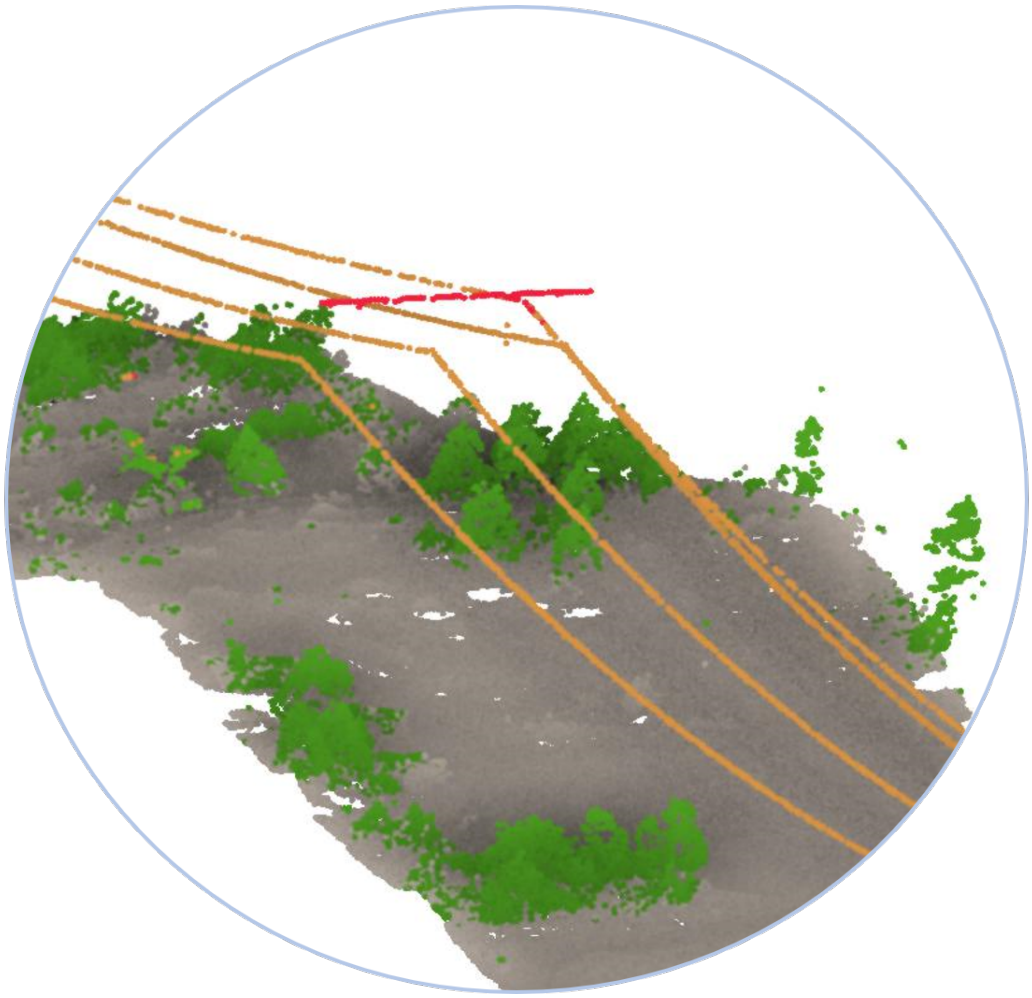
Aalborg University
AAU CPH
Geoinformatics



Comparing machine-learning classifiers of power-lines in point cloud data

in collaboration with Niras A/S

Author:	*Student nr.:*	*Supervisor:*
Tobias Hjelmbjerg Jensen	20155720	Carsten Keßler

January 2020

Title:

Comparing machine-learning classifiers of power-lines in point cloud data

Project:

4. Semester of Geoinformatics

Project Period:

February 2020 - June 2020

Author:

Tobias Hjelmbjerg Jensen

Supervisor:

Carsten Keßler

Number of pages: 59
Finished: 04/06/2020

Abstract:

This study examines the feasibility of a practical implementation of three published deep-learning algorithms developed for point cloud classification and segmentation. This subject is of interest due to the fact that most segmentation of point clouds scenes is currently done manually, which means that there is a great deal to be gained by increasing efficiency through automatic processes.
The study both deals with their accuracy and cost in terms of time needed for training and testing.
In order to understand potential challenges one might be faced with in the case of practical implementation a dataset which differs from standard benchmark dataset was selected for the training and testing of the algorithms. The dataset consists of point cloud data which captures power-line infrastructure and their immediate surroundings. The algorithms are then trained and applied in order to automatically segment the cloud into six distinct classes which include terrain, vegetation, wires, crossbeams and different noise classes.
A range of evaluation metrics are used in order to thoroughly assess the performance of automated segmentation and in order to point out what impact inaccuracy in certain classes might have on the final result as separate raster-based analysis is carried out.

Resumé

Sammenligning af machine-learning klassificeringsmetoder anvendt på højspændingsledninger i punktskysdata

Dette speciale beskæftiger sig med automatisk punktskyssegmentering ved at gennemføre en komparativ analyse af nøjagtigheden og anvendeligheden af tre udvalgte *deep learning* algoritmer, der er udviklet med henblik på segmenterings - og klassificeringsprocesser.

De tre udvalgte algoritmer er PointNet++ [Qi et al., 2017], Superpoint Graph [Landrieu and Simonovsky, 2018] og PointCNN [Li et al., 2018]. De bliver trænet og testet på punktskysdata, der indeholder information vedrørende højspændingsledningskorridorer og det omkringliggende areal. Træning og tests bliver udført med henblik på at segmentere punktskysdata i seks prædefinerede klasser som er: terræn, vegetation, ledninger, højspændingsmaster og to typer af støj. Det anvendte datasæt er ikke offentligt tilgængeligt, men er valgt da det hjælper til at belyse hvilke udfordringer en egentlig implementering ville medføre.

Dette er interessant at undersøge som følge af, at store dele af punktskysprocessering, specielt klassificering og semantisk segmentering, på nuværende tidspunkt udføres manuelt, hvilket kræver mange arbejdstimer og øger tiden det tager at nå fra dataindsamling til endeligt produkt, hvilket en automatisering af klassifikationsprocessen vil kunne effektivisere.

Undersøgelsen anvender en række evalueringsmetoder med henblik på at belyse hvilken algoritme, der er bedst egnet til en egentlig implementering med henblik på at løse en konkret udfordring eller besvare et specifikt spørgsmål. Denne beskæftiger sig både med alment anvendte evalueringsmetoder samt en evaluering med særlig opmærksomhed på nøjagtigheden af den automatisk klassificerede flade som terrænklassen danner sammenlignet med referencedata.

Træning og test af de udvalgte algoritmer resulterede generelt i en segmentering af punktskysdata af høj nøjagtighed med enkelt klasser som når +95% klassificeringsnøjagtighed. Undersøgelsen skaber dermed et vidensgrundlag, der muliggør at vudere hvilke algoritmer, som er bedst egnet til egentlige anvendelsesscenarier udenfor forskning og udvikling.

Dette speciale påviser endvidere, at de undersøgte algoritmer er nået et punkt, hvor nøjagtigheden er høj nok til de enten helt eller delvist kan overtage klassificeringsprocesser, hvilket ville kunne medføre en drastisk reducering i den mængde af ressourcer, der kræves for at gennemføre klassificering af store mængder punktskysdata.

Afslutningsvis diskuteres relevante udfordringer og overvejelser vedrørende specialet samt perspektiverne for fremtidig udvikling af automatiseret punktskysprocessing ved brug af *deep-learning* algoritmer.

Preface

This thesis constitutes the final project on the Geoinformatics master's program at Aalborg University Copenhagen. It has been completed in the spring of 2020 from the 3rd of February until the 6 of June.

The thesis employs an analytical approach to assess the feasibility of automatic point cloud data processing and through an in-depth study of the field results in a thorough understanding of the possibilities and limitations within the field.

I would like to thank my university supervisor Carsten Keßler for guidance and assistance with regards to structuring the thesis.

Furthermore, I would like to thank my colleagues at Niras for providing helpful feedback with regards to the technical aspects of segmentation.

Source Referencing
Source citation and referencing follows the Harvard method. In-text references will be cited with the structure of [Author, Year]. A full list of references can be found at the back of the report. References in this list follow this structure:

- In-text reference
- Author or Organization
- Title
- Type of source
- Publisher or URL
- Year published
- Last date accessed (if internet source)

All figures which are not produced by the author of this paper will be cited.

Contents

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GPS	Global Positioning System
GPU	Graphical Processing Unit
GUI	Graphical User Interface
IoU	Intersection over union
LiDAR	Light detection and ranging
mIoU	mean Intersection over union
ML	Machine Learning
RGB	Red, Green, Blue
SPG	Superpoint Graph
TN	True Negative
TP	True Positive
TT	Total True
VM	Virutal Machine

List of Figures

List of Tables

1 Introduction

Throughout its history, the field of remote sensing has utilised a range of data capturing technologies in order to describe and analyse Earth's physical environments [Campbell and Wynne, 2011]. One of the most recent technologies to be included in remote sensing processes and research is Light Detection And Ranging (LiDAR), a highly flexible and accurate form of data capture that registers a given environment as a volume of points. In its entirety, this volume is called a 'point cloud', which is the type of data this thesis will examine [Weitkamp, 2006].

Machine learning (ML) has also recently seen a marked increase in interest and practical implementation in solving modelling challenges in a wide array of fields and industries. This is possible due to the nature of ML models, which are shaped depending on inputs and outputs that are known to be true [Alpaydin, 2020]. Deep learning (DL), a subset of ML, is a type of modelling that has also seen progress as ML has evolved overall in recent years, including the advent of DL algorithms developed specifically for remote sensing modelling purposes [Zhu et al., 2017].

Because ML and DL modelling is adaptable, it has been possible to combine remote sensing data capture with ML modelling capabilities. This offers insights that would otherwise be difficult or impossible to determine without a combination of the two fields.

