dimensional space should be calculated because height maps are in form of grayscale images and they have a width, height and elevation data as intensity. The Equation 3.6 from the previous chapter is used to calculate a slope.

Remaining five slopes are the slope of five ranges. The height map elevation data has a range between 0 and 1. A division of this range into 5 subranges

are performed. Such as the first subrange is between 0 - 0.2, the second one is between 0.2 - 0.4, the third one is between 0.4 - 0.6, the fourth one is between 0.6 - 0.8 and the last one is between 0.8 - 1. The slope is calculated by finding the minimum and maximum elevation points for each subrange.

The slope data will provide the level of the steepness of a height-map image and it can help to better distinguish between different terrain types.

4.1.3 Partitioning

The height map is divided into 16 equal parts. The resolution of the height map is 1024×1024 and the part resolution is 256×256 each. The height map is divided into parts in order to understand which part of the map has the highest and the lowest ruggedness. Two methods were used to find the ruggedness of height-map image parts. Those methods are as follows: the first derivative of parts (Zero Ruggedness) and the circular ruggedness of parts (Circular Ruggedness). The name in the parenthesis is used in the results table of this study.

4.1.3.1 The First Derivative of Parts

The ruggedness or edge of a height map can be found by taking the first derivative. An edge is a place where the rapid change in the image intensity occurs and to detect these changes the first derivative is taken and the extreme values of the derivative will give the edges. To differentiate the height map a discrete derivative will be taken and the Equation 3.7 from previous chapter, will be used.

A counter is incremented when an edge is detected and in order to detect the extreme values, a threshold is set in a our function.

4.1.3.2 The Circular Ruggedness of Parts

Another way to find the ruggedness in each part is by selecting points by traversing in a circular movement.

The radius of the circle and an angle are defined. Using those parameters the location of a point x and y and the elevation data at this location can be found. In order to find the x and y location, the Equation 3.8 from the previous chapter is used. The flowchart of the operation process is shown in the Figure 4.2.

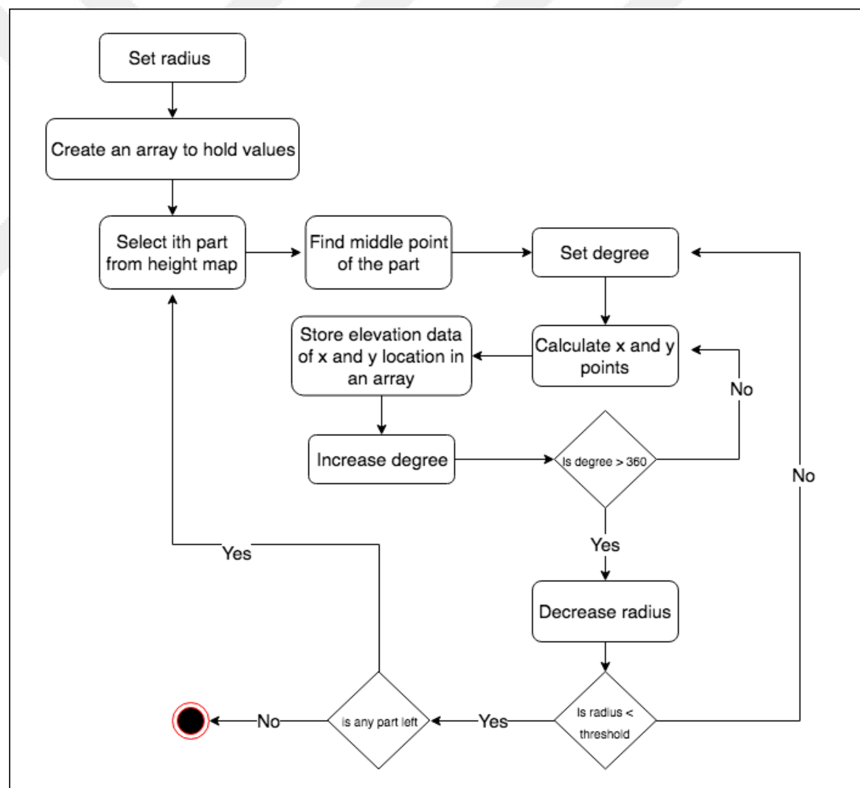


Figure 4.2: Circular Ruggedness Flowchart.

First, the radius parameter is defined and an array is created to hold the elevation data of a point. After in each part, the middle point is found and an angle value is set. By using an angle value and the radius, x and y coordinate point are found. When the point is located the elevation value of that point is stored in an array. Each time the angle value is increased to get another point until the full circle is made. After that, the radius value is decreased and a circle

making process starts again. The whole process is performed for each part of an image. Finally, the first derivative of the resultant array values are taken and from the result the number of zeros is count.

The ruggedness data will provide the level of the ruggedness of a height-map image and it can help to distinguish between terrains with a high and low level of ruggedness.

4.1.4 Histograms

Two separate histograms with different ranges are used in this study. The main goal of separating histogram into two parts is because of data range. Data range changes between 0 - 8880 meters. Because the highest point of the world is taken into account, which is the peak of Mountain Everest.

One histogram has a range of 0 - 9000 meters, named as Histogram Two in this study, and divided into 9 class with a range of 1000 meters. The Figure 4.3 below demonstrates this histogram example.

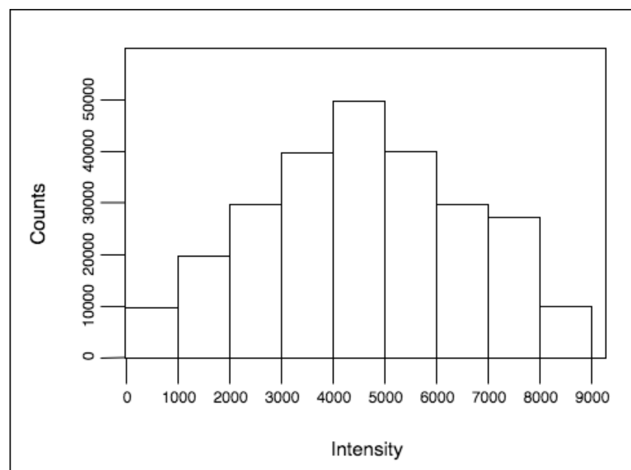


Figure 4.3: Histogram with range between 0 - 9000 meters.

This histogram is very useful to distinguish lowland and sealevel from a mountain types of terrain. But it not very helpful to distinguish sealevel from lowland

types of terrain. Because of this, another histogram is introduced, named as Histogram One in this study, which has a range of 0 - 500 meters and divided into ten class with a range of 50 meters. It is very helpful to distinguish sealevel from lowland types of terrain.

The histogram data results in large numbers because a height map has exactly $1024 \times 1024 = 1048576$-pixel values. Basically, in a histogram, the pixel value going to which class is counted. When all of these class values of the histogram are added the result will give 1048576 and because of that to make the values look small, the normalization can be done by dividing each class value with the 1048576.

The histogram data will provide how the elevation data of a height-map image are distributed in a histogram range and it can help to better distinguish between different terrain types.

4.1.5 Results

After the feature extraction algorithm is complete the result will provide 61 features for each height-map image. These features are stored in an Excel spreadsheet and used later to train an ANN model. The number of features for each category is demonstrated in the Table 4.1 below.

Table 4.1: Number of features in each category.

Category	**Number of features**	**Included**
Statistical Data	4	Mean Standard Deviation (std) Skewness Kurtosis
Slope Data	6	Main Slope Range Slopes
First Derivative of parts	16	Zero Ruggetness of each part
Curcular Ruggetness of parts	16	Circular Ruggetness of each part
Histograms	19	Histogram One and Two

4.2 ANN Implementation

The important details about ANN were mentioned in the previous chapter. In this section, the main architecture of the ANN that is used in our study and how to train the ANN with the extracted features from the previous section will be discussed. The best features to select for the further processes in the ANN will be obtained. In order to select features to get good results on test samples, different combinations of the extracted features are made.

4.2.1 Different Feature Combinations for ANN

As mentioned in the previous section 61 features for each height maps were extracted. The resultant number of features was derived from the five categories discussed in the previous section. In order to find how features influence the performance of an ANN, the different combinations of the extracted features will be used and the combination is basically done within the five categories. The Table 4.2 shows the different combination of the features.

