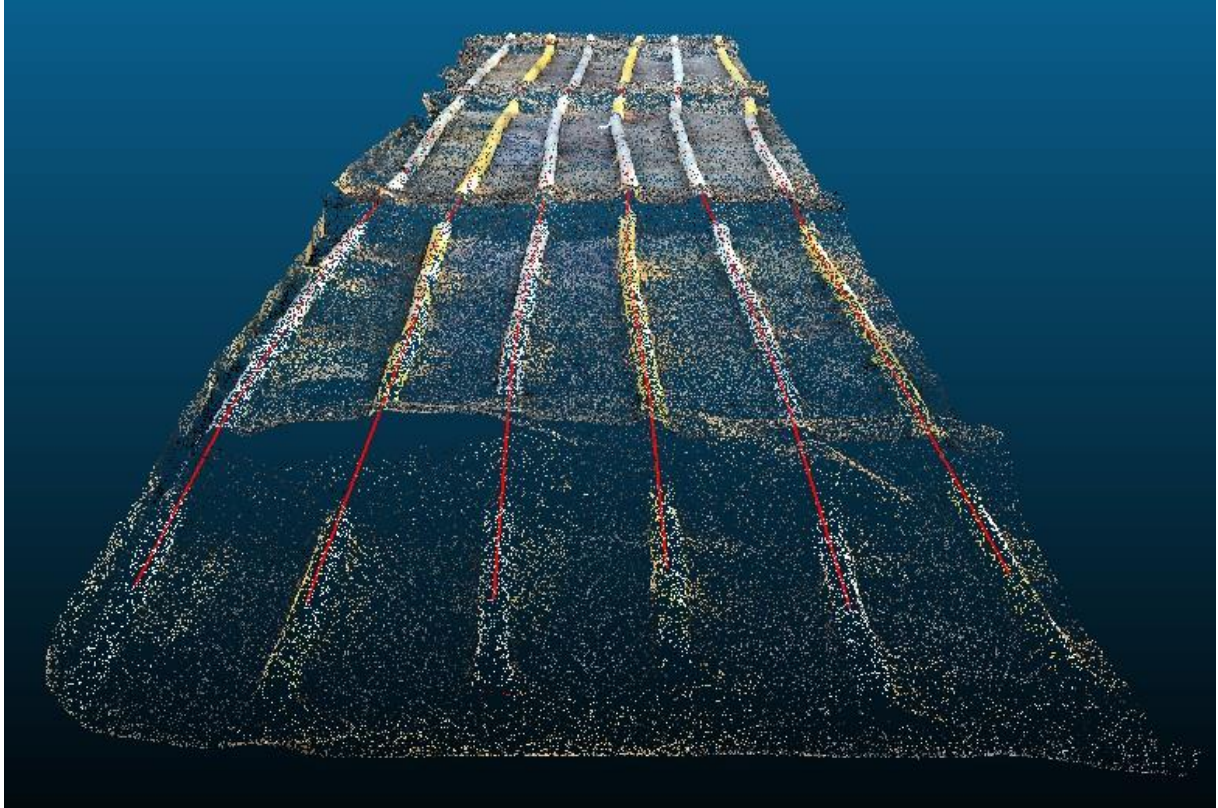


6 MAART 2024



AUTOMATIC RECOGNITION OF UNDERGROUND UTILITIES IN POINT CLOUDS

BACHELOR THESIS CIVIL ENGINEERING

KRUIPER, J.A.

UNIVERSITY
OF TWENTE.

 **siers**

Automatic Recognition of Underground Utilities in Point Clouds

Jorn Kruiper

S2590832

Bachelor Thesis Final Report

Civil Engineering BSc

Faculty Engineering & Technology

University of Twente

Internal Supervisor: DR. IR. L.L. olde Scholtenhuis

Company: Siers Infraconsult

External Supervisor: Mechiel van Manen

External Supervisor: Nima Zarrinpanjeh

6-3-2024

PREFACE

In front of you is the final report of my bachelor thesis 'Automatic Recognition of Underground Utilities in Point Clouds'. This thesis report is the final product of my graduation research to obtain my bachelor's degree in Civil Engineering at the University of Twente. This research was carried out from December 2023 to March 2024.

During my study, I followed the minor 'Smart ways to make smart cities smarter'. Within this minor followed a project on energy transition. This project was about new ways to map the underground utility network. This sparked my interest in registering underground utilities. During this minor, I also had the chance to visit a construction site of Siers.

This thesis helped me to acquire new skills that previously were unfamiliar to me. I learned what machine learning is and how to train machine learning algorithms. Furthermore, I learned how to process point clouds into datasets that can be used for training an algorithm.

I want to thank my supervisor from the University of Twente, Léon olde Scholtenhuis, for providing me with detailed feedback and providing me with new ideas to improve my research. I also want to thank my supervisors at Siers, Nima Zarrinpanjeh and Mechiel van Manen for introducing me to the company and providing the information needed for my research. Together with my supervisors, I had interesting discussions that helped to improve my research.

I hope you enjoy reading the report.

Jorn Kruiper
Oldenzaal, 6-3-2024

EXECUTIVE SUMMARY

Underground utilities are crucial for economic development. Stakeholders involved in the installation of new utilities include network users, utility companies, landowners, and contractors. Contractors like Siers Groep provide services in underground infrastructure, including accurate mapping and registration of cables and pipelines.

The process of mapping underground utilities can be more efficient. One of the novel methods to map utilities is to make a 3D point cloud scan of a trench. Processing the point clouds is still laborious because it is a manual task. Workers at surveying companies do not necessarily have the experience with such a task but also would have a significant amount of additional work in case they need to process 3D point clouds. Automating the process of retrieving the location data of utilities in point clouds enhances efficiency significantly. Automation can be done using machine learning.

The objective of this research is to compare and use machine learning algorithms for automatically recognizing and retrieving underground utilities that are present in 3d point clouds of open trenches and convert the data into geometric shapes. From the objective, one main research question is formulated:

- How can different machine learning algorithms be used to retrieve utilities from point clouds of open trenches?

The main research question is answered by four sub-questions. These are:

1. What types of machine learning algorithms are most useful?
2. How can the machine learning algorithms be used?
3. How should the point cloud data be pre-processed?
4. How can the quality of a model be assessed?

To answer the four sub-questions, first different machine learning algorithms are identified and assessed on a set of criteria. After that, the point cloud data used for this research is pre-processed. In this step, a train and test dataset is created including point clouds representing 'utility' and 'not utility'. From the assessment of the identified machine learning algorithms, three algorithms are chosen. The chosen algorithms are trained on the point cloud training data. The result is a model. To choose the best model for the problem, the quality of the models is assessed. The best model is used for a demonstration case, in which a new point cloud is classified.

Different supervised machine learning algorithms are identified as potential candidates. Three algorithms are chosen to be trained. These are VoxNet, a shallow neural network and PointNet. These algorithms score best on the set of criteria. Random forests, support vector machines, PointNet ++ and the combination of UnitNet, FeatureNet and FinalNet are not trained. The algorithms can be used via different programs. For this thesis, MATLAB is used because there is clear documentation available about using the algorithms in MATLAB.

All three models are trained based on a set of training options. The three models are trained on 10 different combinations of training options. For each model, one option with the best training options is chosen. The three models are then compared and the best model is chosen. This is VoxNet. This model is used for a demonstration case. In the demonstration case, a new point cloud is classified. The utilities classified in this point cloud are converted to polylines that can be used for mapping.

In conclusion, the study contributes to the existing literature on machine learning algorithms' application in underground utility recognition from point cloud data. By comparing and evaluating various algorithms, it provides insights into their suitability for utility recognition tasks. Overall, the study highlights the effectiveness of machine learning algorithms in utility recognition from point cloud data.

TABLE OF CONTENTS

1.	Introduction	1
1.1	Current State of Utility Mapping	2
1.2	New Technologies for Utility Mapping.....	2
1.3	Problem Statement and Research Objective	4
1.4	Research Questions	4
1.5	Report Structure.....	4
2.	Theoretical Points of Departure: Point Cloud Classification Models	5
2.1	Random Forest	5
2.2	Support Vector Machine	6
2.3	Neural Network.....	6
2.3.1	Shallow Neural Network	7
2.3.2	VoxNet	8
2.3.3	PointNet.....	8
2.3.4	PointNet ++	9
2.3.5	UnitNet, FeatureNet and FinalNet	10
2.4	Overview of Machine Learning Algorithms	11
2.5	Training Options	13
3.	Research Methodology	14
3.1	Choose Algorithms to Train	14
3.2	Process Point Cloud Data	14
3.2.1	Preprocess Point Clouds	15
3.2.2	Label Data	16
3.2.3	Compute Normals	17
3.2.4	Merge Data.....	17
3.2.5	Cut Point Clouds Into Smaller Parts	18
3.2.6	Store Point Cloud Parts.....	19
3.3	Train Algorithms	19
3.4	Quality Assessment	20
3.4.1	Demonstration Case	22
4.	Results	24
4.1	Algorithms Trained	24
4.2	Training Results	24
4.2.1	VoxNet	24
4.2.2	Fully Connected Shallow Neural Network	25

4.2.3	PointNet	26
4.2.4	Best Model	27
4.3	Point Clouds Labelled as False Positive or False Negative	27
4.4	Demonstration Case	28
5.	Discussion	32
6.	Conclusion	34
7.	Recommendations for Further Research	36
8.	Bibliography	37
9.	Appendices	39
9.1	Appendix 1, Script for Merging Data	39
9.2	Appendix 2, Description Dataset.....	40
9.3	Appendix 3, Script to Create Train and Test Set	54
9.4	Appendix 4, Script to Cut Point Cloud.....	56
9.5	Appendix 5, Script to Use Trained Model for Demonstration Case.....	58
9.6	Appendix 6, Script to Create Polylines	60
9.7	Appendix 7, VoxNet Script	61
9.8	Appendix 8, Shallow Neural Network Script.....	65
9.9	Appendix 9, PointNet Script	69

TABLE OF FIGURES

Figure 1, an overview of categories in machine learning	3
Figure 2, an example of a Random Forest	5
Figure 3, illustration of the separating hyperplane, margin and support vectors	6
Figure 4, fully connected neural network architecture	7
Figure 5, VoxNet Architecture based on (Maturana & Scherer, 2015)	8
Figure 6, PointNet Architecture based on (Qi, Su, et al., 2017).....	9
Figure 7, PointNet++ Architecture based on (Qi, Yi, et al., 2017)	9
Figure 8, the architecture of UnitNet, FeatureNet and FinalNet based on (Xu et al., 2022)	10
Figure 9, flowchart research methodology	14
Figure 10, a flowchart of processing point cloud data	15
Figure 11, an original point cloud of a utility trench	16
Figure 12, cropped, denoised and downsampled point cloud	16
Figure 13, an unlabelled point cloud of a utility trench	16
Figure 14, a labelled point cloud of a utility trench	16
Figure 15, a point cloud with only utilities	17
Figure 16, point cloud before rotation around the Z-axis	18
Figure 17, point cloud after rotation around the Z-axis	18
Figure 18, a point cloud with utilities that are cut into parts	18
Figure 19, shapes cut from a point cloud that look like utilities	19
Figure 20, an example of training progress VoxNet and shallow neural network	20
Figure 21, an example of a confusion matrix	21
Figure 22, point cloud for demonstration case	22
Figure 23, pre-processed point cloud.....	23
Figure 24, point clouds falsely classified as utility (FP).....	28
Figure 25, point clouds falsely classified as not utility (FN)	28
Figure 26, point cloud after applying the CSF algorithm	29
Figure 27, point cloud after cutting out remaining ground points	29
Figure 28, point cloud cut in parts using the 'label connected components' tool	29
Figure 29, point cloud parts after manual cutting	29
Figure 30, point cloud cut into parts using a MATLAB code	30
Figure 31, point cloud parts cut via CloudCompare classified as 'utility'	30
Figure 32, point cloud parts cut via CloudCompare classified as 'not utility'	30
Figure 33, point cloud parts cut using a MATLAB code classified as 'utility'	30
Figure 34, point cloud parts cut using a MATLAB code classified as 'not utility'	31
Figure 35, point cloud parts with polylines from above	31
Figure 36, point cloud parts with polylines from the side	31

TABLE OF TABLES

Table 1, comparative table for different machine learning algorithms	12
Table 2, training options VoxNet	24
Table 3, results VoxNet	25
Table 4, training options fully connected network	25
Table 5, results fully connected shallow neural network	26
Table 6, training options PointNet	26
Table 7, results PointNet	27
Table 8, comparison of best options for the models	27

GLOSSARY

TERM	DEFINITION
ALGORITHM	A step-by-step procedure for solving a problem or accomplishing a task
CAD	Computer-Aided Design, software used by engineers to create, analyse, and modify designs
CLOUDCOMPARE	An open-source software for 3D point cloud processing and visualization
DEEP NEURAL NETWORK	A neural network with more than 3 layers between the in and output layer
FEATURES	Properties or characteristics of data that are used as input for machine learning algorithms.
FULLY CONNECTED LAYER	A layer in a neural network where each neuron is connected to every neuron in the preceding layer
GIS	Geographic Information System, a system designed to capture, store, manipulate, analyse, manage, and present spatial or geographic data
GITHUB	An online platform for hosting and collaborating on software development projects
GPS	Global Positioning System, a satellite-based navigation system that provides location and time information anywhere on Earth
LIDAR	Light Detection and Ranging, a remote sensing technology that uses laser light to measure distances
MACHINE LEARNING	A subset of artificial intelligence that allows algorithms to learn from training data and improve performance without being explicitly programmed
MATLAB	A programming platform and language for mathematical applications
MODEL	A simplified representation of a system or phenomenon used to make predictions or understand behaviour
NEURAL NETWORK	a type of machine learning algorithm that is inspired by the brain's neural networks
NEURON	A basic unit of a neural network that processes an input to an output using certain weights
PHOTOGRAMMETRY	A technology that combines images that are taken from different angles concerning an object, to create 3D point clouds from them
POINT CLOUD	A set of data points in a coordinate system, typically representing the external surface of an object
POINTNET	A deep learning algorithm for processing point clouds directly.
POINTNET ++	An extension of PointNet, which improves its performance by aggregating local point features
POLYLINE	a connected series of line segments
PYTHON	A programming language used in various fields including data science, machine learning, and web development

RANDOM FOREST	A machine learning algorithm that combines the outcomes of multiple decision trees to reach a single result
RGB COLOURS	A colour model that represents colours using combinations of Red, Green, and Blue
SHALLOW NEURAL NETWORK	A neural network that has up to 3 layers between the in and output layer
SUPPORT VECTOR MACHINE	A supervised machine learning algorithm used for classification and regression analysis
UNITNET, FEATURENET, FINALNET	A combination of machine learning algorithms based on PointNet for construction component classification
UTILITY	a general facility, such as electricity, telecommunications, water, gas and sewerage
VOXEL	A three-dimensional pixel, representing a value in a three-dimensional grid
VOXNET	A 3D convolutional neural network designed for processing voxelized representations of objects
WIBON	‘Wet informatie-uitwisseling bovengrondse en ondergrondse netten en netwerken’, in English ‘law information-exchange above- and underground networks’
XYZ COORDINATES	A coordinate system that represents positions in three-dimensional space using X, Y, and Z axes

1. INTRODUCTION

The system of underground utilities is of vital importance. Where in the past every house, every village and every city had its own water well, oil tank and cesspool, we are now all connected to networks that have taken over these functions (Taselaar, 2009b). Efficient and sustainable infrastructure is vital for economic development, which is why virtually every major urban metropolis is building or extending rapid underground utility tunnel systems to cope with growing populations and the corresponding demand for mobility and infrastructure (Admiraal & Cornaro, 2016).

To understand the process of placing underground utilities, it is crucial to take into account all the stakeholders involved in the installation of new underground utilities. These stakeholders need to understand each other interests to ensure a successful collaboration. The stakeholders involved in the installation of new underground utilities can be divided into four types of groups:

1. Network users;
2. Utility companies that own, manage or provide the utility network;
3. The owner of the land, usually the municipality;
4. Contractor.

The network users consist of citizens, companies and public spaces. Most of the time, users assume that the utilities are in place and working. The task of delivering utilities falls upon the network operator as well as the utility and energy companies. The specific roles and obligations of these providers can vary depending on the type of utility in question. The last party is the owner of the land. The owner has to assign the routes for the utilities. The owner and the network provider have to work closely together while making these decisions (Taselaar, 2009a).

Next to the three groups mentioned above, there is a fourth important stakeholder group. This is the contractor that buries and installs the utilities. Siers Groep is such a construction group. They supply a complete package in the field of underground infrastructure in the Netherlands. This package includes services such as exploratory advice for construction and maintenance. They work on a variety of projects for clients with various complexity (Siers, 2023). One of the main tasks of Siers Groep is to accurately register where they have deployed cables and pipelines.

All utilities have to be properly mapped. The regulation about utility registration can differ per network owner but in most situations, contractors should provide a digital drawing of all cables and pipes. These as-built drawings should be available for third parties conform the act 'Wet informatie-uitwisseling bovengrondse en ondergrondse netten en netwerken', in short WIBON. In English, it means 'law information-exchange above- and underground networks'. This law prescribes the method of information exchange between network operators and contractors. Network operators must provide information about locations of underground utilities to 'het Kadaster' in the form of a digital drawing. This is a Dutch governmental organisation that merges the information. Contractors can request information about the locations of utilities when they want to dig in a certain place.

It differs per municipality what the demands for a digital drawing are. For example, the municipality of Rotterdam has a subsurface management manual with an Appendix about 'requirements and conditions measuring and as-built drawing' (Rotterdam, 2022). This document provides information about quality requirements, collection, administrative data, file format and examples of how different utilities have to be measured. The quality requirements that this document prescribes are significantly higher than the requirements that are set in WIBON. For example, in WIBON, the minimum accuracy is 1 meter and only X and Y coordinates while according to the document used by the municipality of

Rotterdam coordinates should be in meters and have an accuracy of millimetres. Furthermore, X, Y and Z - thus also including elevation - coordinates should be determined. If elevation information is available, it is easier to find utilities back and to reduce damage on utility lines during excavation operations.

Besides the diversity of guidelines and regulations regarding the registration and visualization formats of utilities, there is another serious challenge that complicates the performance of utility measurements that should take place directly on the construction site. This challenge will be the focus for the remainder of this thesis. It is elaborated below.

1.1 CURRENT STATE OF UTILITY MAPPING

Currently, new underground utilities are mapped by a land surveyor. According to the regulations of network owners, a land surveyor should come to the construction site where the construction of the utilities is finished and where the trenches are still open. When the trenches are open, utility lines are visible and physically accessible for the surveyor to measure. The surveyor measures underground utilities.

The problem with mobilizing land surveyors is that they are scarce. Sometimes the utility trenches have already been closed due to safety concerns. Consequently, the mapping of the utilities is not accurate anymore because the utilities are not visible anymore and the surveyor has to guess where the utilities are in the ground. Furthermore, land surveyors need to travel large distances between construction sites, which is also not efficient, and land surveyors are expensive resources too.

Surveyors conventionally map the assets using a total station or a GPS-based point measuring method. GPS positioning is based on trilateration, which is the method of determining position by measuring distances to points at known coordinates (Blewitt, 1997). With these tools positions and heights of assets can be mapped. The surveyor then describes the data collected and processes it into geo-information. The geo-information is often registered in CAD or GIS systems.

The result of measuring utilities with current technologies is a simple 2D network design, which lacks information about three-dimensional geometry and other semantic properties. New technologies could be used to map utilities in much more detail, including information about three-dimensional geometry and other semantic properties.

Due to the aforementioned problems with land surveyors and the lack of information that current measuring technologies provide, contractors need to develop processes so that mapping the utilities can become efficient. In relatively simple construction sites with mostly straight pipes or cables, the task of surveying may for example be executed by foremen or site superintendents by using new technologies.

1.2 NEW TECHNOLOGIES FOR UTILITY MAPPING

While surveying conventionally is done using a GPS-based point measuring method or a total station, there are new technologies that can map aerial information in more detailed ways. New technologies are for example LiDAR or photogrammetry. LiDAR is a remote sensing technology that uses laser light to measure distances. The distances are calculated by measuring the time it takes a laser pulse to return after hitting an object in its path (NOAA, 2023). By combining all the different distances of the laser pulses, a detailed 3D point cloud can be created that represents surfaces or objects in the environment, such as an open trench where utilities are present.

Another technology that can be used to map utilities in open trenches is photogrammetry. This technology combines images that are taken from different angles concerning an object, to create 3D

models from them. Software can recognize the orientation of the camera and control points that are present in overlapping pictures in different images to combine them into a 3D point cloud (Linder, 2016). The more images there are, the more accurate a model is.

If the 3D point clouds resulting from photogrammetric or LiDAR scans are georeferenced, the exact locations of objects in those scans can be retrieved. Such objects can also be utility lines. This means that all points in the point cloud are assigned a coordinate. If the points representing utilities are then extracted, classified or segmented from the point cloud, the locations of a cable or pipeline can be determined and registered in a geographic information system.

Nima Zarrinpanjeh (Zarrinpanjeh, 2023) proposed a novel method for underground utility 3D mapping and detection. A georeferenced 3D model of the trench where the new utility is placed is generated using photogrammetry in combination with satellite navigation. The results are processed to automatically detect utility lines in the 3D model using different learning methods.

One challenge with regard to processing the point clouds from photogrammetric approaches and LiDAR scans is to interpret (classify and segment) the objects that the points in the model represent. While this can be done manually using point cloud processing software, this task is laborious and not preferred (i.e., not economical) for cases of utility scanning. Depending on the size and the amount of utilities in a point cloud, it can take 30 minutes to 2 hours to extract utilities from a point cloud. Instead, extracting or segmenting the utilities from a point cloud can be done using machine learning.

The goal of machine learning is to produce an algorithm that can learn patterns that are present in a dataset to perform a specified task. Machine learning models can be categorised as supervised or unsupervised learning models. Supervised learning includes a series of functions that progress an input to an output based on a series of example input-output pairs. Unsupervised learning finds patterns in input data without using example input-output pairs. This thesis will only look into examples of supervised learning since the data used will be a series of example input-output pairs.

One of the subcategories of supervised learning is classification. In classification models the output is discrete. Examples of classification models are logistic regression, support vector machine, Naïve Bayes, decision tree, random forest and neural network (Singh, 2020). A general overview of the relevant machine learning categories is shown in Figure 1. In construction, deep learning methods are the most common machine learning methods for processing point clouds due to their broad range of applications. Support Vector Machines, Random Forests and clustering are other common methods (Mirzaei & et al., 2022).

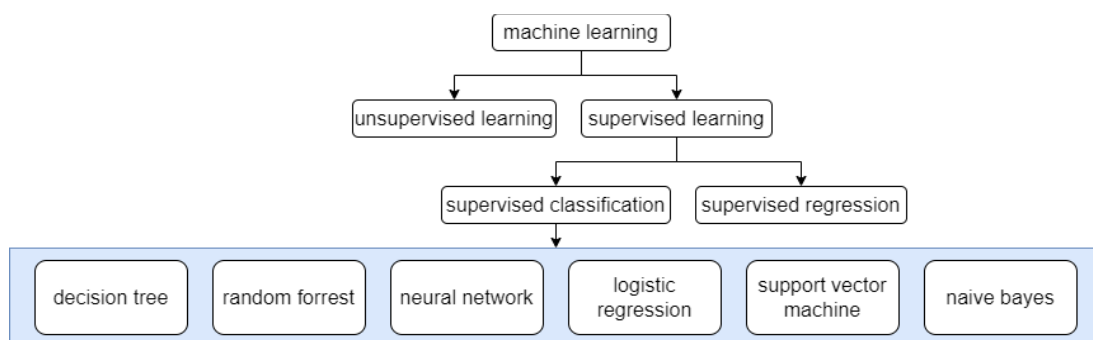


Figure 1, an overview of categories in machine learning

The listed machine learning approaches may be useful to automatically let regions of interest be extracted from point cloud scans of deployed utilities. Automating the process of retrieving the location data of utilities in point clouds enhances efficiency significantly. Such a step requires that objects of

interest have to be recognized in the point clouds by learning algorithms, based on factors such as geometry, height, colour differences, or the position of points relative to other points in the cloud (such as trenches and surroundings). If successful, the location data of the utilities on point clouds could be converted to 3D geometric maps.

There are other examples of studies that classified or segmented point clouds on infrastructure cases. For example, a case study about visualization of 3D cables between overhead utility poles obtained from point clouds was done by Inoue et al. (Inoue, 2021). The researchers evaluated the 3D modelling accuracy of a machine-learning method, a rule-based method, and a combination of both. The machine learning method was the most promising. Another case study was done by Yang et al. (Yang et al., 2023). They proposed a new method for generating topologically consistent BIM models from point clouds of utility tunnels and applied it to a case.

There are studies that show the potential to classify or segment objects in 3D models. To date, however, the literature has not yet provided a machine learning method for automatic extraction of utility data from point clouds. Labelled data is absent and processing algorithms have not been developed for this problem specifically. This is one of the main barriers to the wider adoption of 3D utility scanning in contractors like Siers.