LiDAR point clouds can capture a number of attributes related to each point in an environment besides the spatial dimension. Examples include degree of reflectance and how many times the pulse strikes something on the way to the surface of Earth. These additional attributes make it possible to tell what type of object the point belongs to and thereby distinguish object classes, because reflectance values would differ between a point belonging to a leaf on a tree and a metal object such as a car. This makes it possible to discern between the two classes in a volume of points as opposed to a volume where only geometric attributes are available. These additional attributes therefore enrich the point cloud with information that allow for analysis beyond what is possible using XYZ coordinates. This sort of data enrichment is referred to as semantic segmentation and classification, which this thesis explores by examining the extent to which it is possible to segment a point cloud dataset into a number of distinct object classes using machine learning modelling [Grilli et al., 2017]. That is, this thesis deals with semantic segmentation of point clouds. A comparative analysis will be carried out to compare three state-of-the-art DL algorithms with regard to their segmentation accuracy as well as the cost when applied to a large outdoor dataset. More specifically, an inquiry into the time taken and monetary cost of running the required hardware will be used to estimate the total cost of using the algorithms.

1.1 Problem statement

Using LiDAR for registration of a physical space is very efficient and accurate. One challenge many companies face relates to creating an efficient pipeline for processing and modelling the captured data, which often takes a considerable amount of time. Procedures such as stitching together separate point clouds, each depicting rooms in an indoor scene, to have a complete cloud that depicts an entire building can to a large extent be automated. Meanwhile, procedures such as classification and segmentation are still largely done manually and often outsourced to regions with cheap labour, because it is a time-consuming process without a suitable automatic option.

Recent research and newly published algorithms specifically developed with classification and segmentation tasks in mind have made it possible to partially or fully automate these previously time-consuming tasks. This increases the value offered by the data as a number of new results can be produced by analysing the enriched dataset which has been semantically segmented [Grilli et al., 2017].

This thesis explores the possibilities for automatic semantic segmentation of a large outdoor point cloud dataset using DL algorithms and further evaluates the performance of selected algorithms using accuracy, cost and time metrics.

Research questions

In examining the options related to the semantic segmentation of point cloud data, a number of research questions are put forth in order to give direction to the research process.

- Which machine-learning algorithms are suitable for performing the semantic segmentation task?
- What accuracy can be expected when performing automatic semantic segmentation in large-scale outdoor point cloud scenes using ML algorithms?
- What are the costs related to implementing automatic semantic segmentation costs, measured in time taken and price?

Answering the above questions will provide insight into the utility these DL algorithms offer in the case of practical implementation.

1.2 Background

In order to frame the challenges and opportunities that relate to point clouds as a data type in a general sense, a brief overview will follow, that provides basic information regarding point clouds and their characteristics, capturing, processing and storage methods as well as a description of the research value added by this study. Furthermore, a study of current and relevant literature will be carried out with the intent of guiding further work and limiting the scope of the project.

1.2.1 Point clouds and their utility

LiDAR scanning should be considered as a capturing method when planning how to handle any task relating to remote sensing or reality capture. The method provides a high degree of accuracy and can be deployed using a range of instruments, which makes it very flexible whether the task in question requires scans of small, indoor spaces or extensive outdoor areas [Ussyshkin et al., 2011]. The resulting product consists of a set of point clouds, usually tiled to make further processing faster. Point clouds are commonly produced to provide crucial information for tasks related to construction, planning and maintenance of public utilities as well as a number of other applications [Xu et al., 2008][Eitel et al., 2016].

Characteristics

Point clouds consist of a volume of points in three-dimensional space. Each point is usually linked to additional attributes such as intensity and return number. Intensity is an expression of how strong the return pulse is, which is dependent on the material of the struck surface, while return number is a registration of what number of returns a given pulse has.

Depending on the scale of the project, this volume of points often reaches into the hundreds of millions, which results in sizable workloads throughout processing, analysis and visualisation stages. The most basic representation of a point cloud only holds XYZ-coordinates in either a local or projected coordinate system.

Non-euclidean geometry

Data types such as images, text, audio and 3D spatial data, which is typically used in ML and DL applications can be categorised into one of two categories, namely euclidean data and non-euclidean data [Bronstein et al., 2017]. These two types of data have different intrinsic characteristics, which will be covered in the following.

Euclidean data are types of data that adhere to euclidean geometric principles, e.g. the parallel postulate, the shortest path between two points is defined by a straight line, and the sum of internal angles of a triangle sums to 180°[Honsberger, 1995]. This type of data is often represented in a low-dimensional space such as pixels for images and wave signatures to represent audio. In these cases, distance between different data points adhere to the established rules in euclidean geometry.

Non-euclidean data do not adhere to the rules of euclidean geometry and therefore rules such as straight lines representing the shortest distance between two points is not true. The same is the case for the parallel postulate. Many different data representations fit into the category of non-euclidean data, such as manifolds, graphs and point clouds [Monti et al., 2017].

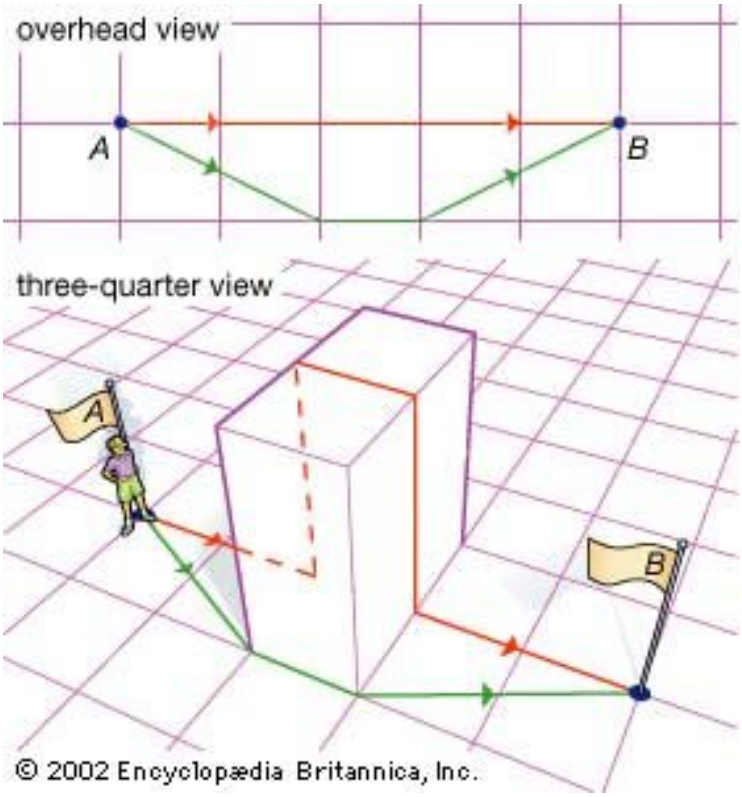


Figure 1.1: Distances in non-euclidean space, source: [Daina Taimina, 2017]

Non-euclidean spaces are divided into hyperbolic spaces and elliptical spaces, which differ in that lines drawn between two data points either diverge or converge. This means that the parallel postulate changes from having one option, drawing a parallel line in euclidean space, to either an infinite number of lines or no lines in hyperbolic and elliptical space, respectively, as seen in figure 1.2 [Krause, 1986].

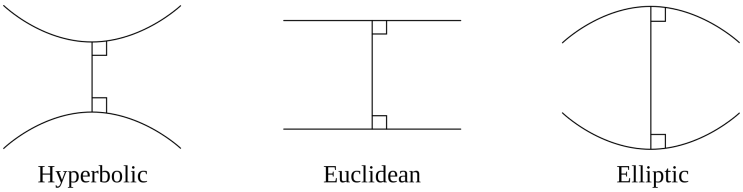


Figure 1.2: Parallel postulate in different spaces, source: [Bowman, 2008]

A case of translating from a non-euclidean domain to a euclidean one is exemplified in transforming a spherical space to a map projection, such as a representation of Earth with a globe to a map projection. Projecting from a globe to a flat surface always distorts either

angles or areas, and this loss of information is also true for other transformations between non-euclidean domains to euclidean ones.

Point clouds belong in the non-euclidean data domain. Rules from euclidean domains do not hold true when applied to the point cloud space, since the shortest distance between two data points is not a straight line due to the requirement that the distance is curved along the space given by the point cloud, as shown in figure 1.3 [Bronstein et al., 2017].

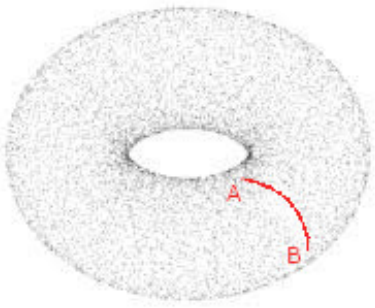


Figure 1.3: Torus shape represented by a point set, source: Public domain

It should be emphasised that the above statement regarding point clouds not adhering to euclidean geometry is an imposed rule. This is put in place to improve algorithm performance, because it is possible, in principle, to measure distances from point to point in a euclidean manner. However, more information is gained by the algorithm if the space is seen as non-euclidean.

Capturing methods

Collecting point cloud data can be done using several different methods, which are all based on the same general principle. The main difference between collection methods is the vehicle the LiDAR sensor is mounted onto. Aerial LiDAR generally refers to when the sensor is mounted onto a plane, and data is collected across a large region. Data can also be collected from a single position on the ground, by mounting the sensor onto a car or a vehicle on tracks or in the form of handheld devices, which is all referred to as terrestrial LiDAR scanning [Esri, 2012]. Some notable differences between terrestrial and aerial scanning are:

Table 1.1: An overview of differing application of aerial LiDAR and terrestrial LiDAR

Aerial LiDAR	**Terrestrial LiDAR**
Low local resolution	High local resolution
Covers large regions	Covers specific areas
Consistent point density	Varying point density

The two collection methods listed above are suited for very different types of LiDAR scanning and highlight the flexibility offered by the capturing method.

