2-11-2022 4:00 PM

Autonomous Rock Segmentation from Lidar Point Clouds Using Machine Learning Approaches

Lauren E. Flanagan, *The University of Western Ontario*

Supervisor: McIsaac, Kenneth, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering

Abstract

Rover navigation on planetary surfaces currently uses a method called blind drive which requires a navigation goal as input from operators on Earth and uses camera images to autonomously detect obstacles. Images can be affected by lighting conditions, are not highly accurate from far distances, and will not work in the dark; these factors negatively impact the autonomous capabilities of rovers. By improving a rover's ability to autonomously detect obstacles, the capabilities of rovers in future missions would improve; for example, enabling exploration of permanently shadowed regions, and allowing faster driving speeds and farther travel distances. This thesis demonstrates how Lidar point clouds can be used to autonomously and efficiently segment planetary terrain to identify obstacles for safe rover navigation. Two Lidar datasets which represent planetary environments containing rock obstacles and sandy terrain were used to train a neural network to perform semantic segmentation. The neural network was based on the RandLA-Net architecture that was designed to efficiently perform semantic segmentation on point clouds using a random sampling algorithm without modifying the point cloud structure. Methods to handle the class imbalance of the datasets were explored to enable the model to learn the minority class and to optimize the model's performance. The model achieved a recall score of 94.46 % and precision score of 84.93 % at a frame rate of 0.6238 seconds/pointcloud on an Intel Xeon E5-2665 CPU, indicating that it is possible to use Lidar point clouds to perform semantic segmentation on-board planetary rovers with similar compute capabilities.

Keywords: Lidar Point Clouds, Semantic Segmentation, Machine Learning, Neural Networks, Planetary Science

Lay Summary

Planetary rovers are designed to explore the solid surfaces of planets such as Mars, and other planetary masses such as the Moon. Because of the distance between Earth and other planets and planetary masses, it can take a long time to communicate with a rover from Earth, so the rover cannot be driven by operators on Earth. Instead, operators on Earth give the rover a location to drive to, and the rover must detect and avoid harmful obstacles on its own, such as rocks, cracks, or cliffs. The current way of locating obstacles is done by using camera images: the rover finds the location of the obstacles in the images and avoids them while it drives. Camera images can be affected by lighting conditions, are not very accurate from far distances, and will not work in the dark. This research proposes using a Light Detection and Ranging (Lidar) sensor rather than a camera to find obstacles. Lidar sensors create point clouds by measuring the distance to their surroundings using laser beams, so they are not affected by lighting conditions, do not require a light source, and are more accurate from distances than cameras. One of the reasons that Lidars have not been used on rovers in the past is because they need a lot of computer power to find obstacles within the point clouds that Lidars make; however, new techniques have been created to reduce the amount of computer power needed to find obstacles in point clouds making it more realistic to use a Lidar on a rover to find obstacles. Datasets that contain rocks and sandy ground to represent planetary surfaces were used to train machine learning models to find the rock obstacles in the planetary scenes. Overall, the machine learning model was able to correctly separate the rock obstacles from the ground and implies that Lidars could be used on future rover missions to improve the rover's ability to detect obstacles.

Acknowledgements

I would like to thank my supervisor Dr. Ken McIsaac for his continuous support and for providing the opportunity to perform this research. I would like to thank Dr. Matt Cross for his extensive knowledge and guidance provided throughout this process. I want to thank my research group for their friendship and advice, and for the online forum discussions about point clouds and machine learning which truly helped to shape my research.

To my supervisors at the European Space Agency, Levin Gerdes and Martin Azkarate, thank you for your invaluable guidance and support during my internship and for encouraging me to fully explore my research topic. To my colleagues at the European Space Agency, thank you for being interested in my project and ambitions, and for making my internship an unforgettable experience.

Finally, to my family, partner, and friends, thank you for your continuous support during this process and for always encouraging me to pursue opportunities that I am passionate about.

Contents

List of Figures

List of Tables

List of Acronyms

ATF Analogue Terrain Facility

CNN Convolutional Neural Network

CSA Canadian Space Agency

DGPS Differential Global Positioning System

ESA European Space Agency

HazCams Hazard Avoidance Cameras

KNN K-Nearest Neighbours

Lidar Light Detection and Ranging

LocSE Local Spatial Encoding

mIOU Mean Intersection Over Union

MRG Multi-Resolution Grouping

MSG Multi-Scale Grouping

NASA National Aeronautics and Space Administration

NavCams Navigation Cameras

NN Neural Network

RGB Red, Green, Blue

ROS Robot Operating System

SMOTE Synthetic Minority Oversampling Technique

UGV Unmanned Ground Vehicle

UTM Universal Transverse Mercator

List of Appendices

Chapter 1

Introduction

1.1 Motivation

The motivation of this thesis is to determine if point cloud data from a Light Detection and Ranging (Lidar) sensor can be used to efficiently segment terrain for safe rover navigation. The goal is to use datasets that represent a planetary environment to identify rock obstacles by performing semantic segmentation of lidar point clouds using machine learning techniques.

Due to communication delay, rover operators on Earth cannot operate rovers on other planetary surfaces in real time. For example, the communication delay between Earth and Mars ranges from five to twenty minutes depending on the distance between the planets based on their positions in orbit [1]. Because of the communication delay, the current method of navigating planetary rovers is to use a blind drive method, where a goal position is set by the operators on Earth, and the rover navigates itself to the goal position. However, this method is not sustainable for future missions with more complex requirements such as increased driving speeds and distances, or for groups of rovers working together. Shreyansh Daftry, a robotics technologist at the National Aeronautics and Space Administration (NASA) Jet Propulsion Laboratory, states that "There's a growing need for future spacecraft to be autonomous, self-aware and have the ability to make critical decisions on their own. ... The only way that we can

scale up the space economy is if we can make our space assets self-sustainable. And artificial intelligence is going to be a key ingredient in making that happen." [2]. Overall, rovers must be capable of detecting natural obstacles including rocks, craters, cliffs, trenches, and sand dunes, and navigating through the obstacles on their own.

A current method for geometric obstacle detection on planetary surfaces is to use cameras. NASA's most recent Mars rover, *Perseverance*, is equipped with six Hazard Avoidance Cameras (HazCams), and two Navigation Cameras (NavCams)[3]. The HazCams provide front and rear views of close by hazards such as large rocks, trenches, or sand dunes [3]. The NavCams are used to detect far away obstacles to detect a safe path for blind drives; the cameras can detect golf ball sized obstacles from 25 meters away [3]. Both the NavCams and the HazCams are stereo cameras, meaning that depth information can be extracted from the image.

Problems can arise when using camera images due to the varying lighting conditions: large shadows can be cast by obstacles making it difficult to apply image processing techniques, lens flares can occur when there is a bright light source, and driving at night or in permanently shadowed regions is not possible without an additional light source. Lidar, on the other hand, is unaffected by lighting conditions since they work by sending out pulsed lasers to measure the distance of surrounding surfaces. Lidar provides a 360-degree view of it's surroundings and can detect surfaces up to 200 meters away.

Previously, using point cloud data required pre-processing techniques that use a large amount of memory and processing power. When working with planetary rovers, both the memory and processing power are limited, making it unrealistic to use Lidar on a rover. However, with recent advancements in machine learning techniques, point clouds can be fed into a Neural Network (NN) without any pre-processing steps beforehand, reducing the required computational power [4]. These advancements make it feasible to use Lidar with the limited computational resources on board a rover.

The ability to autonomously detect rock obstacles using Lidar would provide the following benefits to planetary rover missions:

- Allow for safe rover navigation
- Remove obstacle detection challenges created by lighting conditions
- Produce a detailed map of the rover's surroundings

Currently, Lidar sensors have not been implemented on rovers, and machine learning has been implemented on rovers for autonomous navigation and to sort through collected data to determine which data has significant information that should be sent back to Earth. The Canadian Space Agency (CSA), the European Space Agency (ESA), and NASA have expressed interest in implementing Lidar on future missions and expanding upon the current machine learning techniques. CSA currently uses Lidar on a test rover to detect obstacles and create a traversability map [5]. Additionally, CSA has stated they are working on implementing machine learning algorithms for terrain assessments, and in the future, this may be integrated into the autonomous navigation ecosystem [5]. ESA has designed a system that would enable a rover to explore a permanently shadowed region on the moon using a Lidar, while being powered by a laser from a near by base station [6]. Lidar is ideal to use for this task because it does not require a source of light in order to operate. NASA has published a request for information for a spaceflight-qualified Lidar to be used for a rover mission on the moon. The purpose of the Lidar would be to detect natural terrain hazards. A secondary purpose of the Lidar would be to perform relative localization [7]. NASA has also performed a study on High Performance Spaceflight Computing to allow more complex algorithms to be implemented on-board a rover, such as machine learning algorithms for machine vision and path planning to increase the driving distance that a rover is capable of [8].

1.2 Task and Thesis Contribution

This research explores the feasibility of using machine learning to identify natural hazards on-board a rover in a planetary environment. Lidar data collected in an environment that has been set up to model a planetary environment is used to train a machine learning classification model that can perform per point inference on each point cloud. The classification model is built to use a small amount of processing power by applying the RandLA-Net model which makes use of a random sampling algorithm to lower the computational complexity of the network. The ability to perform inference on point clouds with a low computational complexity makes it possible to deploy the model in real time on-board a rover with limited resources.

The methods being explored this research are a potential new method of hazard detection to be used on-board planetary rovers for determining a safe path of navigation.

1.3 Thesis Outline

This thesis is organized as follows: Chapter 2 contains background information on lidar point clouds and methods for classifying lidar point clouds, as well as a literature review on common techniques used for classifying point clouds. Chapter 3 provides a description of the Lidar datasets used in this thesis and the methods used to determine per-point ground truth labels of the point clouds. Chapter 4 describes the method of developing a model for segmenting rocks from point cloud scenes that represent planetary environments and the method used to test the model's ability to be deployed for real-time use on-board a rover. Chapter 5 presents the results of semantic segmentation on the datasets and the time taken by the model to perform semantic segmentation. Chapter 5 also contains a discussion of the presented results. Chapter 6 presents a summary of the work performed and possible directions for future work.

Chapter 2

Background and Literature Review

This chapter contains relevant background information on point clouds and the techniques used to perform semantic segmentation on point clouds. Specifically, point cloud structure, generation, features, and processing techniques are discussed along with traditional and machine learning based segmentation methods.

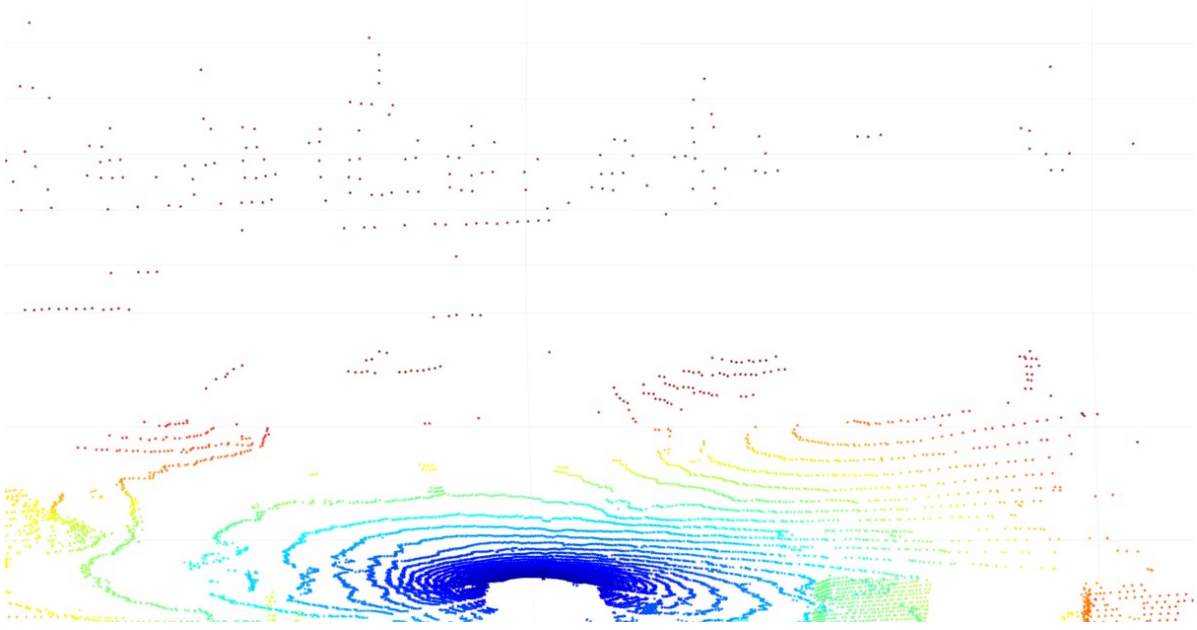
2.1 Point Clouds

2.1.1 An Introduction to Point Clouds

A point cloud is a set of points that represents surfaces in a three-dimensional space. The points are unordered, and the spacing between each point is arbitrary. The point locations are not constrained by a grid, instead, the possible position of each point is continuous, meaning that there are infinite possible locations for a point. Figure 2.1 shows a comparison of an image and a section of a point cloud taken from the same perspective as the image. Figure 2.1a displays the image that is made up of regularly-spaced pixels. The foreground of the image shows sand and rocks, and the background reveals a person standing on a hill lined with trees. Figure 2.1b shows that the spacing of points is not regular; there are dense areas with many points, and sparse areas with few or no points. Even though the spacing of the points are not

regular like the pixels of the image, it is still possible to see the ground plane in the foreground of the point cloud and the person and trees in the background.

(a) Webcam image



(b) Point cloud scan

Figure 2.1: Comparison of a camera image and Lidar point cloud taken from the same point of view at the Canadian Space Agency's Analogue Terrain Facility where the point cloud is coloured based on radial distance

In the simplest form of a point cloud, each point contains information about its distance relative to the sensor that captured the point. The distance information is given as a three-dimensional Cartesian coordinate location and can be represented in matrix form as shown in Equation 2.1, where x_i is a vector containing the Cartesian coordinates and N is the number of points in the point cloud [9].

$$P = \left\{ x_i \, \epsilon \, \mathbb{R}^3 \right\}_{i<N} \tag{2.1}$$

A sample point cloud of a teapot that is plotted on a Cartesian grid is shown in Figure 2.2. The points in the image are coloured based on their height within the image. Depending on the type of sensor used, the data points may contain additional information about the intensity, reflectivity, and/or colour of the surface that the point represents. When these additional features are recorded, the point cloud is then represented by Equation 2.2, where f_i is a feature vector with D dimensions [9].

$$P_F = \left\{ (x_i, f_i) \mid x_i \, \epsilon \, \mathbb{R}^3, f_i \, \epsilon \, \mathbb{R}^D \right\}_{i<N} \tag{2.2}$$

Figure 2.3a shows a point cloud scan of a picnic basket, in which the point cloud contains Red, Green, Blue (RGB) feature information. Figure 2.3b shows an aerial point cloud scan that includes an intensity feature that is displayed as the greyscale colour of the point cloud.

2.1.2 Point Cloud Generation

Point clouds can be generated by range sensors such as time-of-flight sensors and depth cameras. Time-of-flight sensors work by sending out laser beams and capturing information from the laser beams that is returned after reflecting off of a surface. Point clouds generated by time-of-flight sensors are usually sparse in comparison to point clouds generated by depth cameras. Depth cameras capture the depth information about each pixel in an image, and depending on the type, may capture grey-scale or RGB information. The precision can degrade when using

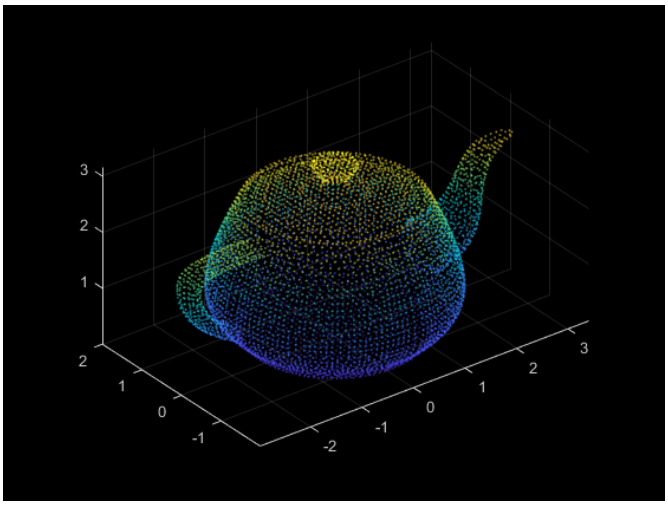
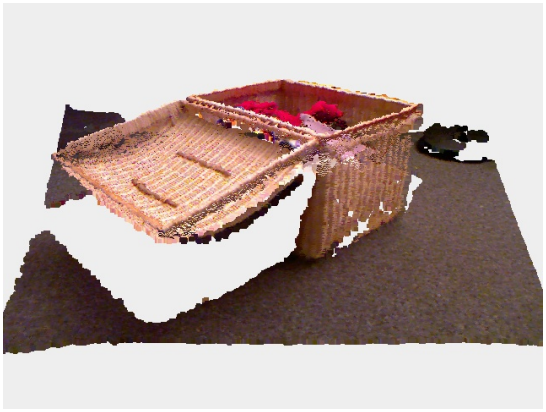
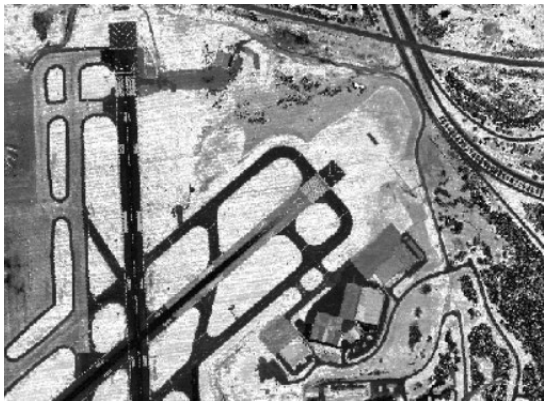


Figure 2.2: Sample point cloud scan. Image generated using MATLAB, taken from the standard MATLAB library.



(a) Point Cloud containing RGB features [10]



(b) Point cloud containing intensity features displayed through the grey-scale colour value [11]

Figure 2.3: Sample Point Cloud Scans showing possible features

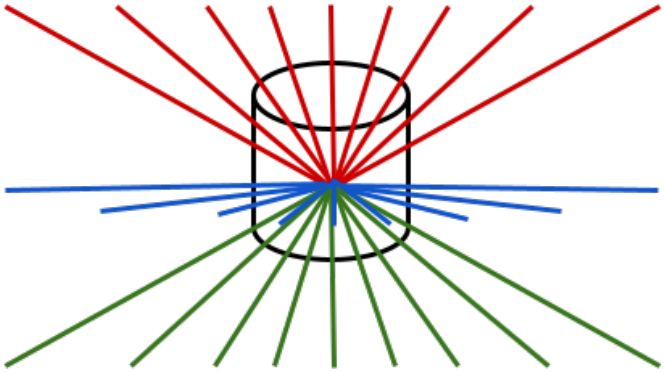


Figure 2.4: Example Lidar scan with a vertical resolution of three and a horizontal resoltion of nine

depth cameras outdoors due to sunlight which may create artifacts from the sunlight interfering with the camera [9]. Lidars are a type of time-of-flight sensor that have centimeter precision over large distances. Lidars work by sending out a laser beam and measuring the time the beam takes to return to the sensor, in order to compute the distance of the object which the beam reflected off of. With each scan performed by a Lidar, it sends out rows (also called channels) of beams, and each row contains a fixed number of points defined by the horizontal resolution. The number of rows of beams sent out by the Lidar is determined by the Lidar's vertical resolution. Figure 2.4 is an example of a Lidar scan with a vertical resolution of three and a horizontal resolution of nine. Although there are a fixed number of rows containing a fixed number of points, the position of the points depends on the position of the surrounding surfaces, which results in unordered data points that are not in a uniform grid.

2.1.3 Point Cloud Features

Features can be extracted from point clouds to determine information about its scene. An important feature for this thesis is the surface normal; it can be computed locally to determine a normal vector for each point by finding the normal to the least squares best fitting plane defined by the point and its K-Nearest Neighbours (KNN) [12]. The direction of the normal

vector cannot be mathematically determined, so it is standard to define the direction of the vectors as pointing towards the sensor to achieve a consistent plane orientation.

2.1.4 Point Cloud Sampling Techniques

Sampling a point cloud consists of selecting a subset of the points within the point cloud. A few methods for point cloud sampling include grid subsampling, farthest point sampling, random sampling, and inverse density importance sampling.

Grid subsampling is conducted by applying a three-dimensional grid over the point cloud and selecting one point within each unit cube. The point is selected based on the most frequently occurring class within the unit cube. Grid subsampling results in a point cloud with a uniform density, and the number of points that remain after sampling can be adjusted based on the grid size [9].

Farthest point sampling is done by first selecting a random point in the point cloud; the farthest point from the initial point is then selected. This process continues by selecting the furthest point from all points that have been selected, until the desired number of points have been sampled. Farthest point sampling produces a uniform sample of the point cloud, but it has a computational complexity of $O(N^2)$ that makes it unsuitable for real-time applications with limited processing power [13].

Random sampling is accomplished by randomly selecting a desired number of points from a point cloud. The computational complexity of random sampling is $O(1)$, which makes it realistic to apply in real-time applications in which there is limited processing power available. A downside of randomly sampling points, however, is that important information can be lost [13].

Inverse density importance sampling selects points based on the inverse density at the point location, meaning that points from less dense areas are more likely to be selected than points from very dense areas. The computational complexity is $O(N)$, which makes it unsuitable for real-time applications with limited processing power [13].

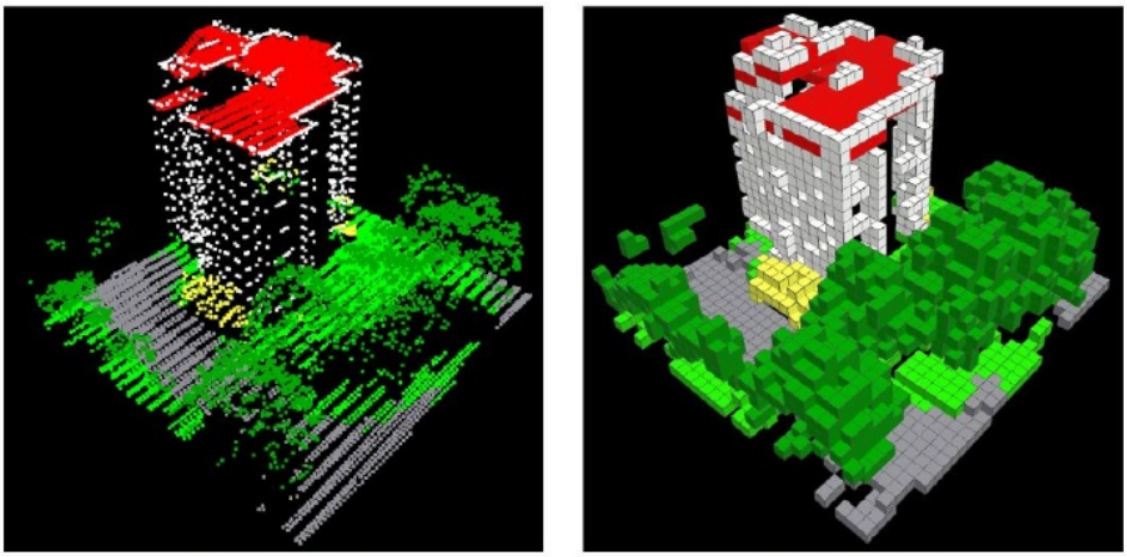


Figure 2.5: An example of a voxelized point cloud [14]

2.1.5 Point Cloud Voxelization

The points that make up point clouds are unordered and have random spacing, which makes them difficult to work with in traditional machine learning algorithms; techniques for working with three-dimensional images, for example, cannot be applied to point clouds because an organized grid of points is required. A process called voxelization is used to transform a point cloud from an unordered and randomly spaced set of points, to an organized three-dimensional grid made up of voxels. Voxelization is done by applying a three-dimensional grid to the point cloud and averaging the points within each cube unit to obtain a uniform three-dimensional grid of data. Figure 2.5 is an example of a point cloud containing a building and trees, and the voxelized version of the point cloud.

Performing voxelization on a point cloud is computationally expensive, making it unrealistic to apply in real-time applications. Recently, new methods in machine learning accept raw point clouds without the need for any pre-processing or the need for voxelization, which reduces the computational complexity required to work with point clouds. These techniques will be discussed in more detail in Section 2.2.

2.2 Point Cloud Classification

2.2.1 Scene Segmentation in Point Clouds

Semantic segmentation in point clouds consists of assigning a label to each point within a point cloud, to develop an understanding of the point cloud scene. The levels of semantic segmentation from simplest to most complex are as follows: object classification, object part segmentation, object detection, instance segmentation, semantic segmentation, and panoptic segmentation [9].

Object classification is typically done using artificial point clouds, in which the entire point cloud represents one object (Figure 2.6a). In this case, one label is determined for all the points within the point cloud. Object classification is typically not used in real-life applications, since the objects contained in the point clouds are isolated from any background scenes, and, therefore, have no spatial context.

Object part segmentation is one step further than object classification. It is also typically done on artificial point clouds, but rather than identifying one single object in the point cloud, the meaningful parts of the object are identified (Figure 2.6b). Object part segmentation is not typically used in real-life applications, since the objects contained in the point clouds are isolated from any background scenes and lack spatial context.

Object detection involves detecting objects within a point cloud scene using bounding boxes and assigning a label to each object (Figure 2.6c). Since a point cloud scene is used, spatial context exists, and it is, therefore, realistic to use object detection in real-life applications. Object detection is a useful technique when the number of objects that appear in a scene is important, but when the exact points belonging to each object is not important.

Instance Segmentation involves detecting objects within a point cloud scene and determining which points belong to each detected object (Figure 2.6d). Since a point cloud scene is used, there is spatial context, and it is realistic to use instance segmentation in real-life applications. Instance segmentation is useful when the exact points belonging to each object is

important, but the background is not important.

Scene segmentation involves assigning a label to every point within a point cloud, and goes one step further than instance segmentation by labelling all of the points according to the object to which they belong, and labelling the points that belong to the background (Figure 2.6e). Because a point cloud scene is used, spatial context exists; therefore, scene segmentation can be applied to real-life applications such as monitoring vegetation growth.

Panoptic segmentation is a combination of scene segmentation and object detection. It involves assigning a label to each point within a point cloud and assigning an instance label to each object contained in the point cloud (Figure 2.6f). Since a point cloud scene is used, spatial context exists, so panoptic segmentation can be applied to real-life applications such as self driving cars.

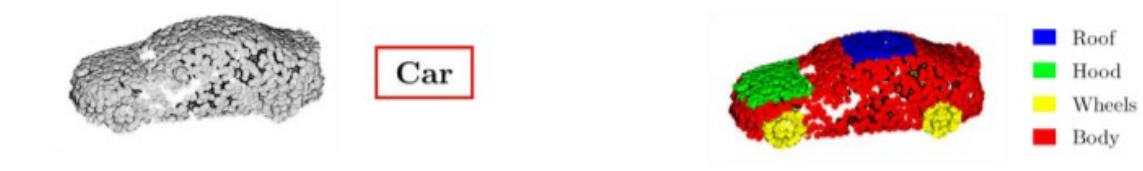
2.2.2 Traditional Point Cloud Segmentation Methods

Traditional methods of detecting obstacles from point cloud data without using machine learning techniques typically involve transforming point clouds into digital elevation maps. By taking the gradient of a digital elevation map, obstacles such as rocks, ridges, and highly sloped regions can be detected as discontinuities. ESA's ExoMars rover, planned to launch in 2022, uses three stereo cameras to produce a digital elevation map of the rover's surroundings to detect obstacles in the terrain by finding discontinuities in the digital elevation map, as shown in Figure 2.7 [15].

2.2.3 Machine Learning Point Cloud Segmentation Methods

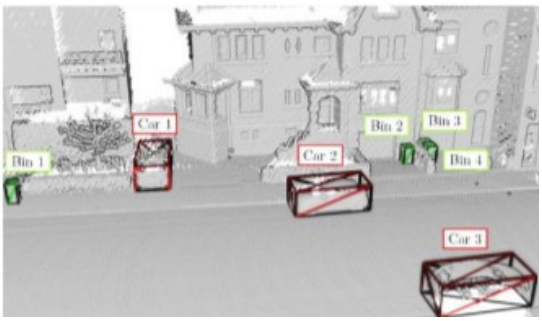
2.2.3.1 Artificial Neural Networks

The design of a NN is based on the nervous system of animals in which neurons in the brain pass signals to one another if the impulse is above a minimum threshold. An artificial NN is made up of perceptrons, which are artificial neurons, and are based off of neurons in the brain.