1.3 PROBLEM STATEMENT AND RESEARCH OBJECTIVE

The process of mapping underground utilities can be more efficient. One of the novel methods to map utilities is to make a 3D point cloud scan of a trench using technologies such as LiDAR and photogrammetry. Processing the point clouds is still laborious because it is a manual task. Workers at surveying companies do not necessarily have the experience with such a task but also would have a significant amount of additional work in case they need to process 3D point clouds. Automating the process of retrieving the location data of utilities in point clouds enhances efficiency significantly.

The research objective is to compare and use machine learning algorithms for automatically recognizing and retrieving underground utilities that are present in 3d point clouds of open trenches and convert the data into geometric shapes.

1.4 RESEARCH QUESTIONS

From the problem statement and the research objective, one main research question is determined:

- How can different machine learning algorithms be used to retrieve utilities from point clouds of open trenches?

The main research question is answered by four sub-questions. These are:

5. What types of machine learning algorithms are most useful?
6. How can the machine learning algorithms be used?
7. How should the point cloud data be pre-processed?
8. How can the quality of a model be assessed?

1.5 REPORT STRUCTURE

This report is structured as follows. The next section covers the theoretical points of departure. Different machine learning algorithms are presented that can interpret objects in point clouds. The theory section is followed by the methodology, which explains how the research is done based on the research questions. It is explained what the steps are to come to a result. The methodology is followed by the results. In this section, results are shown that should answer the research questions. Finally, a discussion and a conclusion are presented followed by the recommendations.

2. THEORETICAL POINTS OF DEPARTURE: POINT CLOUD CLASSIFICATION MODELS

The integration of point clouds and machine learning has broad applications across diverse academic fields including Civil Engineering. It allows researchers to analyse, interpret, and extract valuable information from 3D spatial data, leading to advancements in understanding, automation, and decision-making in their respective domains. The intersection of machine learning and point clouds continues to evolve as machine learning algorithms become more sophisticated, enabling new opportunities for research and innovation.

Mirzaei et al. (Mirzaei & et al., 2022) found that deep learning networks – also known as deep neural networks – are the most common machine learning algorithms for processing point clouds due to their broad range of applications. Support vector machines and random forests are other common methods. To answer the first research question about what types of machine learning algorithms are most useful and how the algorithms can be used, the rest of this section describes what random forests, support vector machines and neural networks are.

2.1 RANDOM FOREST

A random forest is a collection of simple classification trees. Each tree in a random forest is trained using a random subset of the training data. A tree consists of multiple nodes and the split at each node in the tree is created using a random subset of data. By combining the outcomes of the different trees in the forest on a new observation, a class for that observation is determined. The class that is assigned to a new observation is the class that comes out of the forest the most. However, sometimes the outcomes of the trees can be weighed differently. The probability of an observation getting assigned to a class can be calculated by adding the same assignments of single trees and dividing it by the total number of trees (Basener & Basener, 2017). An illustration of a random forest can be seen in Figure 2.

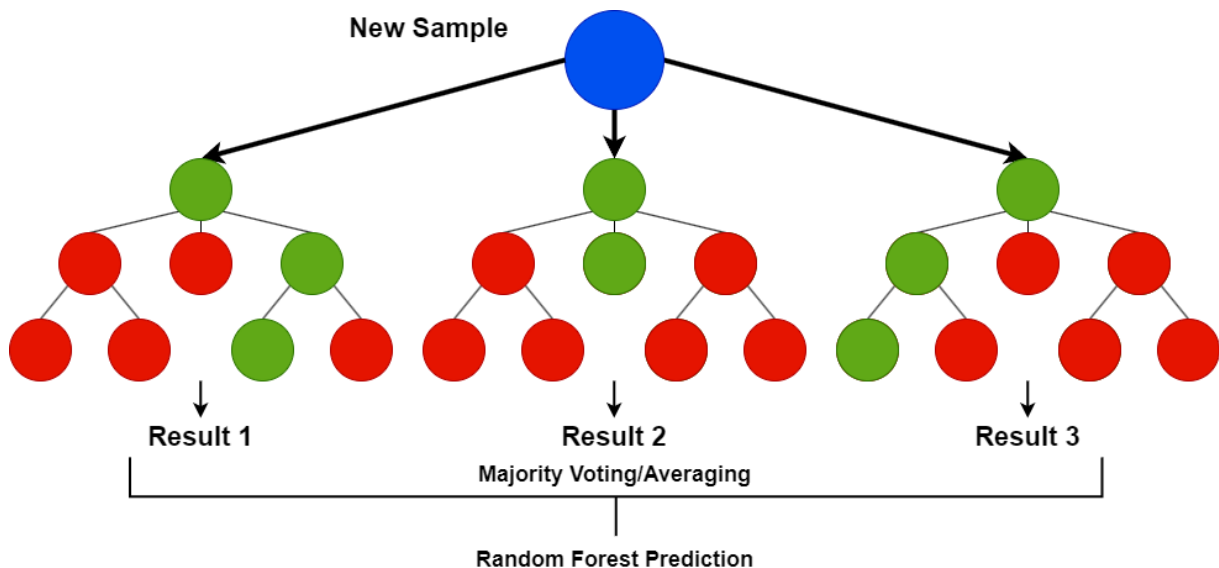


Figure 2, an example of a Random Forest

Each tree in a random forest is trained using a random subset of the training data. The important parameters for random forests are the number of trees, the size of the subset of training data used to train a tree and the number of features that are used for building a tree node. Random forests can deal with large datasets, but it has a negative effect on training time (Desai, 2023). Random forests can be

used via multiple programming languages. It depends on the programming language whether it is easy to use or not. In MATLAB, random forests cannot directly be used for point cloud classification.

2.2 SUPPORT VECTOR MACHINE

Support vector machines or SVMs are a type of machine learning algorithm commonly used for classification problems. SVMs try to find a hyperplane between different predefined classes in the training data. The position of the hyperplane is determined in such a way that it has the largest distance to the closest data points – also called support vectors – of each class that is considered. This is the margin. The support vectors are the subset of training data that identify the separating hyperplane's location. An illustration of the separating hyperplane can be seen in Figure 3. In the figure, the green plus signs can be seen as one category and the red minus signs as another category. The hyperplane splits both categories.

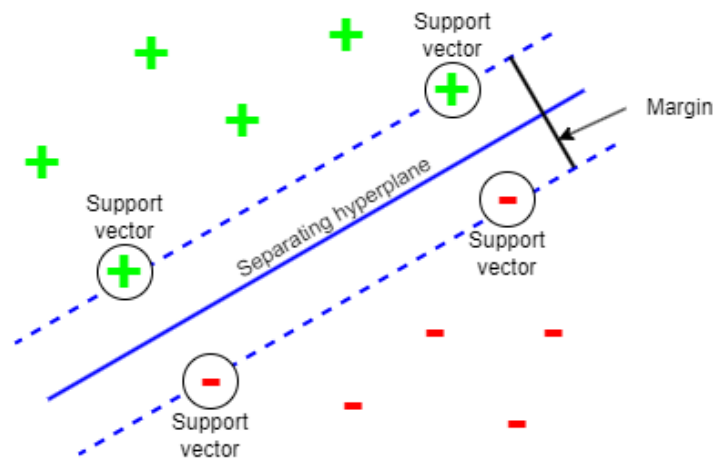


Figure 3, illustration of the separating hyperplane, margin and support vectors

SVMs work well with unstructured and semi-structured data such as text, images and point clouds. However, when the dataset is too large which sometimes is the case with point clouds, the training time can be longer. SVMs are stable because when there is a small change in the data the hyperplane is not greatly affected.

Figure 3 shows a 2-dimensional example, but data often has more dimensions. Therefore it can be difficult to understand an SVM. This could be a problem since point clouds are in a 3-dimensional space. Furthermore, SVMs are sensitive to outliers, so these have to be filtered out before using it.

SVMs can be used via multiple programming languages. It depends on the programming language whether it is easy to use or not. In MATLAB, SVMs cannot directly be used for point cloud classification.

2.3 NEURAL NETWORK

A neural network is a supervised learning algorithm that is inspired by biological nervous systems such as the brain (Dongare et al., 2012). Typically, a neural network consists of 3 parts. These are the input layer, the hidden layers and the output layer. The input layer receives the data and should include the data's features. Features are independent variables or attributes in a data set that act as input (Mcmullen, 2019). One column in a dataset can be seen as one feature. In a point cloud, a feature can be the X, Y or Z coordinate for example.

From the input layer, the data goes to the hidden layers. A hidden layer consists of multiple neurons, hence the name 'neural network'. A neuron processes an input and provides an output that is passed to the next layer. A neuron's input can consist of multiple connections that connect neurons from a

previous layer. The connections can have different weights that determine the strength of the connection. The weights can change during the training of a neural network.

The hidden layers transform the data into the shape of the output layer. For example, when the input is a point cloud and the output is a class label (such as ‘utility’ or ‘not utility’), the hidden layers transform the clouds into a class label. When a network is trained, the weights are set to a random value. During training the weights are adjusted every iteration until training data with the same labels consistently yield similar outputs (Hardesty, 2017).

Depending on the number of hidden layers a neural network can be considered as a shallow neural network or a deep neural network. When a neural network has more than 3 layers, it can be considered as a deep neural network. A deep neural network could be too overengineered for the problem at hand since the point clouds are separated into only two classes, utility and not utility. An alternative for a deep neural network is a shallow neural network. Because of the limited amount of layers, training a shallow network usually requires less time. The interpretation of a shallow network is easier than a deep network but can sometimes still be difficult.

Different deep neural network architectures can solve the classification problem at hand. These are PointNet, PointNet++, VoxNet and a combination involving UnitNet, FeatureNet, and FinalNet. Each of these architectures offers unique characteristics and capabilities and are explained in sections respectively. First, shallow neural networks are explained.

2.3.1 SHALLOW NEURAL NETWORK

Shallow neural networks can be trained. In the program MATLAB. In MATLAB, a network is fully connected for classification (Mathworks, 2023a). In a fully connected neural network, each neuron in one layer is connected to every neuron in the next layer. When not all layers are connected it is called a convolutional neural network. Shallow neural networks can be easily created, since most of the time it only consists of no more than three layers.

To consume point cloud data, a suitable input layer should be used. to easily consume point cloud data, the data should first be converted to 3D images. Multiple other programming languages can also be used to train a shallow neural network. If a network has to be trained on a large dataset, it can be split up into multiple batches that can be used for training. An example of a shallow neural network with three fully connected hidden layers is shown in Figure 4.

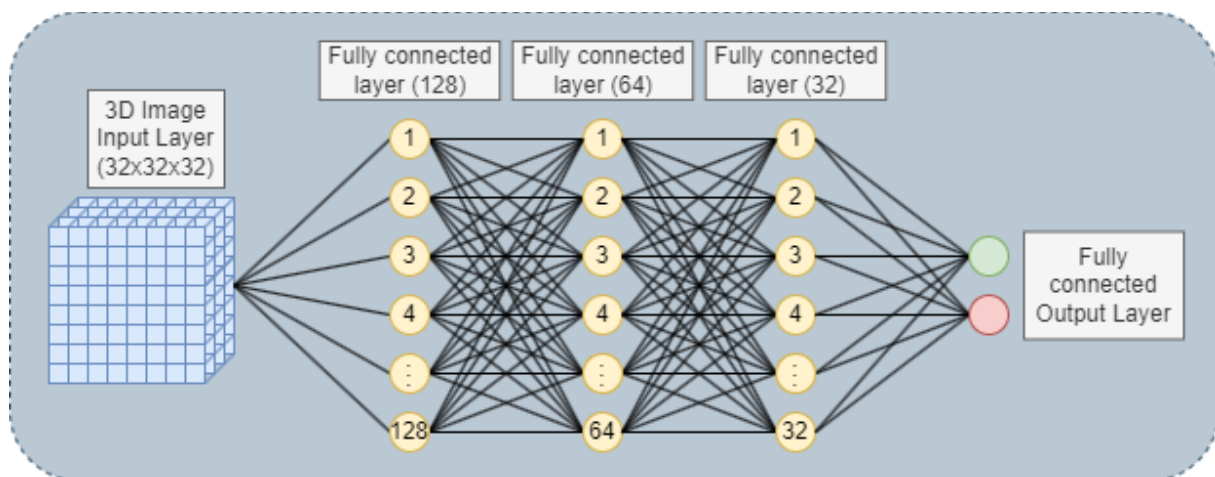


Figure 4, fully connected neural network architecture

2.3.2 VoxNET

VoxNet is a relatively simple convolutional neural network. It is designed by Daniel Maturana and Sebastian Scherer (Maturana & Scherer, 2015) to detect 3D objects in point clouds. VoxNet takes a 3D voxel grid as input and processes it through multiple to classify objects into different categories.

A voxel is a type of geometry in 3D space defined on a regular grid. Voxels can be seen as the 3D counterpart to pixels in 2D images. It is a basic, rasterized unit abstracting and structuring a space of discrete points. A point cloud is divided by voxels, where the divided points inside a given voxel are represented by this voxel (Xu et al., 2021). When a point cloud is voxelized, it is converted to a 3D image of a given grid size. The architecture of VoxNet is shown in Figure 5.

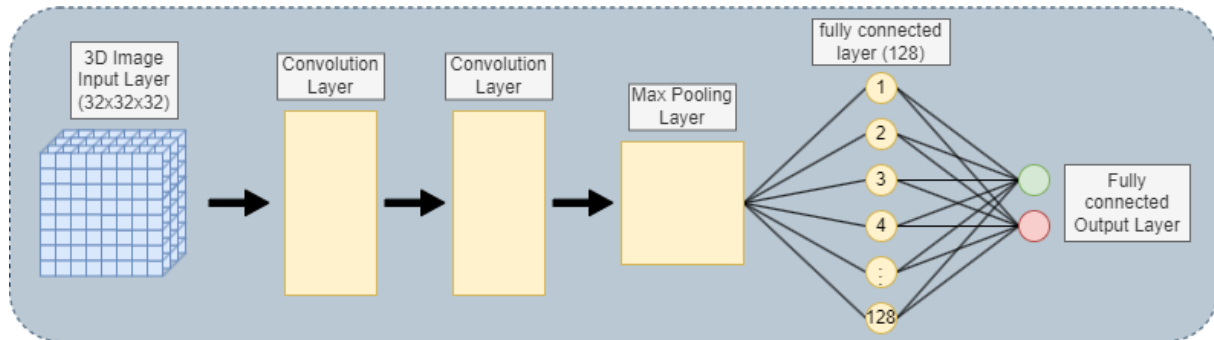


Figure 5, VoxNet Architecture based on (Maturana & Scherer, 2015)

The training time of VoxNet is probably not high, since it is a relatively simple network. If a network has to be trained on a large dataset, it can be split up into multiple batches that can be used for training. The code for VoxNet can be downloaded via GitHub (Qi, 2022). To use the network, TensorFlow or could be used in Python. It is also possible to use VoxNet in MATLAB by using an example of how to use VoxNet in MATLAB (Mathworks, 2021).

2.3.3 POINTNET

One of the most common deep neural network architectures that is used for processing point clouds is PointNet. The type of deep learning method that is used is Convolutional Neural Network and is feature-based. PointNet is specifically designed to handle unordered point clouds. In unordered point clouds, data is stored arbitrarily. Unlike traditional approaches that rely on structured grids, PointNet directly takes a set of 3D points as input without relying on any specific order or structure. PointNet provides a unified architecture for applications ranging from object classification and part segmentation to scene semantic parsing (Qi, Su, et al., 2017).

In the context of this project, classification can be used to classify utility trenches into two classes: 'utility' and 'not utility'. Classification can be used to classify a point cloud that represents an object, for example, a point cloud that represents a chair. The PointNet architecture is shown in Figure 6. It is a complex algorithm. Because of the complexity, training time will likely be high. If a network has to be trained on a large dataset, it can be split up into multiple batches that can be used for training.

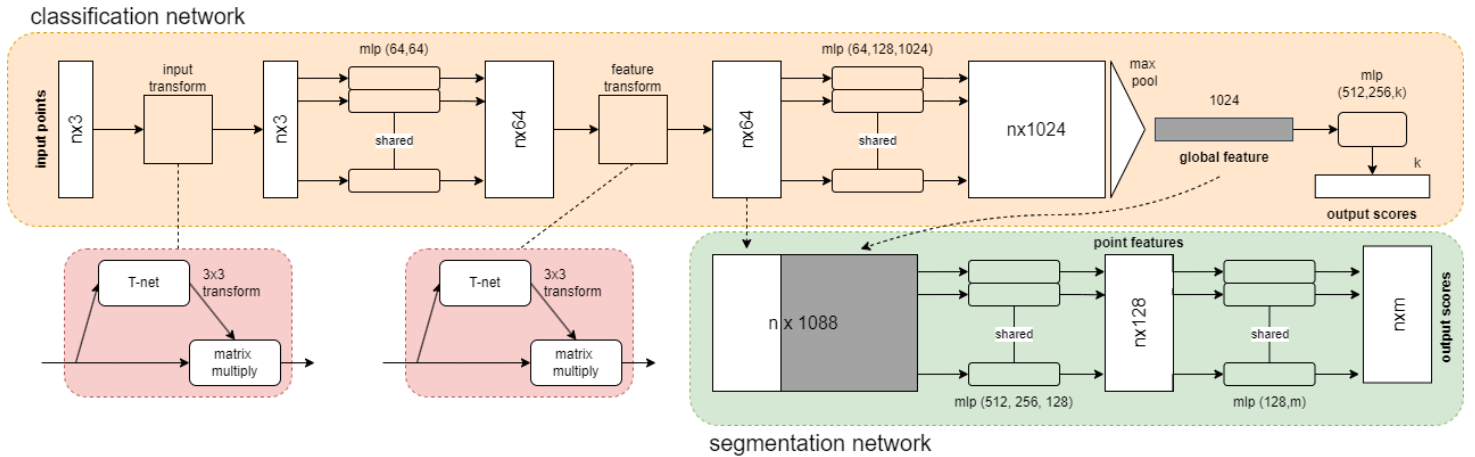


Figure 6, PointNet Architecture based on (Qi, Su, et al., 2017)

The code for PointNet can be downloaded via the online software development platform GitHub (Qi, 2022). To use PointNet, TensorFlow or PyTorch could be used. These are open-source machine learning frameworks. The frameworks can be used via the programming language Python. It is also possible to use PointNet in MATLAB (Mathworks, 2023c).

2.3.4 POINTNET ++

There is also an updated version of PointNet called PointNet++. This feature-based network is an extension of PointNet that addresses the limitation of its ability to recognise fine-grained patterns and generalizability to complex scenes (Qi, Yi, et al., 2017). In PointNet++ a hierarchical neural network architecture is introduced. A point cloud usually has a varying density, which can result in bad performance of networks that are trained on a constant density.

In PointNet ++, density adaptive PointNet layers are introduced that learn to combine features from regions of different scales when the input sampling density changes. These layers are very robust to sampling density variation (Qi, Yi, et al., 2017). The illustration of the complex architecture is shown in Figure 7. Because of the complexity, training time generally is high. If a network has to be trained on a large dataset, it can be split up into multiple batches that can be used for training.

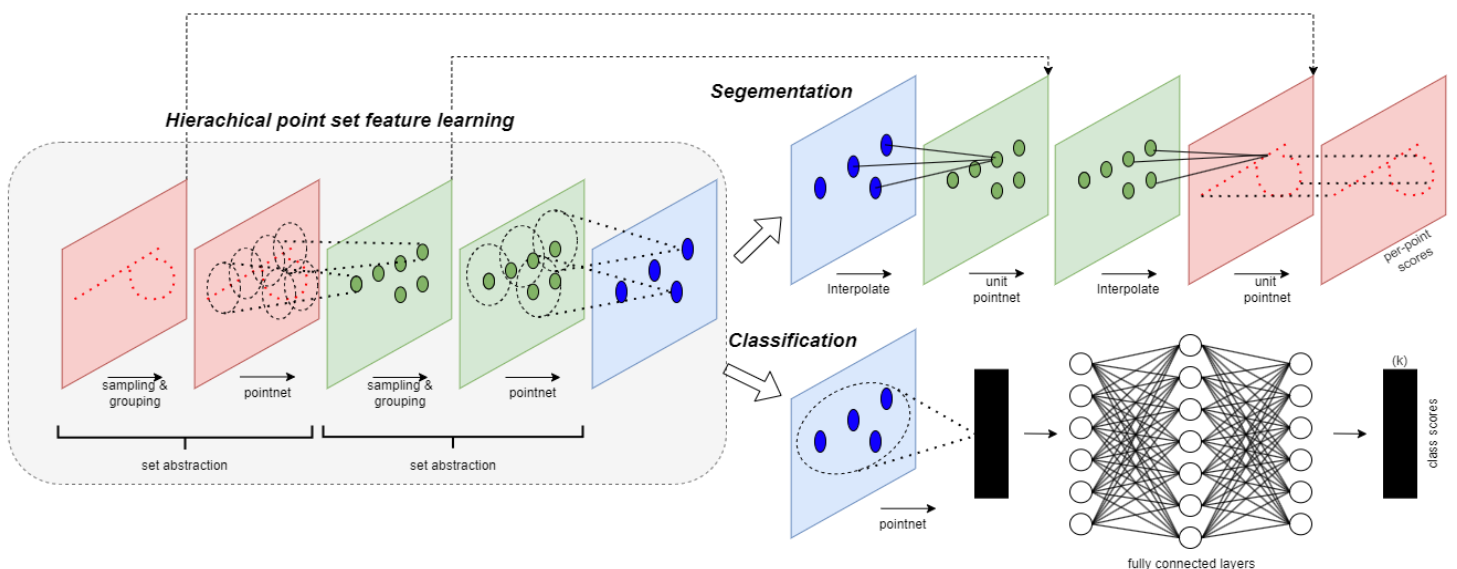


Figure 7, PointNet++ Architecture based on (Qi, Yi, et al., 2017)

Just like PointNet, PointNet ++ can directly consume point clouds. The code for PointNet++ can be found on GitHub (QI, 2018). To use PointNet++ the open-source machine learning framework TensorFlow could be used. The framework can be used via the programming language Python. The segmentation part of PointNet++ can also be used via MATLAB. Because of the complexity of the network, usage could be difficult.

2.3.5 UNITNET, FEATURENET AND FINALNET

A novel method for construction component classification uses a feature-based deep learning network and is presented in a paper by Xu et al. (Xu et al., 2022). Feature fusion is used to combine different features calculated in the algorithm to obtain a class. The deep learning network presented in this paper is shown in Figure 8. It is inspired by PointNet and consists of UnitNet, FeatureNet and FinalNet.

‘The proposed method starts with local and global feature extraction, where global features are processed by the neural network and the traditional shape features are processed separately. Global features are formed by the global coordinates of a point and the shape features - which are the local features – describe the shape of an object in an alternative way, reducing the amount of information stored. Then, a feature map is generated and deep convolution is performed to achieve feature fusion.’ (Xu et al., 2022)

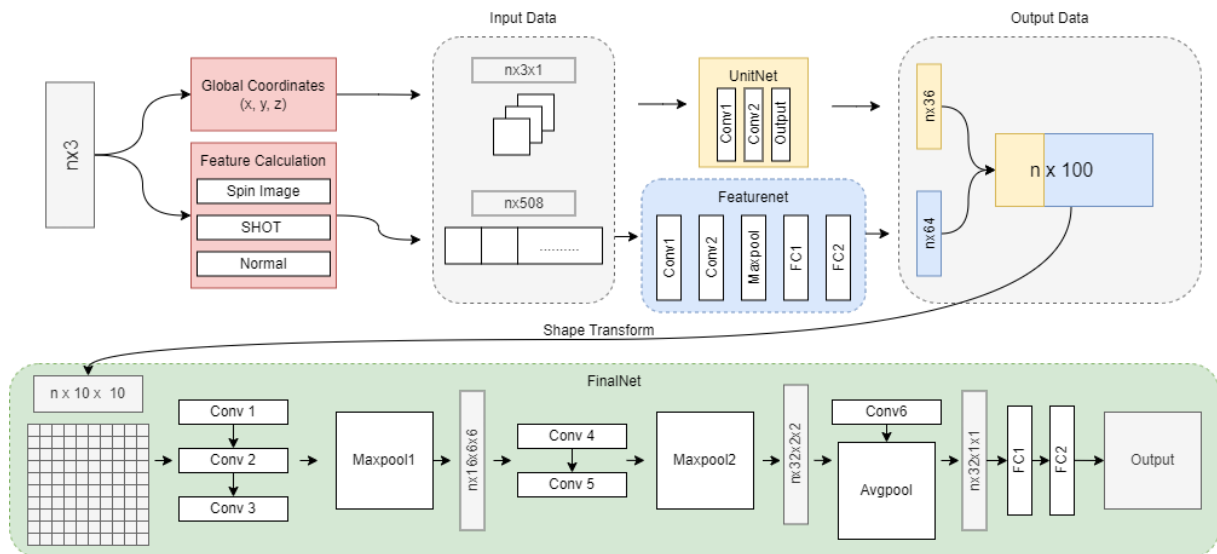


Figure 8, the architecture of UnitNet, FeatureNet and FinalNet based on (Xu et al., 2022)

Although the deep neural network is designed especially for complex construction industry applications it is hard to use, because a code for the network architecture is not available to the public. If the network were to be used, it would have to be recreated. This is a time-consuming task. Because of the complexity, training time will likely be high.