LiDAR scanning can briefly be explained as using highly accurate lasers and sensors to register the surrounding environment. The laser spins and pulses while the sensor registers the return time and signal strength of every pulse. Since the instrument knows the angle, speed as well as start and end times of every pulse, which is crucial in order to determine the position of the struck surface relative to the instrument, an absolute position can then be derived using the knowledge of the instrument's position.

When applied in areas with vegetation, another attribute of LiDAR scanning is that every pulse can pass through multiple layers of canopy and the sensor registers the 1st, 2nd and 3rd returns continuing up until the last return [NOAA, 2020]. This provides information both with regards to types of vegetation and the terrain surface, which is always captured by the last return.

Storage options

Point clouds can be stored in a number of different formats, some of which are proprietary, while others are open-access. The most widely used formats are .LAS and .LAZ [Samberg, 2007]. The latter compresses the data to a high degree, making it suitable for transferring and viewing data; however, it is often more cumbersome to process. These are proprietary formats but nonetheless dominant in the industry. Some open-source options include .pcd, .obj, .ply and a handful of other options that are either binary or ascii-based. The .pcd format was developed with point clouds in mind specifically as it is tied to the PCL-project, an ambitious open-source toolbox that can be used for processing point cloud data [Rusu and Cousins, 2011]. The formats .ply and .obj are not created with point clouds in mind, but rather for 3d representation such as meshes.

Point cloud attributes

Besides registering XYZ coordinates, LiDAR scans capture a number of additional attributes related to each point in the cloud. In the following, each of these additional attributes, which are described in the LAS-file specification, will briefly be presented [Safe software, 2015].

Item	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits (bits 0 – 2)	3 bits	*
Number of Returns (given pulse)	3 bits (bits 3 – 5)	3 bits	*
Scan Direction Flag	1 bit (bit 6)	1 bit	*
Edge of Flight Line	1 bit (bit 7)	1 bit	*
Classification	unsigned char	1 byte	*
Scan Angle Rank (-90 to +90) – Left side	char	1 byte	*
User Data	unsigned char	1 byte	
Point Source ID	unsigned short	2 bytes	*

Figure 1.4: Table containing LAS 1.4 attribute specification, source: [Samberg, 2007]

Intensity - from the LAS specification, it is established that intensity is not a requirement, but should, insofar as possible, be included in the result of LiDAR scans. The stored intensity value is an expression of the signal strength of the return pulse which reveals what kind of material the struck surface is made of.

Return Number - An integer specifying what number of return this signal represents in a series of return pulses from a single outgoing pulse.

Number Of Returns - the total number of returns registered from a single outgoing pulse.

Scan Direction Flag - an integer describing the direction of travel the mirror reflecting the laser pulse was travelling at the time of registering the point. Possible values are either 0 or 1, with 1 representing a left-to-right direction and 0 right-to-left.

Edge Of Flight Line - possible values include either 0 or 1. This value is only 1 when the point in question marks the last point before the scan direction changes.

Classification - integer assigned to each point during post-processing of collected data. The integer representing each class is specified in the LAS specification, which ensures that the most common classes are consistently referred to.

Scan Angle Rank - the angle the laser is emitted at. Valid range is 90 to -90, with 0 representing nadir relative to the plane and 90 representing the most extreme right angle relative to the nadir and -90 being opposite.

User Data - a field reserved for user-specified purposes.

Point Source Id - describes the file from which the point in question originates. This can be very useful as points are merged, deleted and moved during post-processing.

GPS Time - can be stored as GPS-seconds-of-the-week, GPS standard time and GPS time. GPS-seconds-of-the-week are reset every week between Saturday and Sunday, GPS standard time is the time defined in LAS-format, and GPS time is defined as GPS standard time + 1,000,000,000 seconds, meaning it counts from January 6, 1980.

RGB - red, green and blue values collected simultaneously with the LiDAR data using separate imagery data, which can later be fused with the point cloud.

All these attributes offer some utility regarding different workflows during processing of data, whether it concerns noise-reduction, file conversion, merging or classification.

This work aims at improving the classification workflow related to point cloud processing by assessing the capabilities of currently available algorithms used for automatically ascribing class labels to clouds.

1.2.2 Classifying point cloud data

Classification of a point cloud can range from being entirely manual to completely automated. Usually a combination of the two is necessary as time and accuracy are both important factors in the classification effort. Further, the required accuracy in many classification tasks cannot be fulfilled through automatic classification. Manual classification carries obvious downsides in that it is resource-intensive, both in a time and cost sense. It also lacks scalability compared to automatic workflows and is not as consistent, since different people might classify a cloud differently [Yastikli and Cetin, 2016].

A number of different options that utilise clustering and/or shape model fitting algorithms have been available in commercial products for some time. However, these options have not leveraged DL so far, despite showing a lot of promise when used for classification tasks [Grilli et al., 2017]. Semi-automated classification is often supplemented by auxiliary data such as known vectors describing wires, road centrelines and a host of other options. In theory, this is not a requirement when using DL algorithms.

Geometric deep learning

Extending traditional deep-learning techniques to 3D applications is not a simple task. Artificial neural networks and convolutional neural networks rely on the fact that the data they usually operate on, such as images, numbers and text, have a simple structure that adheres to euclidean principles. In order to preserve the relationships between data points in 3D representations such as manifolds, voxel grids and point clouds, it is essential that they are not transformed to a lower dimensional space like an image [Masci et al., 2016]. Applying neural networks to 3D data without transforming it to a lower dimensional space offers additional insight as shown in figure 1.5. Here, a kernel passes through every image pixel (right) and every model node (left), calculating distance to neighbouring regions outputting different results.

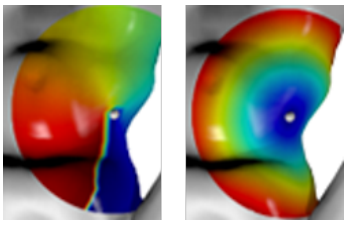


Figure 1.5: Kernel operation on model and image, source: [Flawnson Tong, 2019]

A number of different approaches address the challenges posed by performing deep learning in 3D space, which either operate on derived 3D formats or directly on sensed data such as point clouds.

The earliest attempts were very much inspired by the traditional 2D approach used for images, but instead of observing an object or scene from a single viewpoint, the data is viewed from multiple points of view, each view a 2D representation of the 3D object (1.6, panel 4) [Pang and Neumann, 2016].

Another approach called VoxNet was published some years later. This approach operated on 3D data by shaping the data into a voxel grid, as seen in figure 1.6, panel 2. This representation is reminiscent of a pixel grid from an image, which also highlights the gradual shift from simpler representations to actual 3D representation [Maturana and Scherer, 2015].

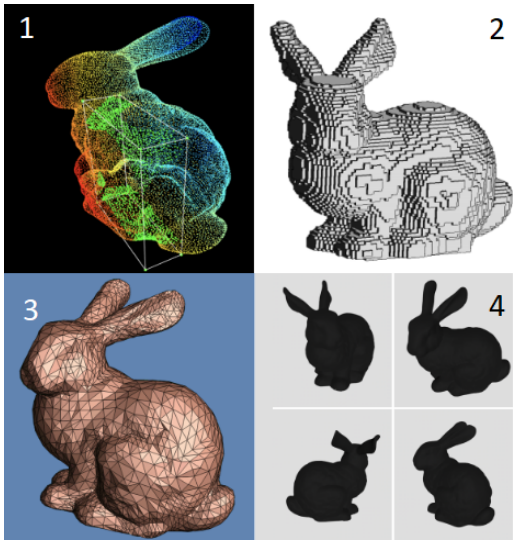


Figure 1.6: Four types of data representations used in geometric deep learning, source: [Flawnson Tong, 2019]

VoxNet in particular utilises a normalised 3D space that is partitioned by a voxel grid and generalizes training data based on the likelyhood of data being present in a given voxel space. While it carries many advantages compared to the multi-view approach, it is sensitive to rotation. This is because the occupancy grid interacts differently once an object is rotated, even though the shape of the object is consistent.

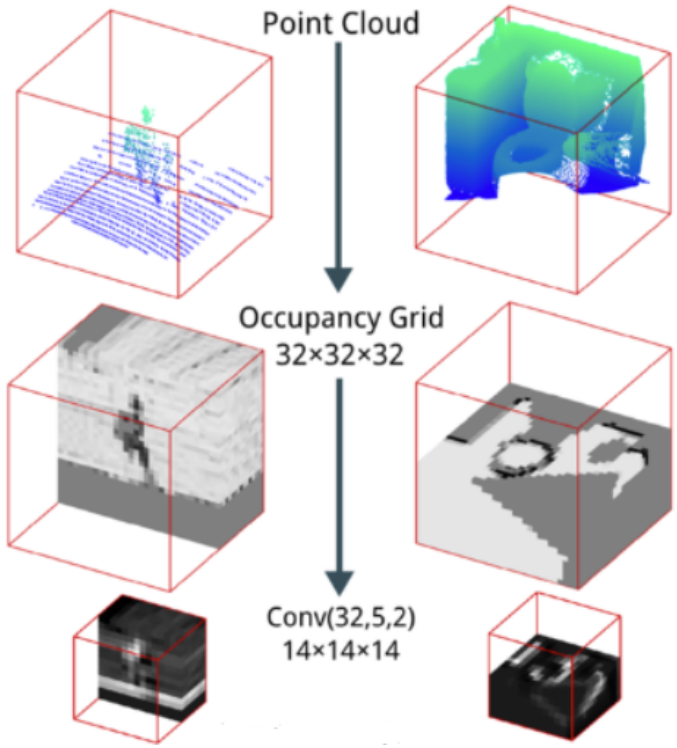


Figure 1.7: Learning based on occupancy grid representation, source: [Maturana and Scherer, 2015]

Panel 1 and 3 shown in figure 1.6 depict point cloud and mesh representations, respectively. These come the closest to depicting the geometry seen in the physical world.

The methodology employed by various algorithms is developed specifically for processing each of the four different representations. Consequently, the methodology differs because of the nature of the data type.

Moving on to mesh and point cloud representations, we arrive at the research field this thesis seeks to examine. In the following, the articles covering the selected algorithms will be described.