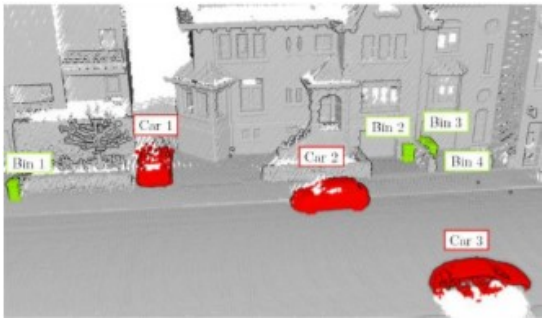


(a) Object classification

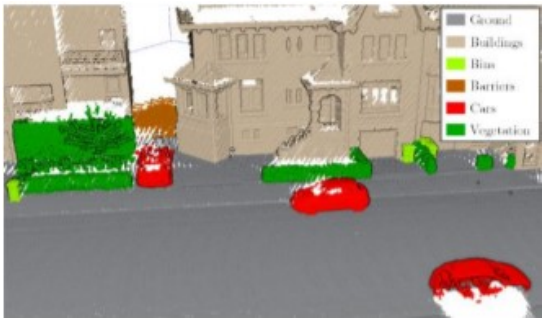
(b) Object part segmentation



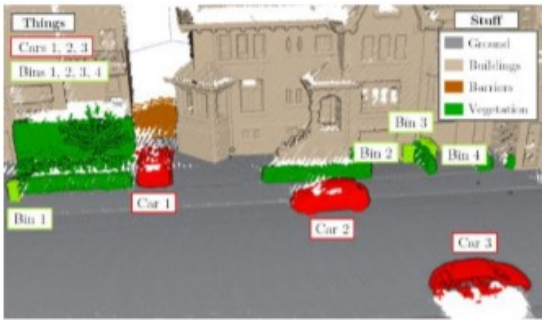
(c) Object detection



(d) Instance Segmentation



(e) Scene segmentation



(f) Panoptic segmentation

Figure 2.6: Levels of semantic segmentation in point clouds [9]

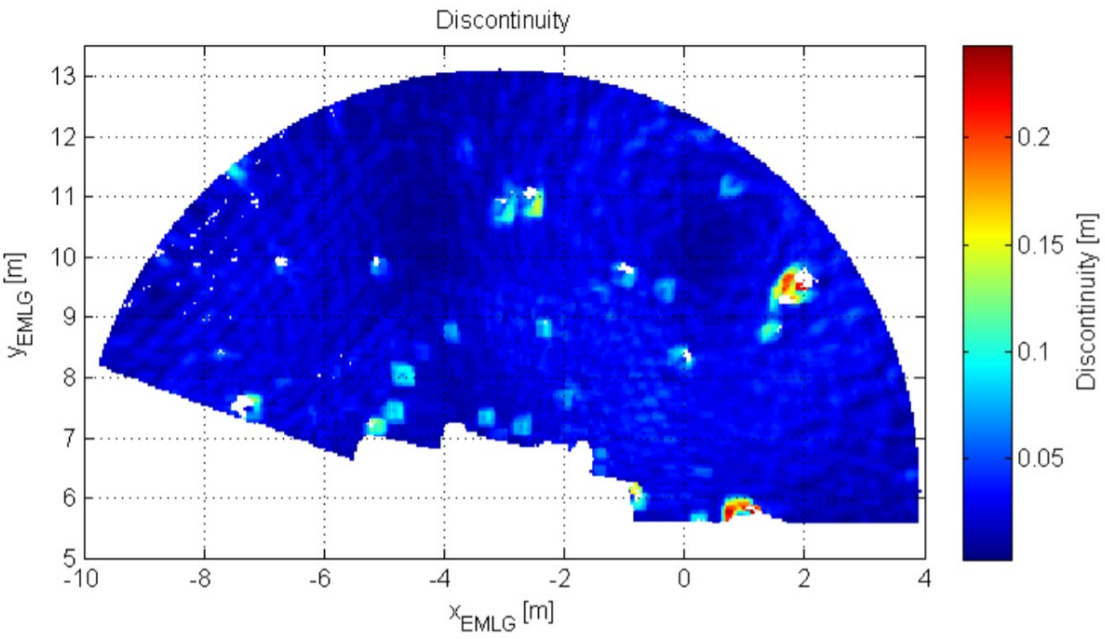


Figure 2.7: Discontinuities in digital elevation map showing location of obstacles [15]

The perceptron on its own is called a single layer perceptron and is a linear classifier that performs binary classification. The structure of a single layer perceptron is shown in Figure 2.8: the n inputs are multiplied by their respective weights and then summed; the weighted sum is passed through the activation function, a Heaviside step function with a threshold that is usually 0.5; and the output is provided as a binary value of one if the sum is greater than the threshold, or zero if the sum is less than the threshold.

For a perceptron to learn, the weights corresponding to each input are updated until the output is a correct classification. If the input data is not linearly separable, the training algorithm will not end because the single layer perceptron is a linear classifier. Because the single layer perceptron is unable to perform non-linear classification, the multi-layer perceptron was developed. The artificial neuron used in a multi-layer perceptron network has one key difference from the single layer perceptron: the activation function. The activation function can be changed based on the goal of the artificial neuron, but all neurons in a single layer have the same activation function that transforms the combined inputs, weights, and biases. By changing the activation function, and adding multiple layers of artificial neurons, the multi-layer

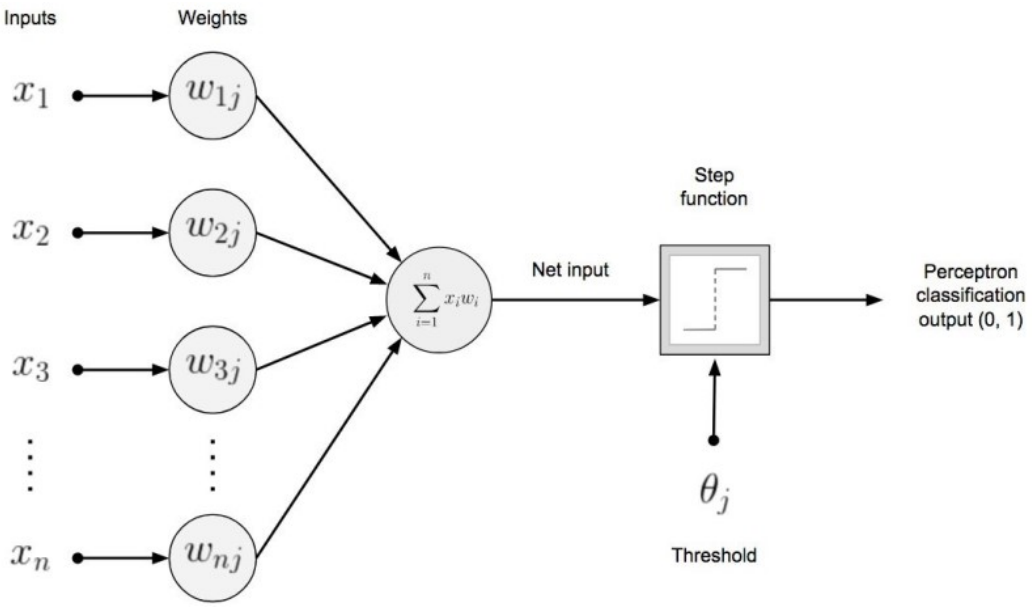


Figure 2.8: Structure of a single layer perceptron [16]

perceptron has the ability to learn non-linear separations of data. A fully connected multi-layer feed-forward NN consists of layers with artificial neurons. The neurons work in parallel and do not require a central control unit. Matrix multiplication is performed between a matrix of inputs and a vector of the weights to produce a vector of output labels. The architecture, shown in Figure 2.9, consists of one input layer, one or more hidden layers, and one output layer. Each layer can have different numbers of neurons, and is fully connected to the adjacent layer, meaning that the output from each neuron in a layer is fed into every neuron in the next layer. The input layer typically has the same number of neurons as the number of inputs. The hidden layer(s) allows a NN to learn non-linear representations of the input data. The weights in the hidden layer(s) are updated as the network learns, and encode the information that the NN has learned from the input data. The output layer provides the overall prediction of the model.

Traditionally, NNs could not be applied to point clouds in their raw format because point clouds are unordered and unstructured. Instead, feature vectors would be extracted from the point cloud to try to capture the information and feed the feature vectors into the NN. Habermann et al. [17] use feature extraction to classify objects in point clouds using NNs. The first step to classify objects in a point cloud is to segment it into smaller sections containing the ob-

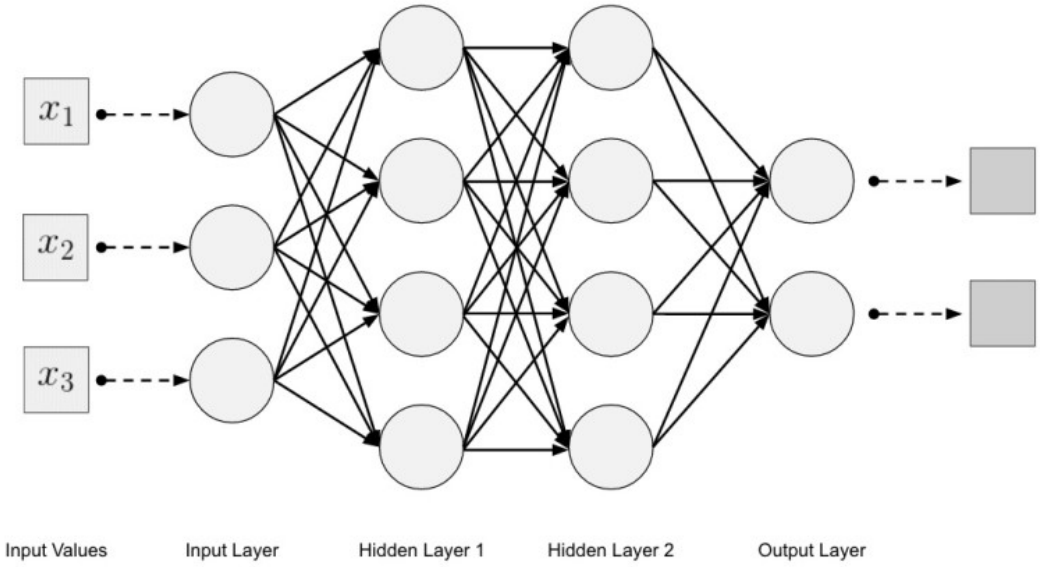


Figure 2.9: Structure of a fully connected multilayer feed-forward neural network [16]

jects of interest. This segmentation is accomplished by first removing the ground plane, which is identified as all points below a specified height threshold. The remaining points are then voxelized to create a three-dimensional occupancy grid, and an activation function is calculated for each voxel to determine groups of voxels; subsequently, feature extraction is performed. The centre of mass, height, width, and depth of each group is calculated, and the points belonging to each voxelized group are transformed so that their centre of mass is the origin of their coordinate system. The points are then projected into 56 grid cells and the number of points in each cell is recorded and normalized by dividing by the total number of points. Each feature vector contains 58 inputs in total:

- 56 inputs include the normalized number of points in each grid cell
- The 57th input is height
- The 58th input is the maximum value of the width and depth

Finally, the feature vectors are fed into a multi-layer perceptron NN to classify the groups of voxelized points. The network achieved 89.5 % accuracy when classifying point clouds of

city scenes containing objects such as people, vehicles, buildings, and trees. One limitation of the method developed by Habermann et al. [17] is that it only works on flat ground because of the threshold method for removing the ground points. Further, if two different objects are located next to one another, their voxels may be grouped together, resulting in an incorrect classification.

Recently, NN architectures have been developed that are capable of accepting point clouds in their raw form, which allows point clouds to be used without any pre-processing. These architectures will be discussed in detail in the following sections: PointNet, PointNet++, and RandLA-Net.

PointNet

PointNet is a NN architecture by Qi et al. [4] that is designed to work on an unordered set of points, to recognize relations between neighbouring points, and to be invariant to certain transformations such as rotations and translations. PointNet takes a set of points in three-dimensional space as an input and performs either object classification, object part segmentation, or scene segmentation. The PointNet network achieves classification and segmentation by using three key modules: the joint alignment network, a max pooling symmetric function, and a local and global information combination structure (Figure 2.10) [4].

The joint alignment network, named T-Net, enables the PointNet network to be invariant to geometric transformations of the input point clouds. T-Net predicts and applies an affine transformation matrix to the input points, which is constrained to be close to orthogonal, so it will not lose information about the input. The joint alignment network results in an increase in optimization stability and better model performance.

The symmetric max pooling function aggregates information from each point and allows the network to be unaffected by the input order of points, which is important due to the unstructured format of point clouds. The result of applying the max pooling function on the transformed elements is a global feature vector that can be used for object classification of the

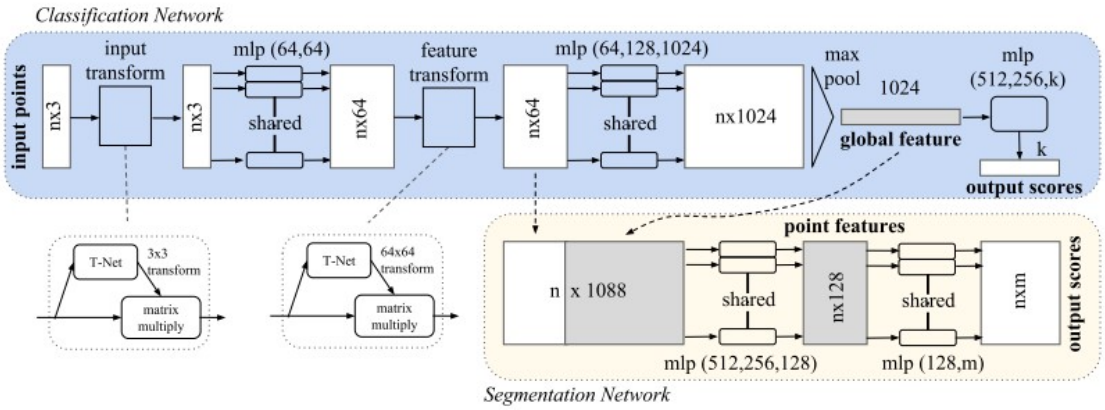


Figure 2.10: PointNet network structure [4]

entire input point cloud.

For segmentation, both local and global features are required, and are obtained using the local and global information combination structure to concatenate the global feature vector with each point in the point cloud. New point features are extracted that contain information on both local and global structures, allowing each point to be classified for either object part segmentation or scene segmentation.

In an efficiency test on a NVIDIA RTX2080Ti GPU, PointNet was able to evaluate 4071 scans within 192 seconds, which is approximately 21 frames per second, and is a realistic rate to apply the model in real time applications [13].

PointNet++

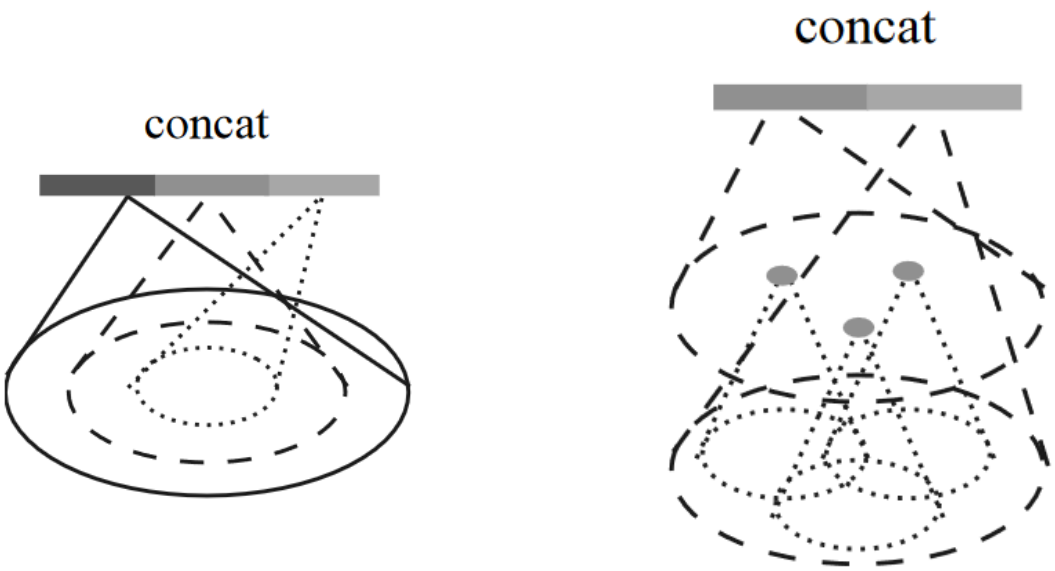
PointNet++ is a hierarchical NN by Qi et al. [18] that is designed to capture local structure and builds upon the ideas presented in PointNet. It works by taking small, overlapping samples to capture fine local details and gradually increases the sample size until all of the features in the point set are known. The feature learning occurs in three layers: sampling, grouping, and PointNet.

The sampling algorithm is the iterative farthest point sampling algorithm, which results in a uniform sample of the point cloud to ensure that both the local and global structures of the point cloud are captured. The points that are selected during sampling are then used as centroids in

the grouping layer that selects a local region around each centroid from the sampling layer. The local region is selected using a ball query method, which is performed by selecting all points within a radius around the centroid point of interest. The ball query method generates a region of a fixed scale around each centroid point, making it easier to generalize the local region feature. After the grouping layer, the coordinates of the points in each group are translated into the local frame relative to the group's centroid. Each group is then fed into the PointNet layer that is used to learn the local patterns of the point cloud.

Since point clouds have a non-uniform density, a problem can arise with generalization: a feature learned in a dense area of a point cloud may be unrecognizable in a sparse area of the point cloud. PointNet++ addresses the problem of non-uniform density in point clouds with two algorithms: Multi-Scale Grouping (MSG) and Multi-Resolution Grouping (MRG). MSG applies grouping layers of different scales by changing the radius during the ball query method (see Figure 2.11a). The PointNet layer is then applied after each scaled grouping layer to extract features of different sizes, and these extracted features are concatenated to create a multi-scale feature vector. Since MSG runs multiple PointNets on each group in each layer, it is computationally expensive, so it is undesirable due to its time-consuming characteristics. The second algorithm used by PointNet++, which is less computationally complex, is MRG in which each region is represented by concatenating two vectors. As shown in Figure 2.11b, the left vector applies multiple PointNets to small regions within the group to obtain summarized features of the group, and the right vector applies one PointNet to all the points in the group. The left vector is used when the density of the group is high, allowing fine detail to be extracted, and the right vector is used when the density of the group is low.

In an efficiency test on a NVIDIA RTX2080Ti GPU, PointNet++ was able to evaluate 4071 scans within 9831 seconds, which is approximately 0.4 frames per second and is not realistic to apply in real-time applications [13].



(a) Multi-Scale Grouping (b) Multi-Resolution Grouping

Figure 2.11: Sampling methods for dealing with varying density in point clouds [18]

RandLA-Net

RandLA-Net is a neural architecture by Hu et al. [13] that is designed to efficiently segment large-scale point clouds through the use of a random sampling technique. It takes an unordered set of points as an input and performs scene segmentation by outputting a label for each point.

A random sampling algorithm is used to randomly select a number of points from the original point cloud. The computational complexity of the random sampling algorithm is O(1), making it possible to implement the network in real-time [13]. Using random sampling means that no extra memory is required for sampling computations; however, random sampling can lead to the loss of important features. To avoid feature loss, RandLA-Net uses a local feature aggregation module to each point in a point cloud in series and contains three neural units: Local Spatial Encoding (LocSE), attentive pooling, and a dilated residual block (Figure 2.12) [13].

The LocSE unit creates awareness of each point's position relative to its neighbouring points by first finding the neighbouring points using the KNN algorithm. Then a relative point position encoding is found for each point based on its KNNs by concatenating the Cartesian

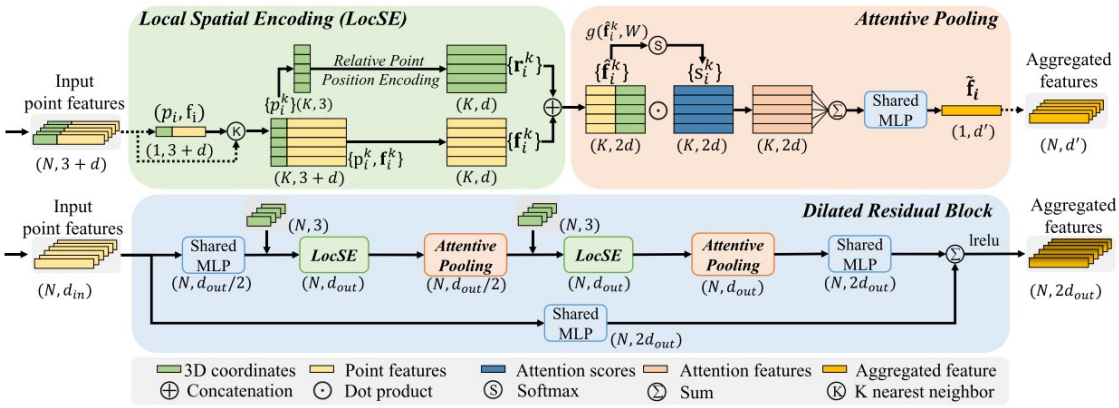


Figure 2.12: Structure of the local feature aggregation module [13]

coordinate and Euclidean distance of all neighbouring points. The final step is to concatenate the relative positioning of the neighbouring points with the already-existing features of each point, resulting in an augmented feature vector. The overall output of the LocSE unit is a new set of features that explicitly encodes the local geometric structure for each point.

Attentive Pooling combines sets of neighbouring point features by automatically learning which features are important. The important features are identified through the computation of attention scores, which are calculated by using a shared multilayer perceptron followed by a softmax function. The result is a unique attention score for each feature, which can be used to select the important features.

The dilated residual block stacks multiple LocSE and Attentive Pooling units in series with a skip connection, which increases the receptive field of each point. An increased receptive field results in each point containing more information about the geometric structure of the original point cloud, so even if points are dropped during random sampling, the geometric information is not lost. Figure 2.13 shows how stacking two dilated residual blocks in series results in an increased receptive field. In the end result shown on the left of the image, the orange dot contains information about K^2 neighbouring points after stacking two dilated residual blocks. Significantly, stacking more units will continue to increase the receptive field of each point, but it will decrease the computational efficiency; as such, the correct balance between increasing the receptive field and the computational efficiency must be established.

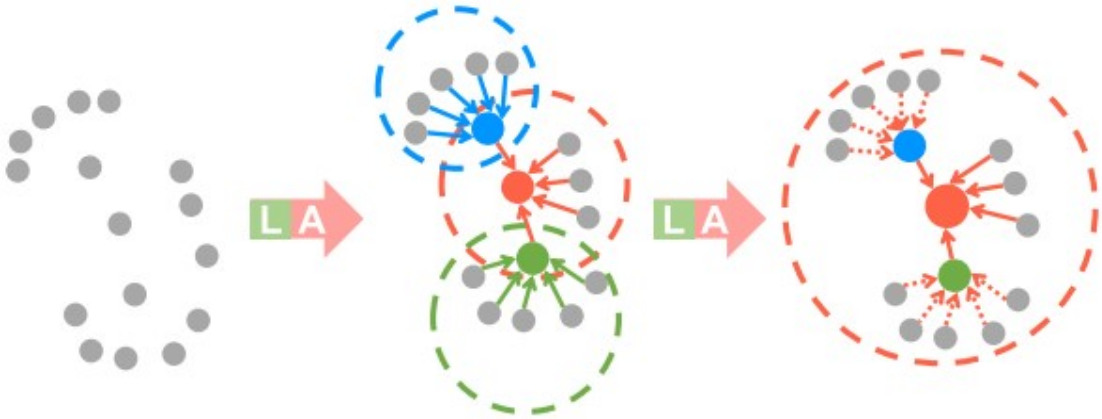


Figure 2.13: The effect of stacking two dilated residual blocks in series [13]

In an efficiency test on a NVIDIA RTX2080Ti GPU, RandLA-Net was able to evaluate 4071 scans within 185 seconds, which is approximately 22 frames per second [13]. At this rate, it is realistic to apply the model in real-time applications.

2.2.3.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of NN architecture that differs from a regular NN. Rather than applying matrix multiplication on each layer, a CNN applies a convolution operation on at least one layer [19]. The hidden layers of a CNN typically contain a repeating pattern of a convolution layer followed by a pooling layer, which allows the CNN to perform feature extraction. The convolution operation is essentially a sum of the product of input data overlapping with a kernel. The kernel is moved over the input data and each convolution is recorded to create the output feature map. An example convolution of a 3×4 input and a 2×2 kernel is shown in Figure 2.14, which produces a 2×3 output. As shown in the image, the kernel is moved over the input data, and each element of the kernel is multiplied with the overlapping element of the input layer, then the products are summed together to create one entry in the output feature map. The pooling layer reduces the spatial size of the data and helps to prevent overfitting.

The motivation to use convolution rather than matrix multiplication comes from the com-

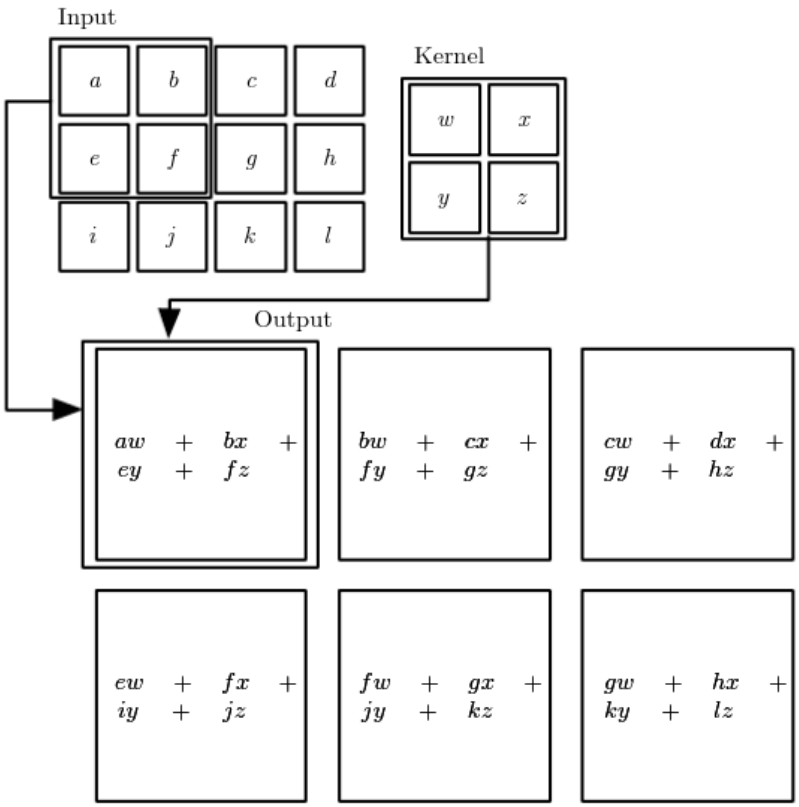
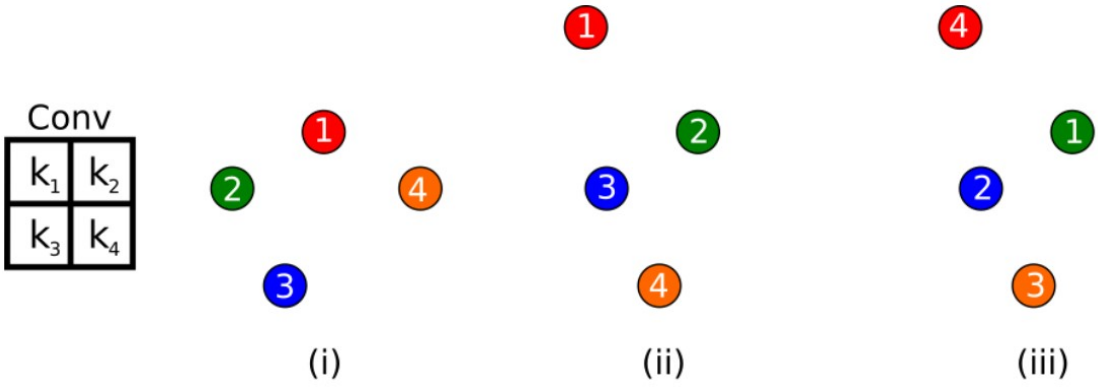


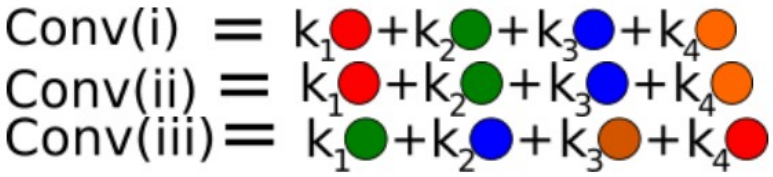
Figure 2.14: Example of a convolution operation [19]

putational complexity that would be required to use a fully connected multi-layer NN for image classification. An image of 100×100 pixels with three channels of RGB data, for example, would require 30,000 weights at each hidden neuron [16]. In comparison, a CNN can be designed in a three-dimensional neuron structure so the width, height, and depth, match the pixel width, pixel height, and channel depth of the image. Additionally, the convolution operation is less computationally complex than matrix multiplication required in traditional NNs.

A CNN is designed to work on data that has a regular grid-like structure. Since point clouds have an irregular structure, they cannot be fed into a CNN in their raw form. Figure 2.15 demonstrates why convolution cannot be applied to unordered and unstructured data. The kernel shown on the left of Figure 2.15a is convoluted with point clouds (i), (ii), and (iii), and the results of the convolutions are given in Figure 2.15b respectively. The colours of the points in the point clouds represent their features, and the numbers of the points represent their input



(a) Three sample point clouds and a convolution kernel [20]



(b) Resulting convolutions [20]

Figure 2.15: Example convolution operation on point clouds showing why convolution cannot be applied to unordered and unstructured data

order. The input order determines the order in which the convolution kernel is applied. Point cloud (i) has a different shape than point clouds (ii) so they should have different convolutions, but the two point clouds have the same input order of points and features which results in conv(i) being equal to conv(ii) as shown in Figure 2.15b. Point cloud (ii) and point cloud (iii) in Figure 2.15a have the same shape and features so their convolutions should be the same, but the input order does not match which results in conv(ii) not being equal to conv(iii) as shown in 2.15b. The fact that conv(i) equals conv(ii), and conv(ii) does not equal conv(iii) demonstrates the importance of being invariant to the input order and shape when working with point clouds. Before a CNN can be applied to point clouds, the point clouds must be transformed into a regular three-dimensional grid.