Table 4.2: Different combinations of the features.

Number of Inputs	**Input Name**	**Description**	**Number of Inputs**	**Input Name**	**Description**
61	Inputs	All Inputs Included	29	InputsNoZeroCircular	Circular and Zero Ruggetness Not Nncluded
45	InputsNoCircular	Circular Ruggetness Not Included	26	InputsNoZerohist	Zero and All Histograms Not Included
51	InputsNoHistOne	Histogram One Not Included	35	InputsNoZeroHistOne	Zero and Histogram One Not Included
52	InputsNoHistTwo	Histogram Two Not Included	36	InputsNoZeroHistTwo	Zero and Histogram Two Not Included
55	InputsNoSlopes	Slopes Not Included	39	InputsNoZeroSlope	Zero and Slope Not Included
45	InputsNoZero	Zero Ruggetness Not Included			

4.2.2 ANN Setup

The results of different feature combinations are obtained in two different ways. The first way is by using Pattern Recognition MATLABs Neural Network Toolbox

for pattern recognition and classification. Another way is by using K-Fold cross-validation technique.

4.2.2.1 Pattern Recognition

To create pattern recognition network MATLABs Neural Network Toolbox is used. Basically, data is selected after the creation of a network by setting a number of neurons in the hidden layer and dividing an original data into train, validation and test data. When those parameters are set a network is trained and the evaluation results are obtained along with the trained neural network model.

As mentioned earlier the original data samples are divided into three kinds of samples randomly, which are train, validation, and test data samples.

- Training: the sample data which is used to train the network.
- Validation: the sample data which is applied to control network generalization, and to terminate the training process when generalization ends advancing.
- Testing: the sample data which is used to test the trained network model to find out the performance of the network.

After the data samples are divided into train, validation and test samples the number of hidden neurons is given and when the network performance is poor the number of neurons can be changed for further training.

Then the training function for the network is selected. A different number of training functions are available to train the network, but the default training function for pattern recognition in Matlab is scaled conjugate gradient (SCG) backpropagation.

SCG backpropagation is a network training function which adjusts bias and weight condition with respect to the SCG method. To bypass the time-consuming

line search the SCG algorithm was developed by Moller. The basic idea of the algorithm is to connect the model-trust region method with the conjugate gradient method. It uses less memory and suitable in low memory situations.

When generalization finishes progressing the training will automatically end, as shown by an expansion in the cross-entropy failure of the validation examples.

After the training ends the results of the trained network is obtained. Those results include percent error and cross-entropy. A percent error shows the portion of samples that are classified incorrectly, 0 indicating no misclassification and 100 indicating maximum misclassification in the result. In cross-entropy, the lower values in the result are better and zero indicates that there is no any error. The evaluation of the network performance is done using cross-entropy and confusion matrices.

Moreover, two plots are available in order to better understand the results. These are confusion and ROC (receiver operating characteristic) plots.

- ROC: a plot applied to control the quality of classifiers. The false and true positive rates are computed for every class over the interval [0, 1].
- Confusion: a plot of a confusion matrix, where the diagonal cells show the correctly classified observations. The column demonstrates the target and the row demonstrates the output.

After the network model is trained and has a good performance the model can be saved for further usage.

4.2.2.2 K-Fold Cross-Validation

The conventional analytical tools for deciding the performance of the system is required to create an efficient machine learning solution. The powerful method, which provides a correct evaluation of the right efficiency of the system is a cross-validation. The sample data is separated into small validation set and a large

training set in a cross-validation. Afterward, for training the train set is applied and to measure the accuracy validation set is applied.

In K-folds cross-validation, the data is randomly sorted and then divided into k folds. The most commonly used values for k is ten and hence the data is divided into ten parts. After the division of data, the cross-validation runs k times and during every iteration, one fold is selected for validation purpose and the rests are used for training. After training completes using validation set the accuracy of a network is measured. Finally, a cross-validation accuracy is computed by taking the average of all accuracy.

The Figure 4.4 below demonstrates the visualization of the ten-fold cross-validation and the accuracy calculation. The Final Accuracy = Average(Round 1, Round2, ...).

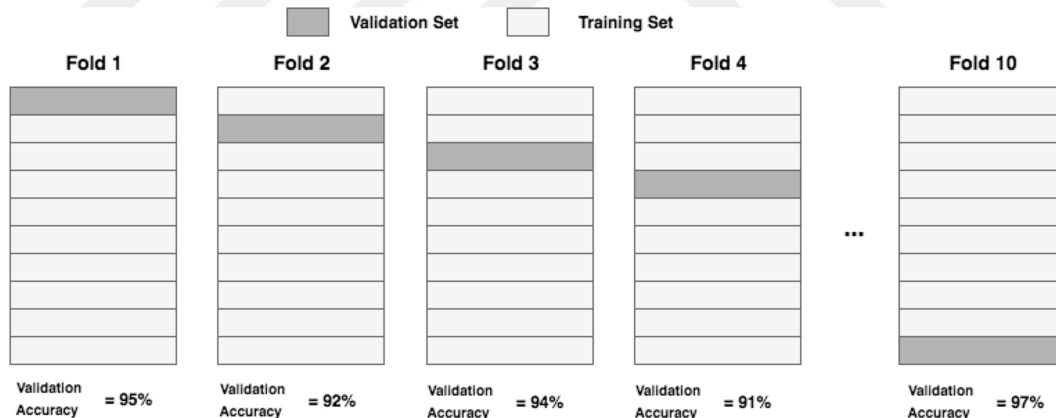


Figure 4.4: 10-folds cross-validation example over the data samples.

It is easy to make k-fold cross-validation in MATLAB. The Figure 4.5 demonstrates the flowchart, which is used to generate the overall accuracy of the network and to find out which feature combination gives the best results.

The results of the different feature combinations with using pattern recognition and K-fold cross-validation will be provided in the next chapter and the detailed analysis is done to choose the correct combination for further processes.

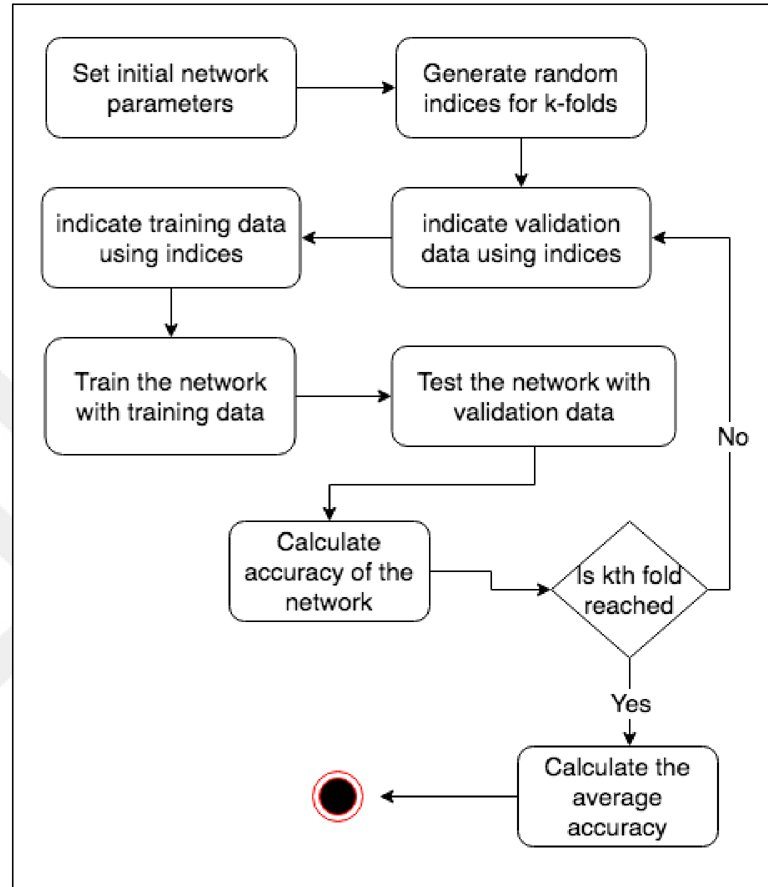


Figure 4.5: K-fold cross-validation flowchart.