2.4 OVERVIEW OF MACHINE LEARNING ALGORITHMS

In this section, an overview of the different machine learning algorithms is shown in a comparative table. The algorithms are assessed on the following criteria:

- Able to consume point cloud data

This point refers to the capability of a machine learning algorithm to consume point cloud data. Some algorithms can directly be trained with point cloud data as input. For other algorithms the data first has to be transformed to a 3D image and some algorithms can handle point cloud data more difficult. First converting point cloud data means that it undergoes an extra step, but when this requires little time it may still be an option.

- Open-source

Open-source software refers to programs whose source code is freely available for users to use. Some algorithms can be downloaded and used directly from the internet, while others are unavailable. When they are unavailable, one should recreate an algorithm based on the architecture. More general algorithms have multiple options that should be determined by the user. Some algorithms can be easily created by the user.

- Complexity algorithm

Complex algorithms include advanced techniques that can be hard to understand and implement. When an algorithm is used as a 'black box' it does not have to be a problem, but when details should be known, it is better to choose an algorithm that is easy to understand.

- Training time

When an algorithm takes a lot of time to train, it may be economically inefficient. Training time depends on how an algorithm can deal with a large dataset and how complex an algorithm is.

- Deal with a large dataset

A dataset with point clouds is usually big. Therefore it is important to know if an algorithm can deal with a large dataset. Some models can deal with a large dataset. Some algorithms can deal with a large dataset but it increases the training time considerably.

- Software

Algorithms can be used in different via different programs. It is described whether an algorithm can be used via MATLAB and/or Python. In the next section, this is explained in more detail.

- Ease of use

Ease of use refers to how user-friendly an algorithm is for its users. Algorithms with a high ease of use feature, clear documentation, helpful tutorials, and user support. Ease of use also refers to the steps it takes to create a working model.

On the next page, a comparative table is displayed that shows how each algorithm scores on different criteria.

Table 1, comparative table for different machine learning algorithms

	<i>Shallow Network</i>	<i>PointNet</i>	<i>PointNet ++</i>	<i>VoxNet</i>	<i>UnitNet, FeatureNet, FinalNet</i>	<i>SVM</i>	<i>Random Forest</i>
<i>Able to consume point cloud data</i>	First, convert to 3D images	Yes	Yes	First, convert to 3D images	Yes	Depending on the programming language, MATLAB no	Depending on the programming language, MATLAB no
<i>Open-source</i>	Yes	Yes	Yes	Yes	No	Yes	Yes
<i>Complexity algorithm</i>	Medium	Hard	Hard	Medium	Hard	Hard	Easy
<i>Training time</i>	Low	High	High	Medium	High	High	High
<i>Deal with a large dataset</i>	Yes	Yes	Yes	Yes	?	No	Yes
<i>Software</i>	MATLAB, Python	MATLAB, Python	MATLAB for segmentation, Python	MATLAB, Python	Not open source	MATLAB, Python	MATLAB, Python
<i>Ease of use</i>	Easy	Medium	Hard	Medium	Hard	Hard	Hard

2.5 TRAINING OPTIONS

When an algorithm is trained, it becomes a model. Different training options can be changed to improve the quality of a model. In this section training options are described that can be altered to improve quality. These are the number of epochs, the batch size, the initial learning rate and the drop period. For models that use 3D images as input, the grid size is a fifth training option.

The first training option is the number of epochs. An epoch is the number of iterations that a training dataset passes through layers in a machine learning model. Commonly, the higher the number of epochs, the better the model performance, but also the more computational effort is needed. One epoch is passed when the entire training dataset is passed through the model once.

The second training option is the batch size of training data that is fed into the training model. Sometimes data needs to be split into batches because it is too large to pass through the model in one go. The number of batches depends on the batch size. If an image dataset has for example 5000 images and the batch size is set at 500, the number of batches is 10.

The third training option is the initial learning rate at which the machine learning needs to achieve an optimal solution. By gradually changing the weights attributed to nodes in the training layers, a model can improve its performance. If this takes place in small steps, learning takes place slower, but the likelihood increases that an optimal model is outputted. The learning rate thus determines at what speed the weights in a model are moved to an optimum (Rakhecha, 2019).

The fourth training option refers to the learning rate drop period during the training process. The period determines after how many epochs the learning rate is dropped by a fixed factor. By dropping the learning rate, the model performs more efficiently as learning can take place in smaller steps when the model converges an optimum. The models have different ways of determining the drop period. In VoxNet and the shallow neural network, the drop period depends on the size of the training set and the batch size.

For models that use 3D images as input, another training option is the grid size. The models that use this input are the shallow neural network and VoxNet. The process of converting a point cloud to a 3D image is called voxelization. A voxel is a type of geometry in 3D space defined on a regular grid. Voxels can be seen as the 3D versions of what pixels are for 2D images. In other words, it is a basic, rasterized unit abstracting and structuring a space of discrete points. A point cloud is divided by voxels, where the divided points inside a given voxel are represented by this voxel (Xu et al., 2021). When a point cloud is voxelized, it is converted to a 3D image of a given grid size. The grid size is an extra training option in these models. If the grid size is for example 32x32x32, it means that the space is divided into 32 parts in the X, Y and Z directions.

3. RESEARCH METHODOLOGY

The research methodology describes how the research is done. The methodology is explained based on the four research questions. First, it is explained how the choice can be made on what type of machine learning algorithms are trained. After that, it is explained how the data can be pre-processed. This is followed by an explanation of how to train the chosen algorithms. Then, it is explained how the quality of the chosen models can be assessed. At last, the trained model with the best quality is used for a demonstration case. In this demonstration case, a new point cloud is classified. In Figure 9, a flowchart of the research methodology can be seen.

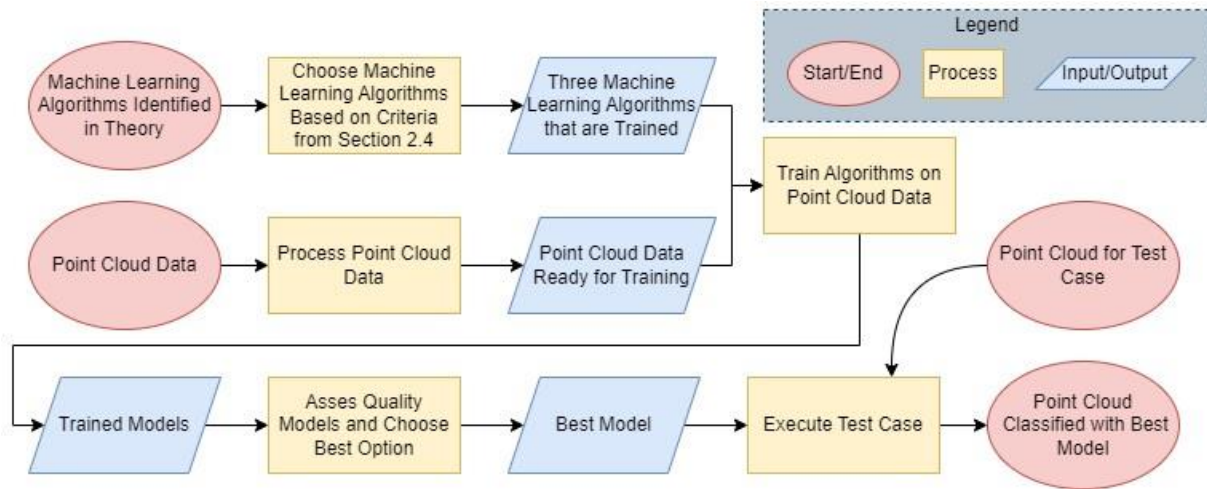


Figure 9, flowchart research methodology

3.1 CHOOSE ALGORITHMS TO TRAIN

In Table 1 in section 2.4, different machine learning algorithms that have been identified are displayed and assessed on the criteria defined in section 2.4. Based on how the algorithms score on the criteria, a choice is made which algorithms are trained. Three algorithms are chosen that best fit the problem at hand. The algorithms can be used via different programs. For this thesis, MATLAB is used because there is clear documentation available about using the algorithms in MATLAB.

3.2 PROCESS POINT CLOUD DATA

This section explains the preparation of point cloud data so it is ready for training the models. First, the raw data undergoes pre-processing, including noise removal, downsampling and cropping. This is followed by labelling the clouds. Additional features are added to enrich the dataset. It is then explained how the data is formatted to be ready for training. The next step is to cut parts of the clouds into smaller pieces for training. Finally, it is explained how the data is stored so it is ready for training. A flowchart of the process is shown in Figure 10.

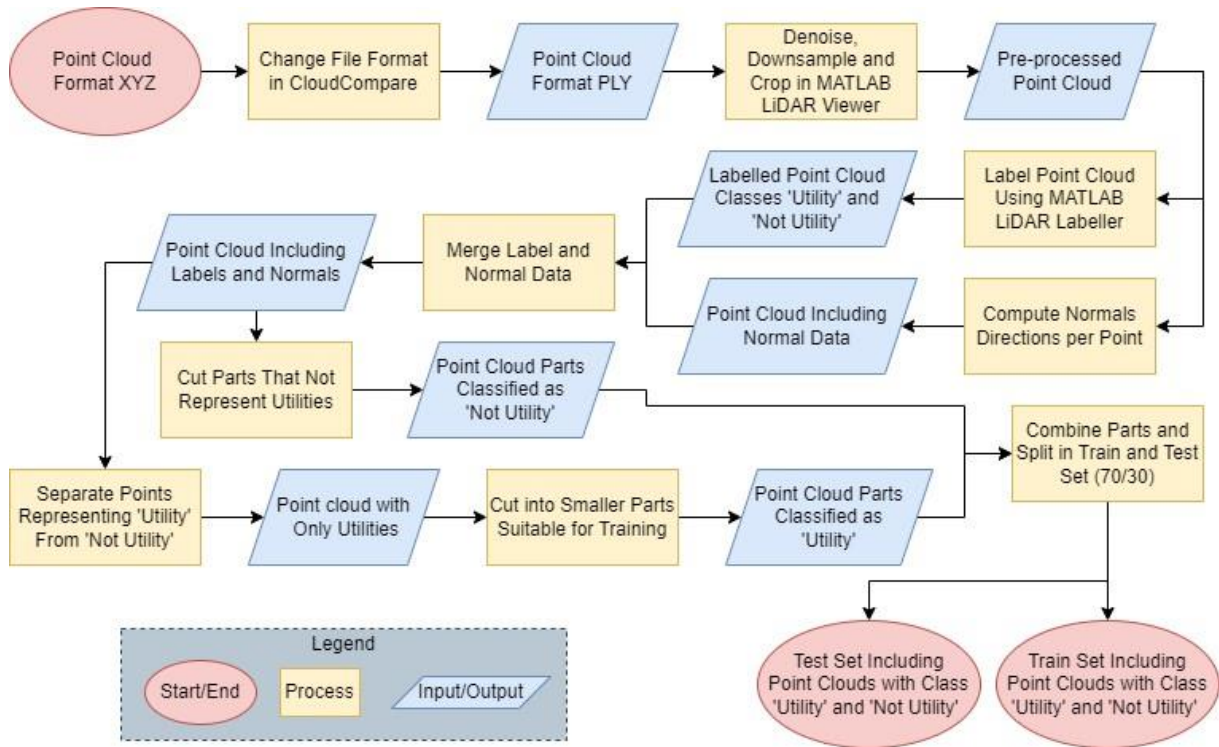


Figure 10, a flowchart of processing point cloud data

3.2.1 PREPROCESS POINT CLOUDS

The point clouds that are used are all saved in an XYZ file format. The files contain information about the location and colour of all the points in a point cloud. The XYZ files have to be transformed into PLY files so they can be used in MATLAB. To do this, I used the program CloudCompare. CloudCompare is a 3D point cloud processing software (CloudCompare, 2024).

The point clouds are pre-processed because originally they contain a large number of points. This is because the computer used for this thesis cannot handle large point clouds well. To preprocess the point clouds, I used the MATLAB LiDAR viewer. The LiDAR viewer is a tool to visualize, analyse and preprocess point cloud data (Mathworks, 2024). The tool is used to denoise, downsample and crop the point clouds. First, all point clouds are denoised using the denoise algorithm. Standard settings for the number of neighbours and the outlier threshold are used. The algorithm looks at the nearest neighbours and if the average distance to its nearest neighbours is above a threshold, the point is removed.

After the denoise algorithm is used, all clouds are downsampled. The Downsample algorithm removes a percentage of points from the cloud. This happens randomly and the default percentage is used. Finally, some clouds are cropped, this is because sometimes there is a large area present in a point cloud that does not have any utility in it. Examples of an original and a cropped, denoised and downsampled point cloud are shown in Figure 11 and Figure 12 respectively.

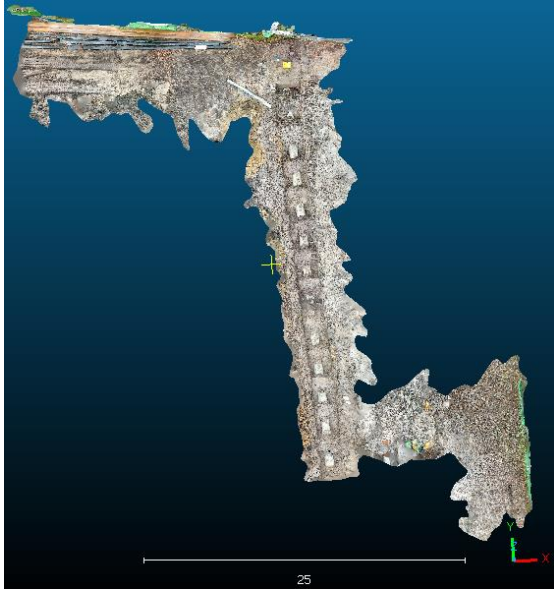


Figure 11, an original point cloud of a utility trench



Figure 12, cropped, denoised and downsampled point cloud

3.2.2 LABEL DATA

After preprocessing the data, I labelled the point clouds. This is done using the MATLAB LiDAR Labeller. The LiDAR labeller is a tool to interactively label ground truth data in a point cloud or a point cloud sequence to generate corresponding ground truth data (Mathworks, 2023b). The pre-processed point clouds are loaded into the LiDAR labeller.

The Labeller is used to draw a shape around a set of points that has to be labelled. In this case, a line is drawn around the utilities to label the points representing a utility. The goal is to retrieve the utilities from a point cloud of a trench. Therefore, the points that are utilities are all labelled as 'utility'. The other points - the ground and other objects - are not labelled. Examples of an unlabelled and a labelled point cloud are shown in Figure 13 and Figure 14 respectively.

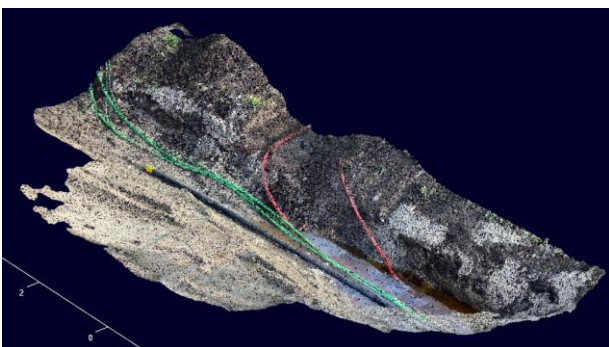


Figure 13, an unlabelled point cloud of a utility trench

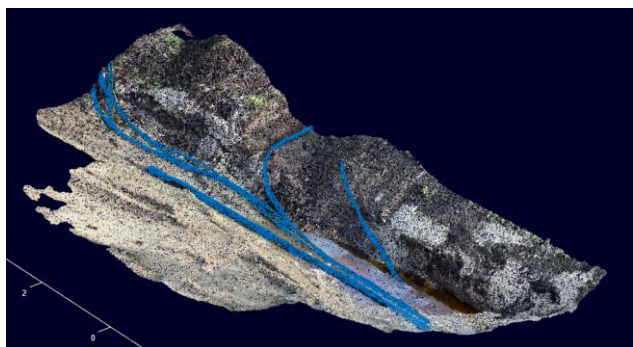


Figure 14, a labelled point cloud of a utility trench

In the right figure, it can be seen that the utilities are labelled blue and get a class value of '1'. Points with this class value thus represent a utility. The rest of the points are not labelled and get a class of value '0', thus not representing a utility. The point clouds are exported as 'ground truth' and contain a table with the coordinates of points and their corresponding class. Ground truth is information that is known to be true and is provided by observation.

30 point clouds were available in Siers and 27 were labelled. I have excluded three clouds from the training set either because there were no utilities present or they were not visible. The 27 point clouds each contain one to six different utility lines. The utilities have different colours and sizes. Some utilities are straight and some contain bends. In appendix 2 I have further described the dataset.

After labelling, data consists of seven features. These are: X, Y and Z for the location of a point, R G and B for the colour of a point and a class value. The X coordinate represents the horizontal position in relation to the origin, the y coordinate the vertical position in relation to the origin and the z coordinate represents the altitude in relation to the origin. RGB stands for Red, Green and Blue and is a colour model used to represent colours by combining different intensities of these three colours. Each colour is represented by an intensity value ranging from 0 to 255.

3.2.3 COMPUTE NORMALS

To get better accuracy from a classification model, more features can be added besides XYZ, RGB and a class. I have added three more features. These are the normals of the points that consist of a normal value in the X, Y and Z directions. Normals are vectors that are perpendicular to the surface of objects represented by the points in a point cloud. Each point in the point cloud represents a location in 3D space, and by calculating the normals at each point, you can determine the orientation of the surface at that point. Normals are necessary to convert a point cloud into a shape. The normals are computed using CloudCompare (CloudCompare, 2016).

3.2.4 MERGE DATA

To train a model, all data of a point cloud should be put together into one file. The pre-processed point clouds contain coordinate and colour data. After labelling the point clouds they are exported as 'ground truth' and contain a table with the coordinates of points and their corresponding class. The point clouds with normal data include coordinates, colour and normal information. I put the data together using a script written in MATLAB. The script can be found in appendix 1. Data is put together per point cloud.

The merging process takes place as follows. A table is created in which all data is stored. The coordinate and colour data from the pre-processed point cloud is stored in the first six columns. The normal data is stored in the seventh eighth and ninth columns. At last, the labels from the labelled point cloud are stored in the tenth column.

After merging the data, I have separated the points that are labelled as 'utility' from the points that are not labelled as 'utility'. This creates a point cloud that has only the points labelled as 'utility' in it. These clouds are loaded into CloudCompare to check if there are any mistakes made during labelling. If this is the case, the labelling is adjusted and the script is rerun. An example of a cloud that has only utilities in it is shown in Figure 15.

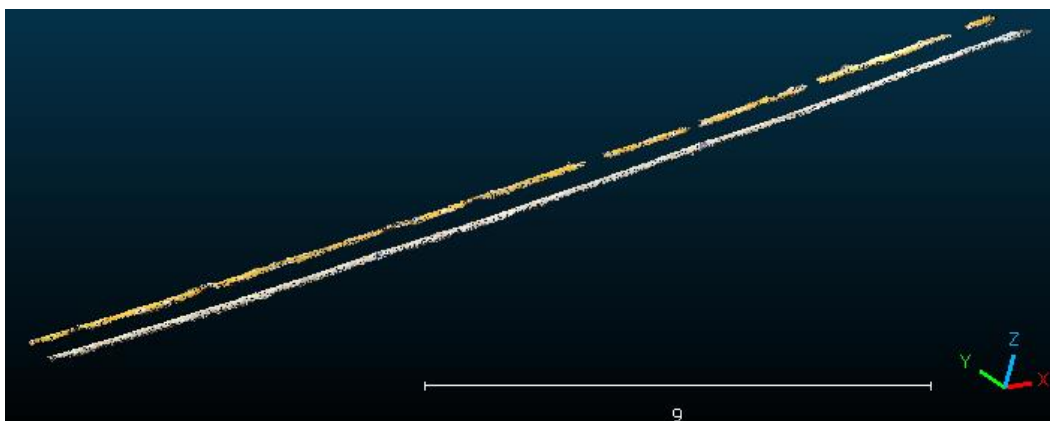


Figure 15, a point cloud with only utilities

3.2.5 CUT POINT CLOUDS INTO SMALLER PARTS

For a machine learning model to recognize utilities in point clouds, it should be trained with smaller fractions of different sizes of point clouds with utilities. In Figure 15 for example, utilities can be seen that have a length of more than 30 meters. A machine learning model should be fed with smaller pieces, so it can also recognize utilities that have different lengths. To create smaller parts, I cut the point clouds with only utilities into smaller pieces of different sizes, ranging from 0.5 to 4 meters. I did this via MATLAB and CloudCompare.

In MATLAB I used the crop function in the LiDAR viewer. Some point clouds lay diagonally in the XY plane. To make cutting easier, the point clouds are rotated around the Z-axis. An example of a point cloud before and after rotation around the Z-axis is shown in Figure 16 and Figure 17 respectively.

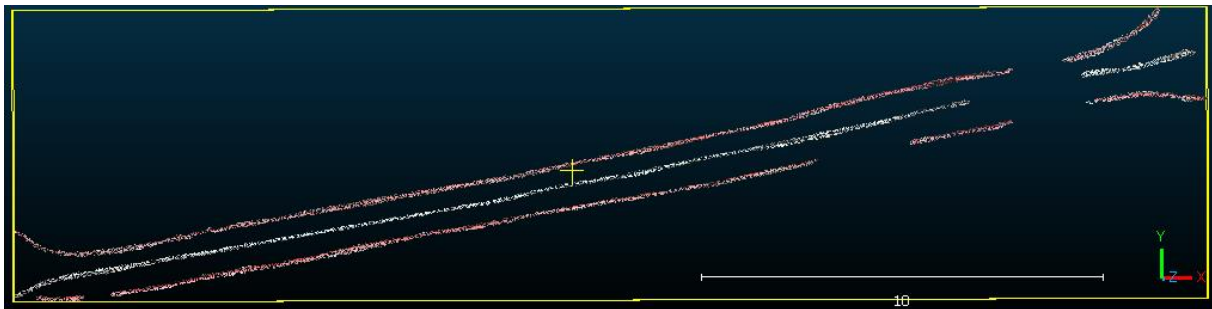


Figure 16, point cloud before rotation around the Z-axis

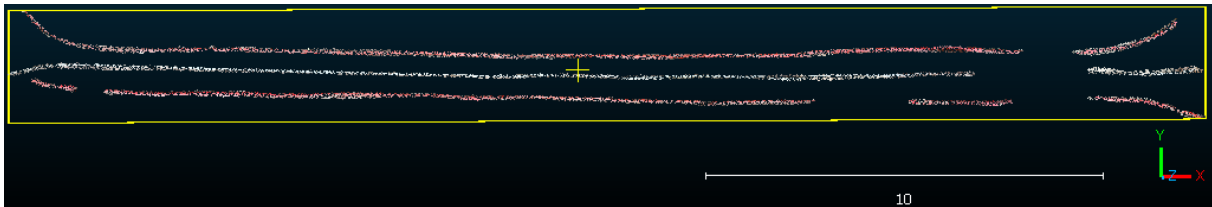


Figure 17, point cloud after rotation around the Z-axis

Utilities smaller than 0.5 meters are not cut into parts. In one point cloud the utilities are all smaller than 0,5 meters. Therefore, 26 of the 27 labelled point clouds are cut into pieces. 23 point clouds were cut into parts using MATLAB. I cut the other three in CloudCompare. In CloudCompare there is a segment option. 'The segment tool allows one to segment selected entities by defining a 2D polygon on the screen. The process can be repeated multiple times. Each time it can be decided to keep the points falling inside or outside the polygon' (CloudCompare, 2018). An example of a point cloud of utilities that is cut into pieces is shown in Figure 18.

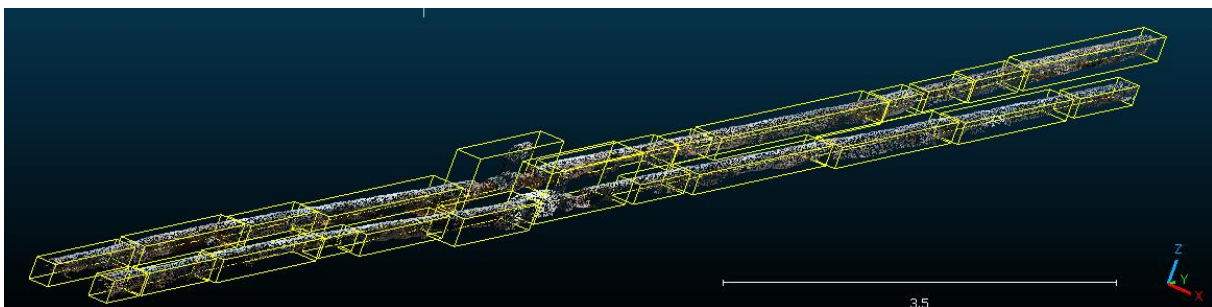


Figure 18, a point cloud with utilities that are cut into parts

There should also be a dataset of point clouds that do not have utilities in it because a model should be able to classify parts of a point cloud into ‘utility’ and ‘not utility’. This can be clouds of parts of soil, grass or other objects that can be present in a point cloud of an open trench with utilities. I cut these parts from the pre-processed clouds including normals. This was done using the segment tool in CloudCompare. Using this tool, shapes of different sizes that look like utilities were cut from 27 different clouds. An example of one cloud is shown in Figure 19.