VoxNet is a three-dimensional CNN architecture by Maturana and Scherer [21] that is designed to detect objects in three-dimensional point clouds. VoxNet takes point cloud segments as an input. Each point cloud segment contains an object, and sometimes background points

surrounding the object. There are two main parts of VoxNet: an estimate of spatial occupancy of the point cloud segment on a volumetric grid also known as voxelization, and a CNN that predicts a class label for each voxelized grid. VoxNet achieves a 92 % accuracy on the ModelNet10 dataset, and a 83 % accuracy on the ModelNet40 dataset. The implementation takes two milliseconds to classify 2000 points while using a Tesla K40 GPU. However, VoxNet is trained to classify sub-sections of point clouds, which means the point cloud scenes recorded by a rover would have to broken into segments and voxelized to be classified. By breaking scenes into segments at random, there would be a risk of splitting an obstacle and making it unrecognizable.

PointCNN by Li et al. [22] develops a hierarchical convolution operation called *X*-Conv. First, points are randomly sampled and passed through an *X*-transformation. The transformation arranges the randomly selected points into an ordered form and stores the local spatial information about the KNN points surrounding the randomly selected points as a feature vector. The computation time to process 4071 scans with 50,000 inference points on a NVIDIA RTX2080Ti GPU using PointCNN is 8142 seconds, which is approximately 0.5 frames per second. At this rate, PointCNN is not realistic to run in real-time applications [13].

2.2.4 Training, Validation, and Testing Sets

In machine learning, it is necessary to split the data into subsets for training, validation, and testing. The training set normally consists of 60 to 80 % of the dataset and is used to fit the model to the dataset by finding optimal weights. The validation set normally consists of 10 to 20 % of the dataset and is used to perform an evaluation on the models performance after each full pass of the training set, also known as one epoch. The validation step allows the model to be evaluated on data that it was not trained on and is used to optimize the model. Since the validation set is used to measure the performance of the model, which is built using the training set, there will still be some bias in the performance metrics calculated on the validation set. The testing set normally consists of 10 to 20 % of the dataset and is used to perform an unbiased

	P' (Predicted)	N' (Predicted)
P (Actual)	True Positive	False Negative
N (Actual)	False Positive	True Negative

Figure 2.16: Layout of a confusion matrix [16]

evaluation of the model's performance after the final model has been tuned on the training set. Evaluation on the testing set provides a final assessment of the model's performance, and the model is not meant to be tuned further after the assessment.

2.2.5 Performance Metrics

Many performance measures exist to evaluate the performance of machine learning models including the confusion matrix, accuracy, precision, recall, and F1 score [16].

A confusion matrix is a table that shows the number of true positives, true negatives, false positives, and false negatives, and tells how well the model is performing. The layout of a confusion matrix is shown in Figure 2.16. A true positive occurs when an instance is predicted as positive, or belonging to a class, and it truly does belong to that class. A true negative occurs when an instance is predicted as being false, or not belonging to the class, and it truly does not belong to the class. A false positive occurs when an instance is predicted to belong to a class, but it truly does not belong to the class. A false negative occurs when an instance is predicted to not belong to a class, but it truly does belong to the class.

Accuracy evaluates the ratio of correct predictions to the total predictions made, shown in Equation 2.3. Accuracy can be misleading if there is a high class imbalance in the dataset; for example, if one class represents 95 % of the data and a 95 % accuracy is achieved it may

appear that the model is performing well. Since the accuracy matches the percentage of data represented by the majority class, it likely means that the model is classifying all points as the majority class and missing the minority class altogether.

$$Accuracy = \frac{\#\ correct\ predictions}{total\ predictions} \tag{2.3}$$

Precision is a measure of the model's ability to produce the same results multiple times under the same conditions, as given in Equation 2.4. It is important to note that a high precision does not indicate that the result is correct, as the model could repeatedly predict an incorrect value.

$$precision = \frac{\#\ true\ positives}{\#\ true\ positives + \#\ false\ positives} \tag{2.4}$$

Recall, also known as sensitivity, is a measure of the true positive rate which indicates how well a model avoids false negatives. The Equation for recall is given in 2.5.

$$recall = \frac{\#\ true\ positives}{\#\ true\ positives + \#\ false\ negatives} \tag{2.5}$$

F1 scores are used to measure the model accuracy for binary classification. It is computed as the mean of the precision and recall scores, as shown in Equation 2.6. An F1 score of 1.0 is the best possible score, and 0.0 is the worst possible score.

$$F1 = \frac{2 \cdot \#\ true\ positives}{2 \cdot \#\ true\ positives + \#\ false\ positives + \#\ false\ negatives} \tag{2.6}$$

2.2.6 Dealing with Class Imbalance in Point Clouds

Class imbalance occurs when a majority class has abundant representation in the data, and a minority class has limited representation in the data. A dataset is considered to have a large class imbalance for majority to minority ratios ranging from 100:1 to 10 000:1 [23]. Having a

largely unbalanced dataset can be problematic because it can lead the network to only learn one class, while still seeming to achieve good results. In point clouds, a large class imbalance can arise when many points belong to the background of the scene, and the objects within the scene contain very few points [24]. The network can achieve good results by learning to classify all points as background points. There are three main categories for dealing with a large class imbalance: data-level methods, algorithm-level methods, and ensemble methods.

Data-level methods focus on sampling techniques. Common sampling methods include under sampling, over sampling, and the Synthetic Minority Oversampling Technique (SMOTE). Under sampling is done by randomly removing instances of the majority class to decrease its representation. Over sampling consists of increasing the representation of the minority class by duplicating the minority class in the dataset. Duplicating the minority class can be done randomly, or by using an algorithm to replicate the data [23]. A visual representation of over sampling and under sampling can be seen in Figure 2.17, which shows the original dataset with a class imbalance, and the size of the resulting datasets after each type of sampling. Problems can arise with both sampling methods. Under sampling the data can result in important information being deleted from the dataset. Whereas over sampling can result in overfitting the model since instances of the minority class are being duplicated, and over sampling increases the size of the training dataset which could result in a dataset that is too large [24]. The SMOTE algorithm generates new instances of the minority class based on KNN in the minority class. Figure 2.18 shows the SMOTE algorithm working to generate new instances of the minority class to create a more balanced dataset. The SMOTE algorithm can run into problems of overfitting, similar to the over sampling method [24].

Algorithm-level methods focus on various algorithms that will help the network learn to classify the data, without being biased to the majority class. Some examples of algorithm-level methods include cost-sensitive and threshold-moving methods. Cost-sensitive methods consist of assigning a cost for each type of misclassification [23]. Cost assignment can be done using a cost matrix, where a cost is assigned for a true negative classification, a true positive classifi-

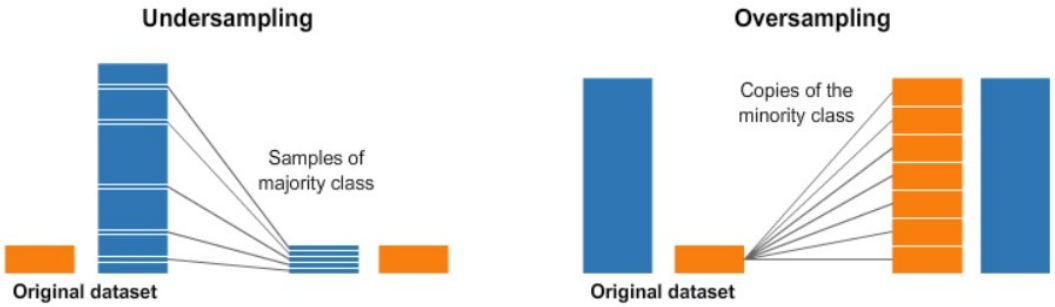


Figure 2.17: Under sampling shown on the left consists of removing samples of the majority class until the data is balanced. Over sampling shown on the right consists of creating copies of the minority class to balance the data [25]

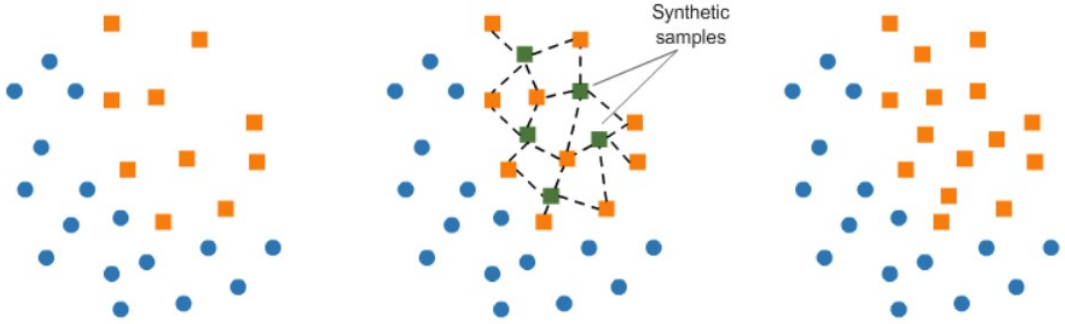


Figure 2.18: The cluster of points on the left show an unbalanced dataset, where the orange points are the minority. The cluster of points in the middle show artificial instances of the orange class being generated by the SMOTE algorithm. Finally, the cluster of points on the right shows a more balanced dataset [25]

cation, a false negative classification, and a false positive classification. The costs are assigned in a way that the cost of misclassification is higher than a correct classification to discourage the network from only learning one class. Further, emphasis can be placed on avoiding false positives or false negatives, depending on the significance of each within the dataset [26]. For example, in medical imaging it is very important to avoid false negatives since a false negative would result in a sick patient being missed, whereas a false positive is less crucial because the result will be checked by a doctor and corrected. In this case where preventing false negatives is very important, the cost of a false negative would be set very high. Threshold-moving methods involve changing the acceptable threshold value for the predicted probability of an instance belonging to a class [27]. Typically, the threshold is set to 0.5 meaning if the probability that an instance belongs to a class is greater than 0.5, the instance is predicted to belong to the class.

Ensemble methods consist of training multiple different classifiers, and making a final classification based on a combined vote from each classifier. Two common ensemble methods are bagging and boosting. Bagging divides the dataset into multiple training sets and generates a classifier for each training set. The predictions of the classifiers are then combined for classification. Bagging results in reduced variability in the final model's predictions [23]. Boosting also divides the dataset into multiple training sets to train multiple classifiers, and iteratively assigns new weights based on the incorrect predictions. Then the results of each classifier are combined into one final classification [23].

Chapter 3

Datasets

This section contains information about two datasets containing Lidar point clouds that were recorded at planetary analogue sites. The processes of data collection and determining ground truth labels for each data set are outlined. It is important for the data to be labelled as either rock or ground in order to use it to train machine learning models to detect rock obstacles in planetary scenes.

3.1 Analogue Terrain Facility Dataset

3.1.1 Data Collection

Lidar point cloud data was collected at the Canadian Space Agency's Analogue Terrain Facility (ATF) in Saint-Hubert, Quebec. The ATF is approximately 80 $\times$ 100 meters with several different types of terrain, geometry, and rocks.

3.1.1.1 Procedure

Point cloud scans were collected using an Ouster Lidar OS1-64 attached to a Clearpath Husky Unmanned Ground Vehicle (UGV) in the Canadian Space Agency's ATF. Data was collected

Figure 3.1: Data collection path where ‘X’ marks the start point and ‘O’ marks the end point

on a path through a sparse boulder field with small and medium sized boulders, and flat sandy terrain. The path is shown in Figure 3.1. Lidar point cloud data was recorded continuously while driving. The Lidar parameters are given in Table 3.1. Noteworthy parameters are the Ouster OS1 Lidar’s 120 meter range and it’s point cloud scan rate of 20 Hz with 65536 points per scan. Overall, 3318 point cloud scans were collected during the traverse through the ATF. The output from the Lidar includes the range, intensity, reflectively, ambient near-infrared, azimuth angle, and time stamp.

3.1.1.2 Rover Setup

The Husky UGV was set up with the following equipment: a Logitech webcam, an Ouster OS1 Lidar sensor, a Goal Zero Yeti 400 battery, bungee cords, a NVIDIA Jetson TX1 board, and a Wavelink antenna.

Robot Operating System (ROS) was used to manage devices on board the UGV, as well as to communicate between the UGV and the user base station. In ROS, each process is called a

Parameter	Value
Horizontal Resolution	1024 channels
Vertical Resolution	64 channels
Range	120 m
Vertical Field of View	45°(±22.5°)
Vertical Angular Resolution	0.35°-2.8°
Precision	±0.7 – 5 cm
Points per Second	1,310,720
Rotation Rate	10 or 20 Hz
Power Draw	14-20 W
Weight	455 g
False Positive Rate	1/10,000

Table 3.1: Ouster OS1-64 Lidar parameters

node. Each node has the ability to communicate with all other nodes by publishing a topic and subscribing to topics from other nodes. ROS was used to communicate with the UGV from a user base station located next to the ATF, where the UGV was controlled using a Logitech game-pad, a laptop, and mission control software that was provided by Mission Control Space Services. The set up of the ROS nodes and topics used are shown in Figure 3.2. The Lidar node published data, which was subscribed to by the user base station and saved to an external hard drive. The video was published from the camera and subscribed to by the user base station where it was displayed on the mission control platform on the laptop. The video was used to see the rover's position and to make decisions on how to control the rover from the user base station. The game controller was then used to publish the input commands to drive the rover, which were subscribed to by the Jetson TX1 on-board the rover and used to control the wheels.

3.1.2 Data Processing

The Ouster OS1 Lidar has a range of 120 meters, which is larger than the size of the ATF. Due to this long range, the point clouds that were collected at the Canadian Space Agency contain background items such as trees and buildings that surround the ATF. Since these features do not exist on other planetary surfaces, a necessary step for this thesis was to remove them from

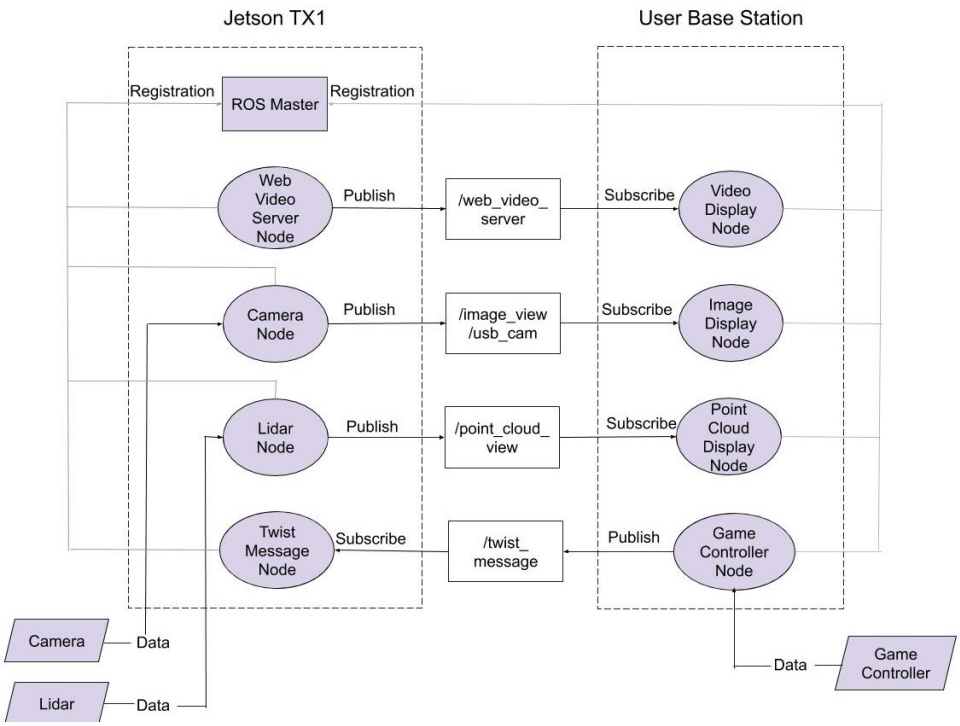


Figure 3.2: ROS nodes and topics used during the ATF dataset acquisition

the point clouds to ensure the scans accurately represent a planetary surface.

As the rover moved around the ATF, the location of the trees and buildings moved around the recorded point clouds. The variation in the position of the trees and buildings means that the point clouds could not simply be cropped to a specified distance. Therefore, it was necessary to develop a way to identify points within the point cloud that belong to the trees and buildings which need to be removed.

A key feature of the points which make up the buildings and trees, is the normal vector to the points. The normal vector of the points that make up these objects is approximately parallel to the ground plane. However, the normal vector to rocks located within the ATF are also approximately parallel to the ground plane. The normal vectors of points representing the buildings, trees, and rocks are shown in Figure 3.3. To account for the fact that the normal vector is parallel to the ground in both the objects that should be kept and the objects that should be removed, a second criteria was considered to identify the points belonging to the buildings and trees: their size. The trees and buildings are made up of a larger number of

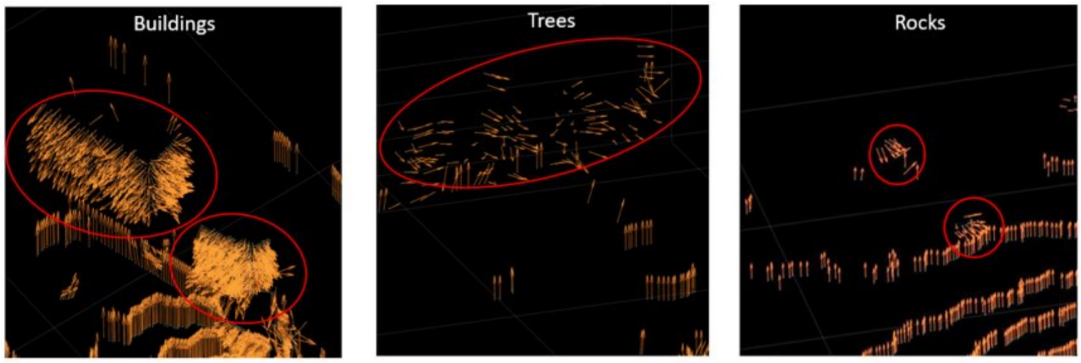
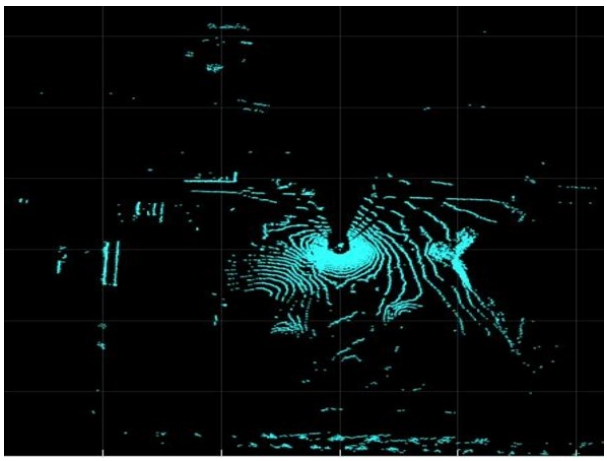


Figure 3.3: Normal Vectors of points belonging to buildings, trees, and rocks

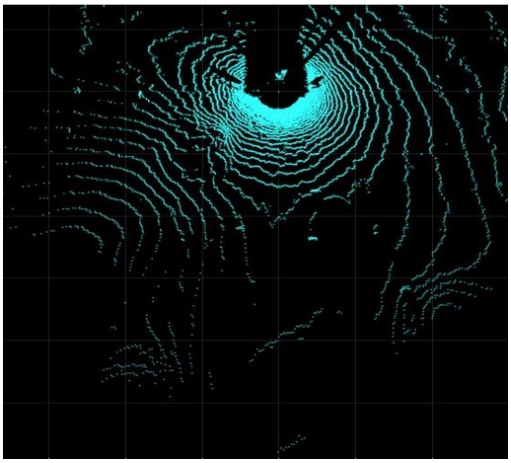
points, and cover a larger area when compared to the rocks within the ATF. Therefore, the points belonging to the trees and the buildings can be eliminated based on their normal vector being parallel to the ground plane, and having greater than KNN that also have a normal vector parallel to the ground plane. The normal vector for each point in the point cloud is computed locally based on the KNN.

After removing points based on their normal value, some small groups of points remained in the point cloud that were located outside of the ATF. To remove these, a de-noising operation was performed by evaluating the distance from a point to it's KNN and removing the point if the average distance is above a specified threshold.

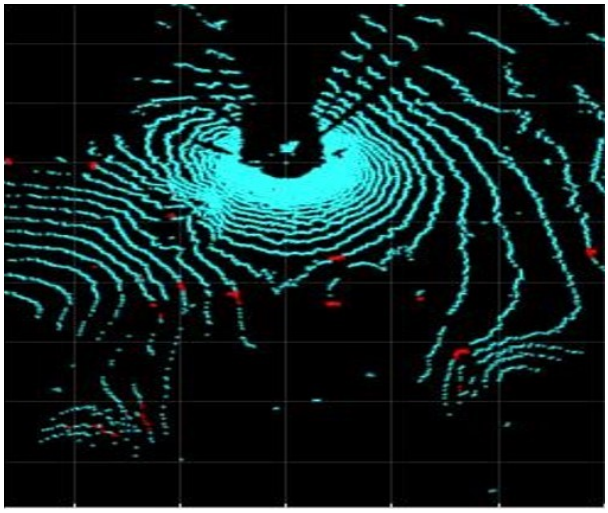
Now that each point cloud only contains points within the ATF, ground truth versions of the point clouds which identify the location of the rocks were obtained. Key features of the points that represent rocks are their intensity, reflectivity, and normal vector. The intensity and reflectivity of the rocks are greater than that of the surrounding terrain; however, this is more visible on larger rocks. The normal values of the points which make up the rocks vary from the normal values of the terrain. A sharp change in the normal direction can be seen at the base of each rock. By combining the change in the normal direction with the change in the reflectivity and intensity values, the rocks can be separated from the terrain which allows for ground truth labels to be applied to the ATF dataset. Figure 3.4 shows the data processing stages including the original point cloud, the point cloud without the surrounding trees and buildings, and the



(a) Original Point Cloud



(b) Point Cloud Without Trees and Buildings



(c) Ground Truth Point Cloud Showing Rock Locations in Red

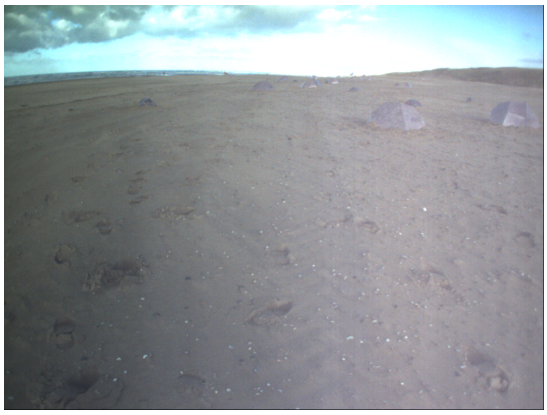
Figure 3.4: Point cloud processing stages for the ATF dataset

ground truth of the point cloud, where the rocks are shown in red.

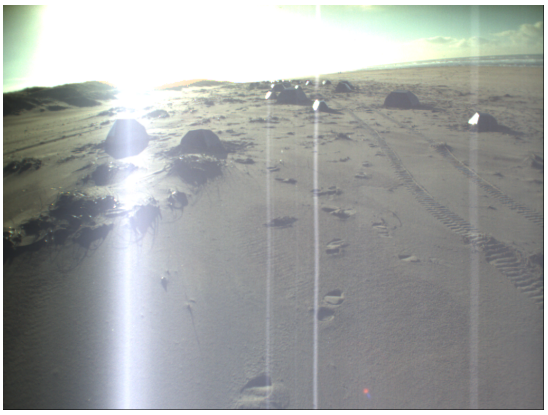
3.2 Katwijk Beach Planetary Rover Dataset

3.2.1 Data Collection

The Katwijk Beach Planetary Rover Dataset was collected at a beach near Katwijk in The Netherlands, by a team at the European Space Research and Technology Centre [28]. Two test runs were conducted during the data collection.



(a) Image taken during traverse 1



(b) Image taken during traverse 2

Figure 3.5: Comparison of lighting conditions in traverse 1 and traverse 2 of the Katwijk Beach Planetary Rover Dataset

The first test run was approximately 2000 meters through a boulder field made up of two hundred and twelve artificial rocks that were placed to model typical boulder fields seen in Mars Reconnaissance Orbiter images. This first run was divided into two traverses due to the significantly different sunlight conditions between the starting point to the turn around, and the turn around back to the starting point. During the first traverse, the sun was to the back of the rover, whereas in the second traverse the sun was in front of the rover. Figure 3.5 shows a comparison of the lighting conditions between the two traverses, where Figure 3.5a was taken while driving away from the sun and the lighting conditions are not as harsh as in Figure 3.5b where the image was taken facing into the sun which created light artifacts in the image. The location and size of each rock in the two traverses, as well as the paths of the two traverses are shown in Figure 3.6.

The second test run, or third traverse, was approximately 200 meters through a boulder field made up of the same two hundred and twelve artificial rocks as in the first run, but in this second test run the rocks were placed to be twice as dense compared to the first test run. Figure 3.7 shows the path taken during the third traverse.

In all three traverses, three different sized artificial boulders were used. The large boulder was 1.897 meters in diameter, the medium boulder was 1.326 meters in diameter, and the small

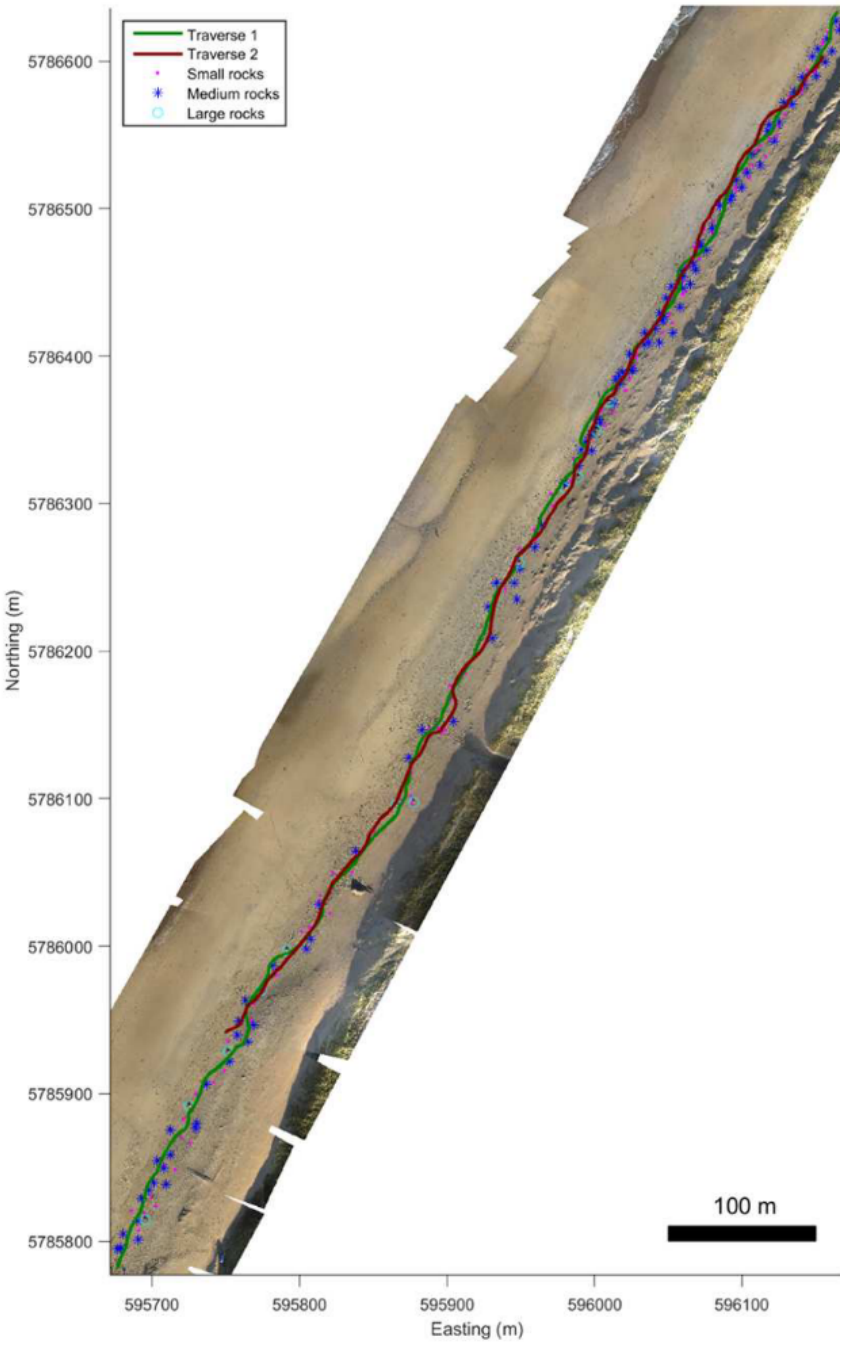


Figure 3.6: Traverse 1 and traverse 2 paths of the Katwijk Beach Planetary Rover Dataset [28]