1.2.3 State-of-the-art review

The algorithms selected for the classification task will be outlined in the following. Pointnet++ [Qi et al., 2017], Superpoint Graph [Landrieu and Simonovsky, 2018] and PointCNN [Li et al., 2018] all vary in their approach to solving the challenges related to classifying point cloud data. A classification using each of them is therefore expected to yield different results.

Pointnet++

Pointnet++ [Qi et al., 2017] builds on previous work from the same authors. The two papers are considered seminal within the field of point cloud classification due to the novel approach presented and the number of successive papers building upon the Pointnet approach. The original Pointnet paper was published in 2017. The Pointnet++ paper was published a year later, and it sought to address some of the issues Pointnet faced in larger scenes and to construct a global feature vector describing the entirety of the scene. The following paragraph briefly describes PointNet and Pointnet++.

Pointnet was pioneering in their approach to working directly on raw point clouds. In order to work directly on point cloud data, three main requirements must be met. First, the neural network architecture must be able to process an unordered input, because it follows from the nature of point clouds that the result of a volume of points fed in an any sequence ends with the same scene. This is achieved by using symmetric functions within the network, which yields better results than first ordering the points, another possible solution to this challenge. Secondly, the model is designed in a way that derives meaningful relationships between neighbouring points, which is necessary in order to distinguish boundaries between parts and objects. Lastly, the original Pointnet model is invariant to transformations such as translation and rotation. This enables the network to recognise objects in a test set, even when they have been rotated compared to the training samples.

In a practical sense, the PointNet algorithm can perform multiple different tasks such as classification, object part segmentation and semantic scene segmentation, the subject of this thesis being the semantic scene segmentation function of Pointnet.

The approach used by the model is slightly different based on the function, with *object classification* starting out with evenly sampling random points along the surface of the object in question and jittering as well as rotating them all together along the vertical axis. This augmented object serves as the training sample, which exemplifies how the model achieves robust performance even when the object is transformed. The approaches of *Object part segmentation* and *semantic scene segmentation* are similar in that they both seek to split a given point cloud into meaningful subsections. The semantic segmentation task starts by sampling 4096 points in a normalised block of points.

Regardless of the set task, Pointnet moves from the input points across multiple transformations, which include the mentioned jittering, rotation and up-scaling local features using multilayer-perceptrons. The goal is to move towards a descriptive global feature

vector that represents the entire scene, see figure 1.8. Once this global feature vector has been established, it is either used for performing the classification task or alternatively concatenated with the local features after they have been transformed. This enables the scene to be segmented as the network now has access to the information stored in the global feature vector.

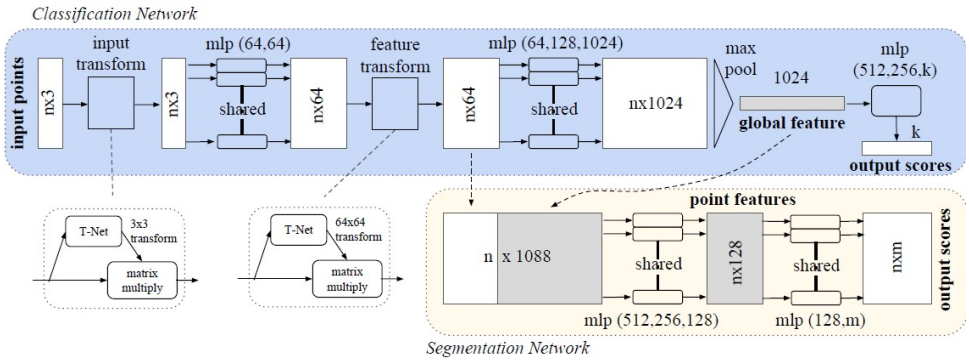


Figure 1.8: Pointnet structure, source: [Qi et al., 2016]

This network ultimately results in output scores for the segmentation and assigns a label to each point.

The one issue faced by Pointnet is that it struggles to learn local structures, resulting in challenges with identifying certain objects in complex scenes. Pointnet++ seeks to address this by using a number of adjustments and additions in handling the point data. Specifically, Pointnet++ recursively applies the original Pointnet algorithm on nested partitions of the scene that are to be classified or segmented. This includes sampling and grouping on the nested partitions and moving from a local neighbourhood of learned features to a global understanding of the scene, which is the result of the understanding gained from training on local neighbourhoods. The architecture of Pointnet++ is shown in figure 1.9, where the hierarchical structure is clear. The classification and segmentation branches are also shown.

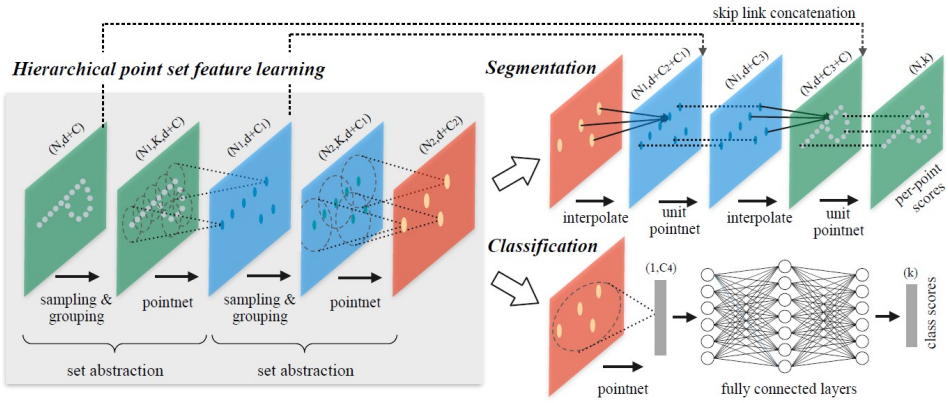


Figure 1.9: Pointnet++ structure, source: [Qi et al., 2017]

It should also be noted that Pointnet++ examines the impact working with non-euclidean distance measures has on segmentation and classification results, as mentioned in section

1.2.1, which shows measurable improvements across a dataset of considerable size.

Superpoint Graph

Superpoint graph [Landrieu and Simonovsky, 2018] has a different approach compared to Pointnet++ as it remarks that the biggest hurdle to 3D segmentation and classification tasks is related to the scale of the input data. In order to address this, a novel approach is demonstrated. This approach draws inspiration from a similar technique used in deep learning applied on images, which utilises superpixels in order to condense the input information while maintaining the information stored in the data. The equivalent term used in this paper is a *superpoint*.

The papers authors accomplish the transformation from the entire input point cloud to a superpoint set by setting some transformation rules based on a few assumptions. An assumption is made that points near each other are more likely to hold similar semantic labels. They add some nuance to this by also requiring that the spatially close point sets should fit a simple primitive, as seen in figure 1.10.

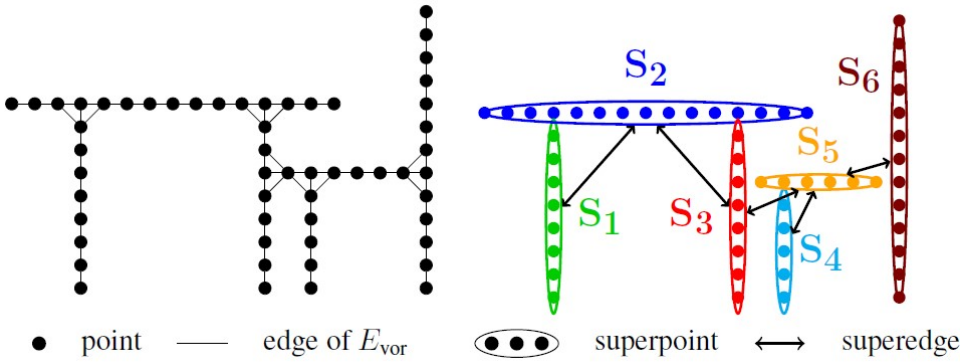


Figure 1.10: Superpoint Method, source: [Landrieu and Simonovsky, 2018]

Once a superpoint set has been calculated, the point features are embedded using Pointnet. For segmentation purposes, they select Edge-conditioned graph convolutions.

The experiments performed up until this point show potential for saving time and processing power by reducing existing datasets to meaningful subsets using a superpoint approach (figure 1.11).

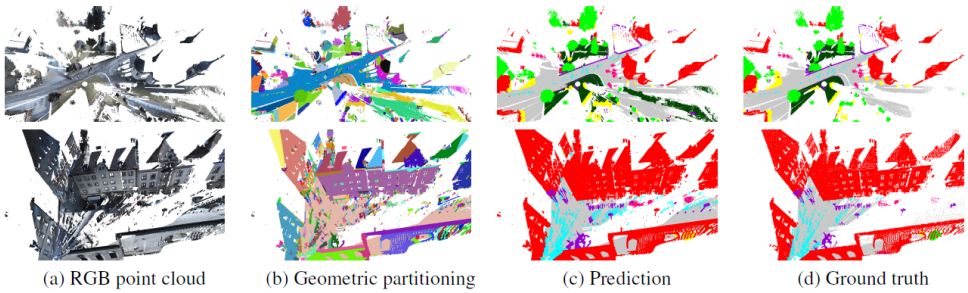


Figure 1.11: Superpoint Segmentation, source: [Landrieu and Simonovsky, 2018]

PointCNN

PointCNN [Li et al., 2018] has a unique approach for tackling the challenges faced when creating an algorithm and network capable of training directly on point cloud data.