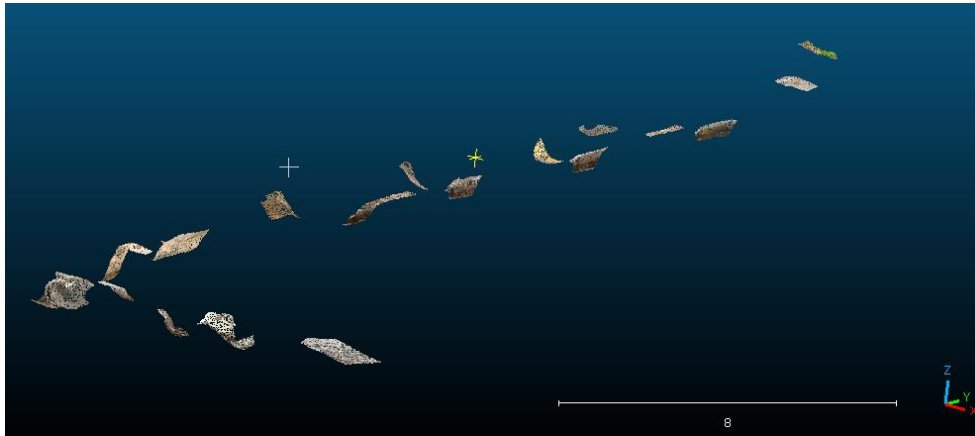


Figure 19, shapes cut from a point cloud that look like utilities

In total I cut the utilities into 552 different point clouds. From the point clouds, I cut 552 parts of different shapes and sizes that are not utilities to create a balanced dataset. In appendix 2, all 27 point clouds are described. It shows the title, a screenshot of the clouds with only the utilities, the number of points of the original and pre-processed cloud and the cloud with only utilities. It also shows in how many parts the utilities are cut and how many parts that are not utilities are cut from the pre-processed point cloud.

3.2.6 STORE POINT CLOUD PARTS

After cutting the point clouds into parts, the parts are saved per point cloud. Utility and not-utility clouds are also saved separately. The dataset for training a machine learning model should consist of all clouds together in a random order. This dataset can be split into a training and testing set. The training set should consist of 70% of the data and the test set 30%. This is a commonly used division (Anwar, 2016). First I stored all the utility and not-utility clouds in two separate folders. Then I used a MATLAB script to randomly distribute all clouds in a train and test set. The script can be found in appendix 3. The training set contains 772 point clouds together with the class ‘utility’ or ‘not utility’ and the test set contains 332 point clouds with the same class description.

All point clouds consist of a different number of points. To train a model, all point clouds should have the same number of points. Therefore, the code I used to create a train and test set also down and up samples the point cloud to a fixed number. The number of points in every point cloud should be equal to 1024. This number is a good trade-off between adequately capturing the shape of the objects while not increasing the computational cost by processing too many points (Mathworks, 2023c).

3.3 TRAIN ALGORITHMS

Models are the output of a trained algorithm. The algorithms are trained on the training set. To compare algorithms that are chosen to be trained, they should be trained on different training options. To get to an optimal model, optimal training options should be chosen. The output of training is a trained network.

I train all models based on four different training options. These are the number of epochs, the batch size, the initial learning rate and the drop period. For models that use 3D images as input, the grid size is a fifth training option. The training options are explained in section 2.5.

All selected models are trained on 10 different combinations of training options. The training process can be monitored to evaluate whether changes to the training options can be made to increase its effectiveness. First, during training, the learning progress can be shown by plotting per iteration what the training accuracy and loss are.

The validation accuracy and loss can also be plotted. Accuracy gives an idea of how well a model is performing by looking at how often a model makes a correct prediction out of all predictions. Loss gives an insight into how far off the predictions are from the actual values. The training time can also be derived from a plot.

An example of a training progress plot is shown in Figure 20. The top graph shows the training accuracy in light blue, the smoothed training accuracy in blue and the validation accuracy in black. The bottom graph shows the training loss in light orange, the smoothed training loss in orange and the validation loss in black.

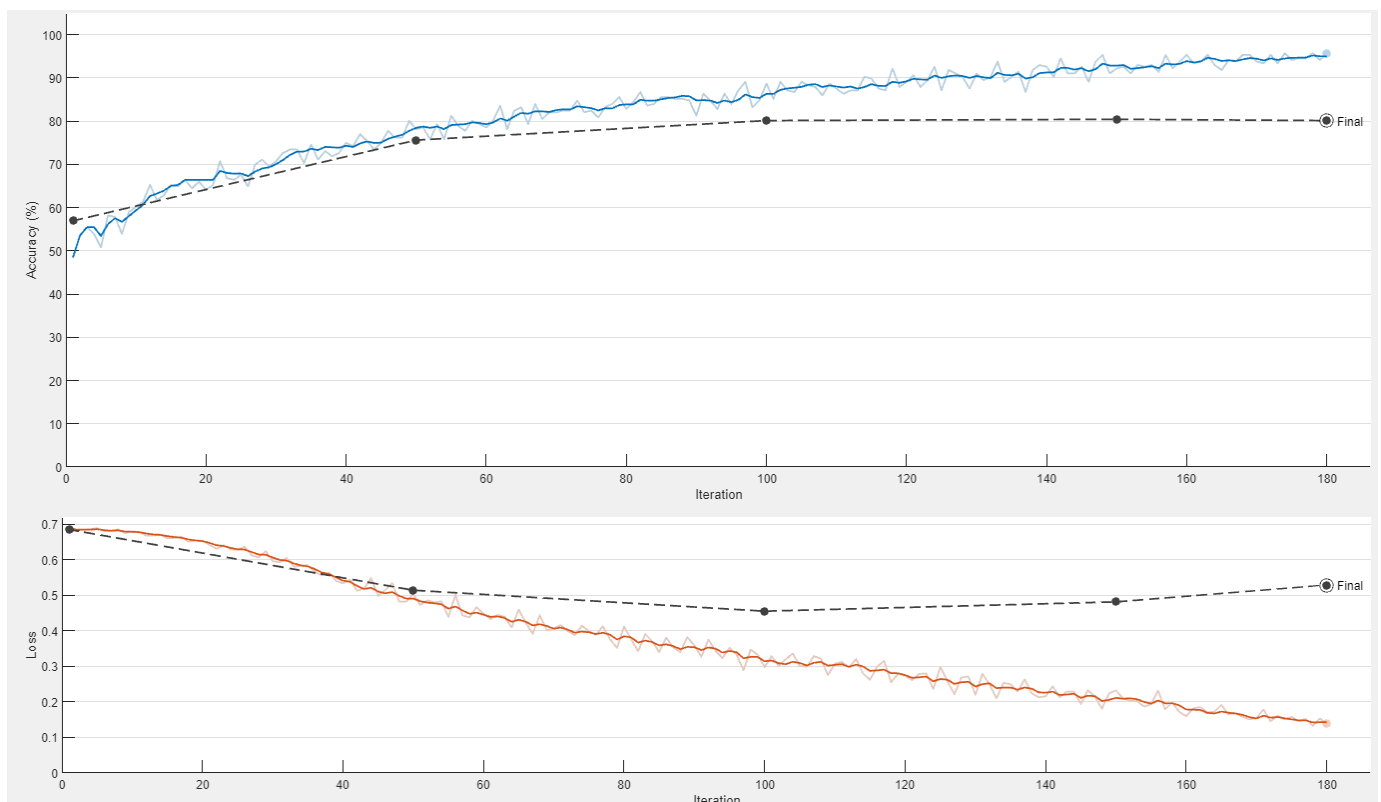


Figure 20, an example of training progress VoxNet and shallow neural network

3.4 QUALITY ASSESSMENT

The output of training is a trained network. The models can be assessed on how well they perform on the test dataset. When a trained model is tested on test data, a confusion matrix can be created. The confusion matrix shows how a model predicts the classes of the input data. An example of a confusion matrix is shown in Figure 21.

The top left box shows how many point clouds that are predicted as a utility are also a utility. This box can also be called 'True Positive' or 'TP'. The bottom left box displays the number of point clouds that

are predicted as a utility but are not a utility. This can also be called ‘False Positive’ or FP. The top right box shows how many clouds are predicted as not utility but are a utility. This can also be called ‘False Negative’ or ‘FN’. The bottom right box displays the number of clouds that are predicted as not utility and are also not utility. This box can also be called ‘True Negative’ or TN. In this case positive refers to ‘utility’ and negative refers to ‘not utility’.

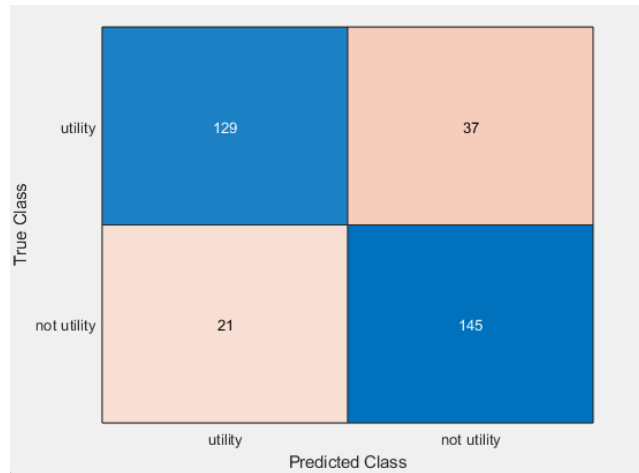


Figure 21, an example of a confusion matrix

Using the values present in the matrix, different quality measures for a model can be calculated. These are precision, recall, accuracy and f1-score. According to Bajaj (Bajaj, 2023), these quality measures are commonly used for classification problems. A combined measure for precision and recall named ‘quality’ is also defined.

Precision is the ratio of true positives to all predicted positives (Bajaj, 2023). The equation can be seen below.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

Recall is the ratio of true positives to all positives in the ground truth data (Bajaj, 2023). The equation for recall is shown below.

$$recall = \frac{TP}{TP + FN} \tag{2}$$

Quality is a general measure of precision and recall combined in a single measure (Wiedemann et al., 1998). The equation can be seen below.

$$quality = \frac{TP}{TP + FP + FN} \tag{3}$$

Accuracy is the most common measure of quality. The number of correct predictions is divided by the total number of predictions. The equations can be seen below.

$$accuracy = \frac{TP + TN}{TP + FN + FP + TN} \quad (4)$$

The f1 score is the harmonic mean of precision and recall. The equation can be seen below.

$$f1 - score = 2 * \frac{precision * recall}{precision + recall} \quad (5)$$

The outcomes of the quality measures per trained model are compared to choose the best-trained model. All models can predict point clouds wrong. Point clouds that are labelled wrong are visualized to see whether the data influences the quality of the model.

3.4.1 DEMONSTRATION CASE

A final step in this study is to feed new data to the model that scored best on the test data. A demonstration case is executed to show whether the trained model can classify new data. For the demonstration case, a new point cloud is used. The cloud contains six pipes and is depicted in Figure 22. This section first explains how the test point cloud is pre-processed. Then it is described how the point cloud is cut into parts using two methods. After that, it is explained how the trained model can predict labels for new data. At last, a method is defined for how pipes that are classified can be converted into polylines.



Figure 22, point cloud for demonstration case

First, the point cloud is pre-processed. This is done by following the steps described in section 3.2.1. The point cloud is first converted from an XYZ to a PLY file. After that, it is denoised, downsampled and cropped. The pre-processed point cloud is shown in Figure 23.



Figure 23, pre-processed point cloud

After preprocessing, the point cloud is cut into pieces so the trained model can classify the parts. This is done in two ways, semi-automatically via CloudCompare and, automatically via MATLAB. These methods are used so that the trained model can classify a whole point cloud directly after preprocessing.

To cut the point cloud in parts using CloudCompare, the Cloth Simulation Filter (CSF) is used (Zhang et al., 2016). This filter separates the point cloud into ground and non-ground measurements. After applying the CSF filter to the point cloud, it is split into a point cloud containing ground points and a point cloud containing off-ground points. The cloud containing off-ground points represents the pipes. Most of the points representing the ground are present in the ground point cloud. However, some points representing the ground are present in the off-ground point cloud. These are cut out manually using the segment tool described in section 3.2.5.

The cloud that is left represents the pipes. The pipes are split using the 'label connected components' tool (CloudCompare, 2015). This tool splits a point cloud into parts separated by a minimum distance. This results in separate point clouds each representing a pipe. To better classify the pipes, they are cut into smaller pieces by using the segment tool described in section 3.2.5. This results in small point clouds with a length varying between 1 and 3.5 meters.

The previous method semi-automatically cuts a point cloud into parts that only represent utilities. To automatically cut the whole point cloud into parts, a MATLAB code is used. The code can be found in appendix 4. The code divides the point cloud into rectangles of random size. It first defines the rectangles in which the point cloud is divided. It then cuts parts from the point cloud using the dimensions of the rectangles.

To classify the point cloud cuts made using the previously mentioned methods, the trained model with the best quality is used. The trained model uses the point cloud parts as input and predicts a class label as output. The clouds predicted as 'utility' and 'not utility' are then saved separately to visualize the output of the model. A code for this procedure can be found in appendix 5.

To properly map a utility, a point cloud of a utility should be transformed into a polyline. A polyline is a line that consists of a sequence of connected line segments. This is done using a MATLAB code. The code can be found in appendix 6. The code finds the centroids of all point cloud parts of a utility line by taking the mean of all X, Y and Z coordinates of a part. The centroids of a utility line are then connected to a line and plotted in a figure together with the point cloud parts to show the result. The centroid coordinates are saved so they can be used in other programs to map polylines.

4. RESULTS

In this section, it is first explained what algorithms are chosen to be trained. This is followed by the results of the trained models. After this, point clouds are shown that are labelled wrong by the trained model. At last, a demonstration case is explained.

4.1 ALGORITHMS TRAINED

Different supervised machine learning algorithms have been identified as potential candidates. One is more suited to the problem than the other. Based on the criteria that are explained in section 2.4, a choice is made on which algorithms are trained to classify the point cloud dataset. Three algorithms are chosen to be trained. These are VoxNet, a shallow neural network and PointNet. These algorithms score best on the criteria defined in section 2.4. Random forests, support vector machines, PointNet ++ and the combination of UnitNet, FeatureNet and FinalNet are not trained.

4.2 TRAINING RESULTS

VoxNet, the shallow neural network and PointNet are all trained via MATLAB. The models are trained on the training dataset and tested on the test dataset that is created by following the steps explained in section 3.2. All three models are trained based on the following training options. These are the number of epochs, the batch size, the initial learning rate and the drop period. For the models that use 3D images as input, the grid size is a fifth training option. It is explained per model what the training options and results are.

4.2.1 VoxNET

The first network architecture that is trained is VoxNet. The script of VoxNet can be found in appendix 7. The point clouds are converted to 3D images of size 32x32x32. In this step the point clouds are voxelized. The next step is to define the layers of the VoxNet training algorithm. An example of the process including training layers can be seen in Figure 5 in section 2.3.2.

Next, the training options are defined. The model is trained for 10 different combinations of training options. The different combinations are chosen by trial and error. The combinations can be seen in Table 2. The eighth option has another option called fc1(256). Fc1 stands for fully connected layer one. The original size of this layer is 128 and it is increased to 256. A visualization of the fully connected layer can be seen on the right side of Figure 5.

Table 2, training options VoxNet

Nr trained model	1	2	3	4	5	6	7	8	9	10
Nr Epochs	60	28	60	60	150	60	60	150	60	200
Batch Size	32	32	64	256	256	32	32	32	32	32
Initial Learn Rate	0,01	0,01	0,01	0,01	0,01	0,01	0,0001	0,001	0,01	0,0001
Drop Period	333	333	666	2666	2666	666	333	333	166	333
Grid Size	32	64	32	32	32	32	32	32	32	32
Other options								fc1(256)		

All trained models are then tested on the test dataset to generate a confusion matrix and calculate precision, recall, quality, accuracy and F1 and training time (see Table 3). The green cells mark the best results per indicator (row). This is the highest value in the row, except for training time, which is the lowest value. The red cells mark the worst results per row.

Table 3, results VoxNet

Nr trained model	1	2	3	4	5	6	7	8	9	10
Precision	0,797	0,771	0,78	0,787	0,809	0,788	0,645	0,772	0,789	0,784
Recall	0,873	0,789	0,789	0,825	0,765	0,849	0,777	0,837	0,813	0,765
Quality	0,714	0,639	0,645	0,675	0,648	0,691	0,544	0,671	0,668	0,632
Accuracy	0,825	0,777	0,783	0,801	0,792	0,81	0,675	0,795	0,798	0,777
F1-score	0,833	0,78	0,784	0,806	0,786	0,817	0,705	0,803	0,801	0,774
Training time	08:35	01:09:23	07:07	17:08	16:53	07:43	08:05	18:53	08:09	26:55

The first trained model is the best option. This model has the highest recall, quality, accuracy and F1-score. It does not score the best on precision and training time, but the differences with the best options are only 0,012 and 1 minute and 28 seconds respectively. The differences between the best and the worst models for precision, recall, quality, accuracy and F1-score are all between 0,10 and 0.17. For training time, there is one outlier. the second trained model has a training time of more than one hour. This is because the grid size was increased from 32x32x32 to 64x64x64, which increases the size of the data eight times.

4.2.2 FULLY CONNECTED SHALLOW NEURAL NETWORK

The second network that is trained is a shallow neural network. It consists of a 3D image input layer, three fully connected layers and a fully connected output layer. The input layer is the same as the input layer of VoxNet, so the point clouds are first voxelized. The fully connected layers are of size 128, 64 and 32 respectively. The network architecture is shown in Figure 4 in section 2.3.1. The MATLAB script can be found in appendix 8.

Next, the training options are defined. The model is trained for 10 different combinations of training options. The different combinations are chosen by trial and error. The combinations can be seen in Table 4.

Table 4, training options fully connected network

Nr trained model	1	2	3	4	5	6	7	8	9	10
Nr Epochs	60	60	60	60	60	60	60	60	60	60
Batch Size	32	32	32	32	32	64	256	64	256	64
Initial Learn Rate	0,01	0,0001	0,0001	0,001	0,001	0,01	0,01	0,001	0,001	0,001
Drop Period	333	333	333	333	333	666	2666	666	2666	333
Grid Size	32	32	64	64	32	32	32	32	32	32

The different trained models are tested on the test dataset. This results in a confusion matrix from which the precision, recall, quality, accuracy and F1-score are derived. The training time of the different options is also measured. The results can be seen in Table 5.

Table 5, results fully connected shallow neural network

Nr trained model	1	2	3	4	5	6	7	8	9	10
Precision	0,821	0,818	0,85	0,836	0,804	0,804	0,809	0,811	0,833	0,81
Recall	0,771	0,759	0,681	0,645	0,765	0,789	0,789	0,777	0,783	0,795
Quality	0,66	0,649	0,608	0,572	0,645	0,662	0,665	0,658	0,677	0,67
Accuracy	0,801	0,795	0,78	0,759	0,789	0,798	0,801	0,798	0,813	0,804
F1-score	0,795	0,788	0,756	0,728	0,784	0,796	0,799	0,794	0,807	0,802
Training time	02:29	02:18	16:26	16:08	02:21	01:34	00:57	01:30	01:02	01:29

For the fully connected network, the ninth trained model is the best option. This model has the highest quality, accuracy and F1 score. It does not score the best on precision, recall and training time, but the differences with the best options are only 0,012, 0,016 and five seconds respectively. The differences between the best and the worst models for precision, recall, quality, accuracy and F1-score are all lower than 0,15. The training time of eight out of 10 models is lower than 2 ½ minutes. For two models, the training time is higher than 16 minutes. This is because of the grid size, which was increased from 32x32x32 to 64x64x64.

4.2.3 POINTNET

The last network that is trained is PointNet. It is used via MATLAB and the script can be found in appendix 9. The script is written with the help of an example created by the developer of MATLAB on how to use PointNet in MATLAB (Mathworks, 2023c). From the example, helper functions are used that define the layers of the network. The network architecture can be seen in Figure 6 in section 2.3.3. From this figure, only the classification network is used.

Next, the training options are defined. The model is trained for 10 different combinations of training options. The different combinations are chosen by trial and error. The combinations can be seen in Table 6.

Table 6, training options PointNet

Nr trained model	1	2	3	4	5	6	7	8	9	10
Nr Epochs	60	60	40	40	60	60	60	60	60	60
Batch Size	128	128	256	64	64	128	32	128	256	128
Initial Learn Rate	0,002	0,001	0,002	0,002	0,001	0,001	0,002	0,0001	0,001	0,001
Drop Period	15	15	15	15	15	300	15	15	15	30

The different trained models are tested on the test dataset. This results in a confusion matrix from which the precision, recall, quality, accuracy and F1-score are derived. The training time of the different options is also measured. The results can be seen in Table 7.

Table 7, results PointNet

Nr trained model	1	2	3	4	5	6	7	8	9	10
Precision	0,684	0,801	0,614	0,734	0,764	0,673	0,754	0,755	0,626	0,684
Recall	0,795	0,777	0,892	0,849	0,759	0,867	0,777	0,723	0,855	0,861
Quality	0,581	0,652	0,571	0,65	0,615	0,61	0,62	0,585	0,566	0,616
Accuracy	0,714	0,792	0,666	0,771	0,762	0,723	0,762	0,744	0,672	0,732
F1-score	0,735	0,789	0,727	0,788	0,761	0,758	0,766	0,738	0,723	0,763
Training time	18:20	18:00	14:04	10:39	16:45	18:25	17:28	18:37	21:13	18:17

For PointNet, the second trained model is the best option. The model has the highest values for precision, quality, accuracy and F1-score. It does not score the best on recall and training time. The differences between the best options are 0,114 and 7 minutes and 21 seconds respectively. Moreover, the scores on recall and training both lie under the average of all options. The differences between the best and the worst models for precision, recall, quality, accuracy and F1-score are all between 0,06 and 0.19. the training times lie between 10 and 22 minutes.

4.2.4 BEST MODEL

In Table 8, a comparison of the best-scoring models of VoxNet, the fully connected neural network and PointNet is shown.

Table 8, comparison of best options for the models

	VoxNet(1)	Shallow NN(9)	PointNet(2)
Precision	0,796703297	0,833333333	0,801242236
Recall	0,873493976	0,78313253	0,777108434
Quality	0,714285714	0,677083333	0,651515152
Accuracy	0,825301205	0,813253012	0,792168675
F1-score	0,833333333	0,807453416	0,788990826
Training time	00:08:35	00:01:02	00:18:00

From this table, it can be seen that the first trained VoxNet model is the best model for recall, quality, accuracy and F1-score. The ninth fully connected neural network is the best option for a low training time. This option also has the highest precision.

4.3 POINT CLOUDS LABELLED AS FALSE POSITIVE OR FALSE NEGATIVE

The scores from the diagnostics tests of the machine learning algorithms show that the accuracy of the models reaches a plateau around 80%. This means that around 20% of the test dataset is labelled wrong by the models. A closer visual inspection shows that the point clouds that are labelled wrong are often similar. Examples of point clouds that are labelled as a utility by the models but are not a utility are shown in Figure 24.



Figure 24, point clouds falsely classified as utility (FP)

In Figure 25, point clouds are shown that are labelled as not utility by the models but are a utility.



Figure 25, point clouds falsely classified as not utility (FN)

4.4 DEMONSTRATION CASE

In this section, the results of the demonstration case are shown. First, it is shown how the point cloud for the demonstration case is cut into parts using two methods. After that, it is shown how the parts are classified. At last, it is shown how the parts can be converted to polylines that can be used for mapping.

To let the trained model identify utilities in the point cloud for the demonstration case, the point cloud should first be cut into smaller parts. This is done semi-automatically via CloudCompare and

automatically via MATLAB. To cut the point cloud in CloudCompare, the CSF algorithm is used to remove points that represent the ground (Zhang et al., 2016). The result can be seen in Figure 26.

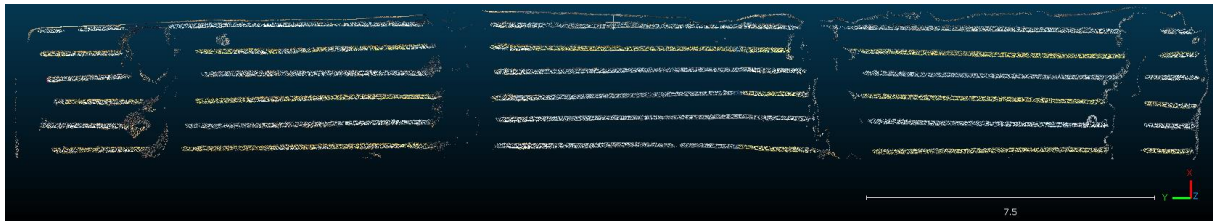


Figure 26, point cloud after applying the CSF algorithm

As can be seen in Figure 26, there are still points present that do not represent a utility. These points are cut out manually using the segmentation tool. The result can be seen in Figure 27.