4.3 CNN Implementation

A detailed explanation of CNN is done in the previous chapter. In this section, the architecture of a CNN that is used in our study with the selected layers are explained. As mentioned in the previous chapter about TensorFlow, this library is going to be used for our CNN implementation.

4.3.1 Architecture of the CNN

The architecture that we are going to use for CNN is not complicated and easy to understand and compute on a CPU. The Figure 4.6 below demonstrates the architecture of our CNN.

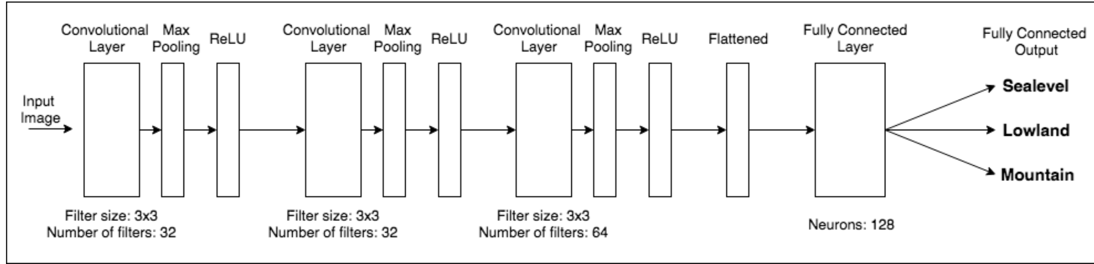


Figure 4.6: Architecture of our CNN.

The input image from all classes (sealevel, lowland, mountain) is given to a convolutional layer. Three convolutional layers are used sequentially. When all convolutional layer has processed the results of an output from the last convolutional layer is flattened. When the results are flattened the two fully connected layers are applied. Three outcomes representing the probability of a sample image being sealevel, lowland or mountain are the results of the last fully connected layer. In total 12 layers are used to build CNN architecture.

4.3.2 Augmenting Data

In order to make CNN give accurate results, augmentation of input data is done because in this study only fifty images for each class are used. To increase the number of input images the augmentation is done. Augmentation of images can be easily done in Matlab.

The following three steps are done in the augmentation process:

- Flip-flopping
- Rotation
- Resizing

The three steps are used in combination with each other. Such as the original image is flipped then rotated and resized giving the augmented image as a result.

The following 15, 30, 45, 60, and 75 angles are used to rotate an image. Each image is rotated with these angle values. The original image size is 1024×1024 and resizing it to 680×680 and 2024×2024 is done. In order to avoid black corners after rotating an image, the middle of the image cropped with the half size of the original image and the result is saved.

After making augmentation for each image we are ending up having 1600 image samples for each class summing up to 4800 samples in total.

4.3.3 Reading Inputs

The 4800 images of sealevel, lowland, and mountain are used as input images. Basically, the input images are divided into 3 parts:

1. Training data: 80% (3840) of images are applied for training.
2. Validation data: the rest 20% (960) of the images are applied for validation to compute accuracy.
3. Test data: 360 separate images, which are not used in either training or validation. Generated by augmenting the 45 test image collected from terrain.party.

4.3.4 CNN Layers

Convolutional layer, flattened and fully connected layer constructs the CNN in this study. In this section, the layers are explained in detail. Such as how input parameters are set for each layer and which methods are used in conjunction with the layers.

4.3.4.1 Convolutional Layer

It is easy to build a convolutional layer in TensorFlow. The programming language used to implement the CNN with TensorFlow library is python. The inputs for a convolutional layer is as follows:

- Input: is a 4D tensor. The input of the first convolutional layer is going to be a number of images, size of an image and number of channel in an image. In the next layers, the output of the previous layer is given.
- Filter: the filter is represented by the variables that are going to be trained. It is a 4D tensor.
- Strides: the number of sliding the filter when doing convolution is defined with this input. In order to not skip images in our batch, we define this parameter as 1.
- Padding: to make output dimension same as input the SAME is used, which fills the input matrix with zero around the border.

Next, a max pooling on the resultant layer is performed after the biases are added. The filter of length 2×2 with a stride of 2 is applied in max pooling.

At a final part, a ReLU is used as an activation function, which applies the ReLU operation over the result of a max pooling. The procedures mentioned above are all done in a single convolutional layer and the pseudocode of this procedures is as follows:

```
Convolutional Layer : Input, Number of Channels, Number of Filters, Conv Filter Size
    Layer = Convolution Operation : Input, filter , strides , padding
    Layer = Layer + biases
    Layer = Max Pooling : Layer, kSize, strides , padding
    Layer = ReLU: Layer
Return Layer
```

4.3.4.2 Flattening Layer

Multi-dimensional tensor is the result of a convolutional layer. In order to convert this result to a single dimensional tensor flattening layer is used. This is simply done by the use of reshaping operation.

4.3.4.3 Fully Connected Layer

A fully connected layer has a similarity with the shallow neural network. The weights and biases are defined first as a random normal distribution. Then a regular z = wx + b process is conducted for all inputs. In order to add a non-linearity to the result, there is a possibility to add ReLU to the layer. The pseudocode of this procedure is as follows:

```
Create Fully Connected Layer : Inputs, Inputs Number, Outputs Number, ReLU Usage
    Weights = create weights : shape = [Inputs Number, Outputs Number]
    Biases = create biases : Outputs Number
    Layer = matmul : input, weights + biases
    If ReLU usage
        Layer = ReLU : Layer
Return Layer
```

4.3.5 Predictions

The softmax activation function is applied to the outcome of a second fully connected layer to obtain the probability values for each class.

To calculate the cost cross entropy with softmax will be used on the output of last fully connected layer and the average will give the cost. To reach the optimum value of the weights the cost needs to be minimized.

4.3.6 Optimization and Accuracy

Most of the optimization functions are implemented in TensorFlow. To reach an optimum value of weight and to measure a gradient, the Adam Optimizer function is used. In an Adam Optimizer function, the objective of minimizing the cost along with a learning rate of 1.00E-4 should be specified.

True and predicted labels are used to calculate the accuracy of training and validation. Since a number of images used in training are higher than validation, training accuracy should give higher results.

The batch size is defined as 48 in our algorithm. The batch size is the number of training samples offered in a single batch. We have 3840 training samples and by dividing this number by the batch size the number of batches will be obtained which is used to make an epoch. In other words, when entire samples passed forward and backward inside the network once, it will form an epoch. So by dividing the 3840 by 48, 80 batches are obtained to make one epoch. The network is going to be trained in 4000 iterations, which is going to make 50 epochs. After each epoch, the training and validation accuracy and the validation loss are reported. From the report, it is possible to observe if the training process in our network is going in the right direction, which means that the accuracy should get increased at each epoch in the training datasets. The report results will be given in the next chapter with a detailed analysis.

Finally, when the CNN training process finishes the model will be saved and it can be used later to test other samples.

4.4 PTG Techniques Implementation

In this section, the implementation of the Diamond-Square algorithm and Perlin Noise are presented.

4.4.1 Diamond-Square Algorithm

The result of an improvement of a midpoint-displacement algorithm is a diamond-square algorithm. It is naturally fractal and easy to implement. The main drawback of this method is a memory and the advantage is the speed.

Implementation of a diamond-square algorithm is done in Unity3D with C# programming language. The flowchart of the algorithm is demonstrated in the Figure 4.7 below.

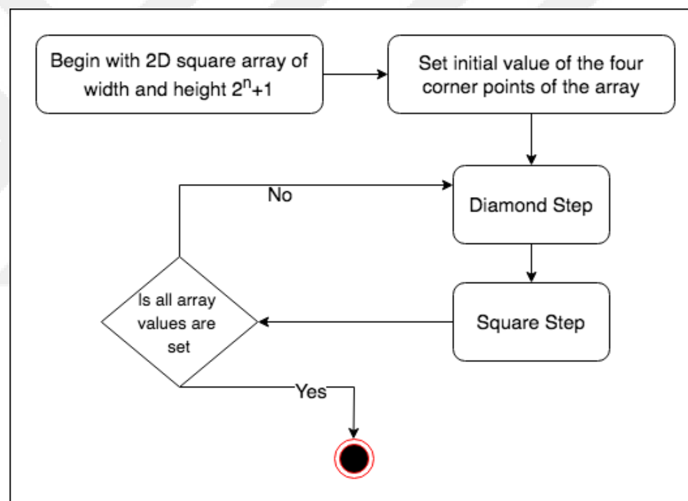


Figure 4.7: Flowchart of Diamond-Square Algorithm.