The novelty and core concept of the PointCNN approach lies in a convolution method they have named X-conv. This is a method of applying convolutions to point clouds while handling the challenge presented by their inherent irregular structure. It is done by creating trainable convolution kernels that can then be applied on the input features, which consist of a point set of representative points and a point set of neighbouring points. These are iteratively convolved, as seen in figure 1.12

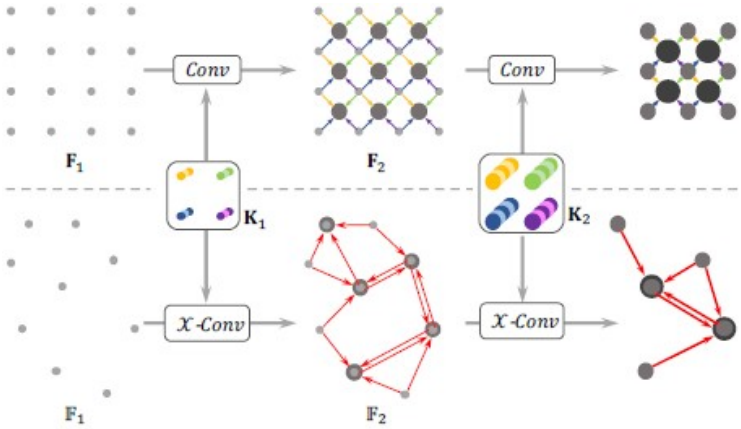


Figure 1.12: X-conv operator, source: [Li et al., 2018]

Furthermore, the network as a whole has separate architectures for classification and segmentation tasks. Both use the X-conv operator for convolution and aggregation of data into fewer representative points. As the convolution progresses, information from a large neighbourhood of points is aggregated into a subset. Then, in the case of classification, a loss is calculated based on the learnt and assigned class. For segmentation purposes the earlier steps in the network are concatenated to the feature vectors in later stages in order to retain both local as well as global information. These operations can be seen in the diagram below (figure 1.13).

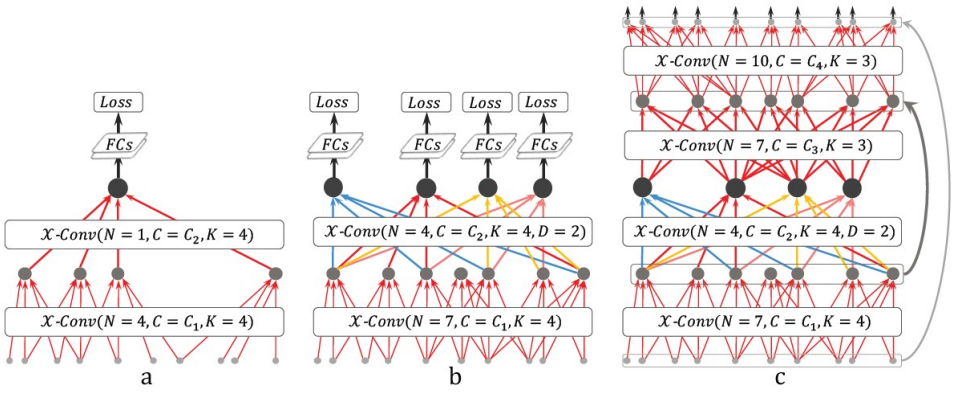


Figure 1.13: PointCNN architecture, source: [Li et al., 2018]

From the varied types of approaches described above, a number of them have proven to be viable in performing accurate classification and segmentation tasks on non-euclidean point data. Their performance and differences when applied on the same dataset, which is not usually used for benchmarking, is of interest when attempting to assess how ready for implementation DL algorithms on 3D data is as a whole.

1.2.4 Project scope

There are a number of avenues and perspectives this thesis could potentially examine if time and resources were not a factor. However, the scope must be limited to some extent, which is why this section will describe the focus of the study.

The thesis will focus on assessing the performance of the three DL algorithms, described in section 1.2.3, when applied on a large outdoor point cloud dataset for semantic segmentation tasks. Accuracy, time and required processing power will be used as evaluation criteria.

The algorithms were selected due to their high reported accuracy on large benchmark datasets, namely Semantic3D, [Rusu, 2010]. This benchmark dataset is often used for evaluating performance. The present study will apply the aforementioned algorithms on a large point cloud dataset that depicts sizeable regions of Sweden's forests and was captured to provide information for maintenance of the Swedish electric utility network. The used dataset is not publicly available, but rather supplied for experimental purposes by Niras A/S.

Method 2

In covering the methodology relevant for the work conducted in this thesis, a number of subjects will be discussed. Firstly, a description of publicly available datasets commonly used and the proprietary one used in this study will be presented. This is followed by an explanation of the programming environment, a point that is deemed worthwhile due to the hurdle presented by establishing suitable environments, which raises a number of requirements related to both software and hardware. Additionally, the tools utilised in preprocessing, intermediate analysis and inspection as well as final evaluation are elaborated upon. Model evaluation ,which includes adjustable parameters, validation methodology and evaluation metrics, will briefly be presented and lead into the final section that describes the scripts and their functions involved in moving from the raw data across the training step and arriving at testing for each of the three selected algorithms.

2.1 Point Cloud data

For some time, it has been the norm to test the performance of new ML and DL algorithms on benchmark datasets. This is done to compare their accuracy and speed to other algorithms, with the intent of demonstrating the viability of the approach [Rusu, 2010]. These come in a number of different formats and data types, and depending on the algorithm being tested, a fitting dataset should be selected.

2.1.1 Benchmark datasets

Common datasets used for ML and DL algorithms in euclidean domains typically hold information about audio, images or text. Below is a short list of datasets that also includes their domain and size.

Table 2.1: Commonly used datasets in euclidean ML and DL testing. source: [Defferrard et al., 2016], [Koehn, 2005], [Deng, 2012], [Krizhevsky et al., 2009]
†in thousands ‡training / testing split

Dataset	Task	Sample count†‡	Format
Mnist	Image processing	60 / 10	28x28 pxs
CIFAR10	Image processing	50 / 10	32x32 pxs
FMA	Audio processing	64 / 16	30 secs
Europarl	Natural language processing	1.900 / 45	Sentences

Since non-euclidean point cloud datasets still find themselves within a niche with regard to modern DL research, these benchmark datasets are not as common. However, a number of these are publicly available, and newly published algorithms are tested on one or multiple of these datasets. The benchmark datasets vary in size and are usually either intended for classification, semantic segmentation and/or object detection, as seen in table 2.2.

Table 2.2: Commonly used datasets in non-euclidean ML and DL testing. source: [Chang et al., 2015], [Armeni et al., 2017], [Geiger et al., 2012], [Hackel et al., 2017]

Dataset	Task	File size	Scene
Shapenet	Classification	30GB	Unique models
S3DIS	Classification	766GB	Large-scale indoor
KITTI3D	Object detection	30GB	Large-scale outdoor
Semantic 3D	Semantic segmentation	100GB	Large-scale outdoor

The datasets which resembles the test dataset used in this study the most, is Semantic3D, which depicts large outdoor urban scenes with multiple classes including terrain, low and high vegetation, scanning artifacts, buildings and cars.

2.1.2 Non-benchmark datasets

The dataset used for testing in this study is not publicly available. In its entirety, it depicts a number of powerline corridors in Sweden and was captured to support maintenance work of the electric utility network.

For the purposes of the study, it was decided to work with a proprietary dataset in order to more accurately determine whether or not DL algorithms at this point in time are ready for implementation in commercial workflows that use and process point cloud data. With this being the case, aspects such as accuracy, especially with regards to modelling terrain and noise correctly, time and cost of processing power, are all relevant factors.

The dataset is split according to the specific powerline corridor they belong to and further split into square tiles of 1000m. The total amount of data is greater than 500GB. However, for development and processing speed purposes, a choice was made in limiting the training and testing data volume to only include a subset of the total available data, despite including more data being expected to generate results of higher accuracy. This subset consists of two selected corridors. One in its entirety acts as training data (LG85), and the other (LG105) provides testing tiles for segmentation purposes and makes up a combined 60GB of point cloud data in .las-format. The testing data covers a stretch of 2,000m with an area of 80,000m^2.

The point clouds hold XYZ, intensity, ReturnNumber, NumberOfReturns, ScanDirectionFlag, EdgeOfFlightLine, Classification, ScanAngleRank, UserData, PointSourceID, GpsTime and RGB values.

2.2 Environment

Because of the specific requirements set by the algorithms, a number of steps must be taken to ensure that they function as intended. Because of this, an outline of the required process will be provided to ensure reproducibility of the presented results, which is limited to the function of the algorithms due to the dataset not being publicly available. The description of the setup will cover general requirements like operating systems (OS), virtual machines, hardware as well as more specific aspect such as firmware, python versioning, compilation tools and the python packages that form the framework for the algorithms.

2.2.1 Virtual machine and OS

The environment used for preprocessing data and running them through the algorithm and evaluating the results is made up of a number of components. The collaboration with Niras and their IT-infrastructure incentivises using Windows OS for file management and sharing. However, it is not feasible to stay within the Windows environment throughout the process due to the algorithms and their component parts not being developed with Windows compatibility. For this reason, a machine running a Linux based OS is needed. Google's Cloud Computing (GCC) platform was selected for this purpose [Krishnan and Gonzalez, 2015]. The following virtual machine (VM) was put together on GCC in order to run the algorithms, which are somewhat computationally expensive and therefore require high-end hardware.

- **CPU** 8vCPUs and 30GB memory
- **GPU** 1 NVIDIA Tesla K80 12GB memory
- **Linux distribution** CentOS 7
- **Disks** 10GB OS disk & 500GB SSD

The essential parts of the specifications listed above is the GPU and the high amount of memory. This is needed as the algorithms are memory-intensive and leverage the fact that using the GPU as opposed to the CPU is much more efficient. Furthermore, all the used algorithms are developed in Linux environments, and some of their components need to be compiled using Linux compilation tools.

In order to access and interact with the virtual machine, GCC offers two options as standard. One is SSH-tunnelling and the other is a platform specific shell called Cloud Shell [Krishnan and Gonzalez, 2015]. In this case, SSH-tunnelling was used to install remote desktop software and a Gnome desktop, which made development and file sharing feasible on the VM.

2.2.2 Software environment

Once the general environment is set up as above or in a similar manner, the python environment along with relevant frameworks and compilation tools should be addressed. More specific information can be found on GitHub [Jensen, 2020].

Firstly, a CUDA version that is both compatible with the selected GPU and the algorithm should be installed along with a suitable driver. This might not function as intended with the most up-to-date driver and CUDA combination as the algorithms are not necessarily kept updated.