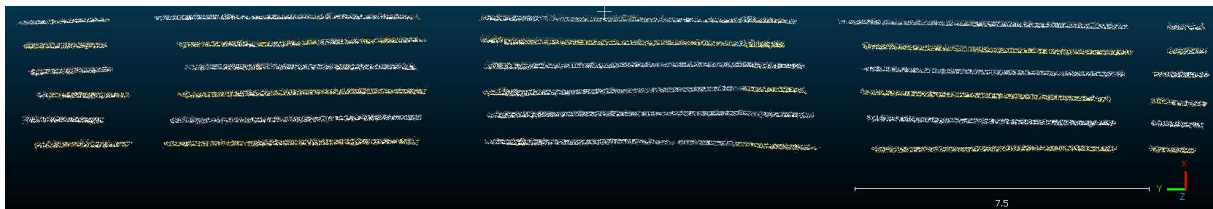


Figure 27, point cloud after cutting out remaining ground points

Using the 'label connected components' tool, the pipes present in the point cloud are cut into parts (CloudCompare, 2015). The result can be seen in Figure 28.

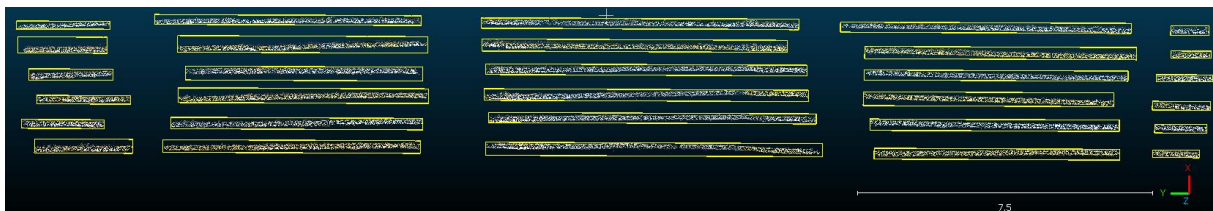


Figure 28, point cloud cut in parts using the 'label connected components' tool

Most parts are too big to properly classify them. Therefore, these parts are cut into smaller pieces manually. The result can be seen in Figure 29.

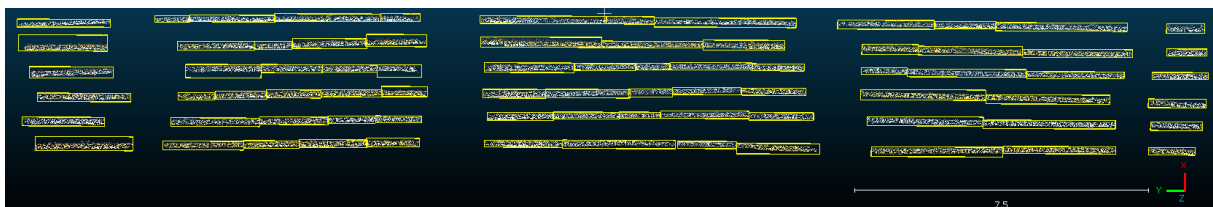


Figure 29, point cloud parts after manual cutting

The test point cloud is also cut into pieces automatically using a MATLAB code. the code cuts the whole point cloud including ground points in parts. The result can be seen in Figure 30.

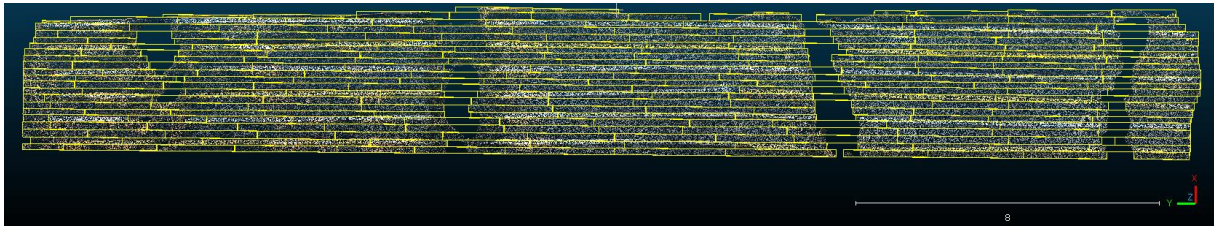


Figure 30, point cloud cut into parts using a MATLAB code

The parts cut by CloudCompare and a MATLAB code shown in Figure 29 and Figure 30 respectively are classified by the trained VoxNet model. The model classifies each part as 'utility' or 'not utility'. In Figure 31, the parts cut with CloudCompare classified as 'utility' are shown. It can be seen that most parts present in Figure 29 are classified as utility.



Figure 31, point cloud parts cut via CloudCompare classified as 'utility'

In Figure 32, the parts cut with CloudCompare classified as 'not utility' are shown. It can be seen that only 5 parts are classified as 'not utility'. The accuracy is therefore 93,5%.

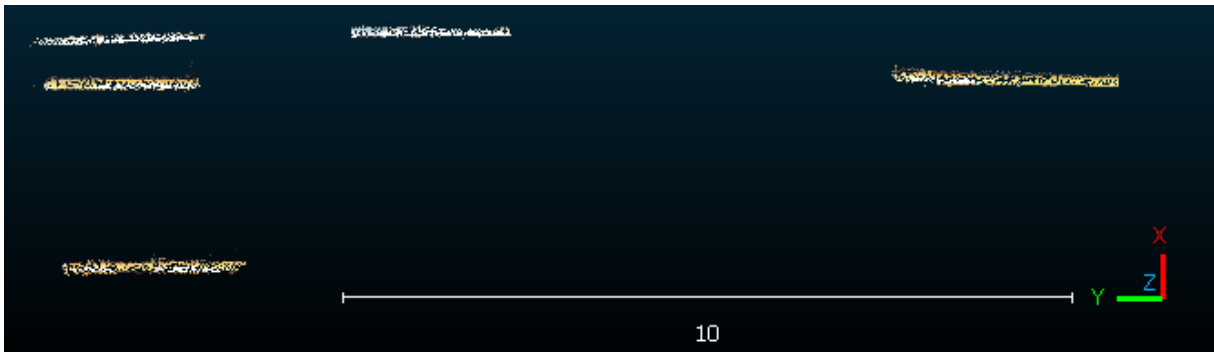


Figure 32, point cloud parts cut via CloudCompare classified as 'not utility'

In Figure 33, the parts cut with MATLAB classified as 'utility' are shown. It can be seen that it is a mix of point clouds that represent a utility and point clouds that contain ground points.

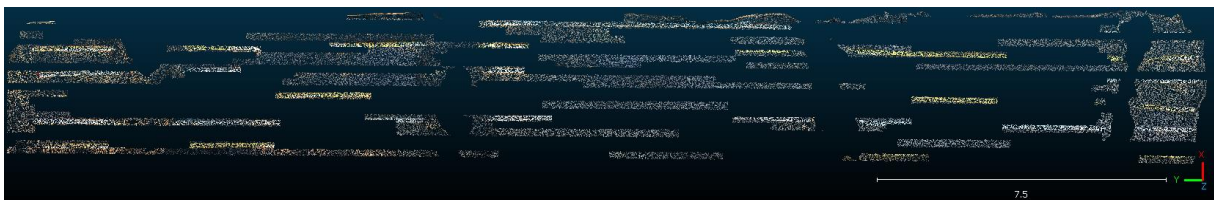


Figure 33, point cloud parts cut using a MATLAB code classified as 'utility'

In Figure 34, the parts cut with MATLAB classified as 'not utility' are shown. Again, it can be seen that it is a mix of point clouds that represent a utility and point clouds that contain ground points.

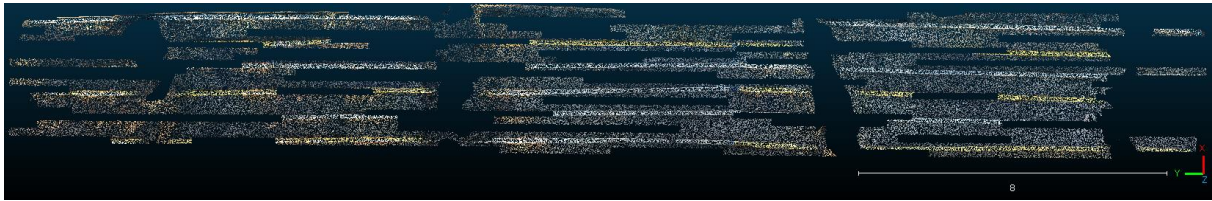


Figure 34, point cloud parts cut using a MATLAB code classified as 'not utility'

The last step is to convert the point clouds to polylines that can be used for mapping. The centroids of the point cloud parts depicted in Figure 31 are first calculated. The centroids per pipe are then connected. In Figure 35 and Figure 36, the polylines can be seen together with the point clouds of the pipes. It can be seen that the polyline roughly follows the centre of the pipes.

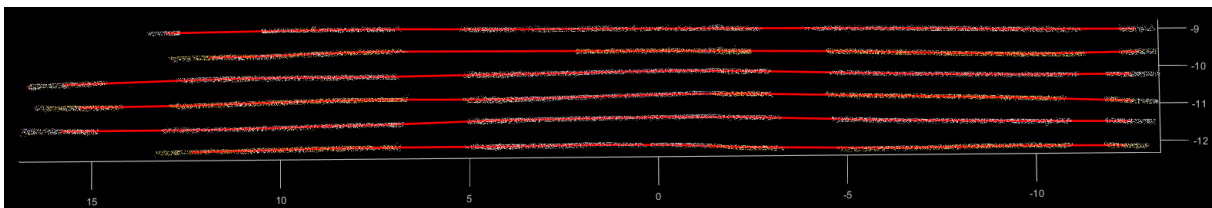


Figure 35, point cloud parts with polylines from above

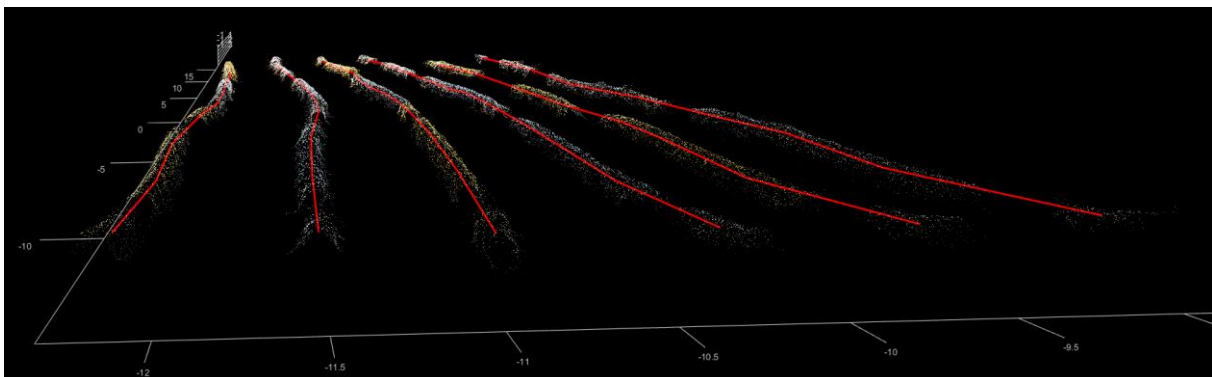


Figure 36, point cloud parts with polylines from the side

5. DISCUSSION

The objective of this research was to compare and use machine learning algorithms for automatically recognizing, and retrieving underground utilities that are present in 3d point clouds of open trenches and convert the data into geometric shapes.

Key findings

The first step in reaching the objective was to identify machine learning algorithms. Different supervised machine learning algorithms were compared for classifying point cloud data. From this comparison, VoxNet, a shallow neural network and PointNet were identified as the best options.

In total, the three different algorithms have been trained on 10 different combinations of training options. From each model, the best scoring was selected. In Table 8 in section 4.2.4, a comparison of the best-scoring models of VoxNet, the fully connected neural network and PointNet is shown. From this table, it can be seen that the VoxNet model is the best for this problem.

Interpretations

The comparison of different machine learning algorithms offered valuable insights into the suitability of various approaches. The decision on which algorithms to train leans on various factors such as the ability to consume point cloud data, whether it is open-source or not, dataset size and via what program it can be used. These criteria were considered most important as the algorithms must perform well according to these standards to be suitable for usage. Other factors such as complexity, training time and ease of use were considered less important because they ultimately do not affect whether an algorithm can be used. According to these criteria, VoxNet, a shallow neural network and PointNet were identified as the best options.

In total, the three different algorithms have been trained on 10 different combinations of training options. For VoxNet, the differences between the best and the worst models for precision, recall, quality, accuracy and F1-score are all between 0,10 and 0,17. For the fully connected network, the differences between the best and the worst models are all lower than 0,15. For PointNet, the differences between the best and the worst models are all between 0,06 and 0,19. VoxNet and the shallow neural network performed worse when the grid size was increased and when the initial learning rate was decreased. PointNet performed worse when the batch size was increased.

It can be said the differences between the best and worst models are not significantly high, so most training options do not influence the quality of the models significantly. The training time for VoxNet and the shallow neural network however contains three outliers. This is because the grid size was increased from 32x32x32 to 64x64x64, which increases the size of the data eight times. This training option should not be changed if the training time should stay low. To further improve the quality of the models, an option is to find other training options that have more influence on the performance.

By comparing the three trained models it was found that VoxNet was the best model. This can be seen in Table 8, comparison of best options for the models. In this table, it can be seen that the differences in precision, quality accuracy and F1 score between the best models for VoxNet, a shallow neural network and PointNet are all lower than 0,06. From these factors, it can be said that there is almost no difference between the models. The differences for recall are higher. The recall for the best VoxNet model is almost 0,10 higher than the recall for the shallow neural network and PointNet.

The biggest differences are in the training time. The shallow neural network has the best training time of 1 minute and 2 seconds. The training time of VoxNet is more than 7 minutes higher and the training time of PointNet is almost 17 minutes higher. If a low training time is wished the shallow neural network

should be chosen. However, VoxNet was chosen above the shallow neural network. This is because training only has to be done once. After training, a model can be used on new data. Processing new data with a trained model does not require much time.

Limitations

During the process of writing this report, limitations have been found. The algorithms have been trained on a new dataset. During the process improvements in efficiency have been identified. The point clouds were cut into smaller parts mostly via MATLAB. This process took a significant amount of time. This process was improved by cutting the parts via CloudCompare. The dataset has a limited size. In total, the algorithms were trained on 772 different point clouds and tested on 332 clouds.

The scores from the diagnostics tests of the machine learning algorithms show that the accuracy of the models reaches a plateau around 80%. This means that around 20% of the test dataset is labelled wrong by the models. The point clouds that are labelled wrong by the different models are often the same. This is probably because of the limited size of the dataset. To further increase the quality of the models, the dataset should be increased in size.

As a final part of this study, the best model was used to assess a demonstration case. A new point cloud was cut into parts using two methods: via CloudCompare and a MATLAB code. The parts that were cut with CloudCompare have a similar shape as the training data. Therefore, almost all parts were classified correctly. The parts cut with the MATLAB code all have a rectangular shape. Sometimes it contains only points representing a utility or the ground and sometimes it is a mix of both. Because of the shape and the content of the parts, the trained model does not predict labels well. To improve the performance of a trained model on new data, the new data thus has to be of roughly the same shape as the training data.

The final part of the demonstration case was to convert utilities in a point cloud to geometrical shapes. The centroids of point cloud parts were used to compute a polyline. The centroids lie 1 to 4 meters apart from each other, limiting the detail of mapping a utility. To improve this, more points should be used to create a polyline. Another option is to convert the point cloud to other shapes with more detail. Other methods should be investigated to reach this objective.

Implications

This report can be used as a benchmark for further research in identifying utilities in point clouds. The algorithms trained in this research can be developed to improve the quality. Other algorithms identified in this research can also be trained for the problem. By further improving the algorithms trained in this research, creating point clouds from utility trenches can be used as a new method for accurately mapping utilities.

6. CONCLUSION

The research objective was to compare and use machine learning algorithms for automatically recognizing, and retrieving underground utilities that are present in 3d point clouds of open trenches and convert the data into geometric shapes. From the objective, one main research question was determined:

How can different machine learning algorithms be used to retrieve utilities from point clouds of open trenches?

This question can be answered by four sub-questions:

What types of machine learning algorithms are most useful?

Through the comparison of various supervised machine learning algorithms, significant insights were gained regarding their suitability based on criteria such as the ability to consume point cloud data, open-source availability, the complexity of the algorithm, training time, the ability to deal with a large dataset, the software that can be used and the ease of use. based on these criteria, three algorithms were identified to be used for training. These are VoxNet, a shallow neural network and PointNet.

How can the machine learning algorithms be used?

The identified algorithms can be used via multiple programming languages. For this thesis, MATLAB was chosen since there is clear documentation available about using the algorithms. Moreover, there was no experience in using other programming languages and time was limited to learn how to use other languages.

How should the point cloud data be pre-processed?

The identified algorithms have been trained on a newly created dataset. The data consists of point clouds labelled as 'utility' and 'not utility' to classify all points in a point cloud. The machine learning algorithms show that the accuracy of the models reaches a plateau around 80% on classifying test data. This means that around 20% of the test dataset is labelled wrong by the models. The point clouds that are labelled wrong by the different models are often the same. This is because of the limited size of the dataset.

As a final part of this study, the best model was used to assess a test case. A new point cloud was cut into parts using two methods. One method ensured that the point clouds had a similar shape as the training data. 93,5% of the parts cut with the first method were labelled right. The other method cut the point clouds into parts of rectangular shape. This method did not work out. Most parts were labelled wrong. Therefore, it can be concluded that new data has to be of similar shape as the training data.

How can the quality of a model be assessed?

The trained models were assessed on 6 criteria. These are precision, recall, quality, accuracy, F1-score and training time. the VoxNet model came out as the best option. The differences for precision, quality accuracy and F1 score between the best models for VoxNet, a shallow neural network and PointNet are all lower than 0,06. From these factors, it can be said that there is almost no difference between the models.

The biggest differences are in the training time. The shallow neural network has the best training time of 1 minute and 2 seconds. The training time of VoxNet is more than 7 minutes higher and the training time of PointNet is almost 17 minutes higher. However, VoxNet was chosen above the shallow neural network. This is because training only has to be done once.

This study contributes to the existing literature on the application of machine learning algorithms in underground utility recognition from point cloud data. By comparing and evaluating various supervised machine learning algorithms, including VoxNet, a shallow neural network, and PointNet, the research provides valuable insights into their suitability for utility recognition tasks. Additionally, the study addresses aspects of data preprocessing, emphasizing the importance of dataset quality.

Overall, this study underlines the effectiveness of machine learning algorithms in utility recognition from point cloud data, highlighting opportunities for improving model robustness and applicability in real-world scenarios.

7. RECOMMENDATIONS FOR FURTHER RESEARCH

Based on the conclusions drawn from this research, recommendations for future research or practical implementation can be made.

Based on the comparison conducted in this study, it is recommended to consider VoxNet, a shallow neural network, or PointNet for utility recognition tasks from point cloud data. These algorithms showed promising results and can be further explored for their effectiveness in different scenarios.

The choice for these algorithms was mainly made because they are available in MATLAB. The programming language was chosen due to clear documentation and time constraints. Future research can explore other programming languages such as Python for implementing the identified algorithms. Other algorithms not chosen in this study can also be investigated via other programming languages.

To improve the accuracy of machine learning models, it is essential to ensure that the dataset represents the variability of real-world scenarios. The training dataset now only consists of 772 samples and the test set consists of 332 samples. Enlarging the dataset with diverse samples can enhance model performance. The size depends on the complexity of the algorithm and the complexity of the problem.

This study only looked at classifying point cloud parts into two classes: 'utility' and 'not utility'. To further improve the quality of the classification models, it is advised to use more than two classes. It is important to classify different utilities, so this class can be split into more classes representing the type of cable or pipe.

Instead of classification, segmentation can also be used to identify utilities in point clouds. It is recommended to also look into point cloud segmentation. When utilities in point clouds are classified, they should first be cut into parts representing 'utility' or 'not utility'. If a point cloud is segmented, it can directly be used as input and the algorithm segments the point cloud into the classes 'utility' and 'not utility'.

The last step in this study was to execute a demonstration case. In this case, the point clouds of the utilities were converted into polylines by connecting the centroids of different point cloud parts. The parts all lay between 0,5 and 4 meters apart from each other. While this gives an indication of where a utility is positioned, it is advised to look at other ways that can convert point clouds into shapes with more detail.

By incorporating these recommendations, future research efforts can further advance the field of utility recognition from point cloud data and contribute to the development of more accurate, efficient, and practical solutions for real-world applications.

8. BIBLIOGRAPHY

- Admiraal, H., & Cornaro, A. (2016). Why underground space should be included in urban planning policy – And how this will enhance an urban underground future. In *Tunnelling and Underground Space Technology volume 55* (pp. 214-220). ScienceDirect.
- Anwar, Z. (2016). *Is there an ideal ratio between a training set and validation set? Which trade-off would you suggest?* <https://www.researchgate.net/post/Is-there-an-ideal-ratio-between-a-training-set-and-validation-set-Which-trade-off-would-you-suggest>
- Bajaj, A. (2023). *Performance Metrics in Machine Learning*. neptune. <https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide>
- Basener, W., & Basener, A. (2017). *Classification and identification of small objects in complex urban-forested LIDAR data using machine learning*. <https://doi.org/10.1117/12.2264641>
- Blewitt, G. (1997). Basics of the GPS Technique: Observation Equations. In *Geodetic Applications of GPS*. Swedish Land Survey.
- CloudCompare. (2015). *Label Connected Components*. https://www.cloudcompare.org/doc/wiki/index.php/Label_Connected_Components
- CloudCompare. (2016). *Normals\Compute*. <https://cloudcompare.org/doc/wiki/index.php?title=Normals%5CCompute>
- CloudCompare. (2018). *Interactive Segmentation Tool*. https://www.cloudcompare.org/doc/wiki/index.php/Interactive_Segmentation_Tool
- CloudCompare. (2024). *Presentation*. <https://www.danielgm.net/cc/>
- Desai, U. (2023). *Mastering Random Forest Algorithm: A Step-by-Step Learning Guide*. Medium. <https://utsavdesai26.medium.com/mastering-random-forest-algorithm-a-step-by-step-learning-guide-f7abf2420a55>
- Dongare, A., Kharde, R., & Kachare, A. D. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1), 189-194.
- Hardesty, L. (2017). *Explained: Neural networks*. Massachusetts institute of technology. <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>
- Inoue, M. M. M. (2021). Visualization of 3D cable between utility poles obtained from laser scanning point clouds: a case study. In *SN Applied Sciences*. Springer International Publishing.
- Linder, W. (2016). Chapter 1 Introduction. In *Digital Photogrammetry A Practical Course* (pp. 1-15). Springer.
- Mathworks. (2021). *Train Classification Network to Classify Object in 3-D Point Cloud*. https://nl.mathworks.com/help/releases/R2021b/vision/ug/train-classification-network-to-classify-object-in-3-d-point-cloud.html#mw_rtc_TrainNetwork3DPointCloudDataExample_D14E8BA8
- Mathworks. (2023a). *Choose Classifier Options*. Mathworks. <https://nl.mathworks.com/help/stats/choose-a-classifier.html>
- Mathworks. (2023b). *Get Started with the Lidar Labeler*. <https://nl.mathworks.com/help/lidar/ug/lidar-labeler-get-started.html>
- Mathworks. (2023c). *Point Cloud Classification Using PointNet Deep Learning*. Mathworks. <https://nl.mathworks.com/help/vision/ug/point-cloud-classification-using-pointnet-deep-learning.html>
- Mathworks. (2024). *Get Started with Lidar Viewer*. <https://nl.mathworks.com/help/lidar/ug/get-started-lidar-viewer.html>
- Maturana, D., & Scherer, S. (2015, 28 Sept.-2 Oct. 2015). VoxNet: A 3D Convolutional Neural Network for real-time object recognition. 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),
- Mcmullen, M. (2019). *What are Features in Machine Learning and Why it is Important?* Medium. <https://cogitotech.medium.com/what-are-features-in-machine-learning-and-why-it-is-important-e72f9905b54d>