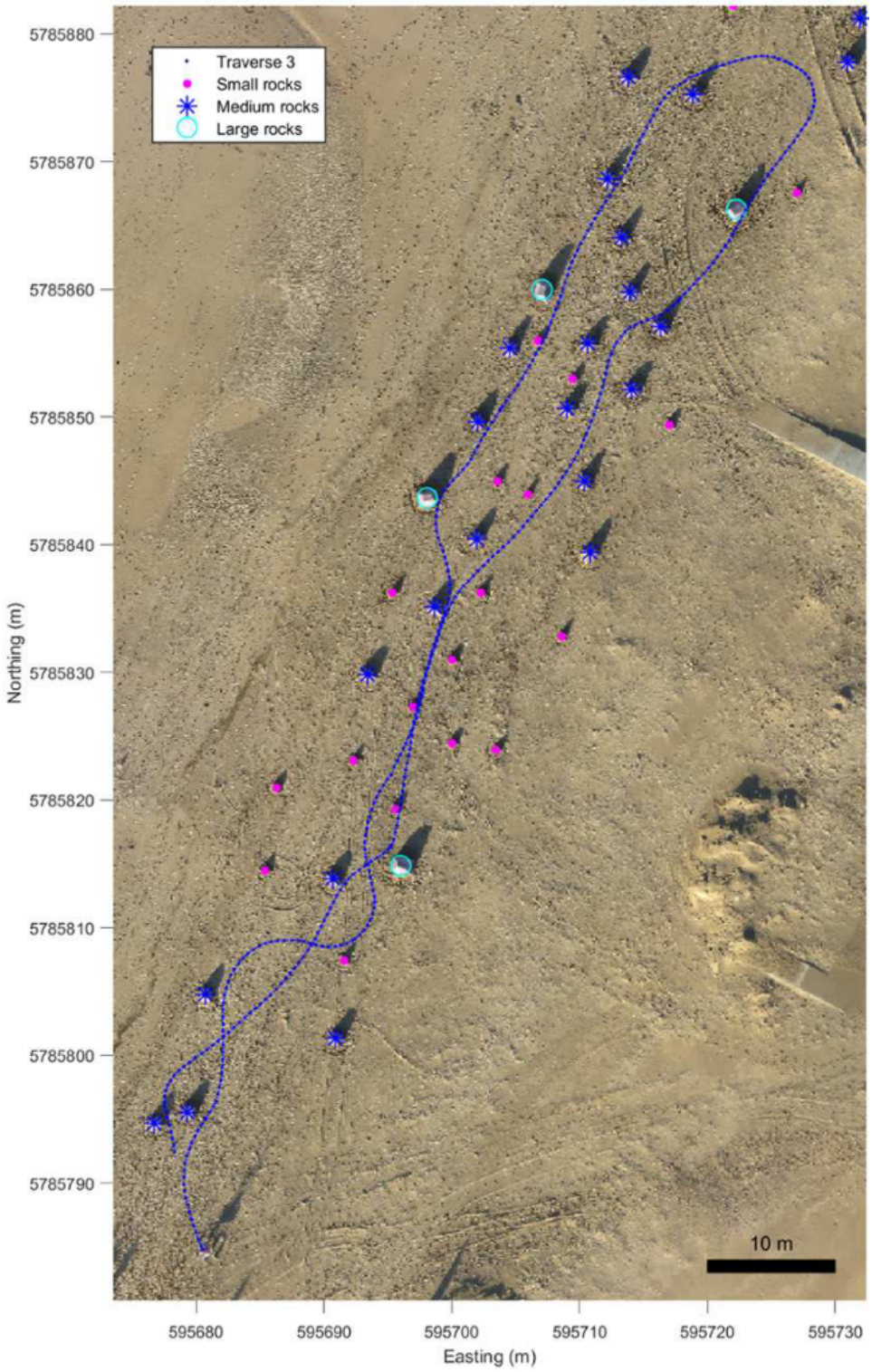


Figure 3.7: Traverse 3 path of the Katwijk Beach Planetary Rover Dataset [28]

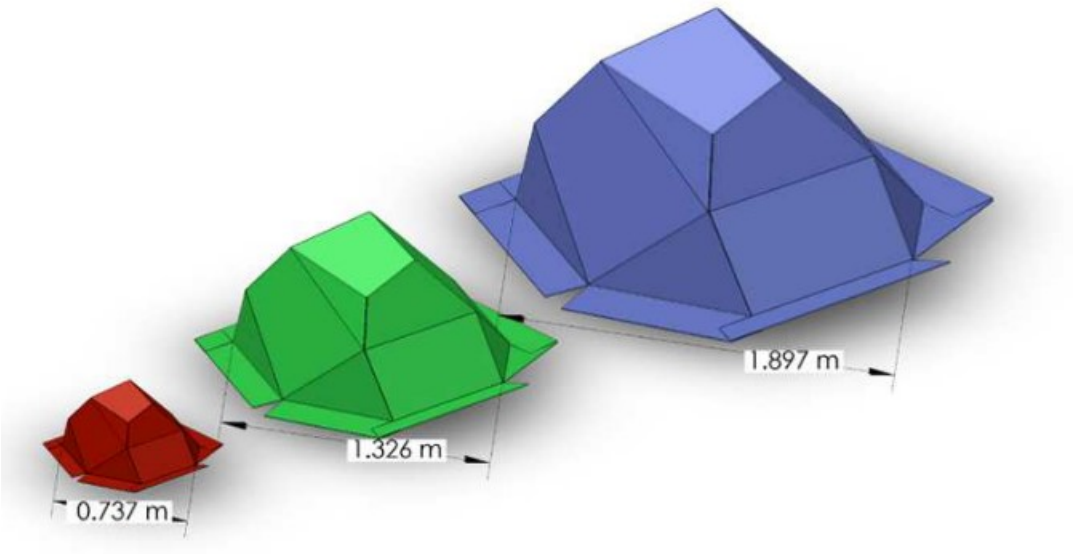


Figure 3.8: Models of the rocks used in the Katwijk Beach Planetary Rover Dataset [28]

boulder was 0.737 meters in diameter. Overall, there were 12 large boulders, 100 medium boulders and 100 small boulders used. A model of each rock is shown in Figure 3.8.

During the data collection process, the rover was equipped with a Velodyne VLP-16 Lidar sensor. The Lidar parameters are provided in Table 3.2. Noteworthy parameters are the Velodyne VLP-16 Lidars 100 meter range and it's point cloud scan rate of 20 Hz with 28928 points per scan. During the first traverse, 15500 point clouds were collected, in the second traverse 10700 point clouds were collected and in the third traverse 9200 point clouds were collected. Additionally, Differential Global Positioning System (DGPS) data was collected every three seconds for the rover position while driving, and for each rock location.

3.2.2 Data Processing

The Velodyne VLP-16 Lidar records data in a local coordinate frame. For data collection, the Lidar was mounted on the rover so that the positive x-axis of the Lidar frame was aligned with the forwards driving direction of the rover, and the positive y-axis was to the left of the rover. The Lidar was mounted so that it was tilted down at an angle of twenty degrees to the ground. By mounting the Lidar in this way, it removed blind spots in-front of the rover and allowed for

Parameter	Value
Horizontal Resolution	1808 channels
Vertical Resolution	16 channels
Range	100 m
Vertical Field of View	30°(±15°)
Vertical Angular Resolution	2°
Points per Second	300 000
Rotation Rate	5 to 20 Hz
Power Draw	8 W
Weight	830 g

Table 3.2: Velodyne VLP-16 Lidar parameters

rocks in front of the rover to be captured by more points as the rover got close to them.

For this thesis, a per-point ground truth model of each point cloud in the Katwijk Beach Planetary Rover Dataset needed to be determined to train machine learning models. In order to determine ground truth models, the Lidar scans were aligned with the known location and estimated heading direction of the rover and overlaid on a georeferenced tiff image. Then, since the location and size of each rock was known, the points overlapping the rock locations could be labelled as rock. To achieve this ground truth model, the following steps had to be taken.

Since the Lidar records a point cloud every 100 milliseconds, and the position of the rover was only taken every three seconds during the traverses, the Lidar scan locations had to be interpolated. Linear interpolation was performed by ordering the unknown scan locations and the known rover locations according to their time stamps, and then performing linear interpolation between each known rover position. A non-linear interpolation method was also considered, to interpolate a function that passed through the known rover locations, and then match the Lidar scan positions to the function based on their time stamp. However, since the rover performed point turns, and the rover's path crosses itself in both the x-axis and y-axis, it was not possible to interpolate a function that would fit the known rover locations. Additionally, because the rover only moved approximately 1.5 meters between each recorded rover location,

a linear interpolation of the Lidar scan locations between each rover location was found to be acceptable.

The next step in aligning the point clouds with the rover location and heading direction was to apply a transformation from the local Lidar frame to the location of the rover in Universal Transverse Mercator (UTM) coordinates taken from the recorded DGPS data. The transform is given in Equation 3.1 and is done by adding the x and y UTM coordinates to the x and y values of all points in the point cloud. This transformation results in the origin of the Lidar data being aligned with the DGPS location on the rover.

$$t_{DGPS} = [p(x) + x_{utm},\ p(y) + y_{utm},\ p(z)] \tag{3.1}$$

A second transformation was applied to translate the Lidar data from the DGPS frame location to the Lidar frame location on the rover. The transformation is given in Equation 3.2 and is done by adding the distance between the DGPS and the Lidar sensors to the x and y coordinates of all Lidar points. After applying the transformation, the Lidar data is aligned with the Lidar location on the rover, but the data still needs to be rotated to properly align with the Lidar frame, as shown in Figure 3.9.

$$t_{lidar} = [p(x) + x_{lidar},\ p(y) + y_{lidar},\ p(z)] \tag{3.2}$$

The next step was to rotate the point cloud so that the Lidar data aligned with the ground plane. The rotation can be determined based on the mounting angle of the Lidar. Since the Lidar was mounted at an angle of twenty degrees to the ground, but the data was collected in the local Lidar frame, a rotation of twenty degrees needed to be applied in order to align the point cloud with the ground plane. The rotation can be done by applying a rotation matrix as defined in Equation 3.3. The effect of this rotation is shown in Figure 3.10. In Figure 3.10a the Lidar does not align with the ground plane because the Lidar is mounted at an angle to the ground, so after applying the rotation into the Lidar frame the point cloud becomes aligned

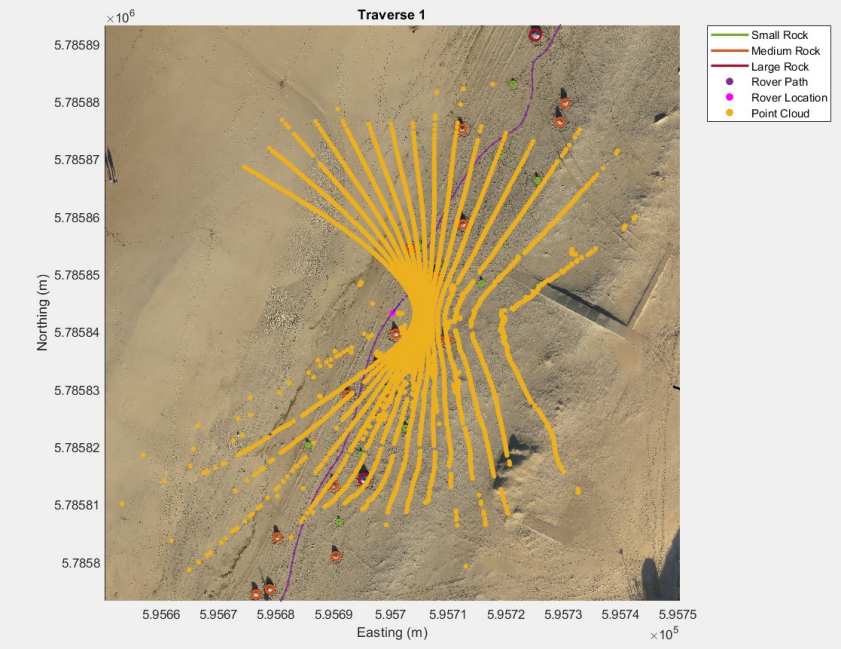


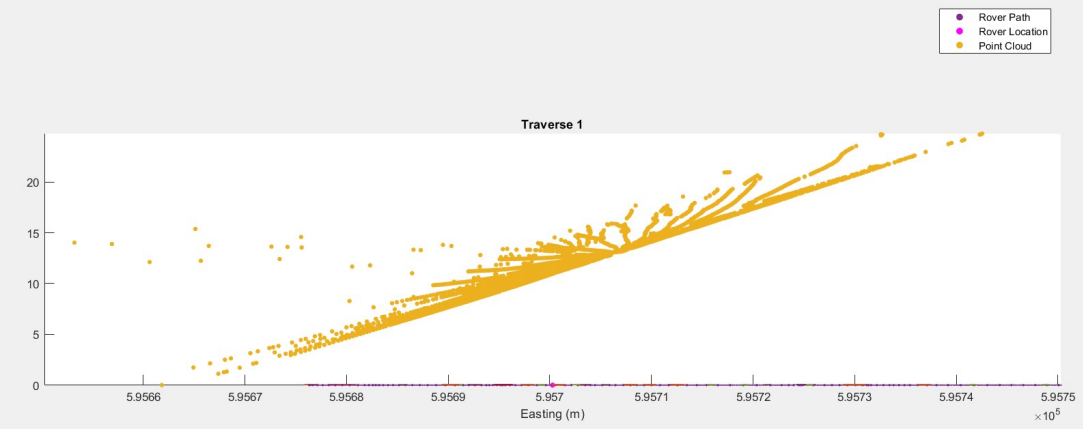
Figure 3.9: Point cloud alignment after translations are applied

with the ground plane as shown in Figure 3.10b.

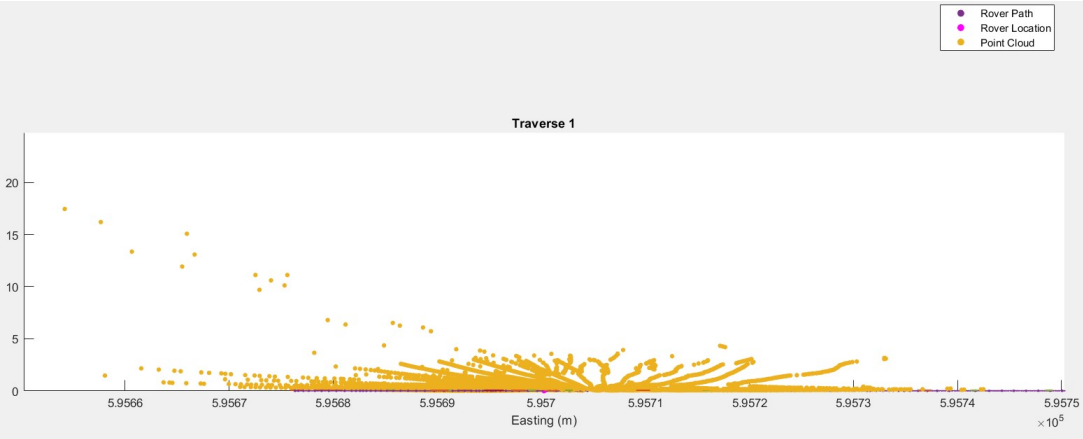
$$R_{ground} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} * P(x, y, z) \tag{3.3}$$

The second rotation serves to rotate the point cloud so that the positive x-axis aligns with the driving direction of the rover. The rotation is required because the Lidar data is defined locally in the Lidar frame where the positive x-axis is aligned with the forward direction, or the heading direction, of the rover. The rotation is defined in Equation 3.4, where the angle θ was calculated as the angle between the x-axis and the estimated heading direction of the rover. Since the ground truth heading direction of the rover was not recorded during the data collection, the driving direction was estimated based on the known locations of the rover. The following methods were tested to calculate the heading direction:

- Using the next known rover location and the current estimated Lidar scan location
- Using the second next known rover location and the current estimated Lidar scan location



(a) Original point cloud orientation



(b) Point cloud after rotation to align with the ground plane

Figure 3.10: Point cloud alignment with the ground plane

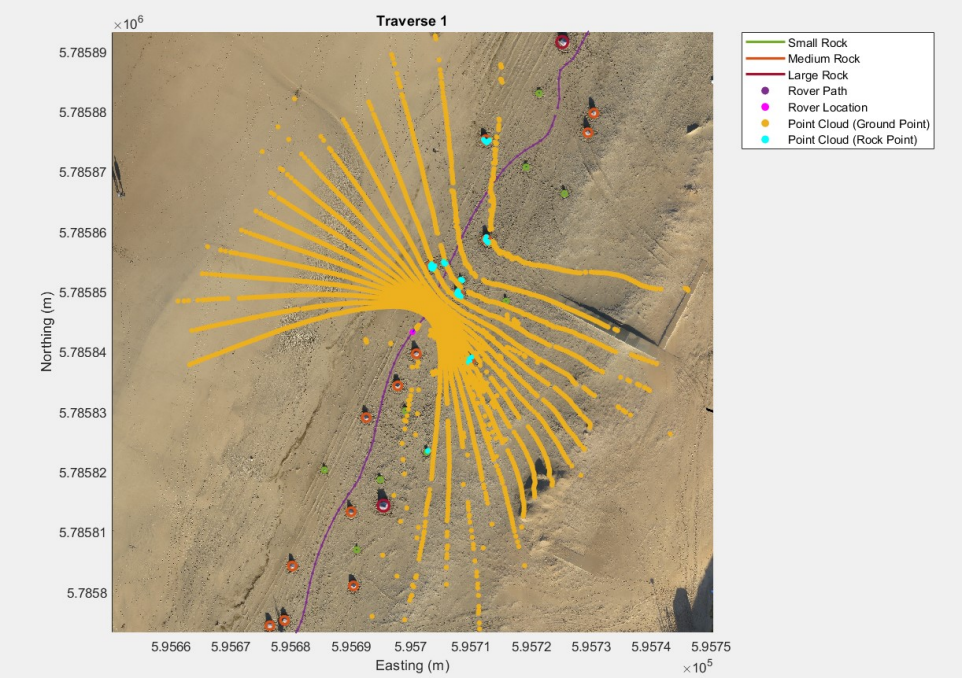


Figure 3.11: Point cloud aligned with estimated heading direction

- Using the next known rover location and the previous known rover location
- Using the second next known rover location and the second last known rover location

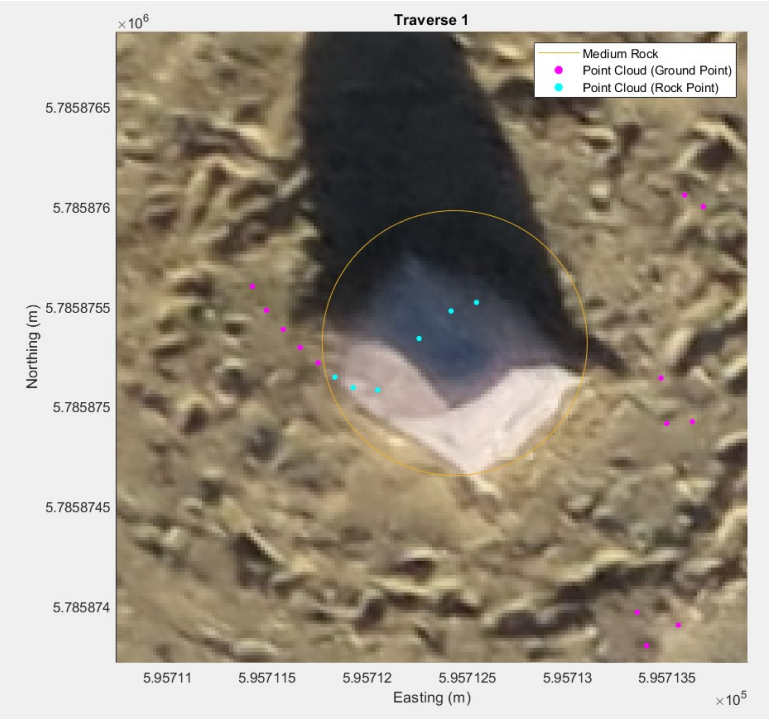
The best heading estimate was found to come from using the second next known rover location and the current estimated scan location. This method resulted in the smoothest heading direction estimate when going around turns, and the most accurate point cloud to rock location alignment overall. Figure 3.11 shows the effect of applying the rotation to align the point cloud with the heading direction, where the forward direction is now aligned with the estimated heading direction of the rover.

$$R_{heading} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * P(x, y, z)) \tag{3.4}$$

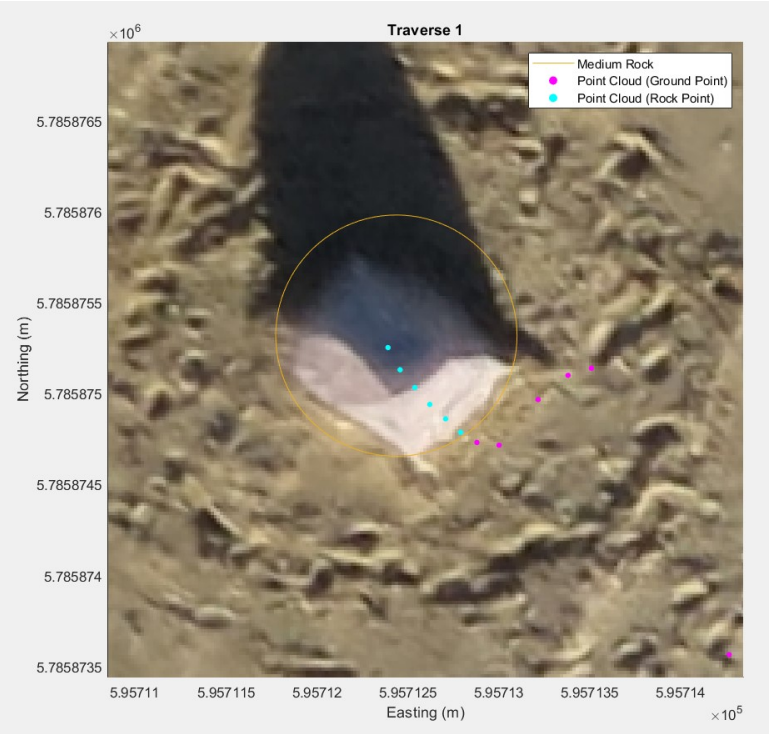
Since the heading direction of the rover was estimated rather than being the ground truth heading direction, there was some variation in the alignment of the point cloud data with the

known rock positions. An example of an over-rotated heading direction estimate is shown in Figure 3.12a and an under-rotated heading direction estimate is shown in Figure 3.12b. For comparison, an example of a correctly rotated point cloud is shown in Figure 3.13. The error in the estimated heading direction is expected when working with real-world data. The DGPS measurements have an accuracy between one to three centimeters, and a small error in the rotation estimate based on the heading direction estimate will result in larger errors for far away points. For example, if the rotation estimate is one degree off, the points at a 50 meter range from the Lidar will be approximately 0.87 meters off from the true rock position.

Having proper ground truth labels of the points that belong to the rock class and the points that belong to the ground class is necessary to be able to use the data to train a machine learning model properly. A method was created in MATLAB to correct the estimated labels. First, a point cloud is displayed in a figure window in MATLAB. The points that belong to the ground class shown in aqua and points that belong to the rock class shown in magenta. The known rock locations, traverse path, and georeferenced tiff image are also displayed in the figure window. Then using the brush tool, the user selects all points that are mislabelled as the ground class and exports an array of coordinated corresponding to the selected point from the figure window to the MATLAB workspace. A function in MATLAB then updates the labels for the points at the selected coordinates. This process is repeated for the points that were mislabelled as the rock class. Once the rock and ground labels have been updated, the user must hit the “enter” key in the MATLAB command window to continue to the next point cloud. An example of an over rotated point cloud where the labels have been updated is shown in Figure 3.14.



(a) Overestimated heading direction resulting in over-rotation of the point cloud



(b) Underestimated heading direction resulting in under-rotation of the point cloud

Figure 3.12: Example of overestimated and underestimated heading directions resulting in misalignment of the points with the rock position

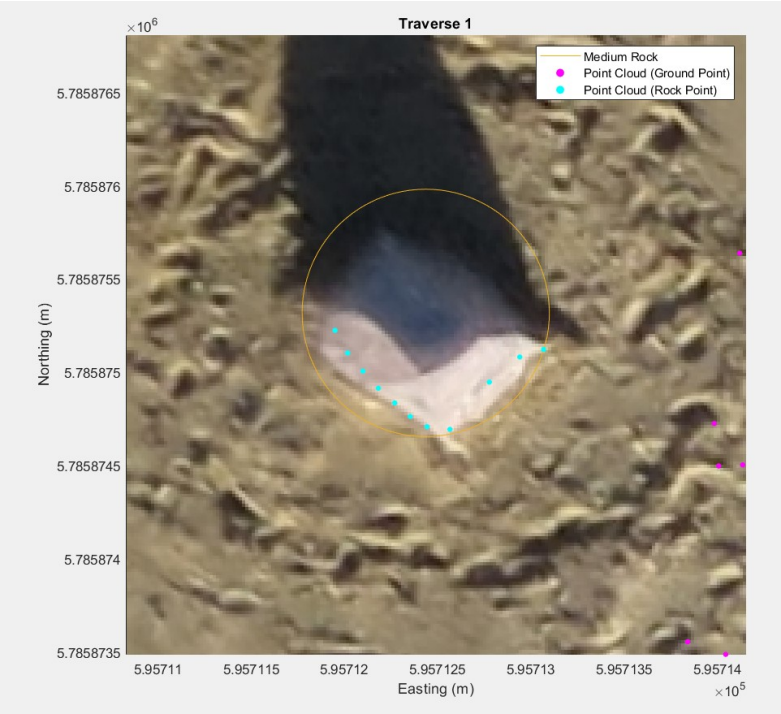


Figure 3.13: Correct estimation of heading direction resulting in correct alignment of point cloud with rock location

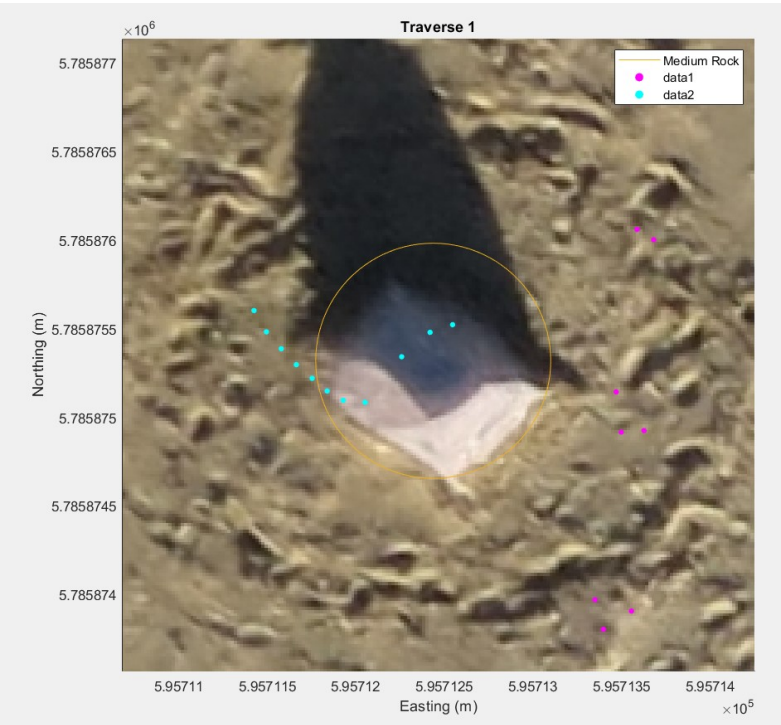


Figure 3.14: Updated labels of a rock in an over rotated point cloud

Chapter 4

Methodology

This chapter outlines the methodology used to perform semantic segmentation of the point clouds contained in the available datasets, and the methodology to test the real-time implementation of the developed models. The following topics will be discussed: the data split, hyperparameter tuning, methods for handling class imbalance, and the method for testing the model performance in real-time.

4.1 Training, Validation, and Testing Split

The data was divided into three parts: the training set, validation set, and testing set. The training set is made up of 60 % of the data and the validation and testing sets are each made up of 20 % of the data. The data from each traverse is sequential, therefore, the data should be split in sequential order to reduce the number of overlapping scans in the training, validation, and testing sets. Each traverse was split so the first 60 % of a traverse was assigned to training, the next 20 % was assigned to validation, and the final 20 % was assigned to testing.

4.2 Experiment One: Semantic Segmentation of the Katwijk Beach Planetary Rover Dataset

The ability to achieve a low computational complexity and accurate segmentation results is important to this thesis to allow for model deployment on-board a rover. Of the networks considered in Section 2.2.3, the network RandLA-Net was selected as a starting point for training a model to detect rock obstacles from point clouds in planetary scenes because the network was able to achieve results of 53.9 % Mean Intersection Over Union (mIOU) when performing semantic segmentation of the SemanticKITTI dataset which contains Lidar scans of city driving scenes with 28 classes, while keeping the computational complexity low: achieving an approximate frame rate of 22 frames per second on an NVIDIA RTX2080Ti GPU [13]. The original RandLA-Net model uses the pre-processing techniques of grid sub-sampling and random cropping to reduce the size of the point clouds used to train the model. The random cropping algorithm works by selecting a point at random and keeping the KNN to that point. The point clouds used for this thesis contain approximately one third the number of points that were used to train the original RandLA-Net model and only two classes; therefore, it is not necessary to reduce the number of points in each point cloud for the training done in this thesis. Further, the grid sub-sampling method and random cropping method could both lead to rock points being removed from the point clouds which should be avoided because there is already a large class imbalance between the rock and ground class in the Katwijk Beach Planetary Rover Dataset and the ATF dataset. For these reasons, the pre-processing steps of grid sub-sampling and random cropping are not applied in this thesis.

A grid search of hyperparameters and methods for handling the class imbalance of the datasets was performed to find the best model for performing semantic segmentation of the planetary datasets. All of the parameter combinations were trained five times and the results were averaged to get a better idea of each model's performance because the random sampling method adds variation to each training set. Early stopping was used while training the models:

Hyperparameter	Value
Number of Layers	4
Batch Size (Training)	6
Initial Learning Rate	1e-2
Number of Epochs	100

Table 4.1: Original hyperparameters of the RandLA-Net model [13]

if the model had not improved with respect to the mIOU within the last 25 epochs, then the training process was stopped.

4.2.1 Hyperparameter Tuning

The hyperparameters of the RandLA-Net model were tuned include the number of layers, the batch size, the initial learning rate, and the number of epochs. The hyperparameters of the original RandLA-Net model that were tuned during this experiment are given in Table 4.1.