Having completed the above steps successfully leads into installing Python. In this case, using Conda is highly advisable, since running the algorithms all depend on different versions of both Python [Foundation, 2020], TensorFlow [Abadi et al., 2015] and Pytorch [Paszke et al., 2017] as well as number of other minor python packages. For this reason, it is crucial to keep environments separate. Appropriate TensorFlow and Pytorch packages should be installed in separate environments, each one intended for running each of the algorithms.

Once this is established, compilation tools such as CMake, of the correct version, should be installed as the some of the algorithms utilise a function from TensorFlow called custom operators, which need to be compiled. These operators are especially particular about versioning of compilation tools, Python packages and CUDA versions, while being central to running the algorithms.

2.3 Tools

The software tools used in this study will briefly be described below.

The selected coding and debugging environment was Virtual Studio Code, which allows for a lot of flexibility and is an efficient approach to Python coding and troubleshooting.

In order to gain a more intuitive understanding of point cloud data scenes and relevant classes, CloudCompare [software, 2020] was selected as a tool for inspection and basic processing. It offers file conversion and processing operations as well as relevant analysis tools.

For batch processing operations, a combination of QGIS [Team, 2020], PDAL [Contributors, 2018] and LAStools [Isenburg, 2020] was used to convert between file types and clip the data along a stored shapefile to reduce the size of the dataset and as a way of dealing with class imbalance issues.

2.4 Model Evaluation

This section will cover the methodology that will be employed to assess model performance. General concepts like cross-validation, hyperparameter optimisation and relevant evaluation metrics will be described and provide a background for the results presented in chapter 3.

2.4.1 Cross-validation

A typical method for evaluating ML and DL algorithms, cross-validation is simply done by withholding a subset of all the available data at the time of training and then introducing the subset at later point for testing purposes [Scikit-learn, 2019]. There are a number of ways one might go about implementing this type of evaluation method, and some of the common methods will be described in the following.

Exhaustive and non-exhaustive make up the general categories that validation methods can fall into [Guo et al., 2017]. As the names suggest, exhaustive methods train and validate across all possible combinations. Non-exhaustive methods use sampling techniques in order to gather representative subsamples that continue to describe the performance of the trained model while keeping the computation requirements low. This study deals with non-exhaustive validation and will therefore cover some of the common methods and explain the one selected in this project.

K-fold cross-validation

K-fold cross-validation is a type of validation that is initialised by deciding on a number of partitions the validation should be performed across. Typical splits are 10 or 5-fold validations that each train a model on all the available data except for the data withheld for validation, which the model is then evaluated on. The performance of the model found through the validation is kept, and a new training and folding is done. After all the models are trained and evaluated, the performance across validation runs is averaged in order to gain nuanced insight into the model performance, see figure 2.1.

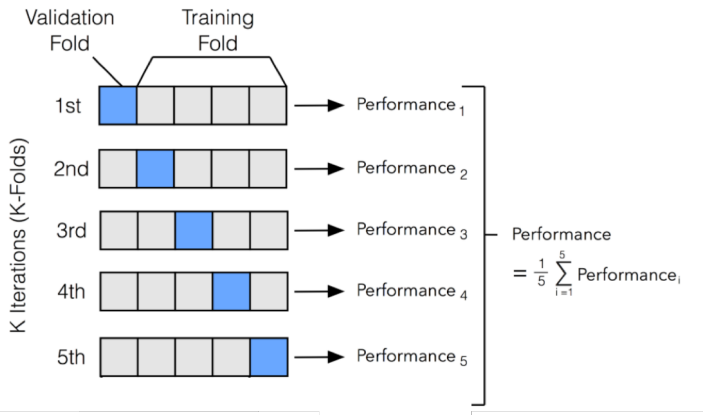


Figure 2.1: 5-fold cross-validation visualized, source: [Scikit-learn, 2019]

Monte-Carlo cross-validation

Monte-Carlo cross-validation is similar to the k-fold method in that multiple runs are performed and then averaged. However, it differs in that the validation subset is randomly sampled and the remaining data used for training. This means that the validation is not limited by the number of partitions. As the validation continues to run, it will eventually approach the results that exhaustive validation methods would have reached, as seen in figure 2.2.

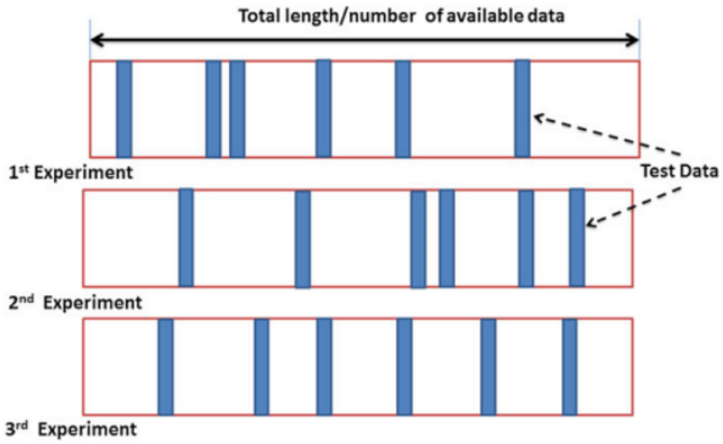


Figure 2.2: Three Monte-Carlo runs, source: [Remesan and Mathew, 2016]

Holdout cross-validation

Holdout cross-validation is a simple approach that is suitable for early development as speed is often a concern. It can be described as being the same as k-fold cross-validation, but having only a single partition. In this partition, the data for validation is the holdout data. Because this data may not be representative for the entirety of the data, this makes the performance measurement somewhat uncertain as the method lacks any kind of averaging. However, it does save considerable time since only one model is trained and validated.

This is also the approach selected in this study due to time constraints, which leaves room for further studies in this area (figure 2.3).

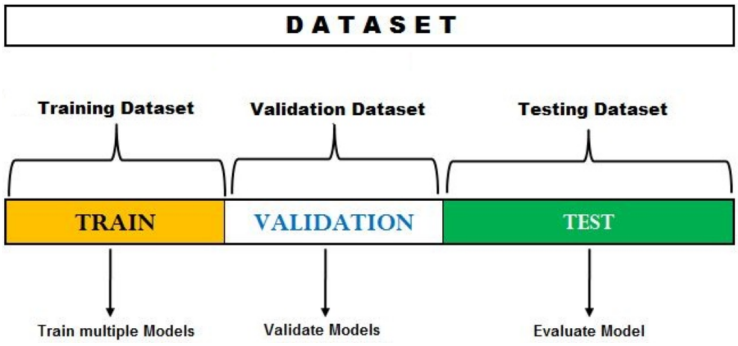


Figure 2.3: Holdout validation method, source: [Archish Rai Kapil., 2018]

2.4.2 Hyperparameter optimisation

The term hyperparameter optimisation deals with the the process of adjusting parameters in various ways in order to find an optimal combination of parameter values, which is evaluated based on a loss calculation [Pier Paolo Ippolito, 2019].

In short, loss is a single metric that describes the degree to which a model manages to predict class labels correctly across the given number of samples contained in a dataset. It manages to summarise whether or not a given modelling effort is better or worse than the previous by taking all variable adjustments into account.

There are a couple of ways to categorise hyperparameter optimisation, which will be explained below. There are also more specific ways, which are going to be outlined because they are commonly used in ML and DL applications.

Grid search

Grid search is an intuitive method for hyperparameter optimisation, which as an input takes a set of possible values for each parameter. These possible values are then used pairwise as settings for training a model. Once all possible combinations are exhausted, the best combination from the defined value sets can be chosen and used for final training. Downsides to this approach include a requirement for thorough understanding of the impact of each parameter on the model. Furthermore, once the parameters become numerous, this method becomes infeasible [Pier Paolo Ippolito, 2019].

Random search

Random search takes a range of values between which possible values for each parameter may lie. Additionally, a number of tests across these values are defined, making this approach more flexible than grid search. Random search has a downside in that the approach is naive, since no information regarding previous runs is stored. Therefore, it does not incorporate prior results in future tests [Pier Paolo Ippolito, 2019].

Stochastic gradient descent

An intuitive way of understanding a stochastic gradient descent is imaging a "landscape of loss", as seen in figure 2.4. The optimisation traverses the landscape by taking steps, the size of which equals the learning rate. These can be dynamic and based on the steepness of the underlying gradient or a static value with fixed step size [Chi-Feng Wang, 2018]. Using this continuous monitoring of performance based on the loss gradient lets the optimiser find a minimum or "valley" where the loss is low.

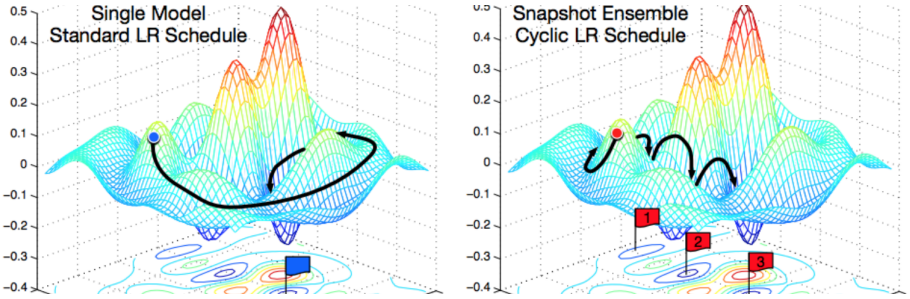


Figure 2.4: Stochastic gradient descent and landscape of loss, source: [Chi-Feng Wang, 2018]

One such optimisation algorithm is the *ADAM optimiser*, which not only detects gradients at given point in the loss landscape, but also the gradient type. This makes it suitable in many different applications, and ADAM is also the optimisation algorithm selected for this study. It is used across all three examined segmentation algorithms [Kingma and Ba, 2014].

2.4.3 Evaluation metrics

Some general considerations should be done when evaluating segmentation tasks are mentioned in the following. Firstly, it is central in the evaluation of algorithm performance to select meaningful metrics that are suitable for describing the quality of the process. Secondly, the class balance should be a point of interest throughout the evaluation, and any steps which can be taken to address class imbalance should be considered.