- Mirzaei, K., & et al. (2022). 3D point cloud data processing with machine learning for construction and infrastructure applications: A comprehensive review. *Advanced Engineering Informatics*, 51, 101501.
- NOAA. (2023). *What is Lidar?* <https://oceanservice.noaa.gov/facts/lidar.html>
- Qi, C. R. (2018). *charlesq34/pointnet2*. <https://github.com/charlesq34/pointnet2>
- Qi, C. R. (2022). *charlesq34/pointnet*. <https://github.com/charlesq34/pointnet>
- Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*.
- Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). *PointNet++: Deep Hierarchical Feature Learning on point sets in a metric space*.
- Rakhecha, A. (2019). *understanding learning rate. towards data science*. <https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de>
- Rotterdam, G. (2022). *eisen en voorwaarden inmeten en as-built tekening*. In *handboek beheer ondergrond* (pp. 12). https://www.cob.nl/hbrinteractief/gemeentebeleid/0599_Rotterdam/Rotterdam%20Bijlage-IV-Eisen-en-Voorwaarden-Inmeten-en-As-Built-Tekening.pdf
- Siers. (2023). *Siers Groep Oldenzaal B.V.* <https://www.siersgroep.nl/>
- Singh, A. (2020). *All Machine Learning Models Explained in 5 Minutes | Types of ML Models Basics*. <https://www.youtube.com/watch?v=yN7ypxC7838>
- Taselaar, F. (2009a). 1.2 Actoren en belangen. In *Inleiding Kabels & Leidingen* (pp. 27-31). Nederlands kenniscentrum voor ondergronds bouwen en ondergronds ruimtegebruik.
- Taselaar, F. (2009b). *Maatschappelijk belang*. In *inleiding kabels en leidingen* (pp. 16-26). Nederlands kenniscentrum voor ondergronds bouwen en ondergronds ruimtegebruik.
- Wiedemann, C., Heipke, C., Mayer, H., & Jamet, O. (1998). *Empirical Evaluation Of Automatically Extracted Road Axes*.
- Xu, Y., Tong, X., & Stilla, U. (2021). *Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry*. *Automation in Construction*, 126, 103675. <https://doi.org/https://doi.org/10.1016/j.autcon.2021.103675>
- Xu, Z., Kang, R., & Li, H. (2022). *Feature-Based Deep Learning Classification for Pipeline Component Extraction from 3D Point Clouds*. *Buildings*, 12(7), 968. <https://www.mdpi.com/2075-5309/12/7/968>
- Yang, L., Zhang, F., Yang, F., Qian, P., Wang, Q., Wu, Y., & Wang, K. (2023). *Generating Topologically Consistent BIM Models of Utility Tunnels from Point Clouds*. In *Sensors* (pp. 6503). MDPI AG.
- Yehoshua, D. R. (2023). *Random Forests*. Medium. <https://medium.com/@roiyehe/random-forests-98892261dc49>
- Zarrinpanjeh, N. (2023). *3D modeling for City Digital twins based on geospatial information*.
- Zhang, W., Qi, J., Peng, W., Wang, H., Xie, D., Wang, X., & Yan, G. (2016). *An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation*. *Remote Sensing*, 8, 501. <https://doi.org/10.3390/rs8060501>

9. APPENDICES

9.1 APPENDIX 1, SCRIPT FOR MERGING DATA

```
clear, clc

%labeledcloud contains coordinates and classes
labelledcloud =
load("VoxelLabelData\pointcloudSequence_PointCloudSequence_Apr14at10_15AM_poly5_Edit_Label_1.mat");

%unlabeledcloud contains coordinates and rgb data
unlabelledcloud = pcread("Point Cloud Sequence_Apr14at10-15AM-poly5_Edit\point_cloud_rgb - Cloud.ply");

%cloudRGBXYZclass contains coordinates, rgb data and classes
cloudRGBXYZclass(:,[1 2 3]) = unlabelledcloud.Location;
cloudRGBXYZclass(:,[4 5 6]) = unlabelledcloud.Color;
cloudRGBXYZclass(:,7) = labelledcloud.L(:,4);

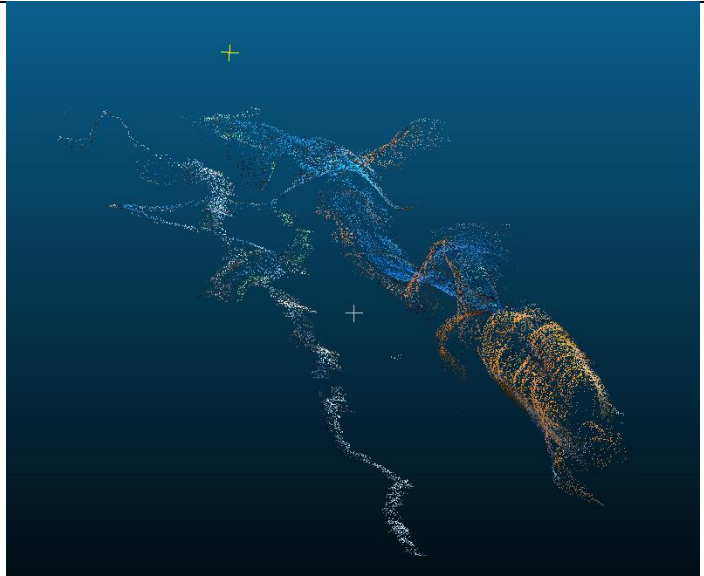
%cloudwithnormals contains coordinates, rgb data and normals
cloudwithnormals = load("cloudwithnormals.txt");

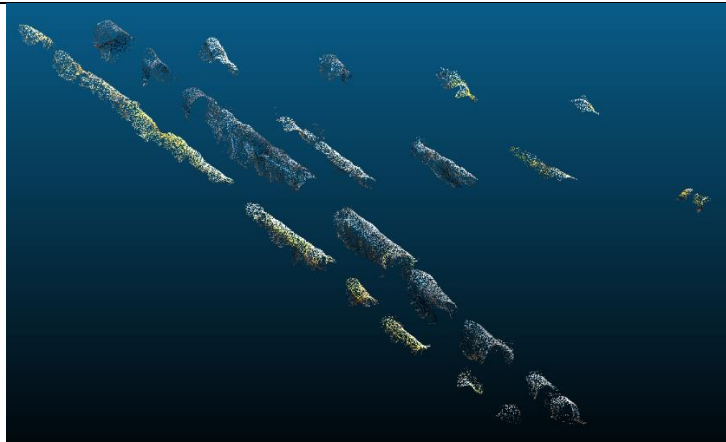
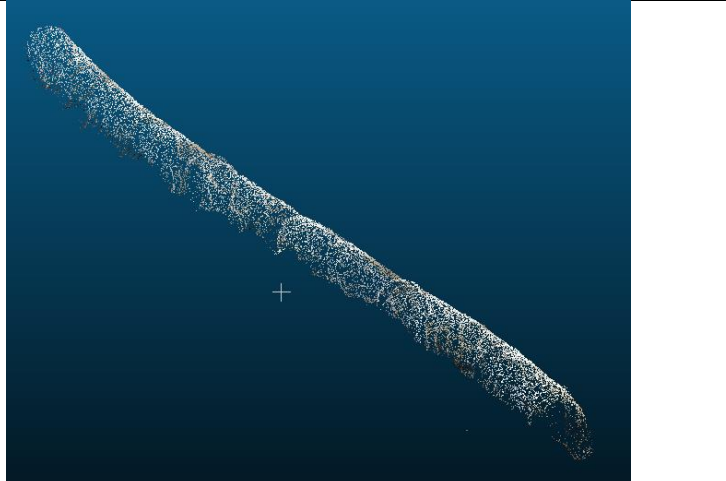
%cloudRGBXYZclassnormals contains coordinates, rgb data, normals and
%classes
cloudRGBXYZclassnormals(:,[1 2 3 4 5 6]) = cloudRGBXYZclass(:,[1 2 3 4 5 6]);
cloudRGBXYZclassnormals(:,[7 8 9]) = cloudwithnormals(:,[7 8 9]);
cloudRGBXYZclassnormals(:,[10]) = cloudRGBXYZclass(:,[7]);
writematrix(cloudRGBXYZclassnormals, 'cloudRGBXYZclassnormals.txt', 'Delimiter', 'tab');


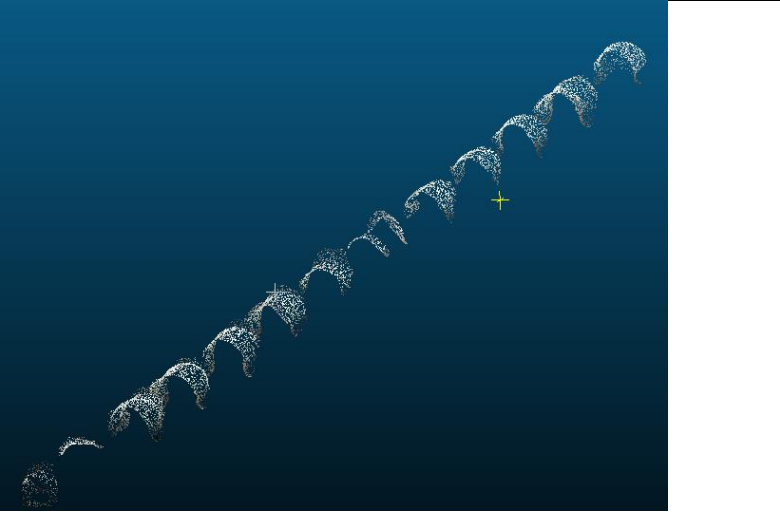
%nonZeroIndices finds the points that are not labelled as utility
nonZeroIndices = find(cloudRGBXYZclassnormals(:,10) ~= 0);



%onlyutilitiescloud is a point cloud with only the utilities present
onlyutilitiescloud = cloudRGBXYZclassnormals(nonZeroIndices,1:10);
writematrix(onlyutilitiescloud, 'onlyutilitiescloud.txt', 'Delimiter', 'tab');
```

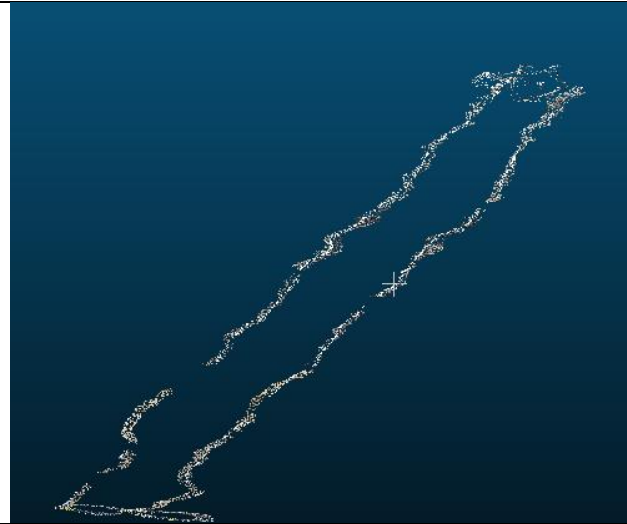
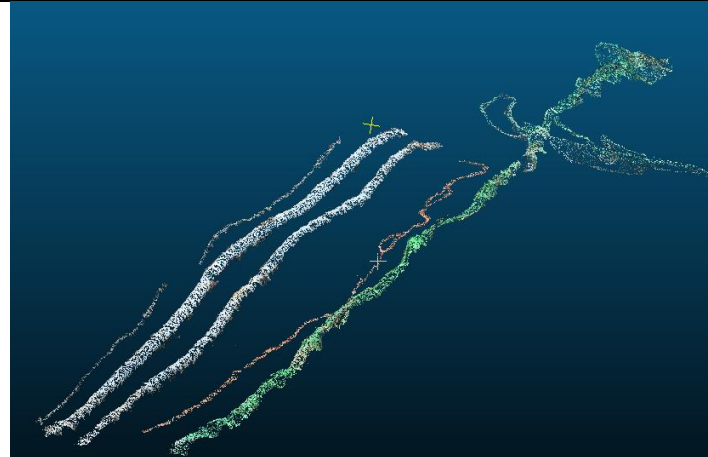
9.2 APPENDIX 2, DESCRIPTION DATASET

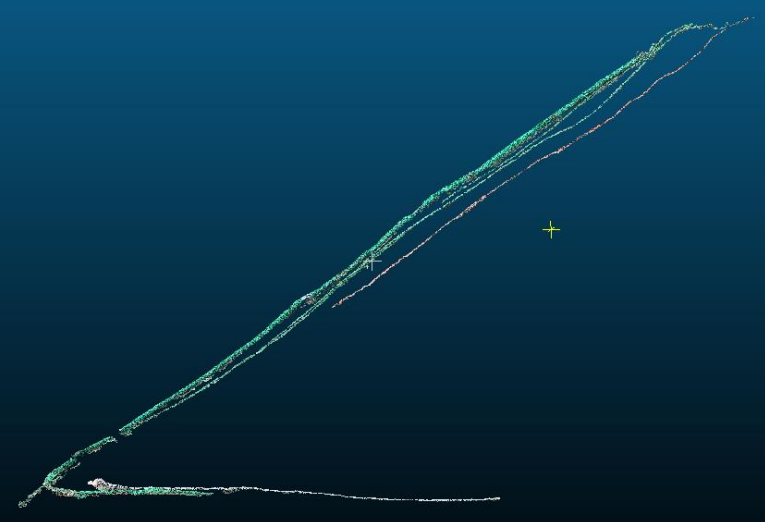
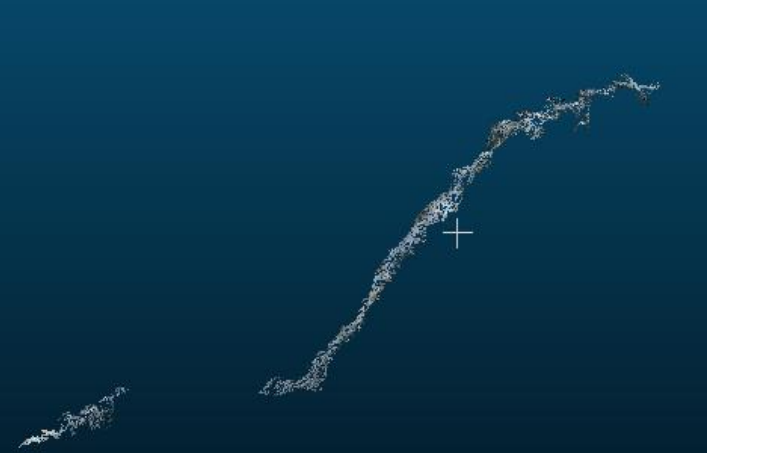
Point cloud	Screenshot of utilities filtered out	Number of points	Dividing in parts
Apr 14 at 10-15		<p>Original Cloud: 2.554.160</p> <p>After preprocessing: 434.230</p> <p>Only utilities: 28.894</p>	<p>Blue: 14 Orange: 17 White: 17</p> <p>Total: 48</p> <p>Not utility: 30</p>



<p>Dec 4 at 2-37</p>		<p>Original Cloud: 4.268.580</p> <p>After preprocessing: 724.831</p> <p>Only utilities: 37.606</p>	<p>Pipe 1: 10 Pipe 2 : 9 Rest: 9</p> <p>Total: 28</p> <p>Not utility: 22</p>
<p>Dec 4 at 2-58</p>		<p>Original Cloud: 1.828.020</p> <p>After preprocessing: 117.535</p> <p>Only utilities: 17.972</p>	<p>Total: 18</p> <p>Not utility: 16</p>

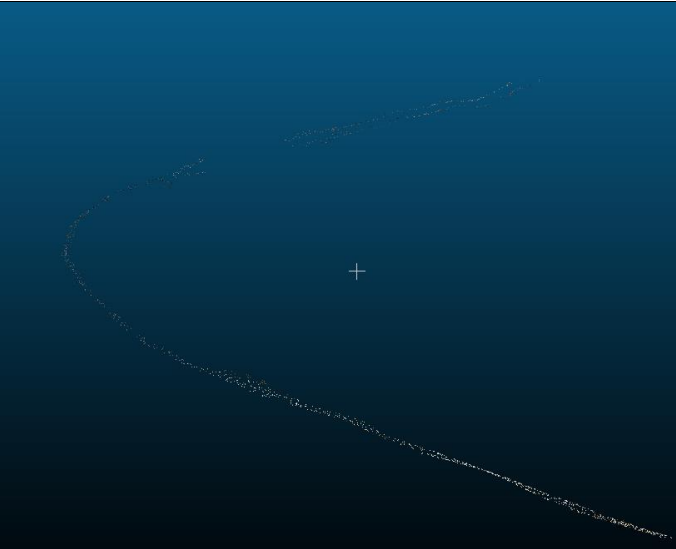
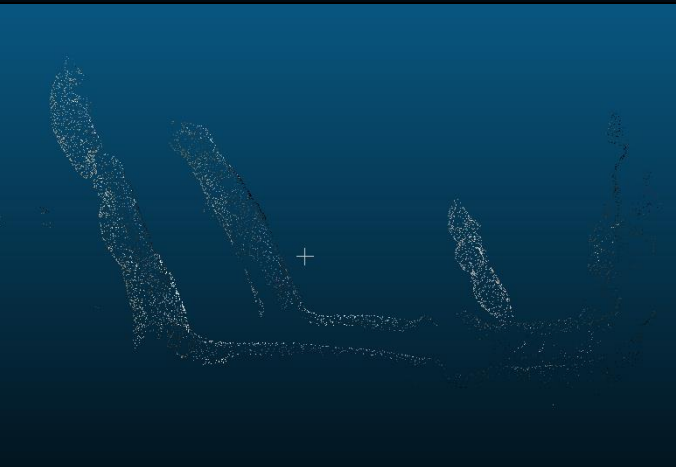
<p>Dec 4 at 11-10</p>		<p>Original Cloud: 2.834.040</p> <p>After preprocessing: 230.832</p> <p>Only utilities: 19.217</p>	<p>Pipe 1: 13 Pipe 2: 10</p> <p>Total: 23</p> <p>Not utility: 15</p>
<p>Dec 5 at 2-48</p>		<p>Original Cloud: 2.682.000</p> <p>After preprocessing: 226.306</p> <p>Only utilities: 10.042</p>	<p>Total: 10</p> <p>Not utility: 18</p>



<p>Dec 5 at 12-08</p>		<p>Original Cloud: 3.730.500</p> <p>After preprocessing: 633.506</p> <p>Only utilities: 10.399</p>	<p>Cable 1: 10 Cable 2: 11 Cable 3: 9</p> <p>Total: 30</p> <p>Not utility: 16</p>
<p>Dec 6 at 4-01</p>		<p>Original Cloud: 2.650.900</p> <p>After preprocessing: 278.461</p> <p>Only utilities: 49.865</p>	<p>Pipe 1: 12 Pipe 2: 13</p> <p>Total: 25</p> <p>Not utility: 23</p>

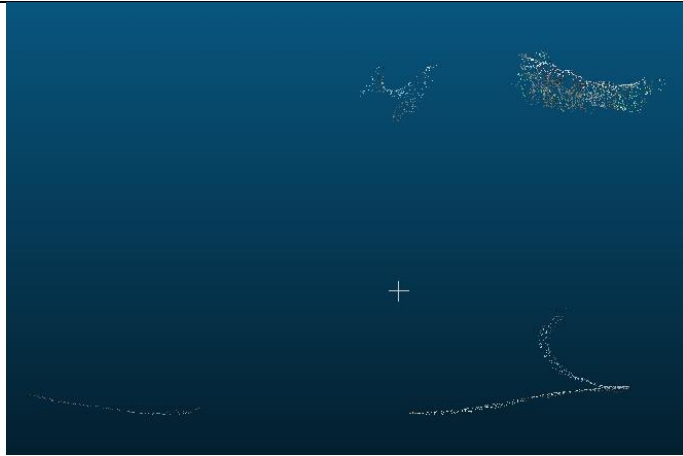

<p>Dec 7 at 9-50</p>			<p>Original Cloud: 3.393.860</p> <p>After preprocessing: 253.581</p> <p>Only utilities: 7.022</p>	<p>Cable 1: 11 Cable 2: 11</p> <p>Total: 22</p> <p>Not utility: 17</p>
<p>Dec 14 at 12-10</p>			<p>Original Cloud: 2.754.600</p> <p>After preprocessing: 467.741</p> <p>Only utilities: 45.759</p>	<p>Cable 1: 15 Cable 2: 7 Cable 3: 9 Cable 4: 10 Cable 5: 9</p> <p>Total: 50</p> <p>Not utility: 18</p>


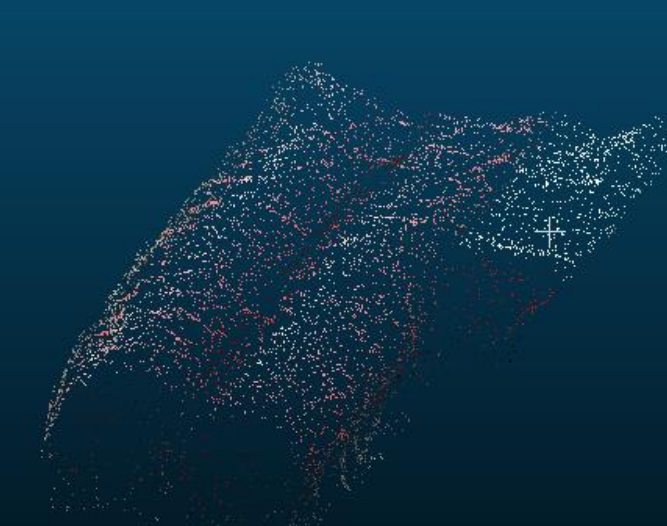
<p>Dec 14 at 12-13</p>		<p>Original Cloud: 2.101.640</p> <p>After preprocessing: 356.965</p> <p>Only utilities: 26.347</p>	<p>Cable 1: 8 Cable 2: 4 Cable 3: 14 Cable 4: 16</p> <p>Total: 42</p> <p>Not utility: 22</p>
<p>Jun 20 at 8-14</p>		<p>Original Cloud: 6.769.540</p> <p>After preprocessing: 830.783</p> <p>Only utilities: 5.820</p>	<p>Total: 6</p> <p>Not utility: 29</p>

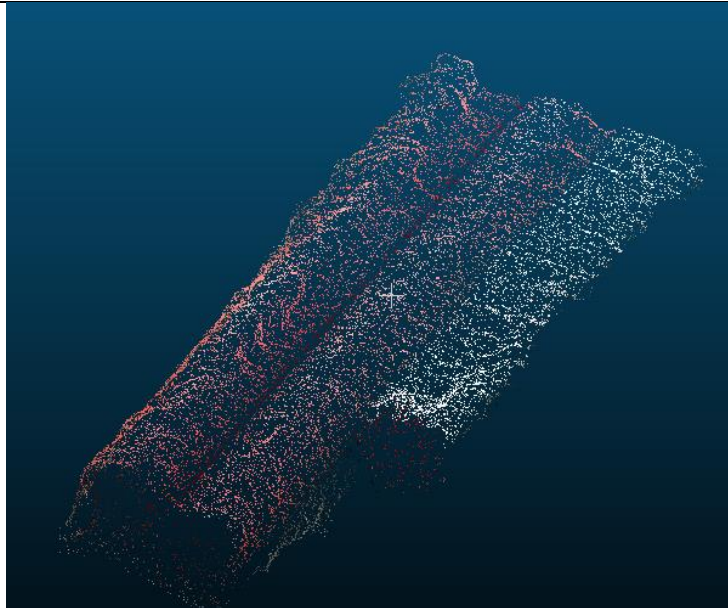

<p>Jun 20 at 8-20</p>		<p>Original Cloud: 8.257.080</p> <p>After preprocessing: 795.241</p> <p>Only utilities: 17.706</p>	<p>Total: 14</p> <p>Not utility: 23</p>
<p>Jun 20 at 8-26</p>		<p>Original Cloud: 2.034.680</p> <p>After preprocessing: 125.952</p> <p>Only utilities: 1.448</p>	<p>Total: 8</p> <p>Not utility: 20</p>

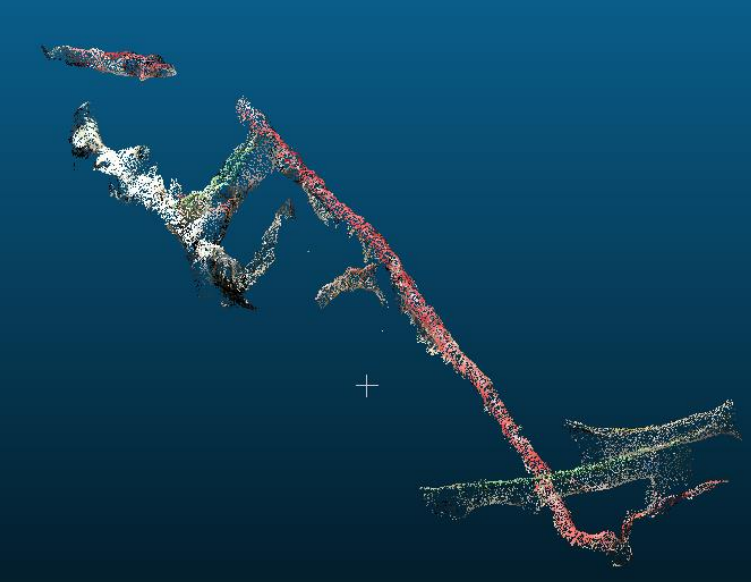

<p>Jun 20 at 9-05</p>		<p>Original Cloud: 1.782.960</p> <p>After preprocessing: 93.135</p> <p>Only utilities: 1.564</p>	<p>Total: 11</p> <p>Not utility: 22</p>
<p>Jun 20 at 10-58</p>		<p>Original Cloud: 2.758.860</p> <p>After preprocessing: 204.335</p> <p>Only utilities: 7.109</p>	<p>Total: 10</p> <p>Not utility: 24</p>