4.2.1.1 Number of Hidden Layers

The number of hidden layers required to train a model depends on the complexity of the dataset and the problem to be solved. By increasing the number of layers, it allows the network to solve increasingly complex problems. If there are too many layers it will cause the model to learn the training data too well resulting in overfitting, but if there are too few layers the model will not be able to learn resulting in underfitting. In the RandLA-Net model, the number of layers affects the receptive field of each point because a local feature aggregation module is applied in each layer. Stacking multiple local feature aggregation modules increases the receptive field of each point which increases the information stored about neighbouring points; therefore, by increasing the number of layers, more information about the local structure is captured within a single point. The number of layers explored during this experiment are listed in Table 4.2.

Hyperparameter	Value
Number of Layers	2, 3, 4
Batch Size (Training)	6, 12
Learning Rate	1e-2, 1e-4
Number of Epochs	200*

Table 4.2: Hyperparameters explored while tuning the model (* with early stopping)

4.2.1.2 Batch size

The batch size is the number of instances that will be passed through the network before updating the model. An instance, in the case of this thesis, is one point cloud from the dataset. After each batch, the error between the expected result and the current result of the model is calculated and used to update the model. Batch Gradient Descent is when the batch size is equal to the number of instances available, for example, the total number of instances in the training set. Stochastic Gradient Descent is when the batch size is equal to one. Mini-Batch Gradient Descent is when the batch size lies somewhere between one and the total number of instances available. For this thesis, Mini-Batch Gradient Descent will be explored; the batch sizes are listed in Table 4.2.

4.2.1.3 Initial Learning Rate

The learning rate of a neural network defines how much the model's weights are updated based on the calculated error after each batch. A learning rate that is too small can cause the model to get stuck during training and will result in a slower training process. A learning rate that is too large will result in a faster training process, however, a large learning rate can cause the model to finish too quickly before optimizing the weights and it can cause the weights to diverge resulting in oscillations in the model performance. The original RandLA-Net model sets the learning rate to decay at a rate of 5% of the current learning rate after each epoch. A decaying learning rate allows for large changes in the weights at the beginning of the training

process and reduces the amount by which the weights change as the training process continues, allowing for fine tuning of the weights at the end of the training process. The learning rates that were explored in this experiment are listed in Table 4.2.

4.2.1.4 Number of Epochs

The number of epochs is the number of full passes over the dataset. One full pass over the dataset means that the model has been trained on all instances in the training dataset. An epoch is made up of the number of batches of the dataset that are defined by the batch size. Early stopping is a technique that can be implemented when a model is being trained. Once the model meets a certain criterion, for example, the error is no longer decreasing after a defined number of epochs, then the training process is stopped. For the experiments preformed in this thesis, the maximum number of epochs was set to 200, and the mIOU was used as the criterion for early stopping. If the model had not made improvements with respect to the mIOU score after twenty-five epochs, then the training process was stopped.

4.2.2 Handling Class Imbalance

Naturally, each point cloud scan in the datasets used for this thesis contains a large number of ground points and a small number of rock points which creates a class imbalance. To deal with the class imbalance, random under sampling of the ground class and cost-sensitive methods are explored in the experiments of this thesis. Over sampling techniques, such as random over sampling and SMOTE, were not considered because the class imbalance is created by the composition of each scan. Duplicating scans or generating artificial scans would not reduce the class imbalance, instead the rock class within each point cloud would have to be increased. To increase the presence of the rock class using over sampling methods on pre-existing point clouds, the rock points would have to be placed at the correct angle to the Lidar sensor, and the ground points in the shadow of the rock would have to be removed. These requirements would

Percentage of Random Under Sampling	20%	40%	60%
Number of Points Remaining in the Point Cloud After Sampling	23142	17357	11571

Table 4.3: Percentages of random under sampling of the ground class explored and the number of points remaining in each point cloud after random under sampling

be very difficult to meet using oversampling techniques.

4.2.2.1 Random Under Sampling

Random under sampling of the ground class is applied by determining the index of all points corresponding to the ground class, and randomly selecting a number of points to delete, N, determined by subtracting the number of points desired after random under sampling from the total number of points in the point cloud scan as shown in Equation 4.1.

$$N = len(points_{total}) - len(RUS) \tag{4.1}$$

Then, N points from the ground class are randomly sampled and deleted. The percentages of randomly sampled points and the corresponding number of points remaining in the point clouds after random under sampling that are explored in this thesis are summarized in Table 4.3.

4.2.2.2 Cost Sensitive Methods

Two cost sensitive methods were explored: a cost matrix, and inverse cost frequency. For this thesis, false negative classifications are much more serious than false positive classifications. A false negative could lead to the rover hitting a rock and getting damaged, whereas a false positive will result in the rover avoiding areas that did not necessarily need to be avoided. The cost matrices explored in the experiments are displayed in Table 4.4. In both matrices the cost of a false negative is set higher than the other costs and there is no cost for a correct

Cost Matrix A	Actual Class	
Predicted Class	0	1
	10	0

(a) Cost Matrix A

Cost Matrix B	Actual Class	
Predicted Class	0	1
	50	0

(b) Cost Matrix B

Table 4.4: Cost matrices explored in experiment one

classification. Cost matrix A assigns a cost of ten for a false negative classification and a cost of one for a false positive classification. Cost matrix B assigns a cost of fifty for a false negative classification and a cost of one for a false positive classification. The inverse cost frequency method takes into account the ratio of the total points in each class relative to the total number of points in the dataset, as shown in Equation 4.2.

$$weight = \frac{num_{class}}{num_{total}} \tag{4.2}$$

Then the cost of predicting a false negative and false positive are calculated as the inverse of the weight plus a scaling factor, as shown in Equation 4.3. The scaling factor is used to limit maximum cost of misclassifying the minority class; for example, if the number of points in the minority class is very small compared to the total number of points in the dataset the weight would be approximately zero, so without a scaling factor the cost of misclassifing the minority class would approach infinity. When the scaling factor is set to 0.02, it limits the maximum cost of misclassifying the minority class to be approximately 50. The resulting costs calculated from the total class imbalance in the Katwijk Beach Planetary Rover Dataset are 29 for a false negative and 1 for a false positive.

$$cost = \frac{1}{weight + 0.02} \tag{4.3}$$

The cost of predicting a false negative is calculated using the weight of the positive class, in this case the rock class. The cost of predicting a false positive is calculated using the weight of the negative class, in this case the ground class.

Cost Matrix A	Actual Class	
Predicted Class	0	5
	30	0

(a) Cost Matrix A

Cost Matrix B	Actual Class	
Predicted Class	0	5
	50	0

(b) Cost Matrix B

Cost Matrix C	Actual Class	
Predicted Class	0	10
	30	0

(c) Cost Matrix C

Cost Matrix D	Actual Class	
Predicted Class	0	10
	50	0

(d) Cost Matrix D

Table 4.5: Cost matrices explored in iteration one of experiment one

4.2.2.3 Iterating the Cost Sensitive Methods

After analyzing the results of the initial grid search presented in Section 5.1, the models had a significantly higher number of false positives than false negatives. The cost sensitive methods were tuned by increasing the cost of a false positive to see if the false positive rate could be decreased while keeping the false negative rate low. The second set of cost matrices considered are given in Table 4.5.

4.3 Experiment Two: Semantic Segmentation on the Analogue Terrain Facility Dataset

Significant differences exist between the two datasets used for this thesis: each Lidar has different parameters resulting in different density representations of the scenes; the ATF dataset has 3 % the number of point clouds and covers 2 % of the distance compared to the Katwijk Beach Planetary Rover Dataset, resulting in the ATF dataset being under-represented; and the artificial obstacles in the Katwijk Beach Planetary Rover Dataset have consistent shape, size, and texture compared to the real rocks in the ATF that vary in shape, size, and texture. For

Model Number	Number of Layers	Batch Size	Learning Rate	Cost Method
1	4	6	1e-2	[0, 5; 30, 0]
2	4	6	1e-2	[0, 10; 30, 0]
3	4	12	1e-2	[0, 10; 50, 0]

Table 4.6: Parameters of the models to be trained and tested on the ATF dataset

all of these reasons, the datasets were not mixed together for training. Instead, after training on the Katwijk Beach Planetary Rover Dataset in Experiment One, the top three performing combinations of model parameters, presented in Table 4.6, were applied to the ATF dataset to test the model's ability to learn representations of real rocks with varying size, shape, and texture. Experiment Two will examine how well the models performs on a more complex dataset compared to Experiment One.

4.4 Experiment Three: Real-time Inference of Point Clouds

The ability of the model to segment point clouds in real time with limited computational power is important so the model is able to be deployed on-board a planetary rover. To test the ability to deploy the model on board a rover, the top three performing models from Experiment One were tested to be used in a real-time simulation using an Intel Xeon E5-2665 CPU which has a processor speed of 2.4 GHz. The time to perform scene segmentation on the 480 point clouds in the test set of the Katwijk Beach Planetary Rover Dataset was recorded and averaged. The results were compared with the segmentation frame rate required for implementing the model on board a rover, and the frame rate which the Lidar used to create the dataset publishes point clouds.

Chapter 5

Results And Discussion

This chapter presents the results of the experiments outlined in Chapter 4 and discusses the implications of the results.

5.1 Semantic Segmentation on the Katwijk Beach Planetary Rover Dataset

5.1.1 Tuning the Hyperparameters and Class Balancing Methods

The first step in this experiment consisted of tuning the number of layers while keeping the other hyperparameters the same. The model was trained with four, three, and two layers and the average validation performance metrics are given in Table 5.1. The averaged validation performance metrics of the model are similar when using four layers and three layers, and are worse when using two layers, indicating that with only two layers, the model is not complex enough to consistently learn the representation of the rock and ground classes in the point clouds. Since there is a large class imbalance, the model naturally achieves very high accuracy by correctly predicting the ground class, so the recall and precision scores are good indicators of the model's ability to predict the minority class, in this case the rock class. The recall score gives the ratio of correctly predicted rock class labels to the true total number of rock class

Number of Layers	Accuracy (%)	mIOU (%)	Recall (%)	Precision (%)	F1 (%)
4	99.16	83.93	92.11	73.13	81.26
3	99.20	83.68	91.86	69.61	80.20
2	98.97	80.00	91.49	64.89	75.43

Table 5.1: Comparison of average performance metrics of models trained with two, three, and four layers

labels. The recall scores were similar between each model, with a decrease of only 0.62 % as the number of layers decreased from four to two, indicating that the number of layers did not have a large impact on the number of false negative predictions. The precision score gives the ratio of correctly predicted rock class labels to the total number of predicted rock class labels. The precision scores decrease by 8.24 % as the number of layers were reduced from four to two layers, and indicates that number of layers has a greater impact on the number of false positive predictions. Figure A.1 in Appendix A shows a comparison of the accuracy and loss during training and validation for the model trained with four, three, and two layers. One key difference seen in the figures is that the model with four layers stopped training just before reaching 50 epochs, whereas the model with three layers stopped training just before 70 epochs, and the model with two layers stopped training around 100 epochs. The number of epochs required for training show that the model with four layers was able to learn faster and achieve better results. Overall, the loss and accuracy plots show that the models were not overfit to the training data, and they were able to learn a representation of the dataset. Since the model performance was found to decrease as the number of layers decreased, the remainder of the experiments are performed on models with four layers only.

A grid search was performed on the remaining hyperparameters and class imbalance techniques outlined in the Experiment One methodology. While performing the grid search with 20 % random under sampling applied, the model was either not able to learn, or it would overfit to the training set. Figure A.2 shows the results of applying 20 % random under sampling on all combinations of the other hyperparameters. The loss plots either do not converge indicating the model is not learning, or the validation loss increases as the training continues indicating

that the model is overfit to the training dataset. Random under sampling likely did not perform well because it changes the density of the ground plane of the point clouds resulting in the network learning an incorrect representation. The network works by taking a random sample of points and using the Local Spatial Encoding unit to create a feature vector for each point that encodes information about the surrounding points. After applying random under sampling to the ground class, the network would learn a less dense representation of the ground during training, and when the full point cloud was fed in for validation, the model could not properly classify the point cloud. Because of these results, no further exploration into random under sampling was done.

The grid search was continued on the rest of the hyperparameters and cost sensitive methods for handling class imbalance. The training and validation loss and accuracy plots of the grid search on are given in Figure A.3 in Appendix A, and show that the loss was able to converge for most of the tested hyperparameters and class balancing methods. The loss and accuracy of the models were less stable when the learning rate was decreased. The average performance metrics for each combination of parameters are given in Table 5.2. The following conclusions about the effect of the hyperparameters and class balancing methods were drawn from the results:

- Decreasing the learning rate resulted in a decrease in the performance of the model
- Increasing the cost of a false negative resulted in the number of false negatives decreasing, but the number of false positives tended to increase
- Increasing the batch size when the learning rate was 1e-2 resulted in the number of false negatives and false positives decreasing

After analyzing the results of the grid search, a question arose as to whether the false positive rate could be decreased by increasing the cost of false positives in the cost matrices. The loss and accuracy plots for training and validation on the new set of cost matrices are presented

Model Description			**Performance**				
Learning Rate	Cost Method	Batch Size	Accuracy (%)	mIOU (%)	Recall (%)	Precision (%)	F1 (%)
1e-2	Inverse Cost	6	99.16	83.93	92.11	73.13	81.26
1e-2	Inverse Cost	12	99.21	85.89	96.52	74.54	84.12
1e-2	[0, 1; 10, 0]	6	97.73	72.56	89.66	50.17	64.34
1e-2	[0, 1; 10, 0]	12	99.20	81.35	93.19	66.60	77.69
1e-2	[0, 1; 50, 0]	6	98.81	74.59	90.47	53.21	67.01
1e-2	[0, 1; 50, 0]	12	98.42	76.36	94.89	55.96	70.40
1e-4	Inverse Cost	6	97.00	67.95	93.49	40.04	56.07
1e-4	Inverse Cost	12	96.41	61.15	90.80	26.64	41.19
1e-4	[0, 1; 10, 0]	6	98.70	76.61	92.13	57.21	70.59
1e-4	[0, 1; 10, 0]	12	98.42	73.86	84.33	54.31	66.07
1e-4	[0, 1; 50, 0]	6	95.94	64.08	92.22	33.21	48.83
1e-4	[0, 1; 50, 0]	12	94.00	58.79	88.64	24.43	38.30

Table 5.2: Performance results of models tested during a grid search of the hyperparameters and methods for handling class imbalance

Model Description			Performance				
Learning Rate	Cost Method	Batch Size	Accuracy (%)	mIOU (%)	Recall (%)	Precision (%)	F1 (%)
1e-2	[0, 5; 30, 0]	6	99.51	89.11	92.55	84.04	88.09
1e-2	[0, 5; 30, 0]	12	99.62	89.10	87.22	88.82	88.01
1e-2	[0, 10; 30, 0]	6	99.65	90.33	91.68	87.45	89.51
1e-2	[0, 10; 30, 0]	12	99.58	88.27	83.20	91.11	86.98
1e-2	[0, 5; 50, 0]	6	99.65	87.31	92.45	79.86	85.69
1e-2	[0, 5; 50, 0]	12	99.33	84.88	90.57	76.01	82.65
1e-2	[0, 10; 50, 0]	6	99.61	88.14	84.19	89.59	86.80
1e-2	[0, 10; 50, 0]	12	99.53	89.25	88.24	88.27	88.26

Table 5.3: Performance results of models tested with the iterated set of cost methods for handling class imbalance

in Figure A.4 in Appendix A and show that the model was able to learn the dataset representation without overfitting. The results of the second grid search, performed with the modified cost matrices, are given in Table 5.3 and show that increasing the cost of a false positive resulted in fewer false positive classifications as seen through the increase in the precision scores. Increasing the batch size tended to result in a increased precision score; however, the recall score tended to decrease which is undesirable because the recall score is linked to the number of false negatives which are very important to minimize because a false negative indicates that a rock point has been missed. When the cost of a false positive was adjusted to 5, the recall scores remained similar to before the cost matrix modifications, which indicates that changing the cost of a false positive to 5 did not have an impact on the number of false negatives. When the cost of a false negative was increased to 10, it resulted in a decrease in the recall scores, which indicates that increasing the cost too much had a negative impact on the number of false negatives.

5.1.2 Performance on the Test Set

The top three performing models based on the F1-scores from the training and validation stage, whose parameters are summarized in Table 5.4, were run on the test set of data to determine the unbiased performance of each model. The results of the test are given in Table 5.5, and are broken down by test area, where test area one and two are from traverse one, test area three is from traverse two and test area four is from traverse three. Additionally, the average performance results over all four test sets of data are provided for each tested model. All three models performed the worst on test area four, which is likely caused by two factors: first, the rocks were placed to be denser in traverse three than in traverse one and two; and second, traverse three had the least amount of point cloud scans, so it was the least represented in the dataset. Another important result is that the models all achieved high recall scores, with scores ranging from 88.50 to 100.00 %, but the precision scores were lower and had a larger range of 50.85 to 96.71 %. A high recall score indicates a low false negative rate which is very important because it indicates that rocks are not being missed in the point cloud scenes. The low precision scores indicate that a high number of false positives are occurring; however, false positives are not a always a bad thing depending on where they occur. If a false positive occurs next to a true rock location, the false positive can be considered padding around the rock; this will be discussed further in Section 5.4. Overall, the model with the best performance based on the F1-score is model number two which achieved an average F1-score of 89.13 %. The performance of the model can be seen in Figure B.1 in Appendix B which shows a comparison of ground truth scans to the models predictions from a bird's-eye view of point clouds from the test sets. Figure B.1b shows a case where all points making up a rock are missed. Figure B.1d shows a case of a partial missclassification of a rock where some points in the rock are correctly predicted and other are missed, and a case of a false detection where ground points are incorrectly predicted as rock but there are not any nearby rocks. Figure B.1h shows a case where all points making up a rock are missed, and a case where several false detections have occurred resulting in a group of false rocks.

Model Number	Number of Layers	Batch Size	Learning Rate	Cost Method
1	4	6	1e-2	[0, 5; 30, 0]
2	4	6	1e-2	[0, 10; 30, 0]
3	4	12	1e-2	[0, 10; 50, 0]

Table 5.4: Parameters of the top three performing models during training and validation on the Katwijk Beach Planetary Rover Dataset

Model Number	Test area	Accuracy (%)	Miou (%)	Recall (%)	Precision (%)	F1 (%)
1	1	99.57	86.65	89.67	96.71	93.06
	2	99.54	87.29	98.13	77.27	86.46
	3	99.78	84.75	100.00	74.27	85.24
	4	99.58	90.60	93.75	57.69	71.43
	Average	**99.62**	**87.33**	**95.39**	**76.49**	**84.05**
2	1	99.54	85.36	88.50	89.76	89.13
	2	99.67	89.72	95.59	89.08	92.22
	3	99.85	88.09	100.00	92.70	96.21
	4	99.65	91.80	93.75	68.18	78.95
	Average	**99.68**	**88.74**	**94.46**	**84.93**	**89.13**
3	1	99.57	86.97	94.13	87.75	90.83
	2	99.43	85.21	98.30	78.24	87.13
	3	99.74	82.99	100.00	64.80	78.64
	4	99.47	88.79	93.75	50.85	65.93
	Average	**99.55**	**85.99**	**96.54**	**70.41**	**80.63**

Table 5.5: Performance of the top three models on the test set of the Katwijk Beach Planetary Rover Dataset

Model Description			Performance				
Learning Rate	Cost Method	Batch Size	Accuracy (%)	mIOU (%)	Recall (%)	Precision (%)	F1 (%)
1e-2	[0, 5; 30, 0]	6	99.32	90.78	93.26	87.46	90.27
1e-2	[0, 10; 30, 0]	6	99.41	91.91	92.43	90.69	91.55
1e-2	[0, 10; 50, 0]	12	99.49	91.44	91.11	90.78	90.94

Table 5.6: Validation performance of the models on the ATF dataset

5.2 Semantic Segmentation on the Analogue Terrain Facility Dataset

The validation results obtained after training the top three models on the ATF dataset, given in Table 5.6, show that the models achieved good precision and recall scores, indicating that the models are capable of learning more complex representations of obstacles in planetary scenes. The ATF dataset contains real rocks, so their size, shape, and texture are different than the artificial rocks placed in the Katwijk Beach Planetary Rover Dataset. The artificial rocks repeat in size and shape throughout the dataset, whereas in the ATF dataset each rock is unique.

The performance of the models on the training set are given in Table 5.7 and show that all three models achieved high recall and precision scores, but model number two performed the best with an F1-score of 91.40 %. Figure B.2 in Appendix B shows a bird's-eye view comparison of the ground truth point cloud labels to the predicted labels from each model. The comparison shows that each model tends to over-predict the rock areas, meaning that there are false positives occurring right next to true rock locations which is not a serious misclassification and will be discussed further in Section 5.4.

Model Description			Performance				
Learning Rate	Cost Method	Batch Size	Accuracy (%)	mIOU (%)	Recall (%)	Precision (%)	F1 (%)
1e-2	[0, 5; 30, 0]	6	97.08	83.54	98.27	81.75	89.25
1e-2	[0, 10; 30, 0]	6	97.28	84.08	96.28	86.99	91.40
1e-2	[0, 10; 50, 0]	12	97.03	83.07	98.17	81.38	88.99

Table 5.7: Performance of the models on the test set of the ATF dataset

5.3 Real-Time Implementation Analysis

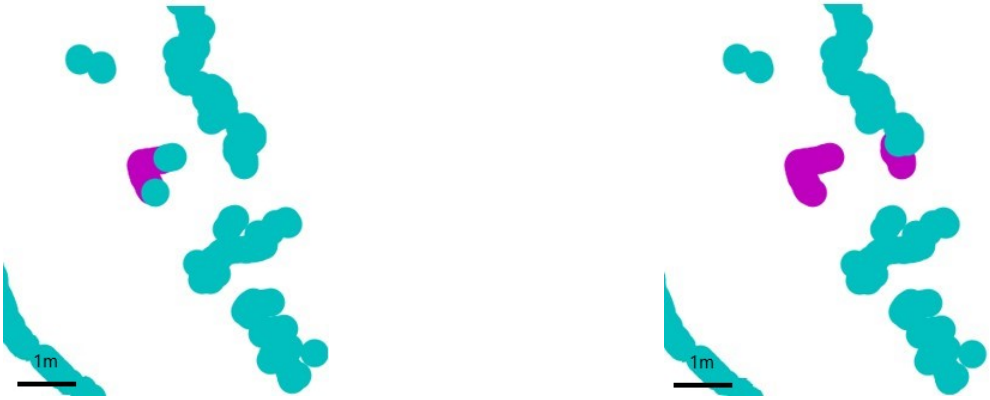
The average time required for the models to perform semantic segmentation of a point cloud scene is presented in Table 5.8. The results show that average time per point cloud required for each model to perform semantic segmentation on the test set is very similar: 0.0068 seconds separate the fastest model from the slowest model. The small time difference between models is likely because the only variation between the tested models is the cost matrix applied to handle the class imbalance. Changing the cost matrix should not affect the models speed, but only the model's ability to predict a correct classifications. If the trained model was applied to the rover, with similar compute capabilities as the tested CPU, it would provide the ability to perform scene segmentation at a rate of approximately 0.63 seconds per point cloud. The most recent NASA Mars rover has a maximum driving speed of 4.2 cm per second on flat, hard ground [29], so the model would allow the rover to know the updated position of surrounding obstacles at approximately every 2.65 cm. The Velodyne VLP-16 Lidar publishes point clouds at a rate of 10 point clouds per second, so the model would be capable of segmenting approximately every sixth point cloud recorded by the Lidar in real-time. The current NASA Mars rover has a processor speed of 200 MHz, which is ten time faster than previous Mars rovers [30] and, as mentioned in section 1.1, work is being done to continue improving the processing capabilities of planetary rovers.

Model Number	Time (Seconds/Point Cloud)
1	0.6306
2	0.6238
3	0.6275

Table 5.8: Average time per point cloud required to perform semantic segmentation on the Katwijk Beach Planetary Rover Dataset test set

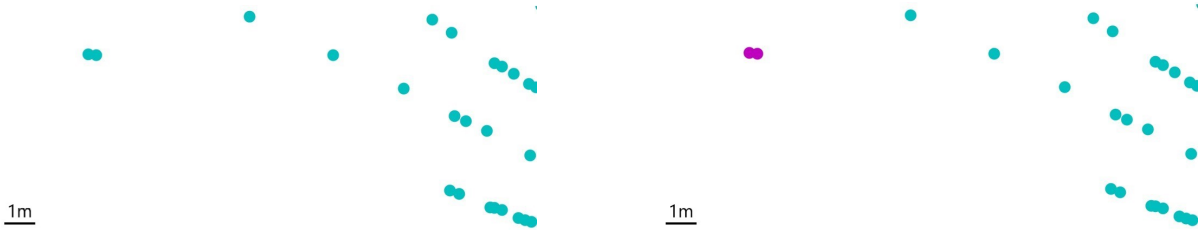
5.4 Impact of False Negative and False Positive Classifications

The seriousness of false negative and false positive classifications depends on two factors: the location of the point and the classification of the neighbouring points. If a false negative occurs but some or all other points belonging to that rock are correctly identified, as shown in Figure 5.1, then the false negative is not very serious because the identified rock area will be padded and avoided. When a false negative occurs and no other points in that rock are correctly identified, as shown in Figure 5.2, then the false negative is a very serious misclassification because a rock has been missed and may damage the rover if it runs into the missed rock. Additionally, false negatives are more likely to occur farther away from the rover because the scan density of Lidar point clouds decrease with distance; therefore, far away rocks have a lower density representation than close rocks and are easier to misclassify. However, false negatives which occur far away from the rover are not as serious because the rover is not close enough to be damaged by the missed rock and as the rover moves closer to the rock, the rock density in the point cloud will increase and the rock will be more likely to be detected. If a false positive occurs and its neighbouring points belong to a rock then the misclassification is not very serious because the rock will be padded and avoided anyways; for example Figure 5.3 shows false positive classifications occurring around rocks that result in the predicted size of the rock being greater than the ground truth. If a false positive occurs and its neighbouring points do not belong to a rock, as shown in Figure 5.4, it will result in the rover unnecessarily avoiding the area and can be serious if the rover drives on a longer path than necessary, or if too



(a) Predicted labels (b) Ground truth labels

Figure 5.1: Bird's-eye view comparison of ground truth labels and predicted labels with false negative classifications where the neighbour rocks points are correctly identified, rock points are shown in magenta and ground points are shown in cyan



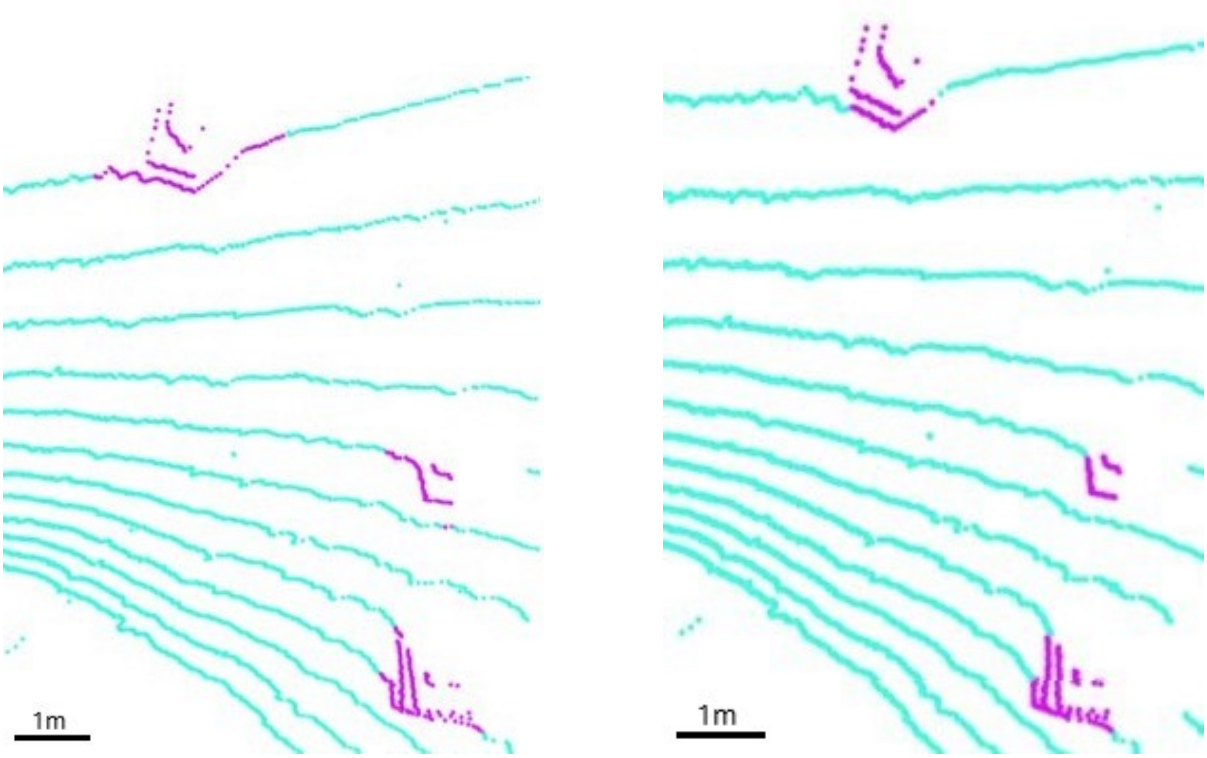
(a) Predicted labels (b) Ground truth labels

Figure 5.2: Bird's-eye view comparison of ground truth labels and predicted labels with false negative classifications resulting in a missed rock, rock points are shown in magenta and ground points are shown in cyan

many false positives occur that are not near true rocks, making the terrain seem untraversable.

5.5 Limitations of the Model

The model is limited in its ability to generalize to variations in point cloud density that the model was not trained on. When the model is trained and tested on point clouds that are generated by the same Lidar sensor, the model can successfully generalize to both dense and sparse representations of the ground and the rocks. However, if the model is trained on point clouds with characteristics from one Lidar sensor and tested on point clouds with characteristics of a different Lidar sensor then the model will be unable to perform segmentation with good results.