Confusion matrix

The confusion matrix is a commonly used method for evaluating performance. It is also called the error matrix, and its structure is shown below in table 2.3. It is very versatile and will therefore form a basic understanding of the algorithm performance of each algorithm examined in this study. A comparison between correctly and incorrectly labelled true/false class (TP, FP, TN, FN) is summarised horizontally and vertically. The total number of class occurrences for predicted and reference sites (P1, P0, R1, R0) is summed in the right-most column and bottom row. Using these metrics, a number of other, often more descriptive metrics, can be derived. The nearest derivatives are called omission error and commission error. The omission error describes the total number of reference samples that are not included in set correctly labelled class samples. Commission error describes the percentage of samples that are incorrectly included in a given class. This also means that omission error in one class results in a commission error in another class [Aggarwal, 2004].

Table 2.3: Confusion matrix structure for binary classification algorithms.

	Reference (R)			
	Class	1	0	Total
Predicted (P)	1	TP	FP	P 1
	0	FN	TN	P 0
	Total	R 1	R 0	TS

Accuracy

Overall accuracy can be used as a metric for performance and is usually included whenever algorithms are evaluated. It is simply a percentage calculated by total number correctly assigned labels over the total number of assigned labels. It is calculated as shown in equation 2.1.

$$OA = (TN + TP)/(TN + TP + FN + FP) \tag{2.1}$$

Overall accuracy should not be used as the only metric to describe performance, since it does a poor job of describing performance in classification tasks that deal with an imbalanced dataset. A simple way of describing the shortcoming solely using accuracy can be explained by the case of a binary classification, as seen in table 2.4.

Table 2.4: Simple binary classification with 90% accuracy.

	Reference (R)			
	Class	1	0	Total
Predicted (P)	1	0	0	0
	0	10	90	100
	Total	10	90	100

And using equation 2.1 we get a high accuracy score, as seen in 2.2, even though the classifier puts all data points into class 0 and therefore does a very poor job of classifying the data into the two desired classes.

$$(90 + 0)/(90 + 0 + 10 + 0) = 0.9 \tag{2.2}$$

Jaccard index

A metric which is often used in segmentation tasks is the Jaccard Index, also sometimes called intersection-over-union (IoU). This metric is a better descriptor of an algorithm's performance in segmentation tasks. The metric is calculated by first calculating the intersection of class 1 ($C1_I$) and then dividing by the union. This is done across classes in order to calculate a mean value (mIoU), as shown in equation 2.3.

$$\begin{aligned} &Class\ 0 && Class\ 1 \\ &I = 90 && I = 0 \\ &U = (90 + 100) - 90 = 100 && U = (0 + 10) - 0 = 10 \\ &IoU = 90/100 = 0.9 && IoU = 0/10 = 0.0 \end{aligned} \tag{2.3}$$

$$mIoU = (0.9+0.0)/2 = 0.45$$

This metric is better suited to describe the actual performance of the segmentation, since both classes are given equal importance regardless of the portion they take up in the dataset in absolute terms [Taha and Hanbury, 2015].

2.5 Python scripts explained

In order to provide a better understanding of the operations, an overview of relevant functions in the involved python scripts will be presented.

2.5.1 Preprocessing scripts

Due to the point cloud files being stored in .las format, some preprocessing was necessary in order to ensure that all algorithms received the input format they expected. In order to do this preprocessing, a couple of different tools were used, which is visualised below in figure 2.5.

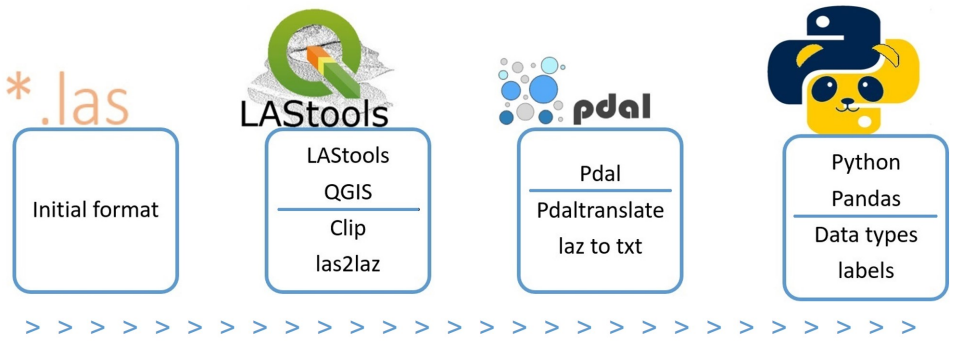


Figure 2.5: Preprocessing pipeline from .las to algorithm-ready ascii-files

The start of the processing pipeline begins with the .las files with classes manually assigned. Some of these will be stripped of their classification info to be used for testing, while the majority will end up retaining their classification info for training purposes. The first operation performed on the data, besides backing up the original, is clipping the clouds using auxiliary vector data, which describes the powerline position. Clipping the data achieves lower processing time due to reducing volume and increasing class balance as the terrain and vegetation classes are far more prevalent than wire and pylon classes further from the powerlines. It should be noted that the class imbalance is still an issue that must be addressed in other ways, but the clip operation alleviates it to a certain extent. Concretely, the clip was performed using LAStools running through QGIS interface, which provides an intuitive GUI and can be used to batch process a list of .las files with some modification. The output of these are stored as .laz to reduce data volume in this intermediate step.

The next step uses Pdal to convert files from .laz to .txt format using the Pdaltranslate function. This is also done by batch processing all *.laz files in relevant folders to *.txt files. This increases the file size substantially due to a lack of compression and transitioning from a binary storage format. However, it is a necessary step in order to use the algorithms without major alterations.

The last step involves reading the .txt files and using the python packages Pandas [Reback et al., 2020] to modify the contents of the files to ensure that the data types match the expected input of the algorithm. Furthermore, all the algorithms rely mostly on X, Y, Z and to some extent intensity and RGB informationl. For this reason, all information that does not relate to these is dropped. This is expected to change once algorithms become more advanced, because a lot of information regarding a scene surely could be gained from including all available data. Pandas is a very flexible python package and would in combination with other available python packages have enabled all the work and processing to be done using python scripts exclusively. The reason for using tools such as Lastools and Pdal is due to their ease of use.

Table 2.5: Overview of tools and packages

Name	**Function**	**Additional information**
LasTools	A point cloud processing toolbox	rapidlasso.com/lastools/
QGIS	Open-source geographic information system	qgis.osgeo.org
Python	Open-source programming language	www.python.org
Pandas	Python package for data analysis	pandas.pydata.org/about/

Having covered required preprocessing was deemed worthwhile to provide some insight into the considerations and steps that should be taken in order to reach a functioning starting point for the algorithms. Next, each of the algorithms and their codebase will be covered non-exhaustively, only highlighting scripts and functions deemed relevant. The

entire codebase for each algorithm can be found in their respective GitHub repositories, which are cited in the source for each algorithm.

2.5.2 PointNet++

source: [ISL, 2019]

The codebase utilised for testing the PointNet++ algorithm is not from the same group who originally created the algorithm. Rather, code developed by Intel Intelligence Systems labs was used due to a more streamlined implementation process. The concrete implementation and selected modifications are covered in the following.

Ensuring that the established python environment complies with package versions specified in the Github repository is required to ensure that each step of the implementation runs expectedly. Concretely, this means that TensorFlow installation, C and C++ compilers as well as CUDA installation and CMake should all function with each other.

Following the directions provided in the repository, the process starts by downloading the benchmark dataset Semantic3d, but given that a custom dataset is used in this case, this can be skipped.

The preprocessing script should be run next, which converts the ascii-files to .pcd files, a much more efficient format for storage and processing purposes.

In order to further speed up the coming training steps, a downsampling script should be run on the data. The one used in this case downsamples by removing all points with a label, which indicates the point is unlabelled. To have this script and all following scripts working as expected, the unlabelled points marked for removal should be labelled 0. Additional points are removed by a voxel sampling the points by averaging them inside each voxel and outputting this as one point.

Using the algorithm also involved compiling custom TensorFlow operators. Following the instructions, the user should ensure a functioning TensorFlow GPU, Cmake and CUDA installation. After compiling successfully, a sanity test should be performed to make sure the operators function as expected. The operators are used in the next script, which involves training.

The training step loads batches from the specified training set and periodically loads validation batches in order to evaluate performance every few epochs. As the network starts out with knowing nothing about the classes and scene in general, the performance is very poor. However, the loss function reaches lower values over a number of epochs, and performance improves. This will be covered in more depth in section 3.

As the performance is evaluated every few epochs, the model performing the best on the data is saved, which can be loaded in later prediction steps.

Having a trained model allows for predicting on cloud datasets that have not been seen by the model so far. The geometries for each class have been generalised by the network and transformation to a stage where they can be applied to unseen data and correctly predict the sets of points belonging to a certain class.

Since the downsampling function referenced in an earlier script retains the index of the points contained in each voxel, the newly predicted classes can be transferred back onto the original files using the provided interpolation script. With this final step, an automatic semantic segmentation has been performed and can be applied to any future point cloud dataset with a similar environment.

2.5.3 Superpoint graph

source: [Landrieu, 2020]

The starting point of running Superpoint graph is mostly identical to PointNet++ with the exception that terrain class should be labelled 0, whereas this label was reserved for unlabelled points in PointNet++. The file structure should otherwise be the same.

Establishing a separate conda environment is required to get Superpoint graph running, since the versioning required by this algorithm differs from PointNet++, and conflicts will arise if they are installed in the same environment. Besides installing Pytorch and CUDA as well as additional packages, which include libraries such as boost and eigen, this algorithm also needs to have custom operators compiled, which are used in the partitioning and downsampling stages.

The files are first partitioned into .h5 files using the partitioning script. This requires a lot of memory, which is the reason why it was recommended earlier. .h5 files are highly efficient in storing point cloud data, but are more cumbersome to interact with than ascii-files. The points that are actually converted to .h5 are also only a subset of the loaded files, since this algorithm also uses a voxel-based downsampling function to reduce the number of points.