Basically, the diamond-square algorithm starts with a two-dimensional square array with width and height size of 2^n+1 and initial values of the four corners points are set. Then the diamond and square steps are done in a loop with a condition that all values of the array are set.

In the diamond step, the center point of a square in the array is computed by taking the average of the four points located in the corners and adding a random value. This process is done for every square in the array.

In the square step, the center point of a diamond in the array is computed by taking the average of the four points located in the corners and adding a random value. This process is done for every diamond in the array.

The size of a random value is decreased with every iteration. When the square steps are performed sometimes the edge points of the array will have only three adjacent values initialized instead of four. To handle such kind of a situation there are many different methods, but the simplest one is to take the average of those three adjacent values.

After all array values are set a Gaussian Smoothing is applied to make the generated terrain smoother.

The Figure 4.8 below demonstrates the height map example generated using a Diamond-Square algorithm that is used in our study.

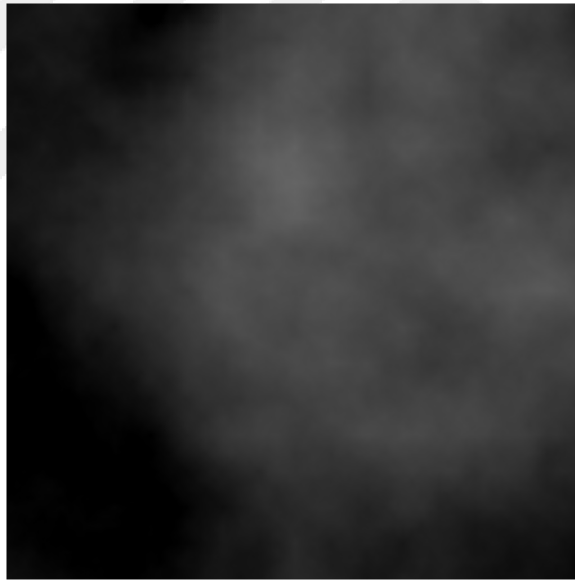


Figure 4.8: Diamond-square height map example.

4.4.2 Perlin Noise

Perlin noise does not generate fractal results and in order to generate fractal results, we are going to use it with the conjunction of Fractional Brownian Motion (fBm). To achieve fBm, several octaves of Perlin noise is generated, each of them is produced with an increased frequency and decreased amplitude and afterward, these octaves are summed together. The persistence value has a range of [0, 1]. The pseudocode of Perlin noise algorithm is as follows:

```
Generate Perlin Noise : x, y, number of octaves, persistence, lacunarity
    Set amplitude to 1
    Set frequency to 1
    Set total noise to 0
    For i = 0 to a number of octaves
        Perlin Value = Perlin Noise ( x* frequency, y*frequency)
        Total Noise  = Total Nose + Perlin Value * amplitude
        Amplitude = amplitude * persistence
        Frequency = frequency * lacunarity
    End of Loop
Return Total Noise
```

The Figure 4.9 below demonstrates the height map example generated using the Perlin noise that is used in our study.

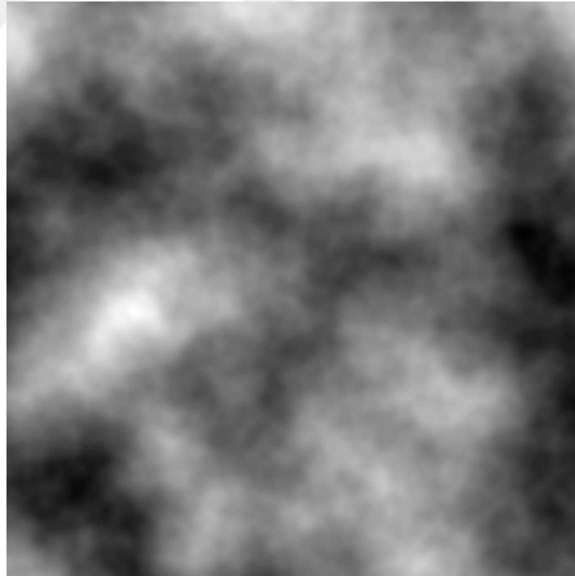


Figure 4.9: Perlin noise height map example.

4.4.3 Feature Extraction of Procedurally Generated Terrains

The feature extraction of procedurally generated terrains is very similar to the feature extraction of real-world height map images. However, there is one important difference that needs to be mentioned when the features are extracted. In the real-world height map feature extraction, the minimum and maximum points of the height map were given by the source in meter unit and those points were used as an input parameter for feature extraction algorithm. But procedurally generated terrains has only range between 0 and 1. The conversion of this ranges to meter unit should be done in order to extract features correctly.

To do the conversion correctly the conversion parameters needs to be defined. To find the necessary parameters the ranges for each class needs to be defined first. In order to define the ranges for each class, all real-world height map data are analyzed and mean, standard deviation, minimum and maximum points are calculated. After calculating those parameters for each class the average of each calculated parameter is taken. The Table 4.3 below demonstrates those results in detail.

Table 4.3: Ranges for each class.

	Sealevel	**Lowland**	**Mountain**
Min	0 - 0.043	0 - 0.075	0.015 - 0.87
Max	0.08 - 0.25	0.085 - 0.65	0.1 - 0.96
Mean	0.004 - 0.082	0.05 - 0.3	0.3 - 0.6
Std	0.0005 - 0.04	0.0045 - 0.14	0.12 - 0.23
Range in meters	0 - 100	0 - 381	0 - 5251
Range in doubles	0 - 0.1	0 - 0.21	0 - 0.953

There is an if-else statement for each range and the counter increment by one when the parameters of the procedurally generated terrains fall into that ranges. After the counter of each class is compared to each other and the one having the highest number wins and the range conversion happens using that counter class parameter.

For example, let's say we procedurally generated terrain using a Diamond-Square algorithm and the min-max value of the generated heightmap are 0 and 0.23 and the standard deviation and mean are 0.054 and 0.051. Then we will look at a table above and find out that min value falls into the range of sealevel and lowland so the counters of that class are incremented. Looking at the maximum point we can see that it falls into the range of sealevel and lowland again. Till now there is a tie between lowland and sealevel classes. So we continue looking for mean and standard deviation. The mean falls into the sealevel and lowland class again so the counter of both classes incremented and standard deviation falls into the range of lowland only. As a result, the counter of lowland class will be equal to 4 and the counter of sealevel class will be equal to 3 and we are going to select 0 - 381 ranges as meter units and 0 - 0.21 units as double points and convert min and max points to meter using the conversion algorithm discussed in the previous section.

Moreover, if the counter of a class will be equal to another classes counter after the calculation is done, the range of which class to select will be done randomly. When the parameter of generated terrain falls out of range of classes the mountain ranges are used for evaluation.

4.4.4 Evaluation of Procedurally Generated Terrains

The terrains are procedurally generated using Diamond-Square Algorithm and Perlin Noise and the features of these terrains are extracted. In order to understand if the desired terrain is generated via procedural methods, the evaluation of those terrains is done using ANN and CNN. The pseudocode below demonstrates the evaluation process in more detail.

```
Begin Loop
    Generate Terrain using Perlin Noise
    Generate Terrain using Diamond−Square Algorithm
    Test the generated terrains using CNN and ANN
    If (CNN or ANN results >= 80% to the selected class)
        End Loop
    Otherwise continue
End Loop
```

As mentioned in the previous chapter the terrains are classified into three classes, such as sealevel, lowland, and mountain. The desired terrain class is selected by the user and the procedural terrain generation techniques start to generate terrains. The generated terrains will be saved as height-map images and the features of each terrain will be extracted. Then the features are tested using ANN and the height-map images are tested using CNN. If the testing results give higher than 80% of the desired terrain class the process will terminate and the final result will demonstrate after how many iterations the desired terrain is reached.

Chapter 5

Results & Discussion

A discussion of the result obtained with ANN and CNN will be done in this chapter. Which feature combination will give the good test results will be observed. Finally, the results of a height map generated with Diamond-Square Algorithm and Perlin Noise will be tested using ANN and CNN model and the outcomes of those test results will be discussed.