(a) Predicted labels (b) Ground truth labels

Figure 5.3: Bird's-eye view comparison of ground truth labels and predicted labels with false positive classifications extending the size of the rocks where rock points are shown in magenta and ground points are shown in cyan

(a) Predicted labels (b) Ground truth labels

Figure 5.4: Bird's-eye view comparison of ground truth labels and predicted labels with false positive classifications that are not next to any true rock locations, rock points are shown in magenta and ground points are shown in cyan

The problem with generalizing between point clouds scans that have different characteristics was evident in Experiment One where the density of the point clouds was changed using random under sampling for training and the model could not generalize to the original point clouds for validation.

Chapter 6

Conclusion

The work in this thesis presents a way to use NNs to efficiently segment obstacles from Lidar point clouds and proves the viability of performing the segmentation on-board a rover. The scope of this thesis was to use pre-processed point cloud data with per-point labels to train NN machine learning models to perform semantic segmentation of point cloud scenes that represent planetary environments, and to detect obstacles within the point clouds. Modifications to a NN called RandLA-Net, including modifying the data processing, tuning the hyperparameters, and applying methods to handle the class imbalance, resulted in the best overall segmentation accuracy of 99.68 %, where each point was identified as either the rock class or the ground class. The model achieved a recall score of 94.46 % and a precision score of 84.93 %. A segmentation rate of 0.6238 seconds/pointcloud was achieved by the model on an Intel Xeon E5-2665 CPU, indicating that the model could be deployed on-board a rover with similar processing capabilities. Overall, the resulting technique developed in this thesis could be confidently used to inform a rover's path planner of the location of surrounding obstacles in environments with sandy terrain with rock obstacles.

6.1 Future Work

A future direction for this research is to train a machine learning model to be robust to identifying different types of terrain and obstacles, and to identifying obstacles in point clouds generated by Lidars with different parameters. A new Lidar dataset that contains a variety of obstacles and terrain types that are present in planetary environments would need to be collected and could be combined with the ATF dataset and the Katwijk Beach Planetary Rover Dataset to train a robust model that is capable of detecting a variety of obstacles.

Another future direction for this research work is to develop a tracking algorithm to track the detected obstacles between sequential Lidar scans. By tracking the detected obstacles, it would allow for the rover to predict the existence of an obstacle if it was undetected in one scan, but it was detected in sequential scans. Further, it could allow the rover to detect false positive classifications when an obstacle is detected in only one scan and not detected in sequential scans.

Bibliography

[1] "Communications." [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/communications/

[2] T. R. Weiss, "How nasa is using ai to develop the next generation of self-driving planetary rovers," Apr 2021. [Online]. Available: https://www.enterpriseai.news/2021/04/01/how-nasa-is-using-ai-to-develop-the-next-generation-of-self-driving-planetary-rovers/

[3] "The cameras on the mars 2020 perseverance rover." [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/cameras/#Engineering-Cameras.

[4] C. R. Qi, H. Su, K. Mo, and L. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Apr 2017.

[5] P. Allard, C. Dubois, S. Gemme, D. Gingras, N. Jackson, T. Lamarche, and M. Picard, "Answers to your questions about the csa's rovers," Dec 2020. [Online]. Available: https://www.asc-csa.gc.ca/eng/rovers/answers-to-questions-about-csa-rovers.asp

[6] "Laser-powered rover to explore moon's dark shadows," May 2020. [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Laser-powered_rover_to_explore_Moon_s_dark_shadows

[7] "Nasa ames solicitation: Planetary (lunar) rover lidar." [Online]. Available: http://spaceref.com/news/viewsr.html?pid=47369

[8] "Machine learning-based analytics for autonomous rover systems (maars)." [Online]. Available: https://www-robotics.jpl.nasa.gov/tasks/showTask.cfm?FuseAction=showTask&TaskID=343&tdaID=700138

[9] H. Thomas, "Learning new representations for 3d point cloud semantic segmentation," Ph.D. dissertation, 11 2019.

[10] "Kinect point cloud basket," Jun 2013. [Online]. Available: https://boofcv.org/index.php?title=File:Kinect_point_cloud_basket.jpg

[11] "Aerial point cloud." [Online]. Available: https://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/GUID-0383FDFE-6B0C-4B69-967A-966F647426C5-web.png

[12] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2, p. 71–78, 1992.

[13] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "Randla-net: Efficient semantic segmentation of large-scale point clouds," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[14] U. Sorgel and S. Schmohl, "Als point cloud classification with convolutional neural networks." [Online]. Available: https://www.ifp.uni-stuttgart.de/en/research/remote_sensing/als_point_cloud_classification/

[15] M. Winter, S. Rubio, R. Lancaster, C. Barclay, N. Silva, B. Nye, and L. Bora, "Detailed description of the high-level autonomy functionalities developed for the exomars rover."

[16] J. Patterson and A. Gibson, *Deep learning: a practitioners approach*. OReilly, 2017.

[17] D. Habermann, A. Hata, D. Wolf, and F. Osorio, "Artificial neural nets object recognition for 3d point clouds," *2013 Brazilian Conference on Intelligent Systems*, Jan 2014.

[18] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet : Deep hierarchical feature learning on point sets in a metric space," Jun 2017.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[20] R. Thalineau, "Deep learning with point clouds," Mar 2019. [Online]. Available: https://www.qwertee.io/blog/deep-learning-with-point-clouds/

[21] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[22] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on x-transformed points," Nov 2018.

[23] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 1, 2018.

[24] H.-I. Lin and M. C. Nguyen, "Boosting minority class prediction on imbalanced point cloud data," *Applied Sciences*, vol. 10, no. 3, p. 973, 2020.

[25] R. Alencar, "Resampling strategies for imbalanced datasets," Nov 2017. [Online]. Available: https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

[26] J. Brownlee, "Cost-sensitive learning for imbalanced classification," Feb 2020. [Online]. Available: https://machinelearningmastery.com/cost-sensitive-learning-for-imbalanced-classification/

[27] Q. Zou, S. Xie, Z. Lin, M. Wu, and Y. Ju, "Finding the best classification threshold in imbalanced classification," *Big Data Research*, vol. 5, p. 2–8, Jan 2016.

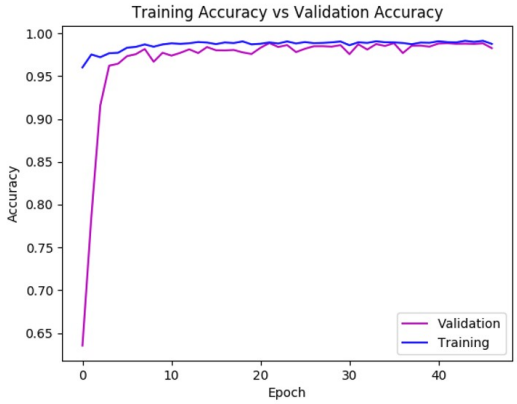
[28] R. Hewitt, E. Boukas, M. Azkarate, M. Pagnamenta, J. Marshall, A. Gasteratos, and G. Visentin, "The Katwijk beach planetary rover dataset," *International Journal of Robotics Research*, Dec 2017.

[29] "Rover wheels." [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/wheels/

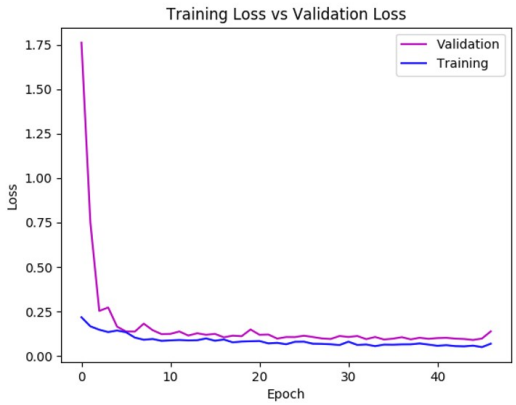
[30] "Rover brains." [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/brains/

Appendix A

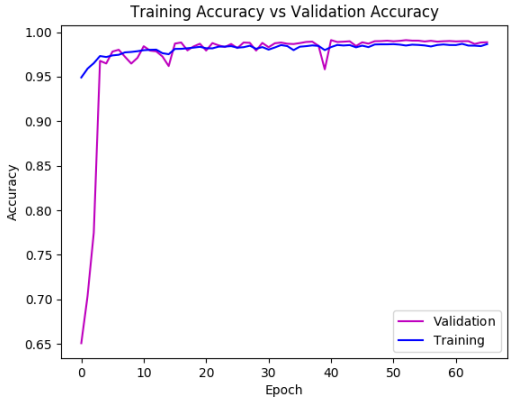
Training and Validation Graphs for Chapter 5



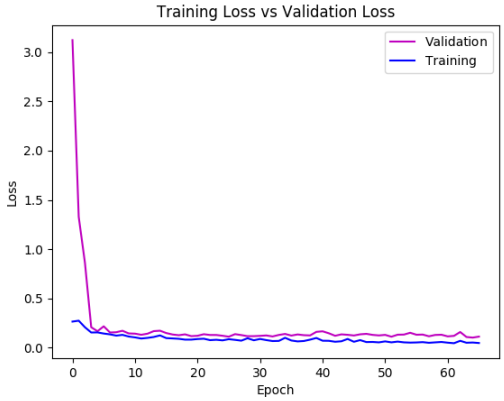
(a) Four layer training and validation accuracy



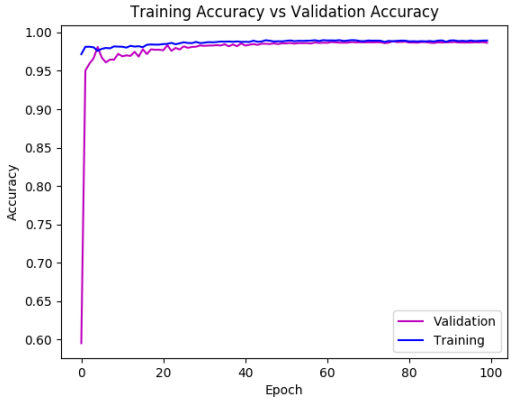
(b) Four layer training and validation loss



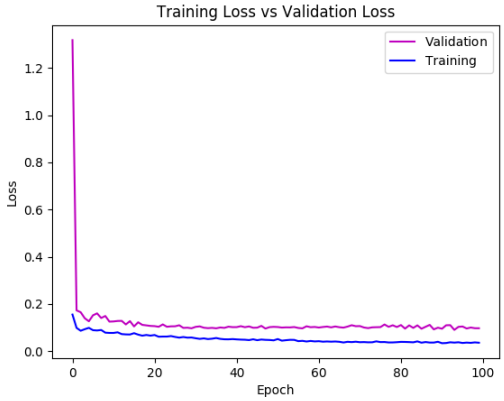
(c) Three layer training and validation accuracy



(d) Three layer training and validation loss

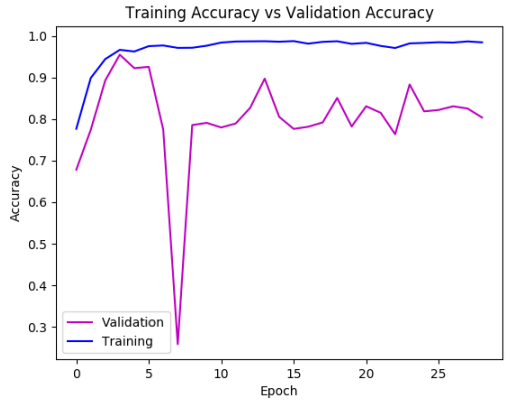


(e) Two layer training and validation accuracy

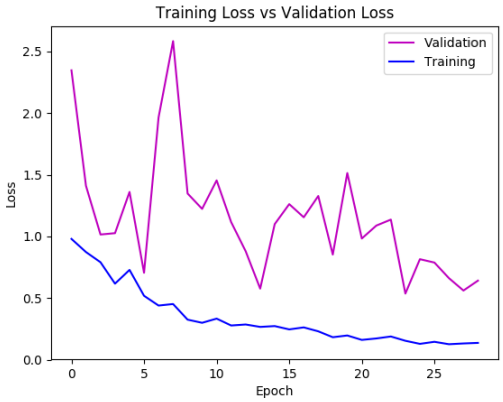


(f) Two layer training and validation loss

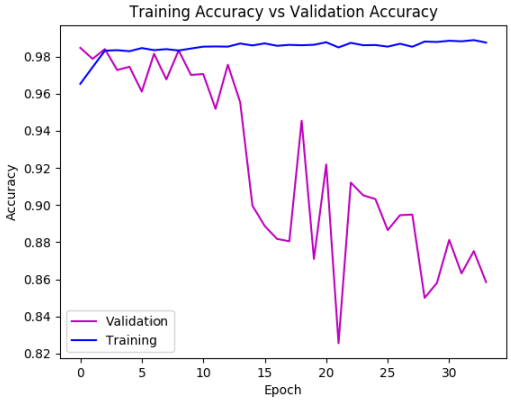
Figure A.1: Training and validation loss and accuracy plots for models with two, three, and four layers



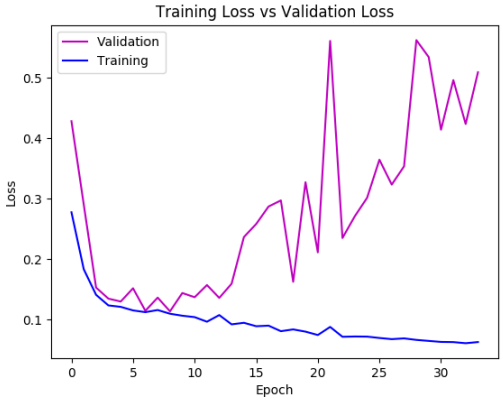
(a) 20% random under sampling with learning rate 1e-2, inverse cost, and batch size 6



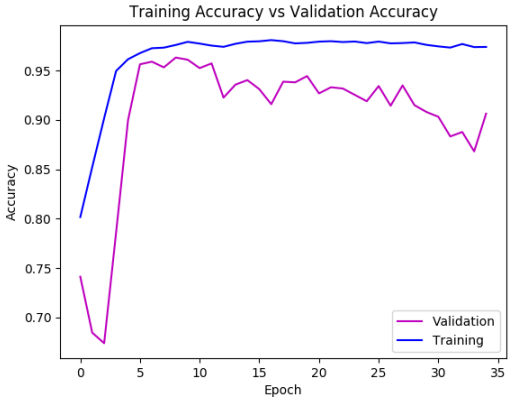
(b) 20% random under sampling with learning rate 1e-2, inverse cost, and batch size 6



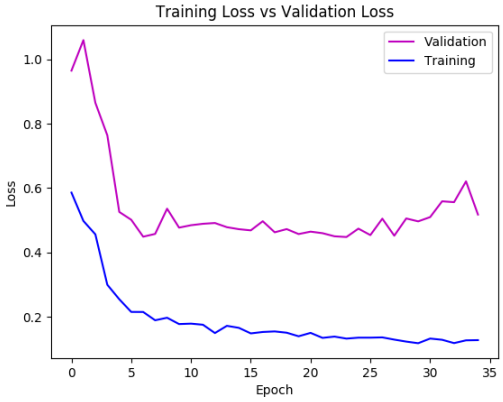
(c) 20% random under sampling with learning rate 1e-2, cost matrix A, and batch size 6



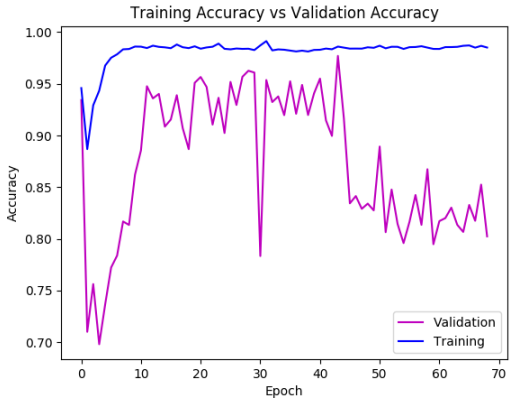
(d) 20% random under sampling with learning rate 1e-2, cost matrix A, and batch size 6



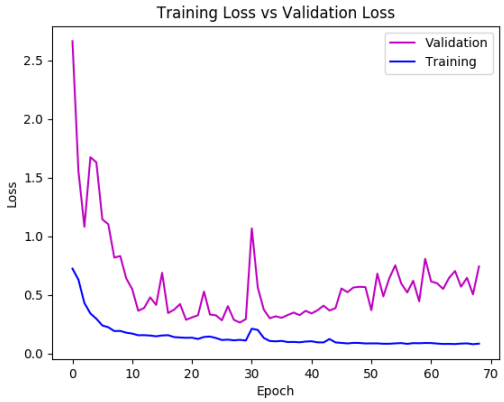
(e) 20% random under sampling with learning rate 1e-2, cost matrix B, and batch size 6



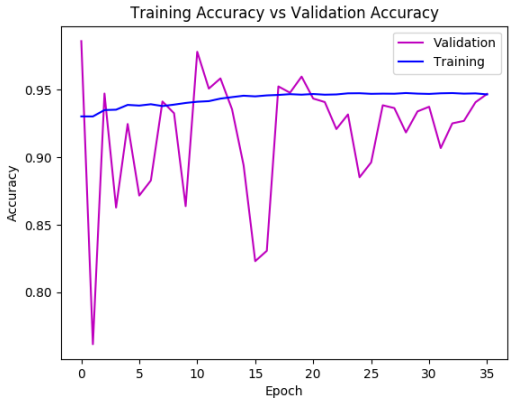
(f) 20% random under sampling with learning rate 1e-2, cost matrix B, and batch size 6



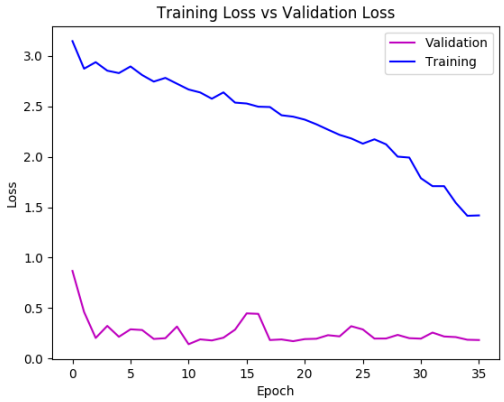
(g) 20% random under sampling with learning rate 1e-2, inverse cost, and batch size 12



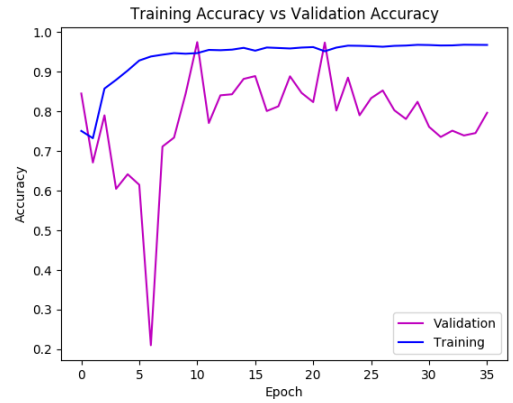
(h) 20% random under sampling with learning rate 1e-2, inverse cost, and batch size 12



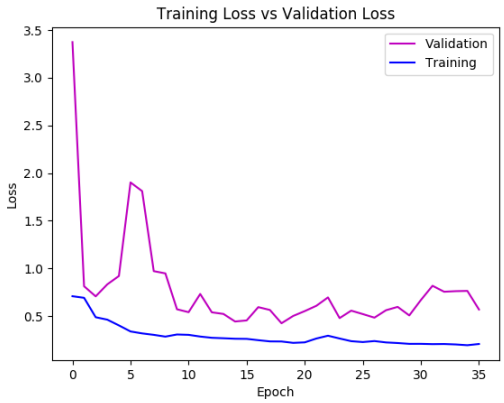
(i) 20% random under sampling with learning rate 1e-2, cost matrix A, and batch size 12



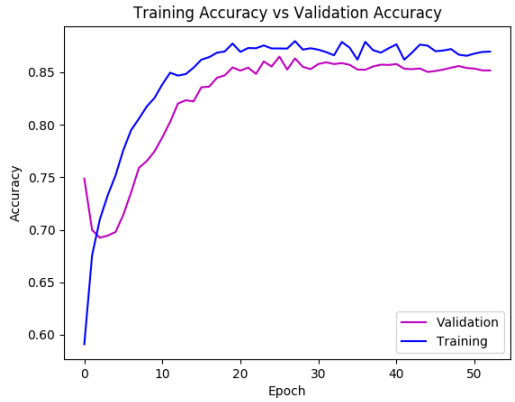
(j) 20% random under sampling with learning rate 1e-2, cost matrix A, and batch size 12



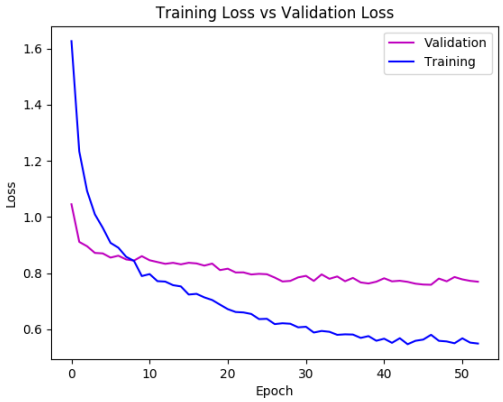
(k) 20% random under sampling with learning rate 1e-2, cost matrix B, and batch size 12



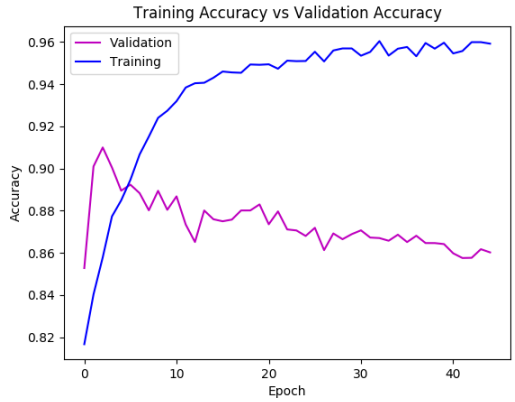
(l) 20% random under sampling with learning rate 1e-2, cost matrix B, and batch size 12



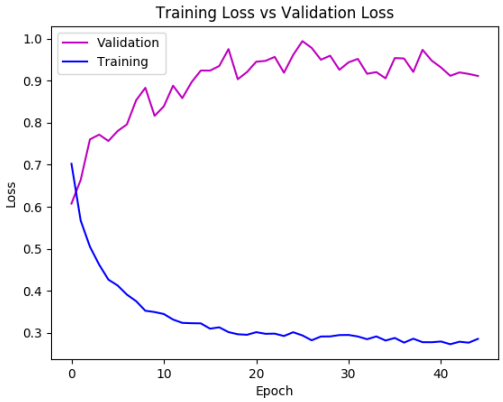
(m) 20% random under sampling with learning rate 1e-4, inverse cost, and batch size 6



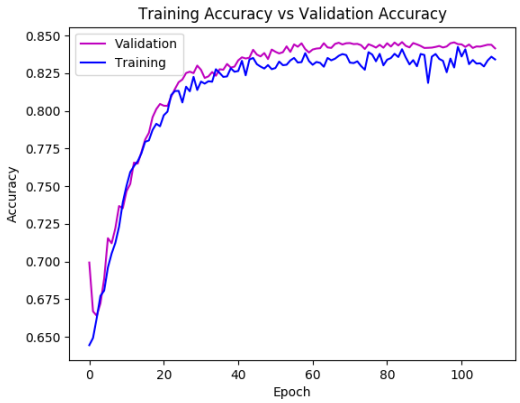
(n) 20% random under sampling with learning rate 1e-4, inverse cost, and batch size 6



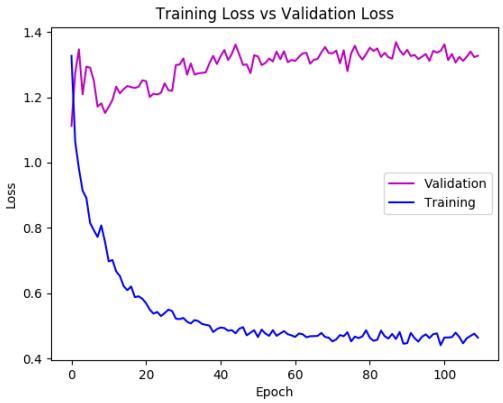
(o) 20% random under sampling with learning rate 1e-4, cost matrix A, and batch size 6



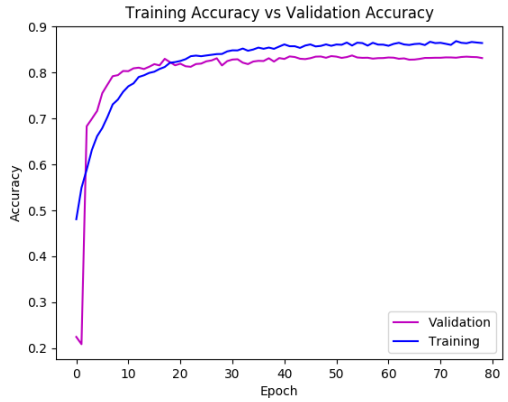
(p) 20% random under sampling with learning rate 1e-4, cost matrix A, and batch size 6



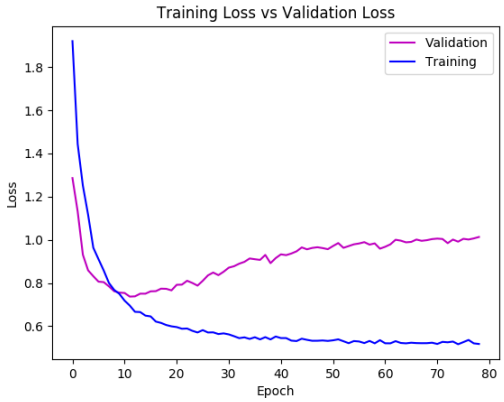
(q) 20% random under sampling with learning rate 1e-4, cost matrix B, and batch size 6



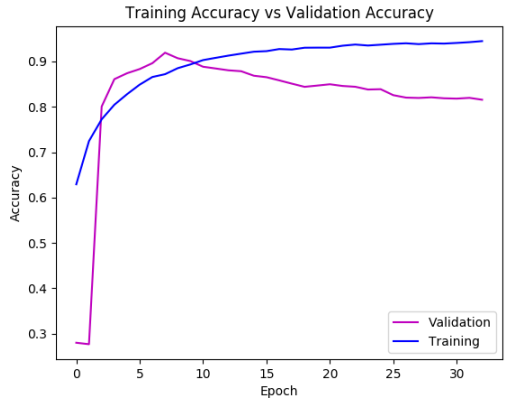
(r) 20% random under sampling with learning rate 1e-4, cost matrix B, and batch size 6



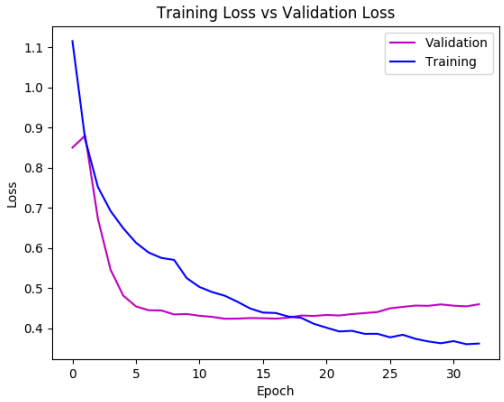
(s) 20% random under sampling with learning rate 1e-4, inverse cost, and batch size 12



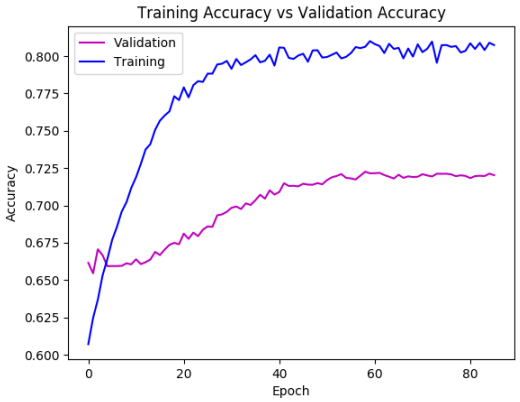
(t) 20% random under sampling with learning rate 1e-4, inverse cost, and batch size 12



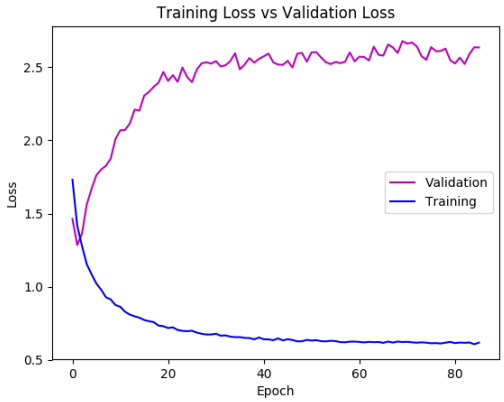
(u) 20% random under sampling with learning rate 1e-4, cost matrix A, and batch size 12



(v) 20% random under sampling with learning rate 1e-4, cost matrix A, and batch size 12

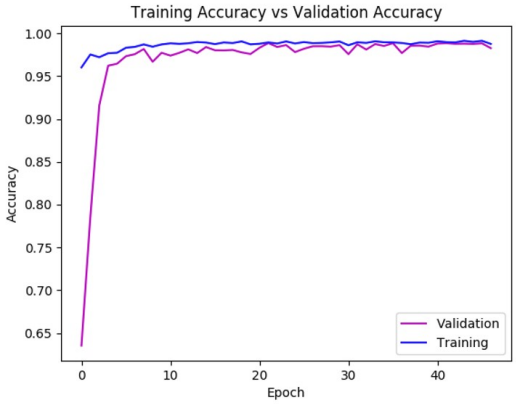


(w) 20% random under sampling with learning rate 1e-4, cost matrix B, and batch size 12