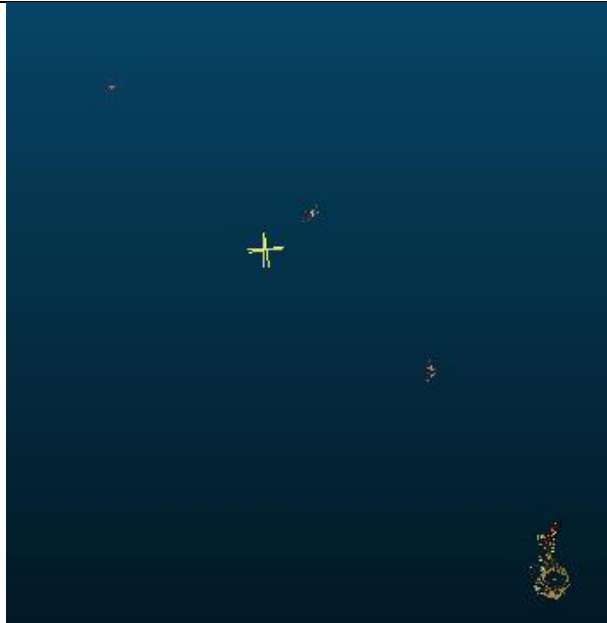
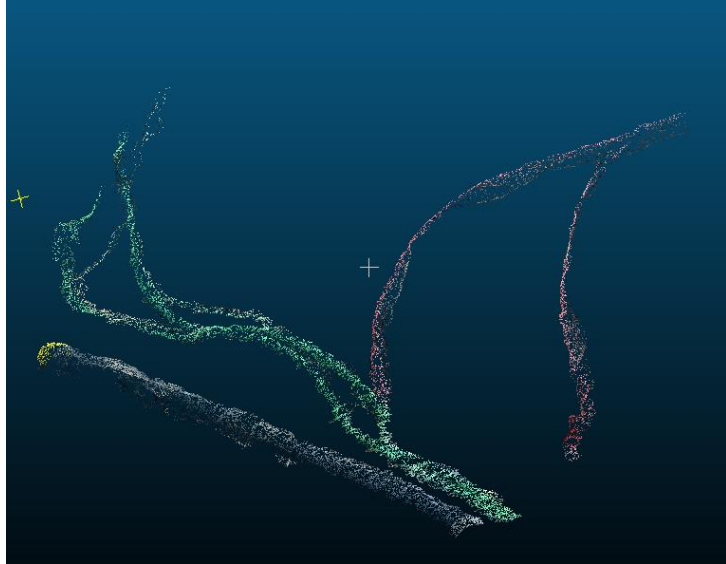
<p>Jun 21 at 8-27</p>		<p>Original Cloud: 4.151.100</p> <p>After preprocessing: 475.691</p> <p>Only utilities: 5.718</p>	<p>Total: 7</p> <p>Not utility: 27</p>
<p>Jun 21 at 8-47</p>		<p>Original Cloud: 2.938.380</p> <p>After preprocessing: 258.718</p> <p>Only utilities: 2.103</p>	<p>Total: 4</p> <p>Not utility: 23</p>

<p>Jun 21 at 9-18</p>		<p>Original Cloud: 5.015.560</p> <p>After preprocessing: 354.807</p> <p>Only utilities: 2.546</p>	<p>Total: 6</p> <p>Not utility: 24</p>
<p>Jun 21 at 10-35</p>		<p>Original Cloud: 3.132.940</p> <p>After preprocessing: 532.075</p> <p>Only utilities: 24.596</p>	<p>Total: 20</p> <p>Not utility: 29</p>

<p>Mar 20 at 9-39</p>		<p>Original Cloud: 1.589.420</p> <p>After preprocessing: 270.063</p> <p>Only utilities: 17.883</p>	<p>Green: 18 Orange: 10 White+blue: 10</p> <p>Total: 38</p> <p>Not utility: 16</p>
<p>Nov 22 at 9-29</p>		<p>Original Cloud: 1.411.640</p> <p>After preprocessing: 239.826</p> <p>Only utilities: 6.504</p>	<p>Total: 5</p> <p>Not utility: 19</p>

<p>Nov 22 at 9-34</p>		<p>Original Cloud: 3.805.560</p> <p>After preprocessing: 646.917</p> <p>Only utilities: 19.876</p>	<p>Total: 8</p> <p>Not utility: 15</p>
<p>Nov 29 at 10-45</p>		<p>Original Cloud: 1.930.100</p> <p>After preprocessing: 329.413</p> <p>Only utilities: 20.738</p>	<p>Red: 38 Green: 26</p> <p>Total: 64</p> <p>Not utility: 19</p>

<p>Sep 7 at 1-11</p>		<p>Original Cloud: 6.658.100</p> <p>After preprocessing: 1.130.831</p> <p>Only utilities: 41.804</p>	<p>Red: 11 Rest: 8</p> <p>Total: 19</p> <p>Not utility: 16</p>
<p>Sep 7 at 1-19</p>		<p>Original Cloud: 3.324.220</p> <p>After preprocessing: 564.516</p> <p>Only utilities: 18.241</p>	<p>Total: 6</p> <p>Not utility: 9</p>

<p>Sep 8 at 11-04</p>			<p>Original Cloud: 1.194.680</p> <p>After preprocessing: 203.162</p> <p>Only utilities: 538</p>	<p>Not utility: 14</p>
<p>Zwolle01</p>			<p>Original Cloud: 2.400.080</p> <p>After preprocessing: 407.453</p> <p>Only utilities: 19.642</p>	<p>Black: 7 Green: 15 Red: 8</p> <p>Total: 30</p> <p>Not utility: 26</p>

9.3 APPENDIX 3, SCRIPT TO CREATE TRAIN AND TEST SET

```

clear, clc

%load the folderpath of the point clouds that are utilities and are not utilities
folderpath_utility = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point clouds\utility';
folderpath_not_utility = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point clouds\not_utility';

%create structure arrays that contain information about the files
filelistutility = dir(fullfile(folderpath_utility, '*.ply'));
filelistnotutility = dir(fullfile(folderpath_not_utility, '*.ply'));

%initialize a cell array to store the point clouds and corresponding class
pointcloudsutility = cell(length(filelistutility),2);

%define number of points each point cloud should contain
numPoints = 1024;

%create a for loop that reads all utility point clouds from a folder and
%stores it in a cell array with the class 'utility'. it also up and
%downsamples the number of points in a cloud to 1024
for i = 1:length(filelistutility)
    filename = filelistutility(i).name;
    filepath = fullfile(folderpath_utility, filename);
    ptCloud = pcread(filepath);
    if ptCloud.Count > numPoints
        percentage = numPoints/ptCloud.Count;
        ptCloud = pcdsample(ptCloud,"random",percentage);
    else
        replicationFactor = ceil(numPoints/ptCloud.Count);
        ind = repmat(1:ptCloud.Count,1,replicationFactor);
        ptCloud = select(ptCloud,ind(1:numPoints));
    end
    pointcloudsutility(i,1) = {ptCloud};
    string = ["utility"];
    cat = categorical(string);
    pointcloudsutility(i,2) = {cat};
end

%initialize a cell array to store the point clouds and corresponding class
pointcloudsnotutility = cell(length(filelistutility),2);

%create a for loop that reads all utility point clouds from a folder and
%stores it in a cell array with the class 'not utility'
for i = 1:length(filelistnotutility)
    filename = filelistnotutility(i).name;
    filepath = fullfile(folderpath_not_utility, filename);
    ptCloud = pcread(filepath);
    if ptCloud.Count > numPoints
        percentage = numPoints/ptCloud.Count;
        ptCloud = pcdsample(ptCloud,"random",percentage);
    else
        replicationFactor = ceil(numPoints/ptCloud.Count);
        ind = repmat(1:ptCloud.Count,1,replicationFactor);
        ptCloud = select(ptCloud,ind(1:numPoints));
    end
    cloudsnotutility(i,1) = {ptCloud};

```

```

    string = ["not utility"];
    cat = categorical(string);
    pointcloudsnotutility(i,2) = {cat};
end

%randomly sort the utility point clouds and split them in a train and test
%set
rng(2,"twister")
numRows = size(pointcloudsutility,1);
indices = randperm(numRows);
trainingsetszeutility = round(0.7 * numRows);
trainingindicesutility = indices(1:trainingsetszeutility);
testindicesutility = indices(trainingsetszeutility+1:end);
trainingsetutility = pointcloudsutility(trainingindicesutility,:);
testsetutility = pointcloudsutility(testindicesutility,:);

%randomly sort the not_utility point clouds and split them in a train and test
%set
rng(3,"philox")
numRows = size(pointcloudsnotutility,1);
indices = randperm(numRows);
trainingsetszenotutility = round(0.7 * numRows);
trainingindicesnotutility = indices(1:trainingsetszenotutility);
testindicesnotutility = indices(trainingsetszenotutility+1:end);
trainingsetnotutility = pointcloudsnotutility(trainingindicesnotutility,:);
testsetnotutility = pointcloudsnotutility(testindicesnotutility,:);

%combine the training sets from utility and not_utility into one training
%set
combinedtrainingset = [
    trainingsetutility,
    trainingsetnotutility
];

%randomly sort the training set
rng(1,"multFibonacci")
indices = randperm(length(combinedtrainingset));
trainingset = combinedtrainingset(indices,:);

%combine the test sets from utility and not_utility into one test set
combinedtestset = [
    testsetutility,
    testsetnotutility
];

%randomly sort the test set
rng(4,"simdTwister")
indices = randperm(length(combinedtestset));
testset = combinedtestset(indices,:);

%save the training and test set in a .mat file
save('trainingset_1024.mat','trainingset')
save('testset_1024.mat','testset')

```

9.4 APPENDIX 4, SCRIPT TO CUT POINT CLOUD

```

clear, clc
%load the point cloud for the test case
test_cloud = pcread("C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point clouds\test_case\Jan9,1-13 for
matlab cut.ply");
%store the x,y and z limits in a variable
xlim = test_cloud.XLimits;
ylim = test_cloud.YLimits;
zlim = test_cloud.ZLimits;
%store the lower and upper limits for x y and z in separate variables
lower_x = xlim(1,1);
upper_x = xlim(1,2);
lower_y = ylim(1,1);
upper_y = ylim(1,2);
lower_z = zlim(1,1);
upper_z = zlim(1,2);
% create an empty array to store the randomly defined x limits and store
% the lower x limit in the array as the first limit
xlimits = [];
xlimits(1) = lower_x;
%define in how much parts the x axis maximally can be divided based on the
%x limits and the minimum distance per part on the x axis
space = diff(xlim)
;
max_amount = floor(space/0.15);

%define x random x limits with respect to the previous x limit. a new x
%limit is between 15 and 20 cm higher than the previous one. when an x
%limit is higher than the x limit of the point cloud, the value is set to
%the x limit of the point cloud.
for i = 2:max_amount
    xlimits(i) = xlimits(i-1)+randi([15,20])/100;
    if xlimits(i) >= upper_x
        xlimits(i) = upper_x;
    end
end

%because of the last for loop it can be the case the the last couple of
%limits have the same value as teh upper x limit these. these are
%transformed to 0 and after that they are deleted.
for i = 1:max_amount
    if xlimits(i) == upper_x
        xlimits(i) = [0];
    end
end

xlimits(numel(xlimits)+1) = upper_x;
xlimits = nonzeros(xlimits)';

%create an empty array in which the y limits can be stored
ylimits = [];

%define y limits that are between 2 and 3 meters higher than the lower y
%limit
for i = 1:numel(xlimits)-1
    ylimits(i) = lower_y + 2 + rand;
end

```

```

%create an empty array to store the dimensions in which the point cloud
%will be cut
roi = [];

%create in total 16 for loops (depending on the size of the point cloud
%that stores the dimensions in which the point cloud is cut. the for loops
%use the same x and z limits. the y limits are constantly redefined by
%picking a random distance between 2 and 3 meters. Only one for loop is displayed)

%1
for i = 1: numel(ylimits)
    roi(i,1) = xlimits(i);
    roi(i,2) = xlimits(i+1);
    roi(i,3) = lower_y;
    roi(i,4) = ylimits(i);
    roi(i,5) = lower_z;
    roi(i,6) = upper_z;
end

%set the y limits that are higher than the upper y limit of the point cloud
%equal to the upper y limit of the point cloud
for i = 1:length(Yehoshua)
    if roi(i,4) > upper_y
        roi(i,4) = upper_y;
    end
end

%delete the rows that define a dimension that lies outside the limits of
%the original point cloud
for i = 1 : length(Yehoshua)
    if roi(i,3) >= upper_y
        roi(i,:) = [0];
    end
end
rowswithzeros = any(roi == 0, 2);
roi = roi(~rowswithzeros,:);

%create an empty cell array in which all point clouds cut from the original
%point cloud can be stored
ptClouds = cell(length(roi),1);

%use a for loop to cut point clouds from the original point clouds. The
%dimensions that are used are stored in the variable 'roi'. Point clouds
%with less than 100 points are removed
for i = 1: length(Yehoshua)
    region = [roi(i,1) roi(i,2) roi(i,3) roi(i,4) roi(i,5) roi(i,6)];
    indices = findPointsInROI(test_cloud,region);
    ptCloud = select(test_cloud,indices);
    if ptCloud.Count > 100
        ptClouds(i) = {ptCloud};
    end
end
ptClouds = ptClouds(~cellfun('isempty', ptClouds));
% store all point clouds
for i = 1:length(ptClouds)
    filename = sprintf('matlabcut_%d.', i);
    pcwrite(ptClouds{i},filename);
end

```

9.5 APPENDIX 5, SCRIPT TO USE TRAINED MODEL FOR DEMONSTRATION

CASE

```

clear, clc

%load the trained network and the test point cloud
load('trainedvoxnet.mat');
folderpath_testclouds = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point
clouds\test_case\cuts_cloudcompare';

%create structure arrays that contain information about the files
filelist_testclouds = dir(fullfile(folderpath_testclouds, '*.ply'));

%initialize a cell array to store the point clouds
testclouds = cell(length(filelist_testclouds),1);
testset = cell(length(filelist_testclouds),1);

%initialize the number of points a point cloud should have
numPoints = 1024;

%create a for loop that reads all point clouds from a folder and
%stores it in a cell array. it also up and downsamples the number of points
%in a cloud to 1024
for i = 1:length(filelist_testclouds)
    filename = filelist_testclouds(i).name;
    filepath = fullfile(folderpath_testclouds, filename);
    ptCloud = pcread(filepath);
    if ptCloud.Count > numPoints
        percentage = numPoints/ptCloud.Count;
        ptCloud = pcdsample(ptCloud,"random",percentage);
    else
        replicationFactor = ceil(numPoints/ptCloud.Count);
        ind = repmat(1:ptCloud.Count,1,replicationFactor);
        ptCloud = select(ptCloud,ind(1:numPoints));
    end
    testclouds(i,1) = {ptCloud};
end

%convert the point clouds to occupancy grids using pccbin
for i = 1:length(testclouds)
    grid = pccbin(testclouds{i, 1},[32,32,32]);
    occupancyGrid = zeros(size(grid),'single');
    for ii = 1:numel(grid)
        occupancyGrid(ii) = ~isempty(grid{ii});
    end
    testset(i,1) = {occupancyGrid};
end

%predict labels for the point clouds using the loaded network
for i = 1:length(testset)
    predicted_labels(i,1) = classify(voxnet,testset{i,1});
end

%store the point clouds with their label in a cell array
cellwithlabels = cell(length(predicted_labels),2);
cellwithlabels(:,1) = testclouds;
for i = 1:length(predicted_labels)
    cellwithlabels(i,2) = {predicted_labels(i,1)};
end

```



```
end

%split the cell array into a cell array for utility and for not utility
for i = 1:length(cellwithlabels)
    if cellwithlabels{i,2} == ["utility"]
        classified_utility{i,1} = cellwithlabels{i,1};
    else if cellwithlabels{i,2} == ["not utility"]
        classified_not_utility{i,1} = cellwithlabels{i,1};
    end
end
end
classified_utility = classified_utility(~cellfun('isempty', classified_utility));
classified_not_utility = classified_not_utility(~cellfun('isempty',
classified_not_utility));

%store the point clouds that are classified as utility
folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point
clouds\test_case\cuts_CC_utility';
for i = 1: length(classified_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(classified_utility{i,1},filepath);
end

%store the point clouds that are classified as not utility
folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point
clouds\test_case\cuts_CC_not_utility';
for i = 1: length(classified_not_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(classified_not_utility{i,1},filepath);
end
```

9.6 APPENDIX 6, SCRIPT TO CREATE POLYLINES

```

clear, clc

%define per pipe the folderpaths for the point cloud parts.
folderpath_pipe1 = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\point cloud data\point
clouds\test_case\cuts_per_pipe\pipe1';

%define a filelist per pipe for the point cloud parts that contains
%information about every file.
filelist_pipe1 = dir(fullfile(folderpath_pipe1, '*.ply'));

%create empty cell arrays in which the point cloud parts can be stored per
%pipe.
pipe1_clouds = cell(length(filelist_pipe1),1);

%use a for loop that reads the filelist containing the names of every file
%and uses the filenames to read the point clouds in the folderpath. the
%point clouds are then stored in the empty cell array.
for i = 1:length(filelist_pipe1)
    filename = filelist_pipe1(i).name;
    filepath = fullfile(folderpath_pipe1, filename);
    ptCloud = pcread(filepath);
    pipe1_clouds(i,1) = {ptCloud};
end

%calculate from every point cloud the mean value for the XY an Z
%coordinates. the points describe the centroid of a point cloud.
for i = 1:length(pipe1_clouds)
    points_pipe1(i,:) =
[mean(pipe1_clouds{i,1}.Location(:,1)),mean(pipe1_clouds{i,1}.Location(:,2)),mean(
pipe1_clouds{i,1}.Location(:,3))];
end

%order the centroids of the point clouds in ascending order based on the
%direction in which the point clouds lie.
if
diff(max(points_pipe1(:,1),min(points_pipe1(:,1))))>diff(max(points_pipe1(:,2),min
(points_pipe1(:,2))))
    points_pipe1 = sortrows(points_pipe1,1);
else
    points_pipe1 = sortrows(points_pipe1,2);
end

%create a figure in which the point cloud parts are shown and in which the
%centroids are connected to each other to create a line that represents the
%centre of the pipe.
figure
hold on

for i = 1:length(pipe1_clouds)
    pcshow(pipe1_clouds{i,1});
end
p1 = plot3(points_pipe1(:,1), points_pipe1(:,2), points_pipe1(:,3), 'b');

% save the points from which the line is computed in a txt file
writematrix(points_pipe1,'points_pipe1','Delimiter','tab');

```

9.7 APPENDIX 7, VOXNET SCRIPT

The VoxNet script is defined below. It is created with the help of an example provided by MathWorks, the developer of MATLAB (Mathworks, 2021).

```
clear, clc

%load the train and test set that consists of point clouds with 1024 points
%each
dsTrain = load("trainingset_1024.mat");
dsTest = load('testset_1024.mat');

%create cell arrays to store the point clouds in column 1 and corresponding
%labels in column 2
trainingset = cell(length(dsTrain.trainingset),2);
testset = cell(length(dsTest.testset),2);

%define label names
labelNames = ["utility","not utility"];

%create a for loop that transforms every point cloud in the training set
%to an occupancy grid of size 32x32x32, store the grid in column 1 and
%corresponding label in column 2
for i = 1:length(dsTrain.trainingset)
    grid = pccbin(dsTrain.trainingset{i, 1},[32,32,32]);
    occupancyGrid = zeros(size(grid),'single');
    for ii = 1:numel(grid)
        occupancyGrid(ii) = ~isempty(grid{ii});
    end
    name = dsTrain.trainingset{i, 2};
    label = categorical(name,labelNames);
    trainingset(i,1) = {occupancyGrid};
    trainingset(i,2) = {label};
end

%create a for loop that transforms every point cloud in the test set
%to an occupancy grid of size 32x32x32, store the grid in column 1 and
%corresponding label in column 2
for i = 1:length(dsTest.testset)
    grid = pccbin(dsTest.testset{i, 1},[32,32,32]);
    occupancyGrid = zeros(size(grid),'single');
    for ii = 1:numel(grid)
        occupancyGrid(ii) = ~isempty(grid{ii});
    end
    name = dsTest.testset{i, 2};
    label = categorical(name,labelNames);
    testset(i,1) = {occupancyGrid};
    testset(i,2) = {label};
end

%store the training and test set in a tall cell array and write the data to
%a folder
trainingtall = tall(trainingset);
testtall = tall(testset);
write("Voxnet_train\",trainingtall);
write("Voxnet_test\",testtall);

%create a train and test datastore from the data saved in the folders
```

```

dsTrain2 = datastore("Voxnet_train\");
dsTest2 = datastore("Voxnet_test\");

%define layers of the neural network
layers =
[image3dInputLayer([32,32,32], 'Name', 'inputLayer', 'Normalization', 'none'),...
    convolution3dLayer(5,32, 'Stride',2, 'Name', 'Conv1'),...
    leakyReluLayer(0.1, 'Name', 'leakyRelu1'),...
    convolution3dLayer(3,32, 'Stride',1, 'Name', 'Conv2'),...
    leakyReluLayer(0.1, 'Name', 'leakyRulu2'),...
    maxPooling3dLayer(2, 'Stride',2, 'Name', 'maxPool'),...
    fullyConnectedLayer(128, 'Name', 'fc1'),...
    reluLayer('Name', 'relu'),...
    dropoutLayer(0.5, 'Name', 'dropout1'),...
    fullyConnectedLayer(2, 'Name', 'fc2'),...
    softmaxLayer('Name', 'softmax'),...
    classificationLayer('Name', 'crossEntropyLoss')];

%display layer graph
voxnet = layerGraph(layers);
figure
plot(voxnet);

%define training options
miniBatchSize = 32;
dsLength = length(trainingset);
iterationsPerEpoch = floor(dsLength/miniBatchSize);
dropPeriod = floor(8000/iterationsPerEpoch);

%store the training options in the 'options' variable
options =
trainingOptions('sgdm', 'InitialLearnRate',0.01, 'MiniBatchSize',miniBatchSize,...
    'LearnRateSchedule', 'Piecewise',...
    'LearnRateDropPeriod', dropPeriod,...
    'ValidationData', dsTest2, 'MaxEpochs', 60,...
    'DispatchInBackground', false,...
    'Shuffle', 'never');

%train the network with the train dataset, the defined layers and the
%options set for training
voxnet = trainNetwork(dsTrain2,voxnet,options);

%evaluate the network, display the accuracy
valLabelSet = transform(dsTest2,@(data) data{2});
valLabels = readall(valLabelSet);
outputLabels = classify(voxnet,dsTest2);
accuracy = nnz(outputLabels == valLabels) / numel(outputLabels);
disp(accuracy)

%display a confusion chart to evaluate the network
figure
confusionchart(valLabels,outputLabels)

%to display clouds that are predicted wrong, first the indices have to be
%found by looking at which output labels do not correspond with the labels
%from the test set. indices are also found for for the clouds that are
%labelled right
wrongpredicted = find(outputLabels ~= valLabels);
rightpredicted = find(outputLabels == valLabels);

```

```
%All test clouds are saved in a cell array together with their class, first
%an empty cell array is created in which the information can be saved
findclouds = cell(length(dsTest.testset),2);
```

```
%the for loop writes the point clouds in the test set together with the
%class label to the empty findclouds cell array
```

```
for i = 1:length(dsTest.testset)
name = dsTest.testset{i, 2};
label = categorical(name,labelNames);
findclouds(i,1) = {dsTest.testset{i, 1}};
findclouds(i,2) = {label};
end
```

```
%in wrongclouds the point clouds are saved that are labelled wrong together
%with their label by looking at the indices of the clouds that are labelled
%wrong. in rightclouds the point clouds are saved that are labelled right
%together with their label.
```

```
wrongclouds = findclouds(wrongpredicted,:);
rightclouds = findclouds(rightpredicted,:);
```

```
%4 empty cell arrays are created in which point clouds that are falsely
%labelled as utility and falsely labelled as not utility and point clouds
%that are labelled right are saved
```

```
false_utility = {};
false_not_utility = {};
right_utility = {};
right_not_utility = {};
```

```
%the for loop writes te wrongly labelled clouds to the correct cell array
%so point clouds that are falsely labelled as utility and falsely labelled
%as not utility are separated
```

```
for i = 1: size(wrongclouds,1)
    if wrongclouds{i,2} == ["not utility"]
        false_utility = [false_utility; wrongclouds(i,:)];
    else if wrongclouds{i,2} == ["utility"]
        false_not_utility = [false_not_utility; wrongclouds(i,:)];
    end
end
end
```

```
%the for loop writes the clouds that are labelled right to the correct cell
%array so that clouds that are labelled right as utility and right as not
%utility are separated
```

```
for i = 1 : size(rightclouds,1)
    if rightclouds{i,2} == ["utility"]
        right_utility = [right_utility; rightclouds(i,:)];
    else if rightclouds{i,2} == ["not utility"]
        right_not_utility = [right_not_utility; rightclouds(i,:)];
    end
end
end
```

```
%the following four for loops save the clouds to the right folders
```

```
folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training\rightutility';
```

```
for i = 1: length(right_utility)
    filename = sprintf('pointcloud_%d.ply',i);
```

```
    filepath = fullfile(folder,filename);
    pcwrite(right_utility{i,1},filepath);
end

folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training\rightnotutility';

for i = 1: length(right_not_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(right_not_utility{i,1},filepath);
end

folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training>falseutility';

for i = 1: length(false_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(false_utility{i,1},filepath);
end

folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training>falseutility';

for i = 1: length(false_not_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(false_not_utility{i,1},filepath);
end
```