5.1 Feature Combination for ANN

Different feature combination mentioned in the previous chapter were tested using pattern recognition and K-fold cross-validation.

Since weights and biases are randomly generated in the Artificial Neural Network the train results will be different each time the network is trained. The feature combinations are trained ten times for each selected feature combination. The network model, which is created after each training, is used to test the testing images and the average results of the tests are taken in order to understand how well the network model classifies the testing images. To better observe the performance of the network model a varying number of neurons are applied.

The Table 5.1 below demonstrates the results generated using 10 hidden neurons and dividing the samples into 70% of training, 15% validation and 15% testing samples.

Table 5.1: Feature combination network results using 10 hidden neurons.

		Cross - Entropy			Percent Error			Testing Results	
Number of inputs	Description	Training	Validation	Testing	Training	Validation	Testing	True Percentage	Failure Percentage
61	All inputs included	3.456	5.338	5.341	1.68	3.7	2.32	95.75	4.25
45	Circular Rugg. not included	1.255	3.4	3.45	4.04	1.3	4.784	94.4	5.6
51	Hist. One not included	1.67	4.68	4.664	2.9	2.8	3.5	96.64	3.36
52	Hist. Two not included	2.1	5.8	5.87	0.48	3.04	4.8	97.67	2.33
55	Slopes not included	2.66	7.58	7.6	1.152	1.95	6.3	97.67	2.33
45	Zero Rugg. not included	2.07	5.78	5.79	1.68	2.825	1.74	97.35	2.65
29	Circular and Zero Rugg. not included	1.67	4.58	4.567	2	3.26	5.867	94.85	5.15
26	Zero and All Hist. not included	1.145	3.3	3.36	2.88	4.779	6.5	95.9	4.1
35	Zero and Hist One not included	1.77	4.8	4.9	2.3	3.257	5.43	97.66	2.34
36	Zero and Hist two not included	2.03	5.77	5.78	0.576	3.045	2.61	98.8	1.2
39	Zero and Slope not included	1.72	4.7	4.79	2.3	1.52	4.78	97.3	2.7

By analyzing the table above it can be seen that the feature combination that does not contains zero ruggedness and histogram two, which is the histogram with a range of 0 - 9000 meters, gives the good results among other feature combinations while testing images.

To better observe the performance of the network model the number of hidden neurons increased to 20 and the samples are divided as in the previous one. The Table 5.2 demonstrates the results generated using these network parameters.

Observing the Table 5.2 it can be seen that the same feature combination as in the previous table gives the good results among other feature combinations.

The number of hidden neurons increased to 30 and the samples are divided as in the previous one to observe the performance of the network model. The Table 5.3 demonstrates the results generated using these network parameters.

It can be observed from Table 5.3 that the same feature combination as in the previous tables gives the good results among other feature combinations.

Table 5.2: Feature combination network results using 20 hidden neurons.

		Cross - Entropy			Percent Error			Testing Results	
Number of inputs	**Description**	**Training**	**Validation**	**Testing**	**Training**	**Validation**	**Testing**	**True Percentage**	**Failure Percentage**
61	All inputs included	2.27	5.9	5.9	1.87	2.4	3.475	96.54	3.46
45	Circular Rugg. not included	1.768	4.895	4.915	2.155	4.995	6.51	95.4	4.6
51	Hist. One not included	1.7575	4.825	4.814	2.25	5.51	3.915	96.67	3.33
52	Hist. Two not included	1.5	4.17	4.14	0.48	2.174	1.74	97.9	2.1
55	Slopes not included	2.284	6.3	6.3	1.7	3	2.61	97.8	2.18
45	Zero Rugg. not included	2.5	7.1	7.1	1.48	3.9	3.26	97.5	2.5
29	Circular and Zero Rugg. not included	2.24	4.586	6.15	1.4	1.955	5.2	96.9	3.1
26	Zero and All Hist. not included	1.88	5.2	5.15	2.68	4.345	3.475	96.5	3.5
35	Zero and Hist One not included	1.374	3.77	3.77	3.64	3.48	8.2	96.8	3.2
36	Zero and Hist two not included	1.95	5.4	5.4	0.384	2.175	2.175	98.6	1.4
39	Zero and Slope not included	2.66	7.37	7.4	1.344	2.6	3.48	97.85	2.15

Table 5.3: Feature combination network results using 30 hidden neurons.

		Cross - Entropy			Percent Error			Testing Results	
Number of inputs	**Description**	**Training**	**Validation**	**Testing**	**Training**	**Validation**	**Testing**	**True Percentage**	**Failure Percentage**
61	All inputs included	2.0757	5.784	5.7987	1.104	2.822	6.085	98.84	1.16
45	Circular Rugg. not included	1.9317	5.356	5.379	2.016	4.563	4.994	95.37	4.637
51	Hist. One not included	1.745	4.929	4.946	2.304	2.169	6.518	98.99	1.01
52	Hist. Two not included	2.8595	8.233	8.201	0.096	2.175	3.915	99.07	0.93
55	Slopes not included	2.114	5.903	5.937	1.584	3.475	4.131	98.54	1.46
45	Zero Rugg. not included	2.6017	7.303	7.321	1.392	1.955	3.477	98.92	1.08
29	Circular and Zero Rugg. not included	2.63	5.02	5.04	1.778	3.912	4.129	95.71	4.29
26	Zero and All Hist. not included	1.67	4.581	4.613	3.072	4.78	6.515	97.42	2.58
35	Zero and Hist One not included	1.626	4.444	4.419	3.218	3.476	4.132	98.15	1.85
36	Zero and Hist two not included	2.448	6.964	6.982	0.288	3.045	3.915	99.1	0.9
39	Zero and Slope not included	2.345	6.621	6.6	1.488	4.13	3.045	98.57	1.43

To make final observation on the performance of the network model the number of neurons increased to 40 and the samples are divided as in the previous one. The Table 5.4 demonstrates the results generated using these network parameters.

From the Table 5.4 it can be observed that the same feature combination as in the previous tables gives the good results among other feature combinations.

Table 5.4: Feature combination network results using 40 hidden neurons.

		Cross - Entropy			Percent Error			Testing Results	
Number of inputs	**Description**	**Training**	**Validation**	**Testing**	**Training**	**Validation**	**Testing**	**True Percentage**	**Failure Percentage**
61	All inputs included	2.726	7.732	7.775	1.248	1.52	7.169	98.95	1.05
45	Circular Rugg. not included	1.532	4.214	4.207	2.736	2.175	4.567	95.17	4.83
51	Hist. One not included	1.699	4.743	4.718	3.124	2.17	3.695	98.66	1.34
52	Hist. Two not included	2.022	5.697	5.699	0.958	3.48	1.305	98.23	1.77
55	Slopes not included	2.431	6.846	6.858	1.104	4.125	3.042	98.58	1.42
45	Zero Rugg. not included	2.502	7.18	6.994	1.584	2.39	3.912	98.45	1.55
29	Circular and Zero Rugg. not included	2.39	6.574	6.592	1.488	4.127	5.215	96.22	3.78
26	Zero and All Hist. not included	1.652	4.597	4.618	2.784	4.785	6.52	97.7	2.3
35	Zero and Hist One not included	1.812	5.012	5.016	3.648	3.04	2.605	97.33	2.67
36	Zero and Hist two not included	2.069	5.879	5.91	1.152	2.61	3.915	99.19	0.81
39	Zero and Slope not included	2.973	8.292	8.331	2.682	4.346	4.347	97.9	2.1

The results above were achieved using the pattern recognition neural network toolbox of MATLAB. The other way to test the feature combination is by using K-fold cross-validation. The K-fold cross-validation is initialized with the 10 and 20 number of hidden neurons and setting the number of a fold to ten. The results of the testing sample using this method are shown in the Table 5.5.

According to the results provided in the Table 5.5 the good combination of feature among other using K-fold cross-validation method is the same as the result of the pattern recognition built-in tool of Matlab. The performance of network, test, and train are calculated by taking the average of all folds. Since ten folds were given the performance results and test results demonstrate the average results of those folds.

After performing pattern recognition and K-fold cross-validation tests on the feature combinations the features zero ruggedness and histogram two should not be included in our final network model, which is going to be used for testing of procedurally generated height maps.