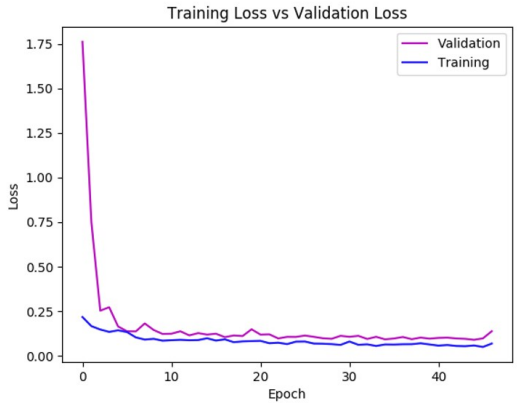


(x) 20% random under sampling with learning rate 1e-4, cost matrix B, and batch size 12

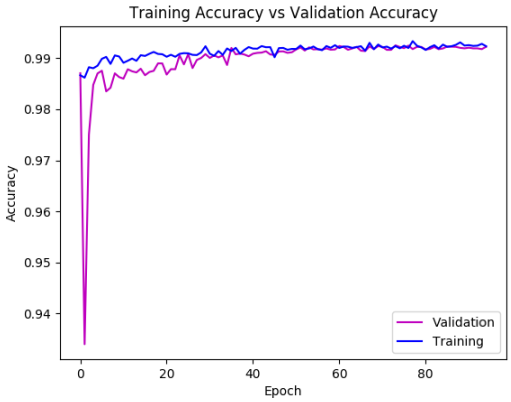
Figure A.2: Training and validation loss and accuracy plots for the hyperparameter search with 20 % random under sampling



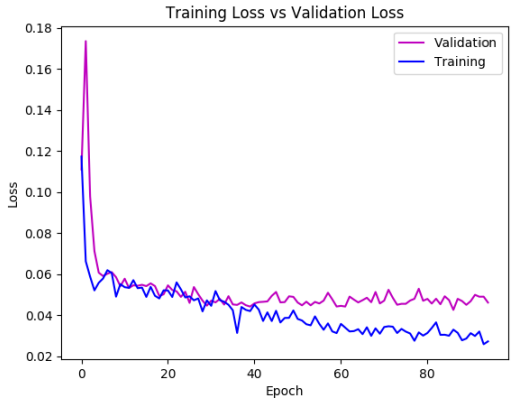
(a) Learning rate 1e-2, inverse cost, and batch size 6



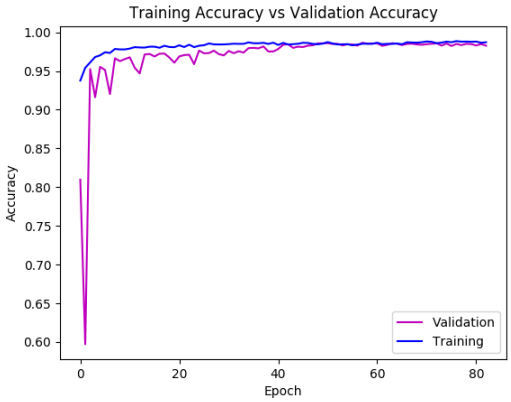
(b) Learning rate 1e-2, inverse cost, and batch size 6



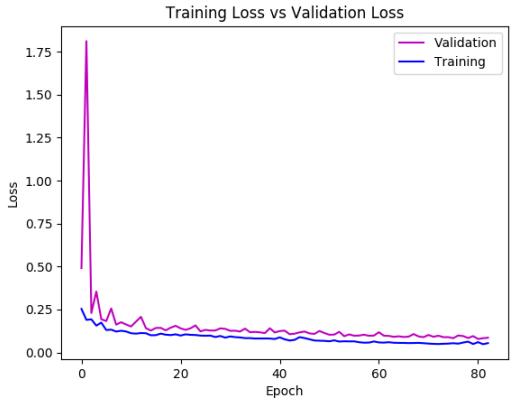
(c) Learning rate 1e-2, cost matrix A, and batch size 6



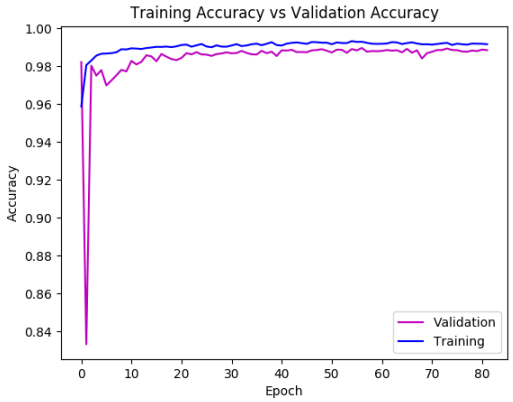
(d) Learning rate 1e-2, cost matrix A, and batch size 6



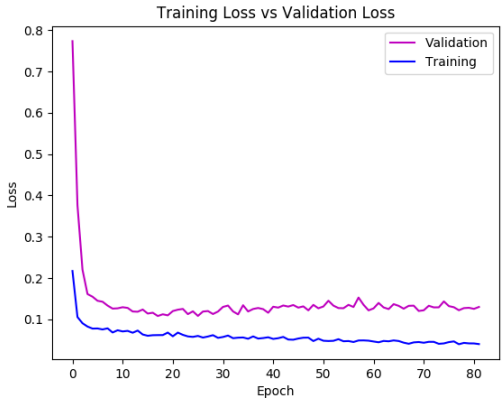
(e) Learning rate 1e-2, cost matrix B, and batch size 6



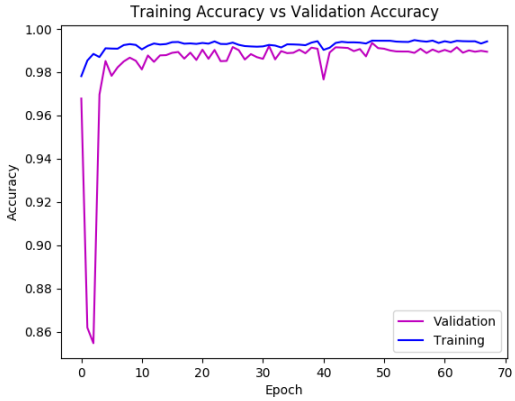
(f) Learning rate 1e-2, cost matrix B, and batch size 6



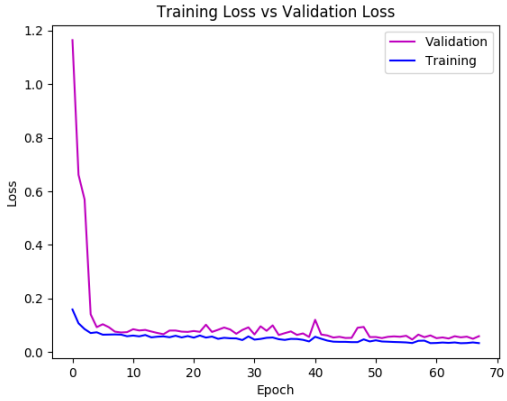
(g) Learning rate 1e-2, inverse cost, and batch size 12



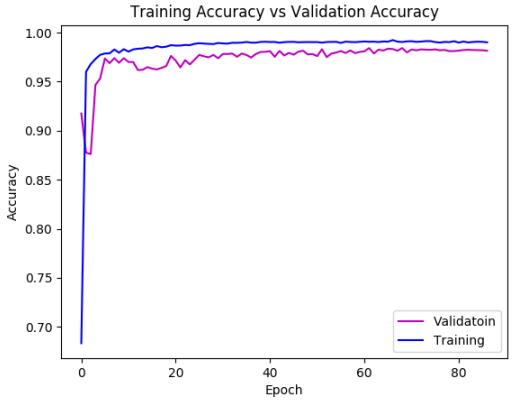
(h) Learning rate 1e-2, inverse cost, and batch size 12



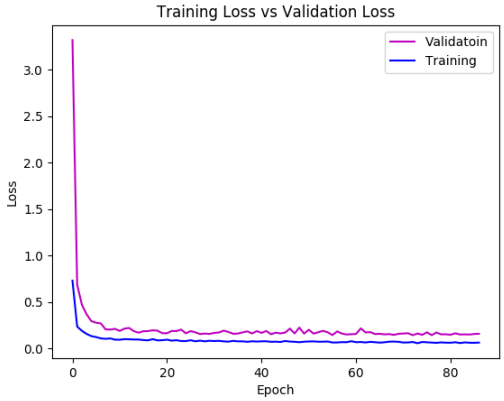
(i) Learning rate 1e-2, cost matrix A, and batch size 12



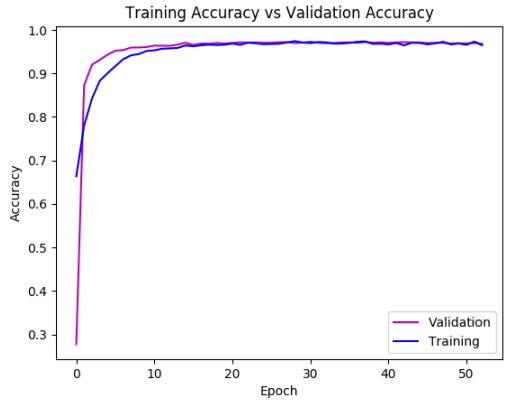
(j) Learning rate 1e-2, cost matrix A, and batch size 12



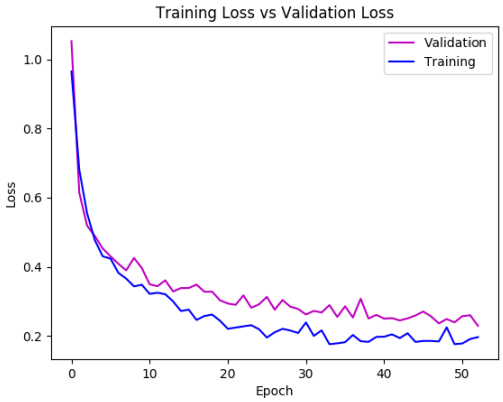
(k) Learning rate 1e-2, cost matrix B, and batch size 12



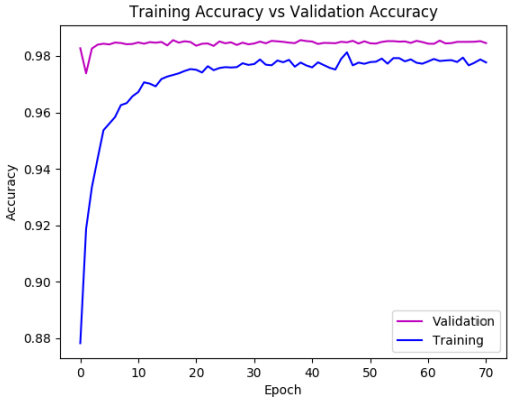
(l) Learning rate 1e-2, cost matrix B, and batch size 12



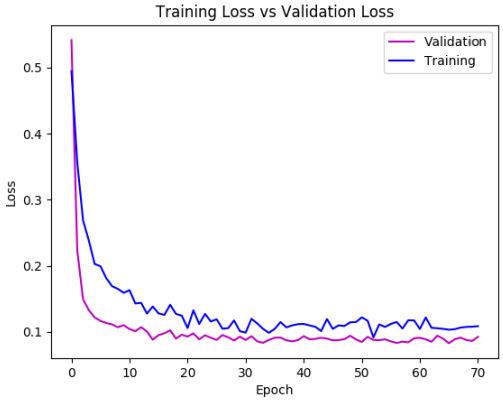
(m) Learning rate 1e-4, inverse cost, and batch size 6



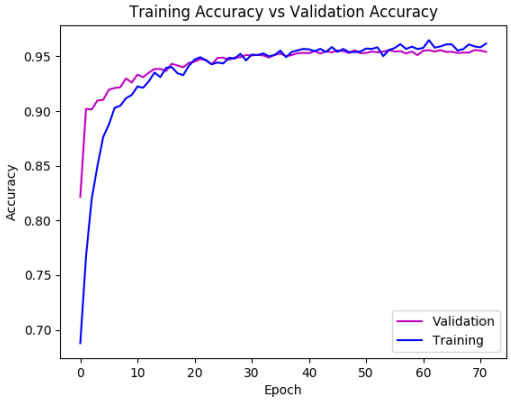
(n) Learning rate 1e-4, inverse cost, and batch size 6



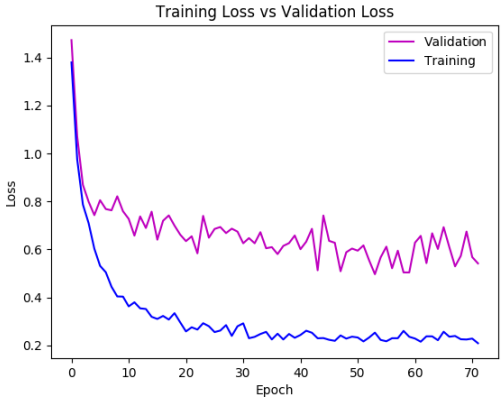
(o) Learning rate 1e-4, cost matrix A, and batch size 6



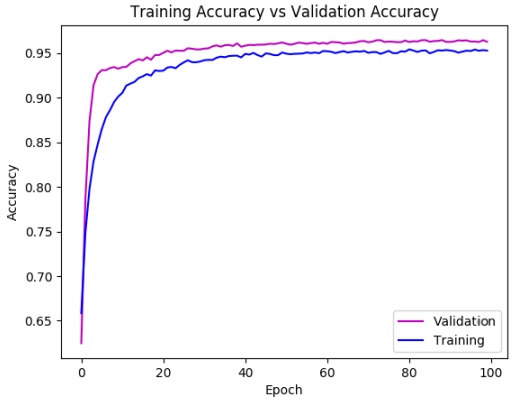
(p) Learning rate 1e-4, cost matrix A, and batch size 6



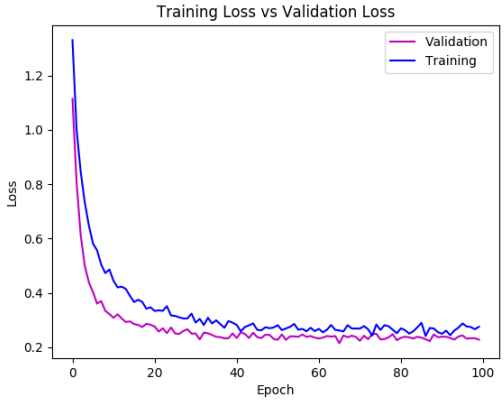
(q) Learning rate 1e-4, cost matrix B, and batch size 6



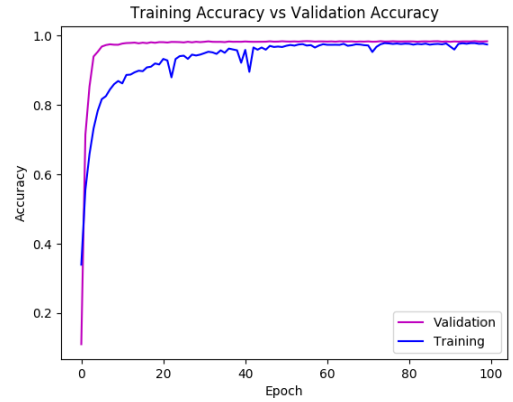
(r) Learning rate 1e-4, cost matrix B, and batch size 6



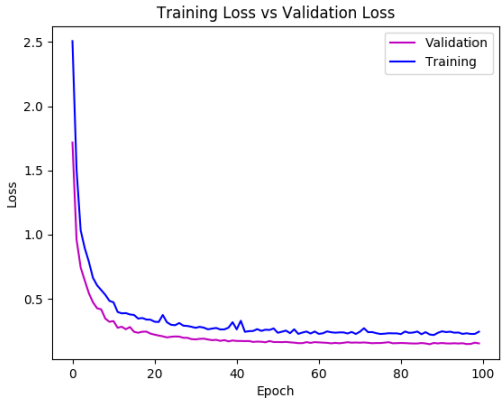
(s) Learning rate 1e-4, inverse cost, and batch size 12



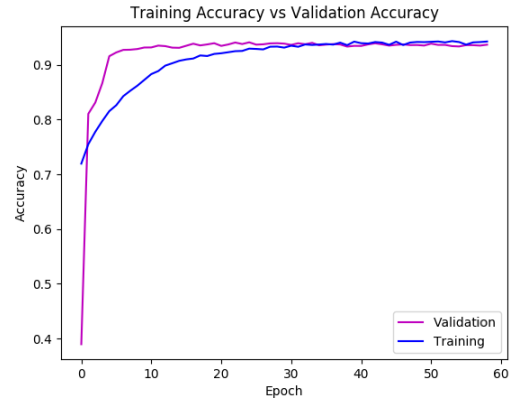
(t) Learning rate 1e-4, inverse cost, and batch size 12



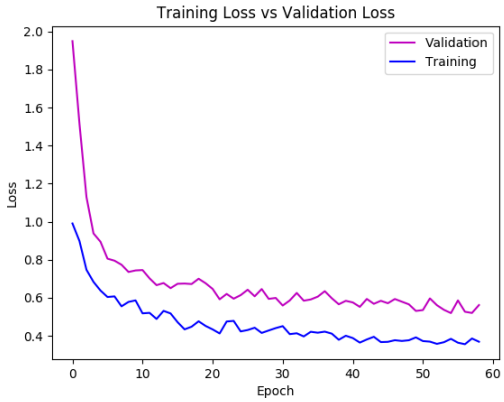
(u) Learning rate 1e-4, cost matrix A, and batch size 12



(v) Learning rate 1e-4, cost matrix A, and batch size 12

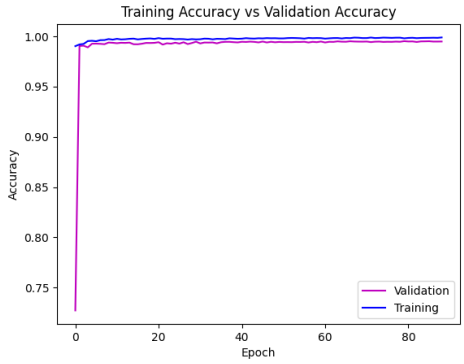


(w) Learning rate 1e-4, cost matrix B, and batch size 12

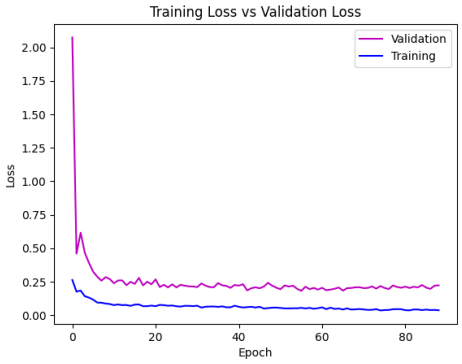


(x) Learning rate 1e-4, cost matrix B, and batch size 12

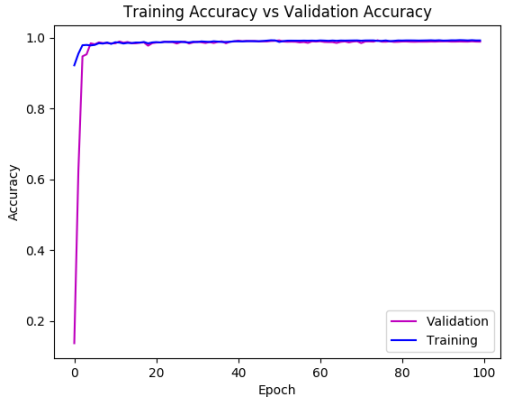
Figure A.3: Training and validation loss and accuracy plots for the hyperparameter search without random under sampling



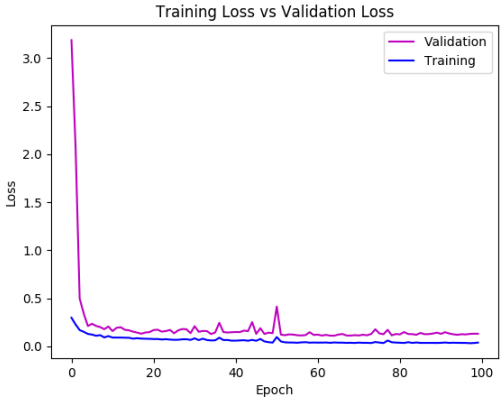
(a) Cost matrix A and batch size 6



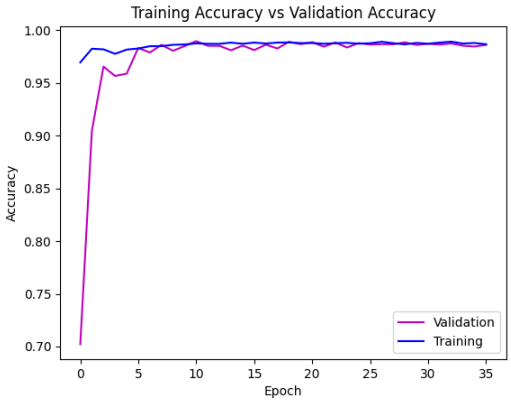
(b) Cost matrix A and batch size 6



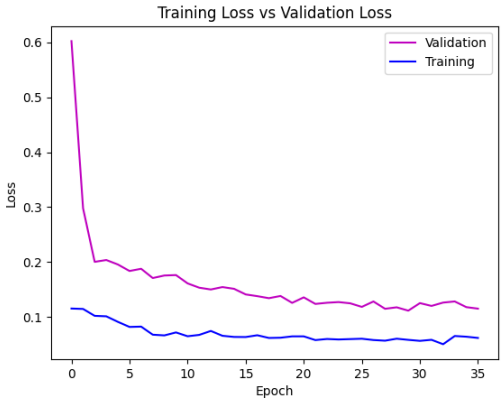
(c) Cost matrix A and batch size 12



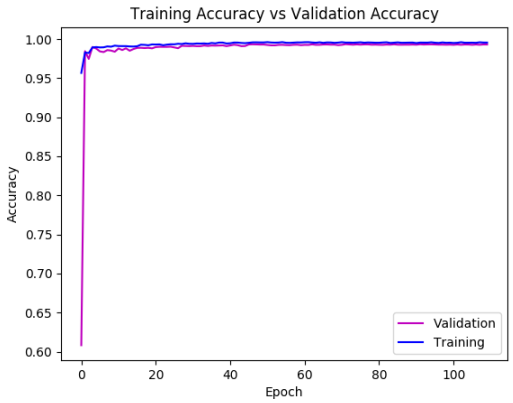
(d) Cost matrix A and batch size 12



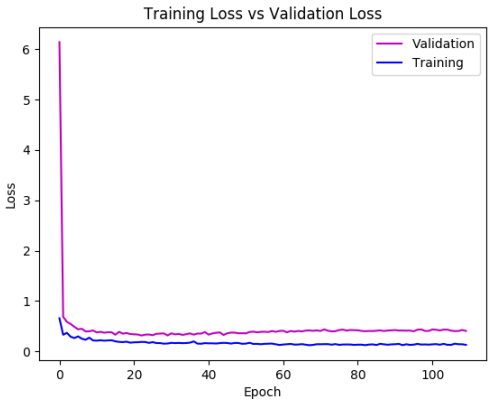
(e) Cost matrix B and batch size 6



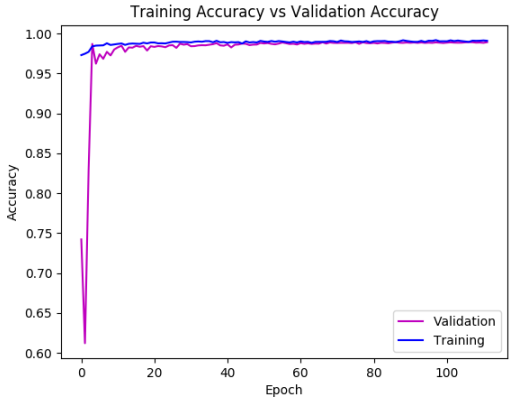
(f) Cost matrix B and batch size 6



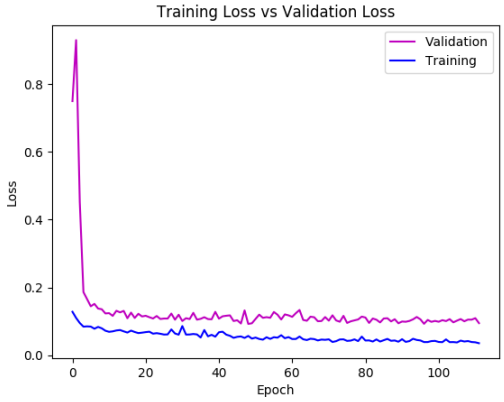
(g) Cost matrix B and batch size 12



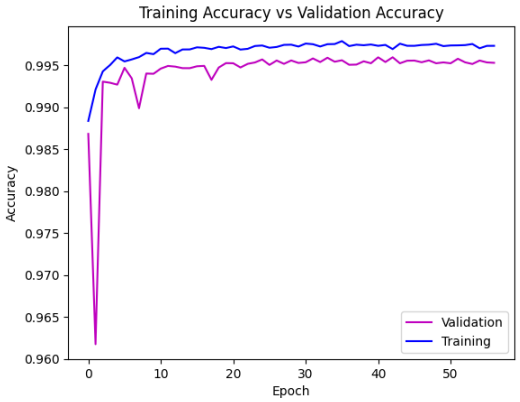
(h) Cost matrix B and batch size 12



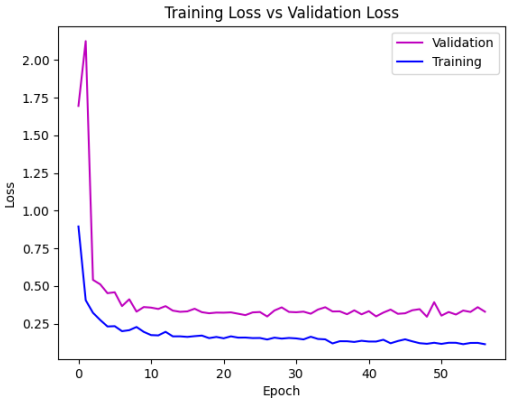
(i) Cost matrix C and batch size 6



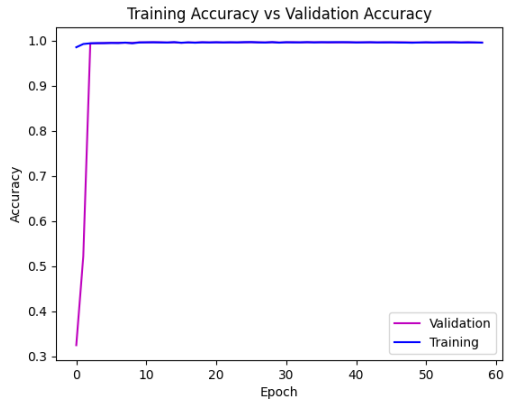
(j) Cost matrix C and batch size 6



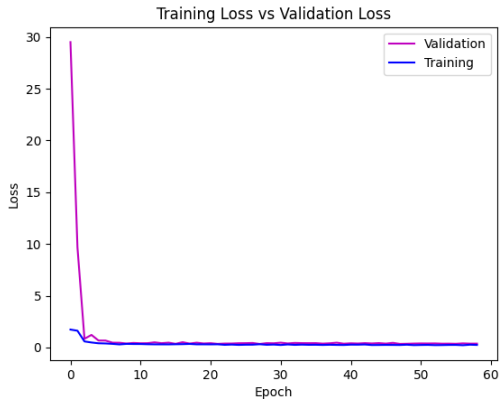
(k) Cost matrix C and batch size 12



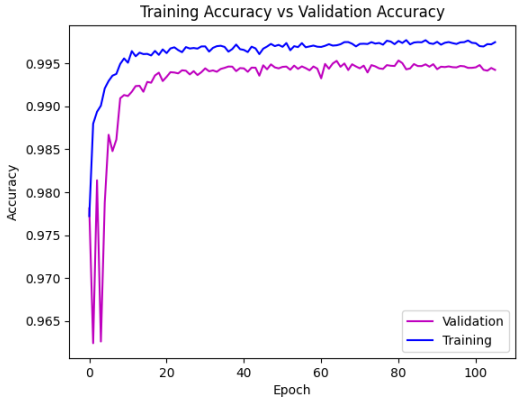
(l) Cost matrix C and batch size 12



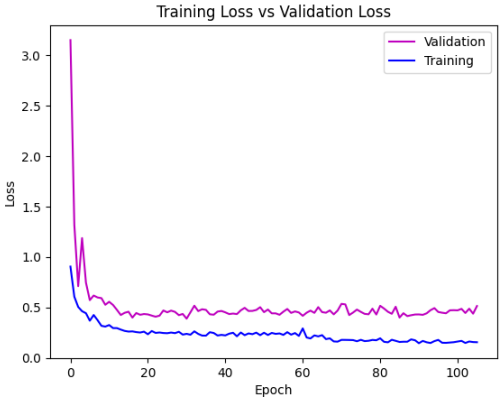
(m) Cost matrix D and batch size 6



(n) Cost matrix D and batch size 6



(o) Cost matrix D and batch size 12

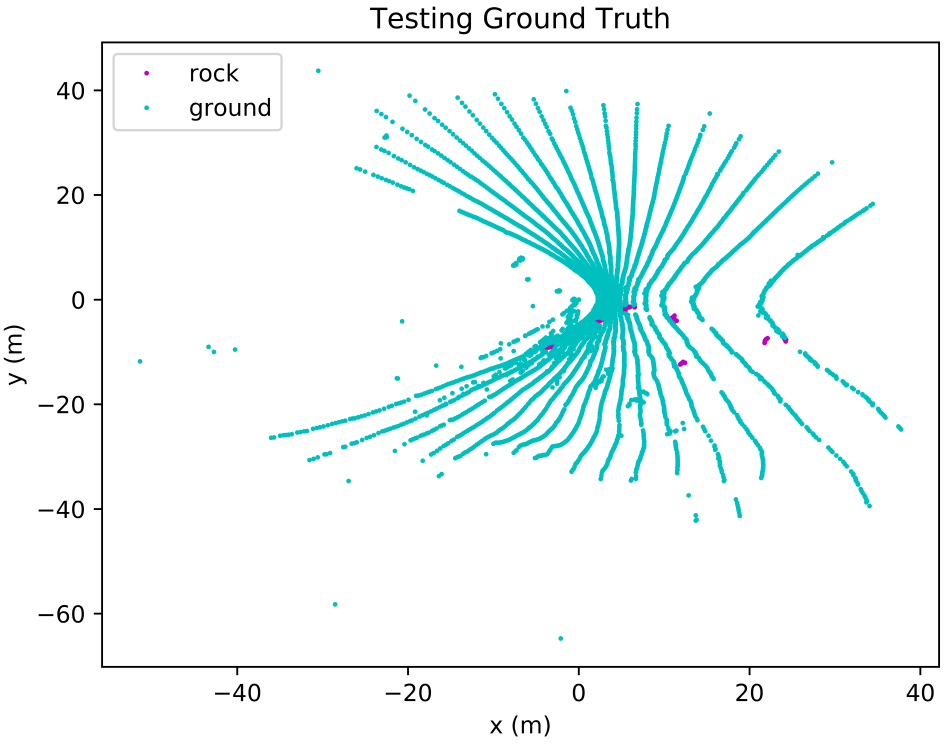


(p) Cost matrix D and batch size 12

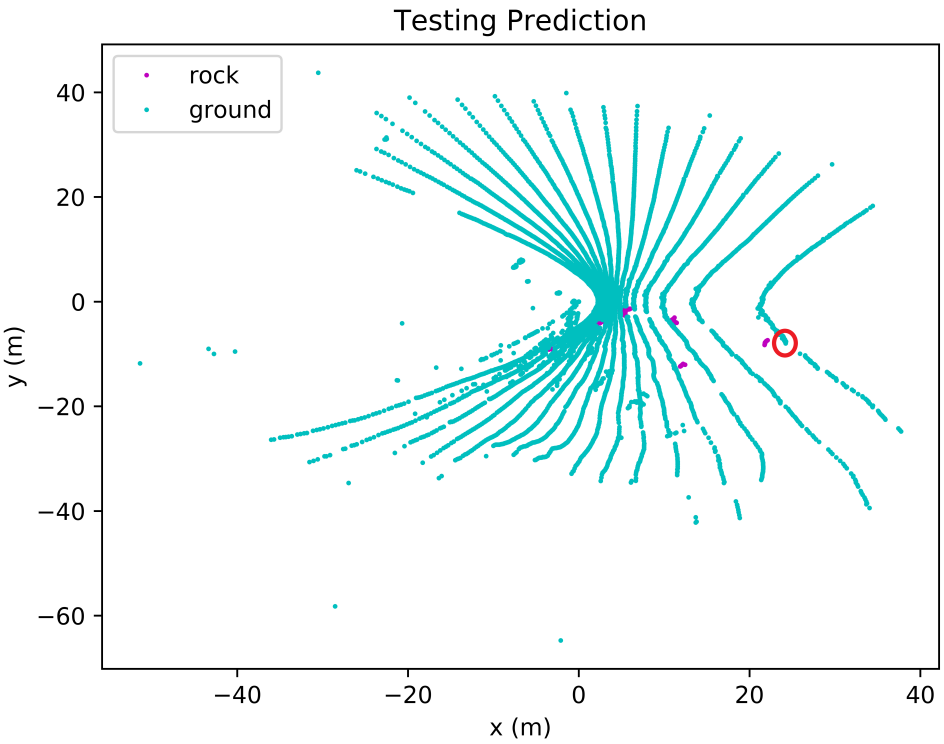
Figure A.4: Training and validation loss and accuracy plots for the iterated cost matrices