9.8 APPENDIX 8, SHALLOW NEURAL NETWORK SCRIPT

```

clear, clc

%load the train and test set that consists of point clouds with 1024 points
%each
dsTrain = load("trainingset_1024.mat");
dsTest = load('testset_1024.mat');

%create cell arrays to store the point clouds in column 1 and corresponding
%labels in column 2
trainingset = cell(length(dsTrain.trainingset),2);
testset = cell(length(dsTest.testset),2);

%define label names
labelNames = ["utility","not utility"];

%create a for loop that transforms every point cloud in the training set
%to an occupancy grid of size 32x32x32, store the grid in column 1 and
%corresponding label in column 2
for i = 1:length(dsTrain.trainingset)
    grid = pccbin(dsTrain.trainingset{i, 1},[32 32 32]);
    occupancyGrid = zeros(size(grid),'single');
    for ii = 1:numel(grid)
        occupancyGrid(ii) = ~isempty(grid{ii});
    end
    name = dsTrain.trainingset{i, 2};
    label = categorical(name,labelNames);
    trainingset(i,1) = {occupancyGrid};
    trainingset(i,2) = {label};
end

%create a for loop that transforms every point cloud in the test set
%to an occupancy grid of size 32x32x32, store the grid in column 1 and
%corresponding label in column 2
for i = 1:length(dsTest.testset)
    grid = pccbin(dsTest.testset{i, 1},[32 32 32]);
    occupancyGrid = zeros(size(grid),'single');
    for ii = 1:numel(grid)
        occupancyGrid(ii) = ~isempty(grid{ii});
    end
    name = dsTest.testset{i, 2};
    label = categorical(name,labelNames);
    testset(i,1) = {occupancyGrid};
    testset(i,2) = {label};
end

%store the training and test set in a tall cell array and write the data to
%a folder
trainingtall = tall(trainingset);
testtall = tall(testset);
write("fc_train\ ",trainingtall);
write("fc_test\ ",testtall);

%create a train and test datastore from the data saved in the folders
dsTrain2 = datastore("fc_train\ ");
dsTest2 = datastore("fc_test\ ");

```



```

%define layers of the neural network
layers = [
    image3dInputLayer([32 32 32 1], "Name", "image3dinput")
    fullyConnectedLayer(128, "Name", "fc")
    fullyConnectedLayer(64, "Name", "fc_1")
    fullyConnectedLayer(32, "Name", "fc_2")
    fullyConnectedLayer(2, "Name", "fc_3")
    softmaxLayer("Name", "softmax")
    classificationLayer("Name", "classoutput")];

%display layer graph
fullyconnectedlayers = layerGraph(layers);
figure
plot(fullyconnectedlayers);

%define training options
miniBatchSize = 32;
dsLength = length(trainingset);
iterationsPerEpoch = floor(dsLength/miniBatchSize);
dropPeriod = floor(4000/iterationsPerEpoch);

%store the training options in the 'options' variable
options =
trainingOptions('sgdm', 'InitialLearnRate', 0.01, 'MiniBatchSize', miniBatchSize, ...
    'LearnRateSchedule', 'Piecewise', ...
    'LearnRateDropPeriod', dropPeriod, ...
    'ValidationData', dsTest2, 'MaxEpochs', 60, ...
    'DispatchInBackground', false, ...
    'Shuffle', 'never', 'Plots', 'training-progress' );

%train the network with the train dataset, the defined layers and the
%options set for training
fullyconnectedlayers = trainNetwork(dsTrain2, fullyconnectedlayers, options);

%evaluate the network, display the accuracy
valLabelSet = transform(dsTest2, @(data) data{2});
valLabels = readall(valLabelSet);
outputLabels = classify(fullyconnectedlayers, dsTest2);
accuracy = nnz(outputLabels == valLabels) / numel(outputLabels);
disp(accuracy)

%display a confusion chart to evaluate the network
figure
confusionchart(valLabels, outputLabels)

%to display clouds that are predicted wrong, first the indices have to be
%found by looking at which output labels do not correspond with the labels
%from the test set. indices are also found for for the clouds that are
%labelled right
wrongpredicted = find(outputLabels ~= valLabels);
rightpredicted = find(outputLabels == valLabels);

%All test clouds are saved in a cell array together with their class, first
%an empty cell array is created in which the information can be saved
findclouds = cell(length(dsTest.testset), 2);

%the for loop writes the point clouds in the test set together with the
%class label to the empty findclouds cell array
for i = 1:length(dsTest.testset)

```

```

name = dsTest.testset{i, 2};
label = categorical(name,labelNames);
findclouds(i,1) = {dsTest.testset{i, 1}};
findclouds(i,2) = {label};
end

%in wrongclouds the point clouds are saved that are labelled wrong together
%with their label by looking at the indices of the clouds that are labelled
%wrong. in rightclouds the point clouds are saved that are labelled right
%together with their label.
wrongclouds = findclouds(wrongpredicted,:);
rightclouds = findclouds(rightpredicted,:);

%4 empty cell arrays are created in which point clouds that are falsely
%labelled as utility and falsely labelled as not utility and point clouds
%that are labelled right are saved
false_utility = {};
false_not_utility = {};
right_utility = {};
right_not_utility = {};

%the for loop writes te wrongly labelled clouds to the correct cell array
%so point clouds that are falsely labelled as utility and falsely labelled
%as not utility are separated
for i = 1: size(wrongclouds,1)
    if wrongclouds{i,2} == ["not utility"]
        false_utility = [false_utility; wrongclouds(i,:)];
    else if wrongclouds{i,2} == ["utility"]
        false_not_utility = [false_not_utility; wrongclouds(i,:)];
    end
end
end

%the for loop writes the clouds that are labelled right to the correct cell
%array so that clouds that are labelled right as utility and right as not
%utility are separated
for i = 1 : size(rightclouds,1)
    if rightclouds{i,2} == ["utility"]
        right_utility = [right_utility; rightclouds(i,:)];
    else if rightclouds{i,2} == ["not utility"]
        right_not_utility = [right_not_utility; rightclouds(i,:)];
    end
end
end

%the following four for loops save the clouds to the right folders
folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training\rightutility';

for i = 1: length(right_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(right_utility{i,1},filepath);
end

folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training\rightnotutility';

```

```
for i = 1: length(right_not_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(right_not_utility{i,1},filepath);
end

folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training>falseutility';

for i = 1: length(false_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(false_utility{i,1},filepath);
end

folder = 'C:\Users\jornk\OneDrive - Universiteit
Twente\Afstuderen\afstuderen\training>falseutility';

for i = 1: length(false_not_utility)
    filename = sprintf('pointcloud_%d.ply',i);
    filepath = fullfile(folder,filename);
    pcwrite(false_not_utility{i,1},filepath);
end
```

9.9 APPENDIX 9, POINTNET SCRIPT

The PointNet script is defined below. It is created with the help of an example provided by MathWorks, the developer of MATLAB (Mathworks, 2023c).

```
clear, clc

%load the train and test set that consists of point clouds with 1024 points
%each
dsTrain = load("trainingset_1024.mat");
dsTest = load('testset_1024.mat');

%create cell arrays to store the point clouds in column 1 and corresponding
%labels in column 2
trainingset = cell(length(dsTrain.trainingset),2);
testset = cell(length(dsTest.testset),2);

%define label names
labelNames = ["utility","not utility"];
classes = ["utility","not utility"];

%create a for loop that writes the training point clouds and corresponding labels
to
%a cell array with the point clouds in the first column and the labels in
%the second column
for i = 1:length(dsTrain.trainingset)
name = dsTrain.trainingset{i, 2};
label = categorical(name,labelNames);
trainingset(i,1) = {dsTrain.trainingset{i, 1}};
trainingset(i,2) = {label};
end

%create a for loop that writes the testing point clouds and corresponding labels
to
%a cell array with the point clouds in the first column and the labels in
%the second column
for i = 1:length(dsTest.testset)
name = dsTest.testset{i, 2};
label = categorical(name,labelNames);
testset(i,1) = {dsTest.testset{i, 1}};
testset(i,2) = {label};
end

%{
%store the training and test set in a tall cell array and write the data to
%a folder
trainingtall = tall(trainingset);
testtall = tall(testset);
write("pointnet_train\ ",trainingtall);
write("pointnet_test\ ",testtall);
%}

%create a train and test datastore from the data saved in the folders
dsTrain2 = datastore("pointnet_train\ ");
dsTest2 = datastore("pointnet_test\ ");

% use the @preprocesspointcloud function to transform the point clouds
% stored in the first column of the cell array to a dataset that consists
% of x,y and z coordinates of 1024 points per cloud. The points are rescaled
```

```
% between 0 and 1
dsTrain2 = transform(dsTrain2,@preprocessPointCloud);
dsTest2 = transform(dsTest2,@preprocessPointCloud);

%define the parameters of the first layer of the pointnet encoder, which is an
input
%transform layer. It uses the initialize transform function at the end of
%the script
inputChannelSize = 3;
hiddenChannelSize1 = [64,128];
hiddenChannelSize2 = 256;
[parameters.InputTransform, state.InputTransform] =
initializeTransform(inputChannelSize,hiddenChannelSize1,hiddenChannelSize2);

%define the parameters of the second layer of the pointnet encoder, which
%is a shared MLP layer. It uses the initialize shared MLP function at the
%end of the script
inputChannelSize = 3;
hiddenChannelSize = [64 64];
[parameters.SharedMLP1,state.SharedMLP1] =
initializeSharedMLP(inputChannelSize,hiddenChannelSize);

%define the parameters of the third layer of the pointnet encoder, which is
%a feature transform layer. it uses the initialize transform function at the
%end of the script
inputChannelSize = 64;
hiddenChannelSize1 = [64,128];
hiddenChannelSize2 = 256;
[parameters.FeatureTransform, state.FeatureTransform] =
initializeTransform(inputChannelSize,hiddenChannelSize,hiddenChannelSize2);

%define the parameters of the fourth layer of the pointnet encoder, which
%is a shared MLP layer. It uses the initialize shared MLP function at the
%end of the script
inputChannelSize = 64;
hiddenChannelSize = 64;
[parameters.SharedMLP2,state.SharedMLP2] =
initializeSharedMLP(inputChannelSize,hiddenChannelSize);

% define the parameters of the pointnet classifier. It consists of a
% shared MLP, fully connected layer and a softmax activation layer. It uses
% the initialize classification MLP function at the end of the example
inputChannelSize = 64;
hiddenChannelSize = [512,256];
numClasses = numel(classes);
[parameters.ClassificationMLP, state.ClassificationMLP] =
initializeClassificationMLP(inputChannelSize,hiddenChannelSize,numClasses);

%define the training parameters
numEpochs = 60;
learnRate = 0.001;
miniBatchSize = 128;
l2Regularization = 0.01;
learnRateDropPeriod = 15;
learnRateDropFactor = 0.5;

%initialize options for adam optimization
gradientDecayFactor = 0.9;
squaredGradientDecayFactor = 0.999;
```

```

avgGradients = [];
avgSquaredGradients = [];

%define the training loop

% Create a minibatchqueue to batch data from training and validation
% datastores. It uses the batch data function at the end of the script
% to batch the point cloud data and one-hot encode the label
% data.
numOutputsFromDSRead = 2;
mbqTrain = minibatchqueue(dsTrain2,numOutputsFromDSRead,...
    "MiniBatchSize", miniBatchSize,...
    "MiniBatchFcn",@batchData,...
    "MiniBatchFormat",["SCSB" "BC"]);

mbqVal = minibatchqueue(dsTest2,numOutputsFromDSRead,...
    "MiniBatchSize", miniBatchSize,...
    "MiniBatchFcn",@batchData,...
    "MiniBatchFormat",["SCSB" "BC"]);

% Use the configureTrainingProgressPlot function at the end of the
% script, to initialize the training progress plot to display the training
% loss, training accuracy, and validation accuracy.
[lossPlotter, trainAccPlotter,valAccPlotter] = initializeTrainingProgressPlot;

numClasses = numel(classes);
iteration = 0;
start = tic;
for epoch = 1:numEpochs

    % Iterate through data set.
    while hasdata(mbqTrain)
        iteration = iteration + 1;

        % Read next batch of training data.
        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and the
        % modelGradients function.
        [gradients, loss, state, acc] =
dlfeval(@modelGradients,XTrain,YTrain,parameters,state);

        % L2 regularization.
        gradients = dlupdate(@(g,p) g +
l2Regularization*p,gradients,parameters);

        % Update the network parameters using the Adam optimizer.
        [parameters, avgGradients, avgSquaredGradients] =
adamupdate(parameters, gradients, ...
            avgGradients, avgSquaredGradients, iteration,...
            learnRate,gradientDecayFactor, squaredGradientDecayFactor);

        % Update the training progress.
        D = duration(0,0,toc(start),"Format","hh:mm:ss");
        title(lossPlotter.Parent,"Epoch: " + epoch + ", Elapsed: " +
string(D))
        addpoints(lossPlotter,iteration,double(gather(extractdata(loss))))
        addpoints(trainAccPlotter,iteration,acc);
    end
end

```

```

        drawnow
    end

    % Evaluate the model on validation data.
    cmat = sparse(numClasses,numClasses);
    while hasdata(mbqVal)

        % Read next batch of validation data.
        [XVal, YVal] = next(mbqVal);

        % Compute label predictions.
        isTraining = false;
        YPred = pointnetClassifier(XVal,parameters,state,isTraining);

        % Choose prediction with highest score as the class label for
        % XTest.
        [~,YValLabel] = max(YVal,[],1);
        [~,YPredLabel] = max(YPred,[],1);

        % Collect confusion metrics.
        cmat = aggregateConfusionMetric(cmat,YValLabel,YPredLabel);
    end

    % Update training progress plot with average classification accuracy.
    acc = sum(diag(cmat))./sum(cmat,"all");
    addpoints(valAccPlotter,iteration,acc);

    % Udate the learning rate.
    if mod(epoch,learnRateDropPeriod) == 0
        learnRate = learnRate * learnRateDropFactor;
    end

    % Reset training and validation data queues.
    reset(mbqTrain);
    reset(mbqVal);
end

%create a confusion chart to display the model results
figure
chart = confusionchart(cmat,classes);

%calculate the accuracy
acc = sum(diag(cmat))./sum(cmat,"all")

%the model gradients function takes as input the model parameters, state
%and a mini batch of input data. it returns the gradients of the loss with
%respect to the learnable parameters and the corresponding loss.
function [gradients, loss, state, acc] = modelGradients(X,Y,parameters,state)

% Execute the model function.
isTraining = true;
[YPred,state,d1T] = pointnetClassifier(X,parameters,state,isTraining);

% Add regularization term to ensure feature transform matrix is
% approximately orthogonal.
K = size(d1T,1);
B = size(d1T, 4);
I = repelem(eye(K),1,1,1,B);
d1I = dlarray(I,"SSCB");

```

```

treg = mse(dlI,pageTimes(dlT,permute(dlT,[2 1 3 4])));
factor = 0.001;

% Compute the loss.
loss = crossentropy(YPred,Y) + factor*treg;

% Compute the parameter gradients with respect to the loss.
gradients = dlgradient(loss, parameters);

% Compute training accuracy metric.
[~,YTest] = max(Y,[],1);
[~,YPred] = max(YPred,[],1);
acc = gather(extractdata(sum(YTest == YPred)./numel(YTest)));

end

% the pointnet classifier function takes as input the point cloud data,
% parameters, model state and the flag istraining. the output is the
% probability distribution, model state and feature transformation matrix
function [dlY,state,dlT] = pointnetClassifier(dlX,parameters,state,istraining)

% Invoke the PointNet encoder.
[dlY,state,dlT] = pointnetEncoder(dlX,parameters,state,istraining);

% Invoke the classifier.
p = parameters.ClassificationMLP.Perceptron;
s = state.ClassificationMLP.Perceptron;
for k = 1:numel(p)

    [dlY, s(k)] = perceptron(dlY,p(k),s(k),istraining);

    % If training, apply inverted dropout with a probability of 0.3.
    if istraining
        probability = 0.3;
        dropoutScaleFactor = 1 - probability;
        dropoutMask = ( rand(size(dlY), "like", dlY) > probability ) /
dropoutScaleFactor;
        dlY = dlY.*dropoutMask;
    end

end

state.ClassificationMLP.Perceptron = s;

% Apply final fully connected and softmax operations.
weights = parameters.ClassificationMLP.FC.Weights;
bias = parameters.ClassificationMLP.FC.Bias;
dlY = fullyconnect(dlY,weights,bias);
dlY = softmax(dlY);
end

% the pointnet encoder function processes the input point cloud data using
% the input transform, shared MLP, feature transform and shared MLP layers
% and a max operation and gives as output the result of max operation
function [dlY,state,T] = pointnetEncoder(dlX,parameters,state,istraining)
% Input transform.
[dlY,state.InputTransform] =
dataTransform(dlX,parameters.InputTransform,state.InputTransform,istraining);

% Shared MLP.

```



```

[dLY, state.SharedMLP1.Perceptron] =
sharedMLP(dLY, parameters.SharedMLP1.Perceptron, state.SharedMLP1.Perceptron, isTrain
ing);

% Feature transform.
[dLY, state.FeatureTransform, T] =
dataTransform(dLY, parameters.FeatureTransform, state.FeatureTransform, isTraining);

% Shared MLP.
[dLY, state.SharedMLP2.Perceptron] =
sharedMLP(dLY, parameters.SharedMLP2.Perceptron, state.SharedMLP2.Perceptron, isTrain
ing);

% Max operation.
dLY = max(dLY, [], 1);
end

% the shared multilayer perceptron (MLP) processes the input data using
% multiple perceptrons and gives the result of the last perceptron
function [dLY, state] = sharedMLP(dlX, parameters, state, isTraining)
dLY = dlX;
for k = 1:numel(parameters)
    [dLY, state(k)] = perceptron(dLY, parameters(k), state(k), isTraining);
end
end

% the perceptron function processes the input data using convolution, batch
% normalization and relu operation and gives as output the result of the
% relu operation
function [dLY, state] = perceptron(dlX, parameters, state, isTraining)
% Convolution.
W = parameters.Conv.Weights;
B = parameters.Conv.Bias;
dLY = dlconv(dlX, W, B);

% Batch normalization. Update batch normalization state when training.
offset = parameters.BatchNorm.Offset;
scale = parameters.BatchNorm.Scale;
trainedMean = state.BatchNorm.TrainedMean;
trainedVariance = state.BatchNorm.TrainedVariance;
if isTraining
    [dLY, trainedMean, trainedVariance] =
batchnorm(dLY, offset, scale, trainedMean, trainedVariance);

    % Update state.
    state.BatchNorm.TrainedMean = trainedMean;
    state.BatchNorm.TrainedVariance = trainedVariance;
else
    dLY = batchnorm(dLY, offset, scale, trainedMean, trainedVariance);
end

% ReLU.
dLY = relu(dLY);
end

%the data transform function processes the input data using a shared MLP, a
%max operation and a second shared MLP to predict a transformation matrix.
function [dLY, state, T] = dataTransform(dlX, parameters, state, isTraining)

```

```

% Shared MLP.
[dLY,state.Block1.Perceptron] =
sharedMLP(dlX,parameters.Block1.Perceptron,state.Block1.Perceptron,isTraining);

% Max operation.
dLY = max(dLY,[],1);

% Shared MLP.
[dLY,state.Block2.Perceptron] =
sharedMLP(dLY,parameters.Block2.Perceptron,state.Block2.Perceptron,isTraining);

% Transform net (T-Net). Apply last fully connected operation as W*X to
% predict transformation matrix T.
dLY = squeeze(dLY); % N-by-B
T = parameters.Transform * stripdims(dLY); % K^2-by-B

% Reshape T into a square matrix.
K = sqrt(size(T,1));
T = reshape(T,K,K,1,[]); % [K K 1 B]
T = T + eye(K);

% Apply to input dlX using batch matrix multiply.
[C,B] = size(dlX,[3 4]); % [M 1 K B]
dlX = reshape(dlX,[],C,1,B); % [M K 1 B]
Y = pagemtimes(dlX,T);
dLY = dlarray(Y,"SCSB");
end

%The initializeTransform function takes as input the channel size and the
%number of hidden channels for the two shared MLPs, and returns the initialized
%parameters. The parameters are initialized using He weight initialization
function [params,state] = initializeTransform(inputChannelSize,block1,block2)
[params.Block1,state.Block1] = initializeSharedMLP(inputChannelSize,block1);
[params.Block2,state.Block2] = initializeSharedMLP(block1(end),block2);

% Parameters for the transform matrix.
params.Transform = dlarray(zeros(inputChannelSize^2,block2(end)));
end

%the initialize shared MLP function takes as input channel size and number
%of hidden layers and gives as output the initialized parameters. The
%parameters are initialized using He weight initialization.
function [params,state] = initializeSharedMLP(inputChannelSize,hiddenChannelSize)
weights = initializeWeightsHe([1 1 inputChannelSize hiddenChannelSize(1)]);
bias = zeros(hiddenChannelSize(1),1,"single");
p.Conv.Weights = dlarray(weights);
p.Conv.Bias = dlarray(bias);

p.BatchNorm.Offset = dlarray(zeros(hiddenChannelSize(1),1,"single"));
p.BatchNorm.Scale = dlarray(ones(hiddenChannelSize(1),1,"single"));

s.BatchNorm.TrainedMean = zeros(hiddenChannelSize(1),1,"single");
s.BatchNorm.TrainedVariance = ones(hiddenChannelSize(1),1,"single");

params.Perceptron(1) = p;
state.Perceptron(1) = s;

for k = 2:numel(hiddenChannelSize)

```

```

    weights = initializeWeightsHe([1 1 hiddenChannelSize(k-1)
hiddenChannelSize(k)]);
    bias = zeros(hiddenChannelSize(k),1,"single");
    p.Conv.Weights = darray(weights);
    p.Conv.Bias = darray(bias);

    p.BatchNorm.Offset = darray(zeros(hiddenChannelSize(k),1,"single"));
    p.BatchNorm.Scale = darray(ones(hiddenChannelSize(k),1,"single"));

    s.BatchNorm.TrainedMean = zeros(hiddenChannelSize(k),1,"single");
    s.BatchNorm.TrainedVariance = ones(hiddenChannelSize(k),1,"single");

    params.Perceptron(k) = p;
    state.Perceptron(k) = s;
end
end

%The initializeClassificationMLP function takes as input the channel size,
%the hidden channel size, and the number of classes and returns the initialized
%parameters. The shared MLP is initialized using He weight initialization
%and the final fully connected operation is initialized using random Gaussian
values.
function [params,state] =
initializeClassificationMLP(inputChannelSize,hiddenChannelSize,numClasses)
[params,state] = initializeSharedMLP(inputChannelSize,hiddenChannelSize);

weights = initializeWeightsGaussian([numClasses hiddenChannelSize(end)]);
bias = zeros(numClasses,1,"single");
params.FC.Weights = darray(weights);
params.FC.Bias = darray(bias);
end

% this function initializes parameters using He initialization
function x = initializeWeightsHe(sz)
fanIn = prod(sz(1:3));
stddev = sqrt(2/fanIn);
x = stddev .* randn(sz);
end

%this function initializes parameters using Gaussian initialization
function x = initializeWeightsGaussian(sz)
x = randn(sz,"single") .* 0.01;
end

%The preprocesspointcloud function transforms the point clouds
%stored in the first column of the cell array to a dataset that consists
%of x,y and z coordinates of 1024 point per cloud. the points are rescaled
%between 0 and 1
function data = preprocessPointCloud(data)

if ~iscell(data)
    data = {data};
end

numObservations = size(data,1);
for i = 1:numObservations
    % Scale points between 0 and 1.
    xlim = data{i,1}.XLimits;
    ylim = data{i,1}.YLimits;

```

```

    zlim = data{i,1}.ZLimits;

    xyzMin = [xlim(1) ylim(1) zlim(1)];
    xyzDiff = [diff(xlim) diff(ylim) diff(zlim)];

    data{i,1} = (data{i,1}.Location - xyzMin) ./ xyzDiff;
end
end

%this function is used to create a confusion metric
function cmat = aggregateConfusionMetric(cmat,YTest,YPred)
YTest = gather(extractdata(YTest));
YPred = gather(extractdata(YPred));
[m,n] = size(cmat);
cmat = cmat + full(sparse(YTest,YPred,1,m,n));
end

%this function is used to create a training progress plot
function [plotter,trainAccPlotter,valAccPlotter] =
initializeTrainingProgressPlot()
% Plot the loss, training accuracy, and validation accuracy.
figure

% Loss plot
subplot(2,1,1)
plotter = animatedline;
xlabel("Iteration")
ylabel("Loss")

% Accuracy plot
subplot(2,1,2)
trainAccPlotter = animatedline('Color','b');
valAccPlotter = animatedline('Color','g');
legend('Training Accuracy','Validation Accuracy','Location','northwest');
xlabel("Iteration")
ylabel("Accuracy")
end

% this function splits data into batches
function [X,Y] = batchData(ptCloud,labels)
X = cat(4,ptCloud{:});
labels = cat(1,labels{:});
Y = onehotencode(labels,2);
end

```