Table 5.5: Results of the 10-folds cross-validation

Neural Pattern Recognition with K Fold (K=10)							
		NN Input Parameters	**Performance**			**Testing Results**	
Number of Inputs	**Description**	**Number of Hidden Neurons**	**Network**	**Test**	**Train**	**True Percentage**	**Failure Percentage**
61	All Inputs Included	10	5.30E-05	11.7341	1.2003	95.44	4.56
61	All Inputs Included	20	0.0024	12.2504	1.2554	95.97	4.03
45	Circular Ruggetness Not Included	10	4.82E-06	14.3115	1.6393	94.53	5.47
45	Circular Ruggetness Not Included	20	8.55E-08	13.0172	1.4604	95.12	4.88
51	Histogram One Not Included	10	3.39E-07	14.6089	1.7596	93.42	6.58
51	Histogram One Not Included	20	0.0089	13.6497	1.4823	96.31	3.69
52	Histogram Two Not Included	10	2.35E-06	11.5945	1.2017	97.95	2.05
52	Histogram Two Not Included	20	1.44E-07	12.6	1.4848	97.53	2.47
55	Slopes Not Included	10	0.0017	11.9917	1.3858	95.21	4.79
55	Slopes Not Included	20	9.71E-06	11.302	1.2891	94.65	5.35
45	Zero Ruggetness Not Included	10	3.03E-07	11.2702	1.2509	96.23	3.77
45	Zero Ruggetness Not Included	20	1.37E-07	11.7486	1.227	95.12	4.88
29	Circular and Zero Ruggetness Not Included	10	7.45E-08	14.468	1.7017	94.54	5.46
29	Circular and Zero Ruggetness Not Included	20	0.0037	14.207	1.7113	93.876	6.124
26	Zero and All Hist Not Included	10	4.89E-03	14.4452	1.6955	92.4	7.6
26	Zero and All Hist Not Included	20	3.32E-07	12.9554	1.4099	93.21	6.79
35	Zero and Hist One Not Included	10	3.52E-07	14.4524	1.5242	96.432	3.568
35	Zero and Hist One Not Included	20	0.0044	12.8297	1.4708	95.432	4.568
36	Zero and Hist Two Not Included	10	7.27E-08	10.8152	1.2184	98.553	1.447
36	Zero and Hist Two Not Included	20	7.91E-04	11.2019	1.2617	98.323	1.677
39	Zero and Slope Not Included	10	1.81E-07	11.2927	1.2583	96.342	3.658
39	Zero and Slope Not Included	20	1.72E-07	11.9321	1.3896	96.974	3.026

To create the final neural network model 36 generated features were used, zero ruggedness and histogram two were excluded, with ten number of hidden neurons and by dividing the samples into 70% of training, 15% validation and 15% testing samples.

The figures below demonstrate the ROC and confusion matrix of the trained network model.

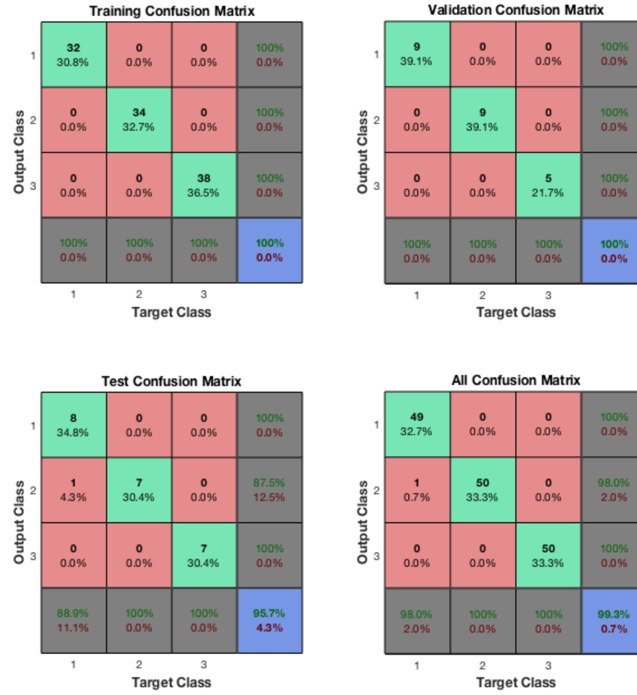


Figure 5.1: Confusion Plot.

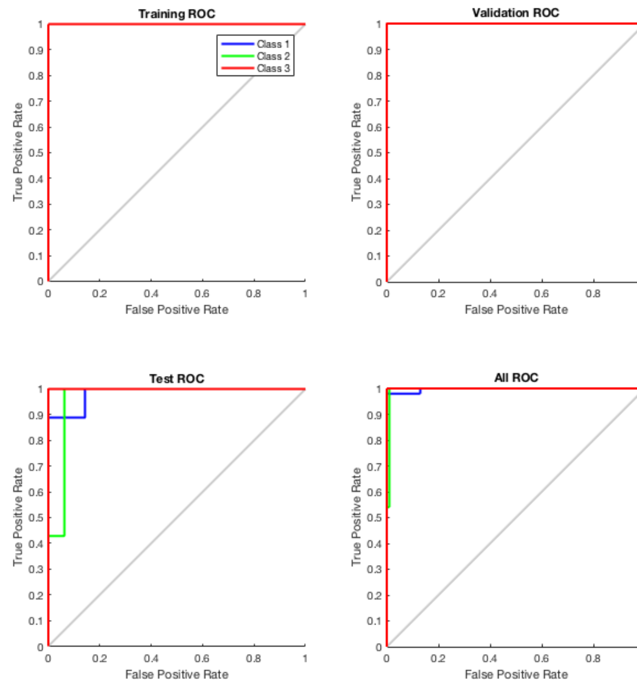


Figure 5.2: ROC Plot.

It can be observed that there was one confusion in the test sample and the sealevel was confused by lowland in that sample. However, in general, the network model gives good results. This model is saved for testing procedurally generated height maps.

5.2 Convolutional Neural Network Test

The Convolutional Neural Network was created using a TensorFlow library. The number of input images was augmented in order to train the CNN. The test image was also augmented and after the augmentation, the number of test images increased to 360, each class containing 120 images. The CNN model was created with 80% of the image samples to train and the remaining 20% of the image samples were used to validate the network. The following results are obtained during the training and validation of the CNN.

```
(targetDirectory) ganirahmon CNN-tf-8-bit-augment $ python train.py
Reading training images
Reading sealevels images (Index: 0)
Reading lowlands images (Index: 1)
Reading mountains images (Index: 2)
Reading input images complete.
Training-set images number:   3840
Validation-set images number: 960

Epoch 1 --- Training Accuracy: 33.3%, Validation Accuracy: 29.2%, Validation Loss: 1.090
Epoch 2 --- Training Accuracy: 95.8%, Validation Accuracy: 89.6%, Validation Loss: 0.159
Epoch 3 --- Training Accuracy: 95.8%, Validation Accuracy: 89.6%, Validation Loss: 0.136
Epoch 4 --- Training Accuracy: 95.8%, Validation Accuracy: 95.8%, Validation Loss: 0.120
Epoch 5 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.113
Epoch 6 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.106
Epoch 7 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.102
Epoch 8 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.098
Epoch 9 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.095
Epoch 10 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.089
.
.
.
```

```
Epoch 25 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.057
Epoch 26 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.053
Epoch 27 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.058
Epoch 28 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.048
Epoch 29 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.047
Epoch 30 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.046
Epoch 31 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.043
Epoch 32 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.054
Epoch 33 --- Training Accuracy: 97.9%, Validation Accuracy: 97.9%, Validation Loss: 0.057
Epoch 34 --- Training Accuracy: 97.9%, Validation Accuracy: 100.0%, Validation Loss: 0.033
.
.
Epoch 46 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.005
Epoch 47 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.004
Epoch 48 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.005
Epoch 49 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.004
Epoch 50 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.005
```

When entire samples are passed forward and backward inside the network once will form an epoch. In each epoch, the training, validation accuracy and validation loss are calculated. The predicted and true values are used to calculate the accuracy of training and validation. The results demonstrate that the training process in the network is going in the right direction, which is indicated by the increase in accuracy at each epoch.

After creating the CNN model test image samples were used to make the test on the generated model. The Table 5.6 below show the results generated after testing test image samples.