Appendix B

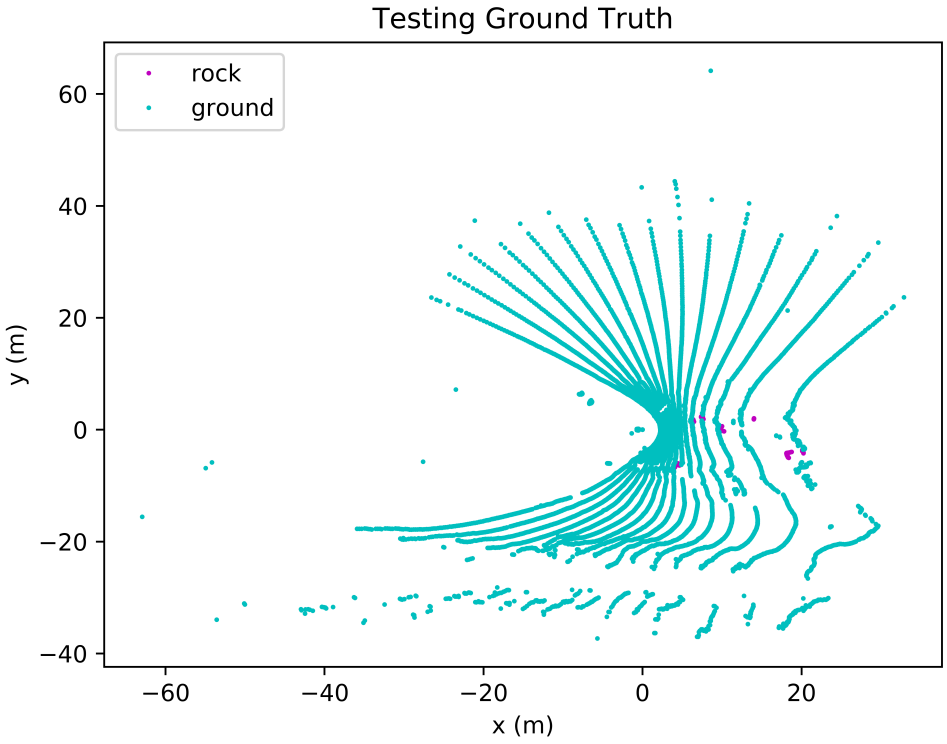
Point Cloud Figures with Ground Truth and Predicted Labels for Chapter 5



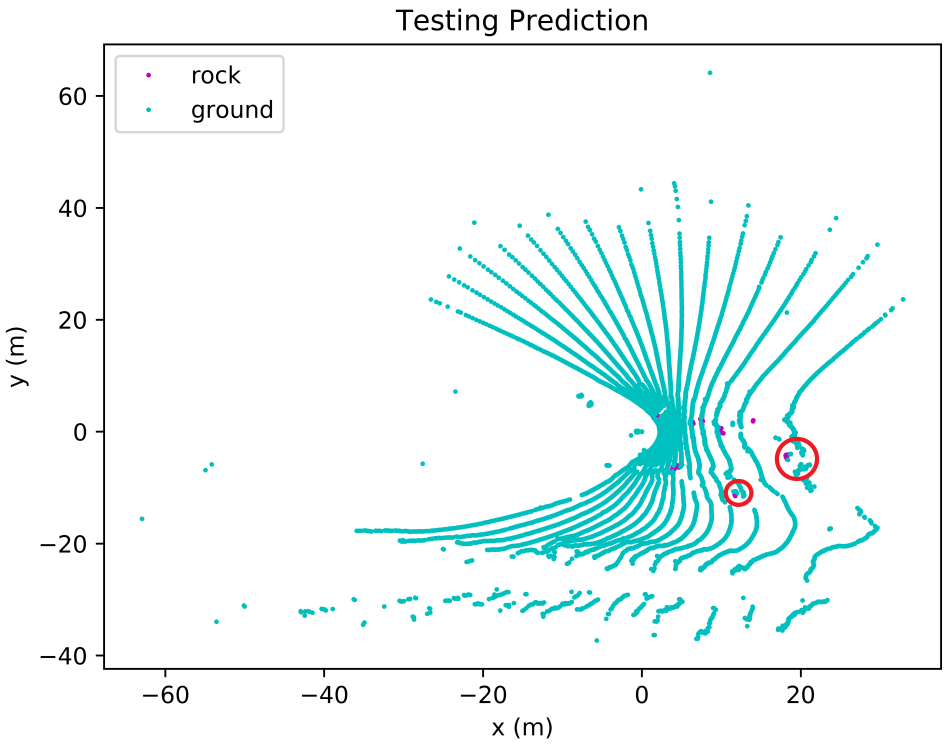
(a) Ground truth of point cloud from test area one



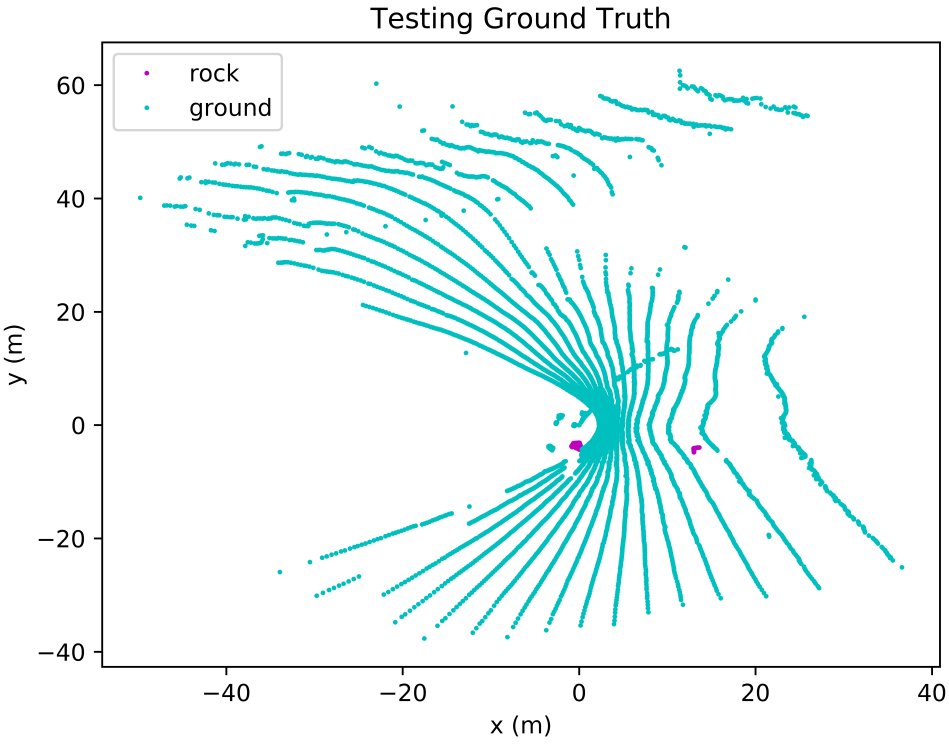
(b) Prediction of point cloud from test area one



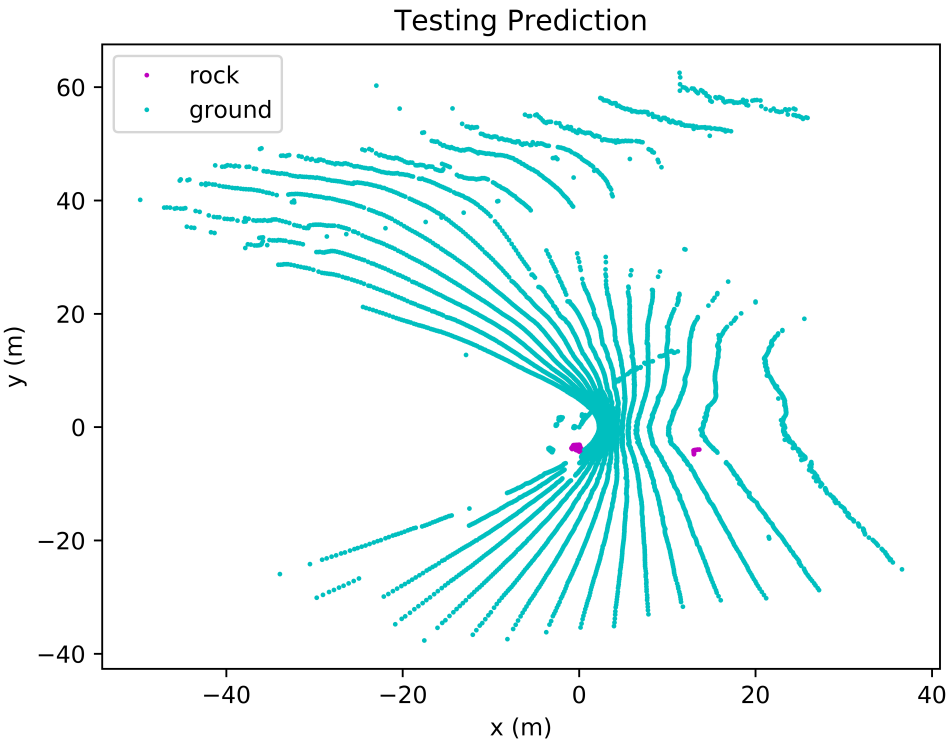
(c) Ground truth of point cloud from test area two



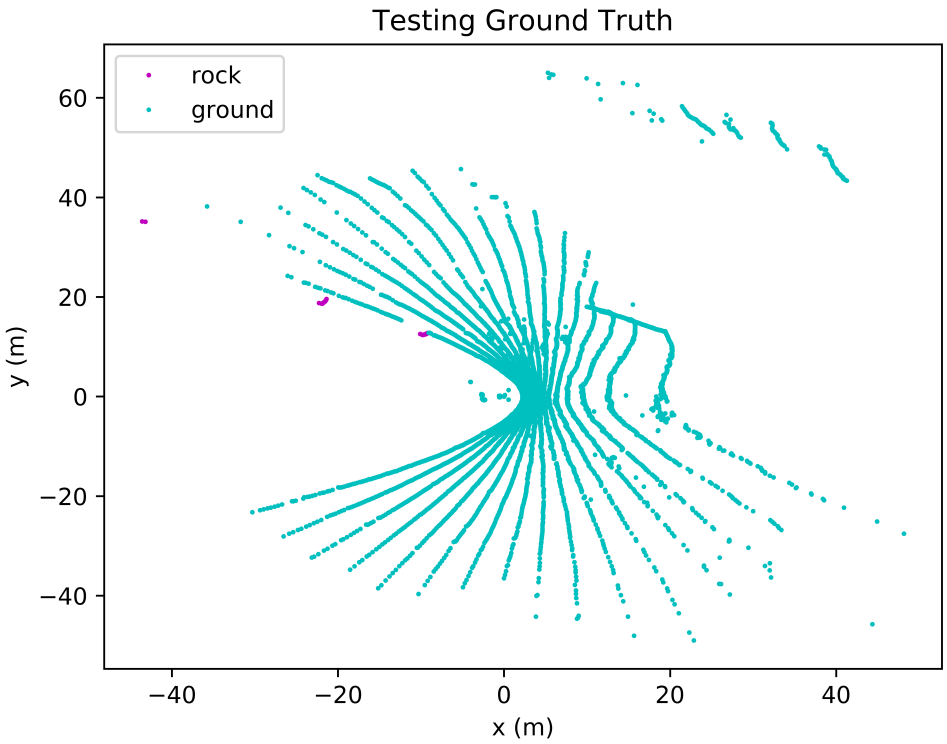
(d) Prediction of point cloud from test area two



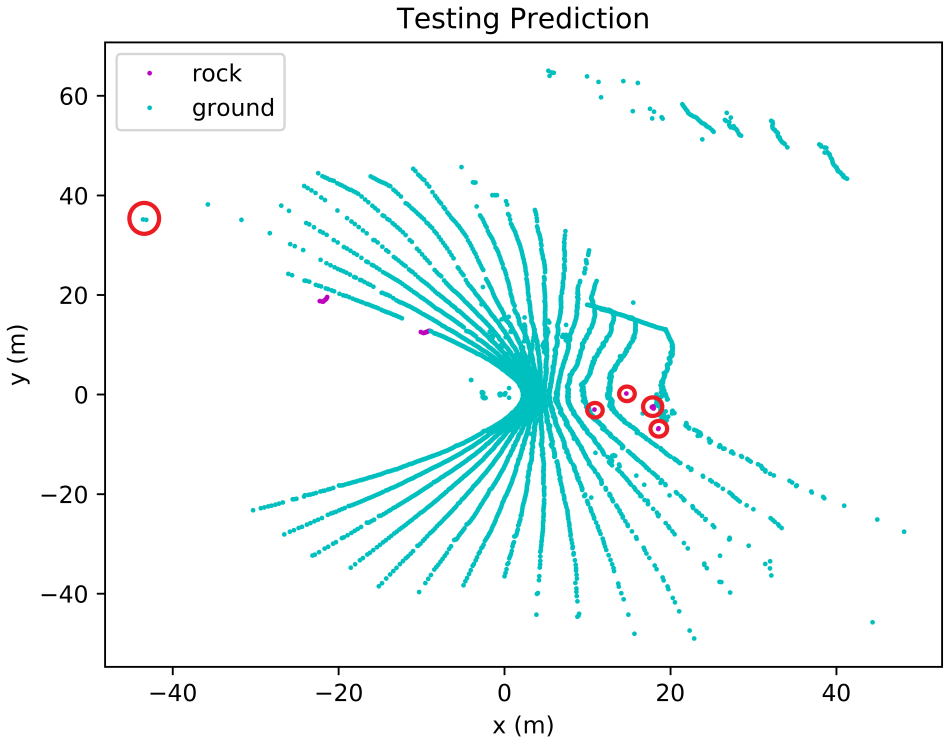
(e) Ground truth of point cloud from test area three



(f) Prediction of point cloud from test area three

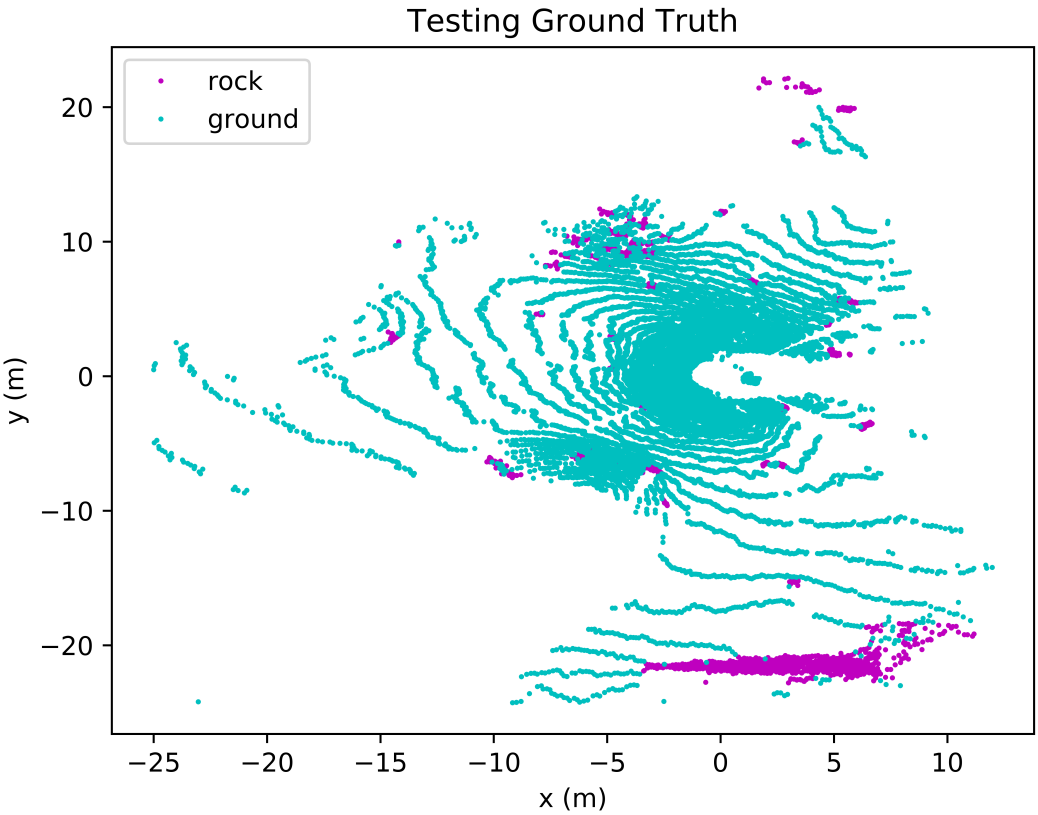


(g) Ground truth of point cloud from test area four

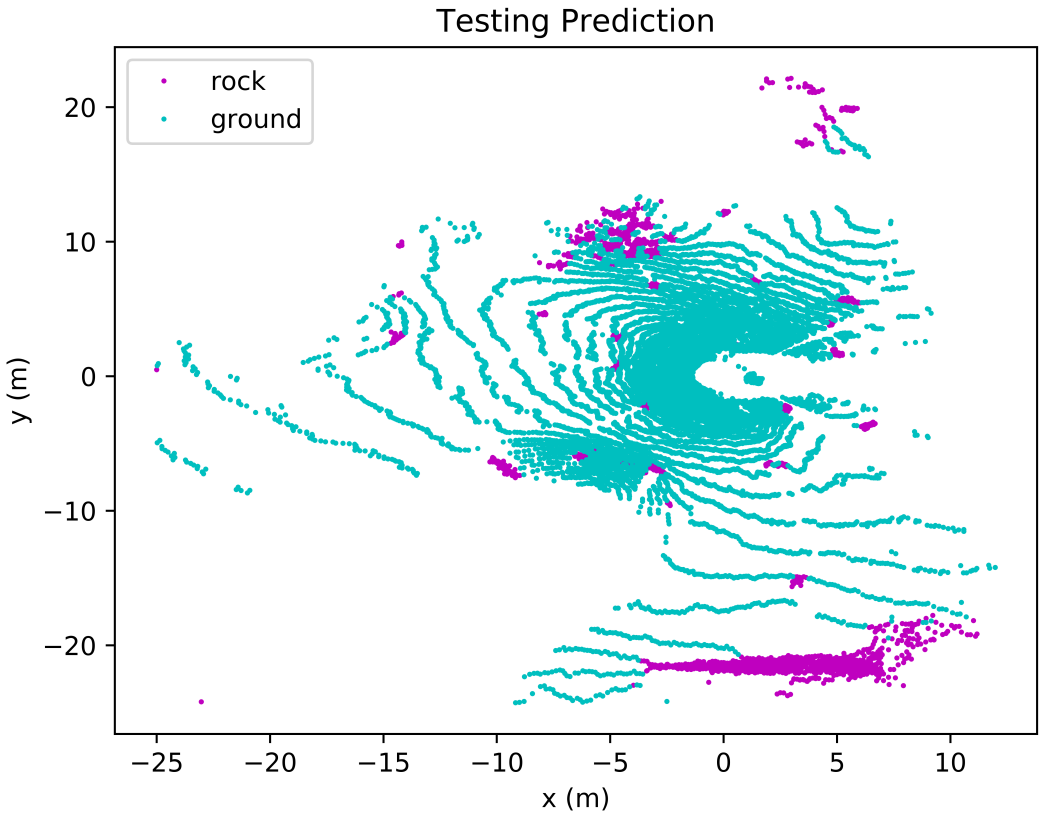


(h) Prediction of point cloud from test area four

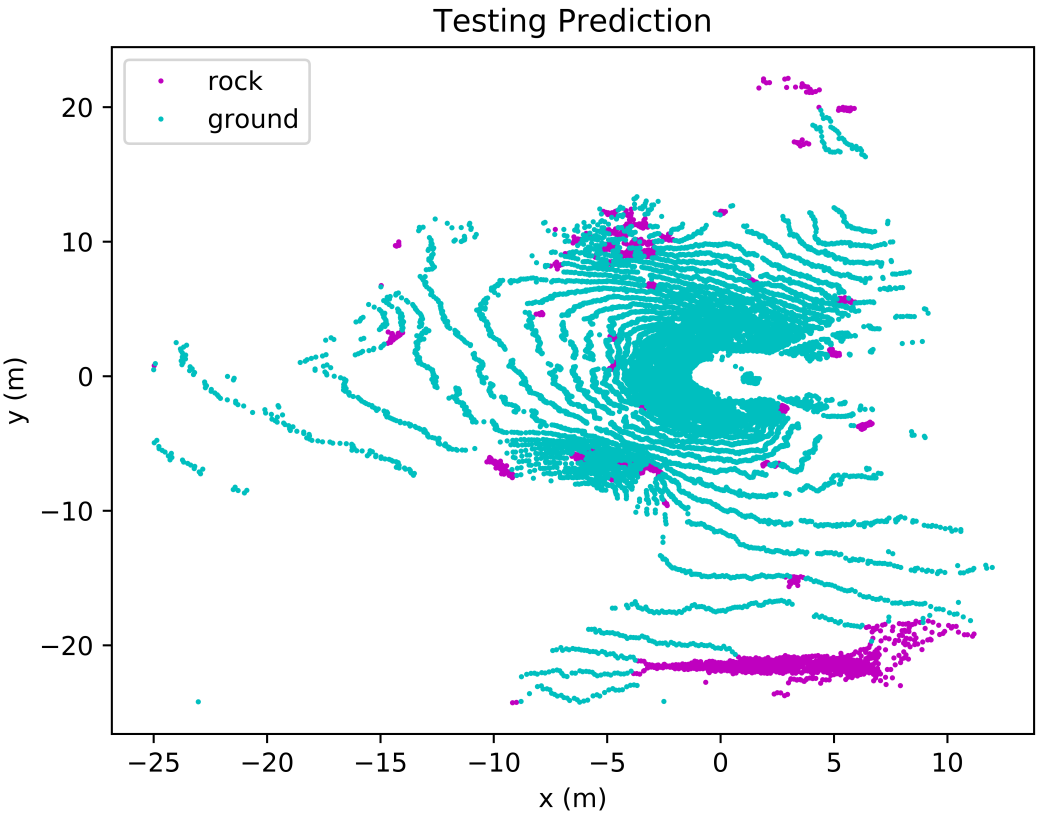
Figure B.1: Comparison of ground truth and predicted labels for four point clouds from the test set of the Katwijk Beach Planetary Rover Dataset where misclassifications are circled in red



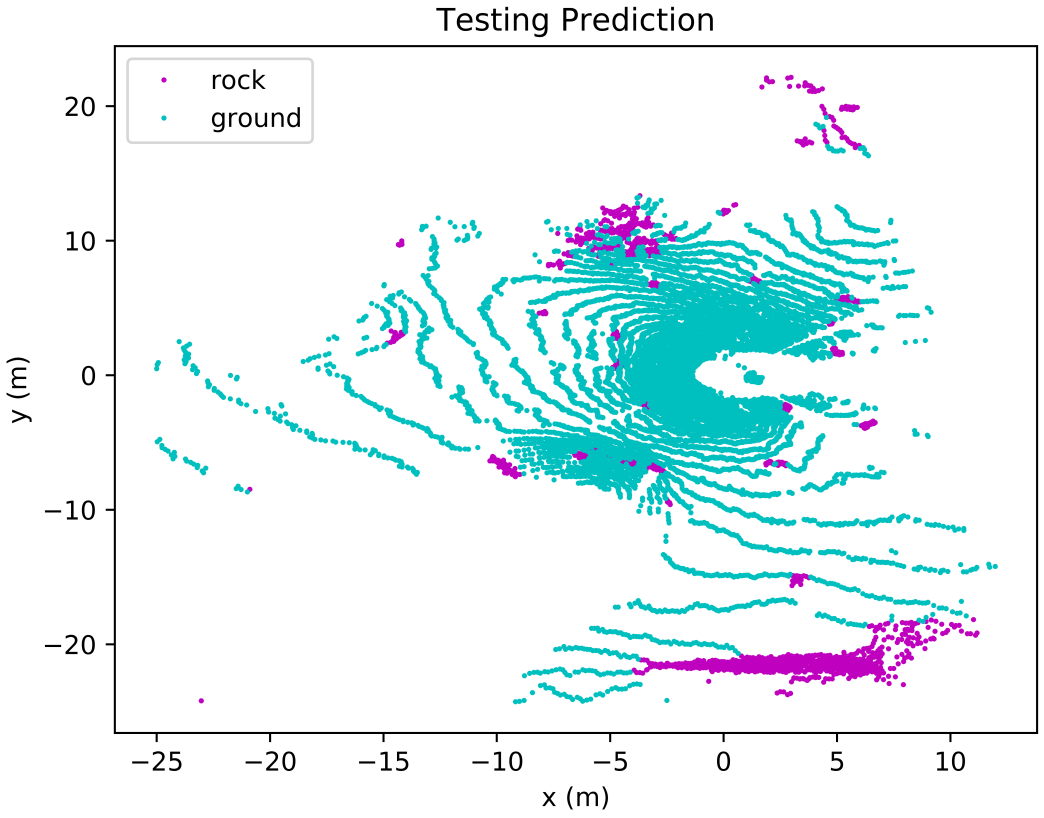
(a) Ground truth labels for a point cloud from the test set of the ATF dataset



(b) Model 1's predicted labels for a point cloud from the test set of the ATF dataset



(c) Model 2's predicted labels for a point cloud from the test set of the ATF dataset



(d) Model 3's predicted labels for a point cloud from the test set of the ATF dataset

Figure B.2: Comparison of ground truth labels to the predicted labels from three tested models on a point cloud from the test set of the ATF dataset

Curriculum Vitae

Name:

Post-Secondary Education and Degrees:	Western University London, ON 2015 - 2019 B.E.Sc. Western University London, ON 2019 - 2021 M.E.Sc.
Related Work Experience:	Planetary Robotics Intern European Space Agency Noordwijk, Netherlands 2021 Teaching Assistant Western University 2019 - 2021 Research Assistant Western University 2019 - 2021
Presentations:	L. Flanagan, K. McIsaac, and M. Cross, ”Autonomous Rock Segmentation from Lidar Point Clouds for Planetary Rover Navigation” in *Western University Engineering Symposium*, 2021