Table 5.6: CNN test image samples results

		Test Samples		
		Sealevel	**Lowland**	**Mountain**
Results	**Sealevel**	81.57%	2.06%	0.00%
	Lowland	18.43%	97.94%	7.44%
	Mountain	0.00%	0.00%	92.56%

Overall Results	
True Percentage	90.69%
Failure Percentage	9.31%

The CNN model has the overall of 90.69% in evaluating the test samples correctly. There are some problems in evaluating sealevel. The model confuses sealevel with lowland in some cases, but the overall result is good and the test of procedurally generated terrains is done using this CNN model.

5.3 Evaluation of Procedurally Generated Terrains

To procedurally generate terrains two techniques are used, which are Diamond-Square Algorithm and Perlin Noise. The terrain generated with these algorithms are saved as 8-bit PNGs. The feature extraction algorithm is used to extract features of the generated terrain.

The evaluation of the generated terrain is done using ANN and CNN. If we want to have a sealevel terrain as a result, the algorithm will generate terrain, save as height map image and extract features. Then the features are tested using ANN model and height map image is tested using CNN model. If the result of the testing gives higher than 80% for that target class generation of new terrains stops and the number of iterations to reach the target terrain is observed.

The Table 5.7 demonstrates after how many iterations the sealevel is generated using Diamond-Square Algorithm.

Table 5.7: Sealevel generation using Diamond-Square Algorithm.

	ANN Result						
	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6	Iteration 7
sealevel	0	0	0	0.1231	0	0	0.9176
lowland	1	1	1	0.8769	1	1	0.0824
mountain	0	0	0	0	0	0	0
	CNN Result						
	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6	Iteration 7
sealevel	1.80E-05	5.37E-18	8.02E-18	5.77E-07	9.08E-15	9.36E-10	8.90E-01
lowland	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.10E-01
mountain	1.89E-05	1.20E-13	1.76E-12	1.27E-09	1.39E-13	3.21E-10	1.09E-04

It can be seen that the sealevel is generated at seventh iteration and the previous iterations generated lowland terrains.

To render the generated height-map different colors used to represent different regions. The colors region were accurately identified by testing the original height-maps. The Figure 5.3 shows colors representing each regions to render the height-map.

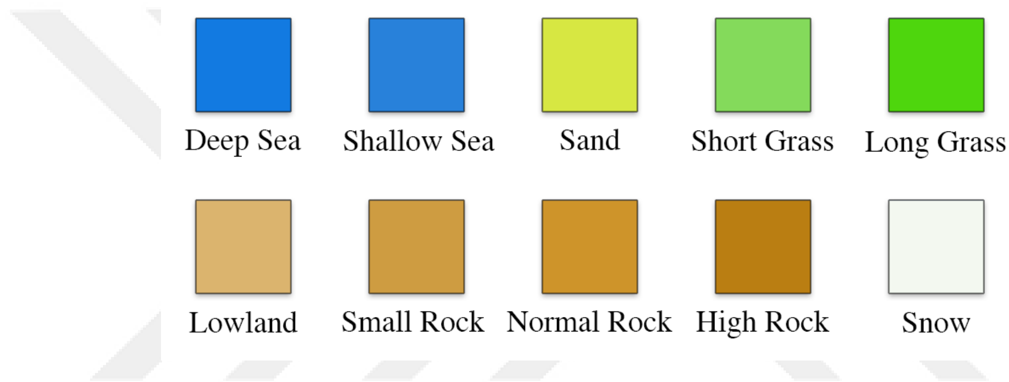


Figure 5.3: Color region representation.

The Figure 5.4 demonstrates the height-map generated at seventh iteration as a two-dimensional image on the left and the matching three-dimensional rendering on the right. To generate and render height-map images Unity3D was used.

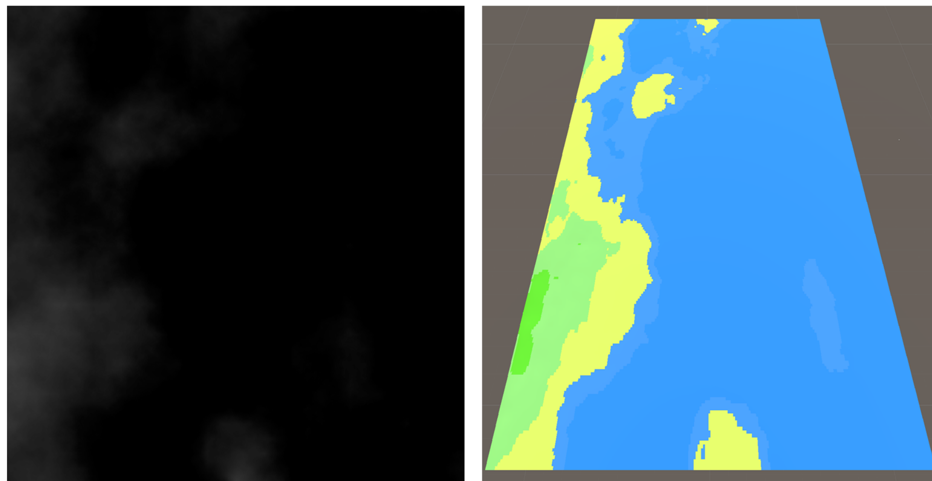


Figure 5.4: 2D height-map image (left). 3D render of height-map image (right).

The Table 5.8 demonstrates after how many iterations the lowland is generated using Diamond-Square Algorithm.

Table 5.8: Lowland generation using Diamond-Square Algorithm.

	ANN Result					
	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6
sealevel	1	0.9972	0.9999	1	1	0
lowland	0	0.0028	0.0001	0	0	1
mountain	0	0	0	0	0	0
	CNN Result					
	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6
sealevel	8.80E-01	1.00E+00	1.00E+00	1.00E+00	6.62E-01	7.67E-13
lowland	1.20E-01	2.30E-06	1.75E-04	2.73E-06	3.37E-01	1.00E+00
mountain	4.56E-04	2.91E-06	3.60E-06	1.11E-06	5.30E-04	3.06E-12

It can be seen that the lowland is generated at sixth iteration and the previous iterations generated sealevel terrains. Also, it can be observed that our Diamond-Square Algorithm is good in generating lowland and selevel, but not good in generating mountain.

The Figure 5.5 demonstrates the height-map generated at sixth iteration as a two-dimensional image on the left and the matching three-dimensional rendering on the right.

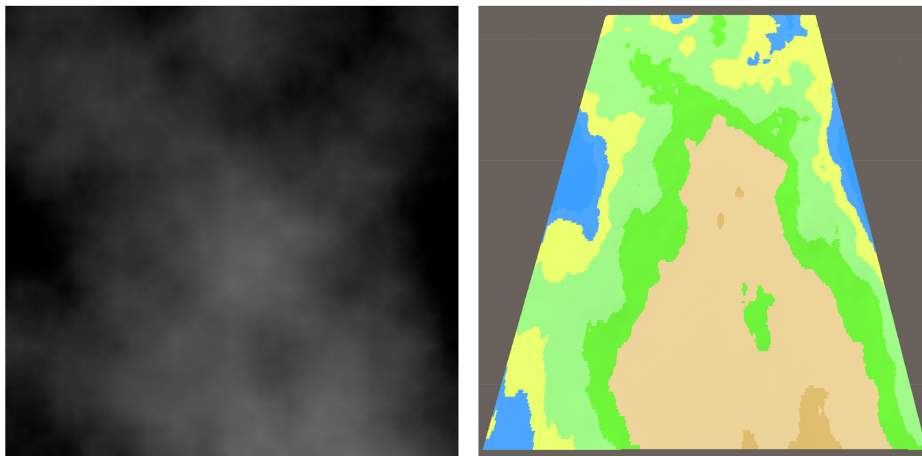


Figure 5.5: 2D height-map image (left). 3D render of height-map image (right).

The Table 5.9 demonstrates after how many iterations the mountain is generated using Perlin Noise.

Table 5.9: Mountain generation using Perlin Noise

	ANN Result			
	Iteration 1	Iteration 2	Iteration 3	Iteration 4
sealevel	0	0	0	0
lowland	0	0	0	0
mountain	1	1	1	1
	CNN Result			
	Iteration 1	Iteration 2	Iteration 3	Iteration 4
sealevel	0.00E+00	0.00E+00	0.00E+00	0.00E+00
lowland	5.44E-19	8.66E-15	1.86E-12	1.68E-20
mountain	1.00E+00	1.00E+00	1.00E+00	1.00E+00

It can be observed that the mountain is generated in first iteration and the iterations after that also generate mountain regions. For generating mountain Perlin Noise is a good choice.

The Figure 5.6 demonstrates the height-map generated at first iteration as a two-dimensional image on the left and the matching three-dimensional rendering on the right.

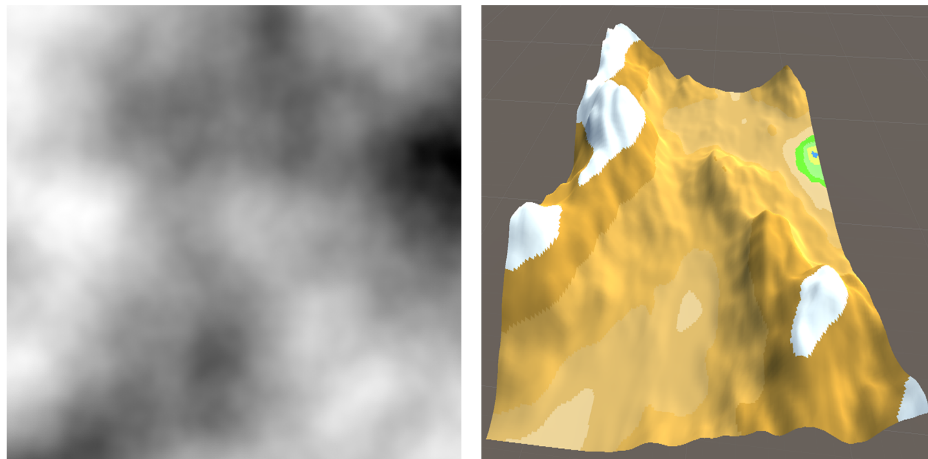


Figure 5.6: 2D height-map image (left). 3D render of height-map image (right).

Chapter 6

Conclusion & Future Work

Procedural terrain generation techniques produce terrains which are usually fractal in nature, but the main disadvantage of this techniques is that they are difficult to control. Because of this, the evaluation with the help of neural network helps to interpret the generated terrains correctly and hence generate the type of terrain that is required. Since procedural terrain generation techniques generate terrains using pseudo-random number generator, each time different terrain will be generated. Because of it the targeted type of the terrain sometimes may take longer iterations to be generated.

The contribution of this study is that with the help of Artificial and Convolutional Neural Networks procedurally generated terrains are evaluated to match the user requirements. In addition, Artificial and Convolutional Neural Networks are trained with the real-world map data to make an interpretation.

Diamond-Square algorithm and Perlin noise were used to generate terrains and the outcomes of those algorithms were evaluated. After the evaluation, it can be observed that Diamond-Square algorithm tends to generate terrain of sealevel and lowland class. However, Perlin noise tends to generate terrain of mountain class.

There are many aspects to improve for the future works. New features can be extracted to improve the working of Artificial Neural Network and to interpret the result more correctly. Instead of increasing the number of real-world map data using augmentation the new real-world map data can be obtained from sources. By doing so the Convolutional Neural Network will give the more correct result while doing an interpretation of the terrains. Some improvements in the architecture of the Convolutional Neural Network can be done to make network model predict correctly. The normalization of the generated terrains using Perlin Noise can be done in order to generate a different type of the terrain classes.

BIBLIOGRAPHY

[1] US Entertainment Software Association (ESA). 2017. *Two-Thirds of American Households Regularly Play Video Games*. Article. Available from: <http://www.theesa.com/article/two-thirds-american-households-regularly-play-video-games/>. [19 April 2017].

[2] Togelius, J., N. Shaker, and M. J. Nelson. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. Chapter 1: Introduction, 1-15.

[3] Hendrikx, M. et al. 2013. *Procedural Content Generation for Games: A Survey*. ACM Trans. Multimedia Comput. Commun. Appl. 9, 1, Article 1 (February 2013), 22 pages.

[4] Greeff, G. 2009. "Interactive voxel terrain design using procedural techniques." MSc Thesis. University of Stellenbosch.

[5] Fowler, J. Robert, and James J. Little. 1979. *Automatic Extraction of Irregular Network Digital Terrain Models*. SIGGRAPH '79 Conference Proceedings. August, 1979. vol. 13 ; no. 2: pp. 199-207.

[6] Crause, J. 2015. "Fast, Realistic Terrain Synthesis." MSc Thesis. University of Cape Town.

[7] Fournier, A., D. Fussell, and L. Carpenter. 1982. *Computer Rendering of Stochastic Models*. In Communications of the ACM. Vol. 25. No. 6. ACM Press. New York, NY. 371-384.

[8] Christopher, E. 2015. *Visualization of the Diamond Square Algorithm*. Available from:

<https://commons.wikimedia.org/wiki/File:Diamond_Square.svg>. [20 August 2015].

[9] Travis, A. 2011. "Procedurally Generating Terrain." Study. Morningside College.

[10] Perlin, K. 1985. *An Image Synthesizer*. In Proceedings of SIGGRAPH '85. ACM Press. New York, NY. 287-296.

[11] Perlin, K. 2001. *Noise hardware*. In Real-Time Shading SIGGRAPH Course Notes. Olano M., (Ed.).

[12] Worley, S. 1996. *A cellular texture basis function*. Proceedings of the 23rd annual conference on computer graphics and interactive techniques. acm.org. pp. 291-294.

[13] Rose, T. J., and A. G. Bakaoukas. 2016. *Algorithms and Approaches for Procedural Terrain Generation - A Brief Review of Current Techniques*. 2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES), Barcelona, pp. 1-2.

[14] Raffe, W., F. Zambetta, and X. Li. 2011. *Evolving patch-based terrains for use in video games*. Genetic and Evolutionary Computation Conference, GECCO'11. 363-370.

[15] Zhou, H., J. Sun, G. Turk, and J. M. Rehg. 2007. *Terrain Synthesis from Digital Elevation Models*. in IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 4, pp. 834-848.

[16] Saunders, L. R. 2006. "Terrainosaurus: Realistic Terrain Synthesis Using Genetic Algorithms." MSc Thesis. Texas A&M University.

[17] Qicheng, L., G. Wang, F. Zhou, X. Tang, and K. Yang. 2006. *Example-Based Realistic Terrain Generation*. In Proceedings of the 16th international conference on Advances in Artificial Reality and Tele-Existence (ICAT'06), Zhigeng P., A. Cheok, M. Haller, R. H. Lau, and H. Saito (Eds.). Springer-Verlag, Berlin, Heidelberg, 811-818.

[18] Digne, J. et al. 2017. *Interactive example-based terrain authoring with conditional generative adversarial networks*. ACM Trans. Graph. 36, 6, Article 228, 13 pages.

[19] Teong, J. O., R. Saunders, J. Keyser, and J. J. Leggett. 2005. *Terrain generation using genetic algorithms.*GECCO '05, Hans-Georg Beyer (Ed.). ACM, New York, NY, USA, 1463-1470.

[20] Terrain Party. *The easiest way to get real-world height maps for Cities: Skylines*. Available from: <http://terrain.party/>. [27 November 2017].

[21] e-Handbook of Statistical Methods. 2012. *Engineering Statistics Handbook. Chapter 1.3.5.11: Measures of Skewness and Kurtosis*. Available from: <https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm>. [April 2012].

[22] Gözde, Y. T. 2017, "Advanced Data Analysis", lecture notes distributed in Advanced Data Analysis MATH 658 at Izmir University of Economics, Izmir on 15 March 2017.

[23] Ujjwalkarn. 2016. *A Quick Introduction to Neural Networks*. Available from: <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/>. [9 August 2016].

[24] Vzquez, F. 2017. *Deep Learning made easy with Deep Cognition*. Available from: <https://becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351>. [21 December 2017].

[25] Fauset, L. 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Laurene Fausett (Ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[26] Ujjwalkarn. 2016. *An Intuitive Explanation of Convolutional Neural Networks*. Available from: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>. [11 August 2016].

[27] Official TensorFlow Website. 2018. Available from: <https://www.tensorflow.org/>.