Having downsampled the points, the Superpoint graph can be computed for each file, which is structured as explained in section 1.2.3.

With the superpoint graphs established, the algorithm can efficiently train on a markedly reduced dataset while maintaining most of the information crucial to describing the geometry present in the scene. Due to volume of data being low, the training stage can be executed far faster than what was the case for PointNet++.

2.5.4 PointCNN

source: [Yangyan Li and Chen, 2020]

PointCNN as a whole tool set can perform a number of functions with regard to both classification and segmentation tasks. As standard, the algorithm functions with a number of benchmark datasets. For semantic segmentation, the scripts related to the Semantic3D dataset will be modified in order to run as expected with the non-benchmark dataset.

Similarly to what was outlined in the two previous procedures, the ascii-files containing the point cloud data are converted to .h5 files using a preparation python script. These .h5 files further split the tiled ascii-files into parts. In order to keep track of the way these fit together, a second script is run to generate filelists pointing to training, validation and testing data.

From this point, training can begin after adjusting the settings to have the expected input match the actual input with regard to data dimensions and data types. The training can be monitored by using Tensorboard, which generates charts containing information on loss and accuracy for both training and validation data.

Having trained the algorithm, testing can begin by loading a checkpoint, which contains a saved state of the previously trained model. This is then recalled and applied to the testing data. Having run the testing script, the previously split data can be written back into the original state using a merging script, which saves the .h5 files in a .ply format to make processing more straightforward.

Results 3

The following chapter will detail the specific stages of automatic segmentation using the three algorithms outlined in section 1.2.3.

3.1 Neural network segmentation

This section will present relevant metrics and visualise a number of tiles from the semantic segmentation of the powerline dataset. The presented metrics will differ due to the fact that the toolsets that come with each algorithm do not provide the same options for outputting certain metrics. However, there is enough overlap between the three algorithms to perform a thorough comparison with regard to performance in accuracy as well as time and cost.

3.1.1 Pointnet++

Pointnet++ had a training duration of 19 hours, 34 minutes and 33 seconds. During this period, the algorithm trained through 25 epochs by studying the training set. It also went through five additional epochs which used validation data for evaluation purposes.

Epoch count is the most intuitive metric for measuring how far along the algorithm is in the training process, but this was not available for all aspects of training. Some of the following charts therefore express the process by "wall time", which is simply time as tenths of a second since the process began.

While training, the algorithm attempts to minimise the loss value by seeking a minimum using the ADAM optimiser [Kingma and Ba, 2014]. This process can be seen below in figure 3.1, where learning rate and decay is adjusted as the step increases. Once the step resets and a new minimum for the loss functions is found, the learning rate and decay resets to an earlier value. Step, loss and learning rate are scaled by a factor 1 x 10^1, 1 x 10^6 and 1 x 10^9, respectively, in order to better visualise and compare all four parameters.

Figure 3.1: Pointnet++ loss minimisation

One should keep in mind that even though learning rate, decay and steps are fluctuating, it is with the end goal of minimising loss.

Commenting on some of the interesting and expected behaviour observed in visualising the process, it is notable that loss fluctuates and increases starkly as steps reset. This is the expected behaviour as the algorithm attempts to find a point in the "landscape of loss" with a minimal loss value and manages to reduce loss notably as the process comes to its end point. The increase in decay and decrease in learning rate happens synchronously.

With the segmentation complete, accuracy metrics are available for both training and validation phases. For training, both general metrics such as overall accuracy and mean class IoU will be reported, as seen in figure 3.2. Additionally, Pointnet++ provides IoU metrics regarding each class, which are also interesting to examine.

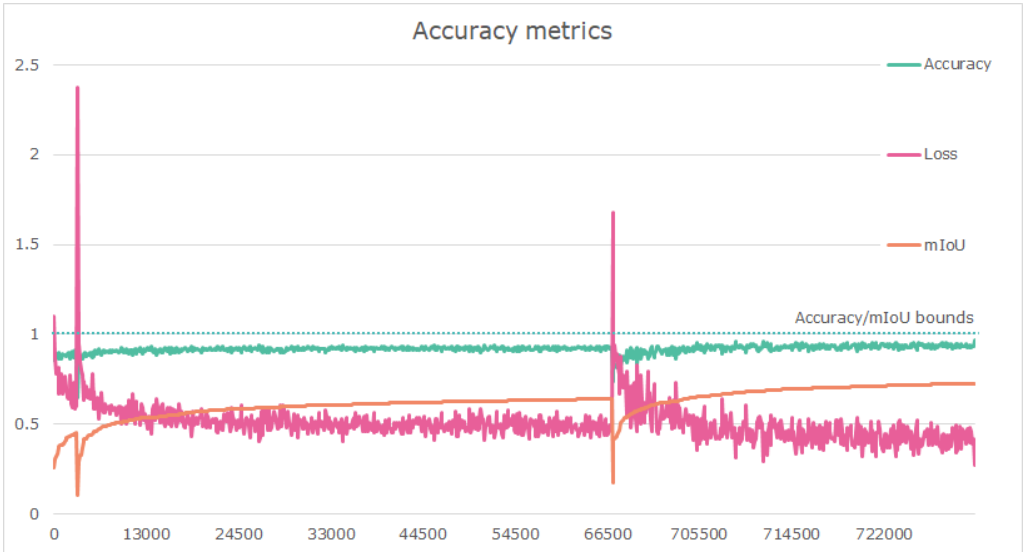


Figure 3.2: Pointnet++ accuracy metrics

The figure shows a small increase in accuracy over the course of the training process approaching 1.0 as marked by the dashed horizontal line labelled accuracy/mIoU bounds. The decrease in loss is identical to the one displayed in figure 3.1. As expected, sudden decreases are seen in both mIoU and accuracy as loss momentarily increases. mIoU ends at a noticeably higher point than the beginning of the training and seemingly increases at a steady rate. Had more training time been allowed, better performance would with all likelihood have been achieved. The difference in the rate of improvement for accuracy and mIoU is caused by the characteristics of the classes in question and the class imbalance within the dataset.

Examining the training process on a per epoch basis, seen in figure 3.3, shows similar picture, but the process is seen as more smooth.

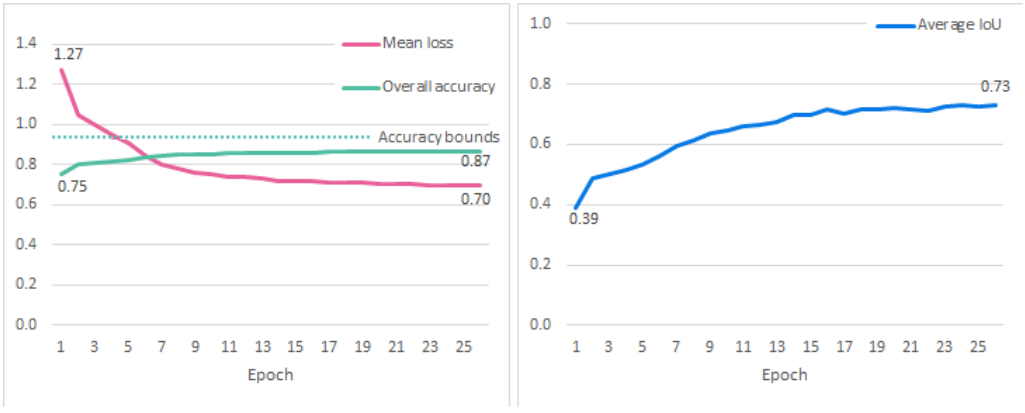


Figure 3.3: Pointnet++ overall performance per epoch, loss and accuracy (left) and average IoU (right)

The following charts visualise the models performance on each class, beginning with the two largest classes, terrain and vegetation, seen to the right in figure 3.4.

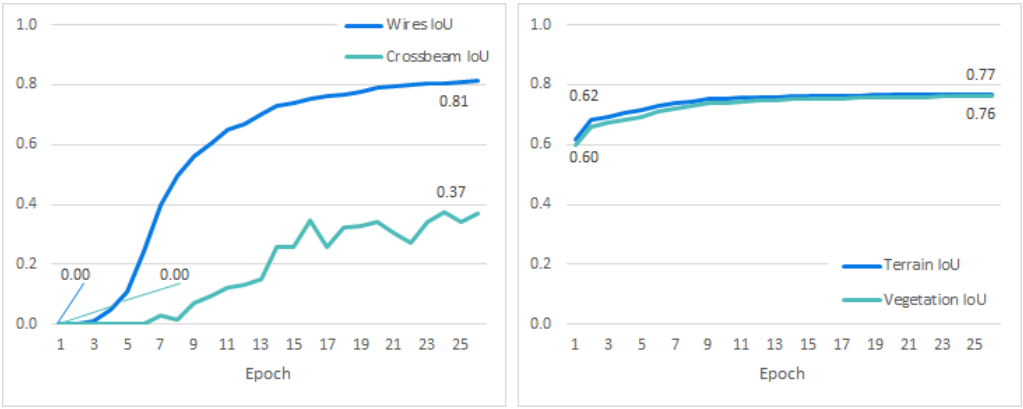


Figure 3.4: Pointnet++ performance on classes wires and crossbeams (left) terrain and vegetation (right)

The pattern seen for these classes is a simple increase in IoU that is initially sharp and then levels off. The two classes are generally predicted with comparable IoU. The geometrical characteristics of the classes are very different from others in the dataset and plenty of training points are available, which raises some questions as to why the IoU is not higher. The start point is unsurprisingly very high for both classes when taking into consideration that the algorithm does not know anything at this point in training. This is due to the two classes being so dominant. Given a random distribution of class label assignments, a large number of labels will therefore be correctly assigned by chance.

Examining the wires and crossbeam classes, a more interesting pattern reveals itself. Starting out, the algorithm knows nothing about the classes in question, and due to this, it completely misses by randomly assigning labels due to the low degree of occurrence of these in the dataset. The actual start is not 0.00, but this is displayed in the chart due to rounding. For the wires, a substantial increase in IoU is seen across epochs with quick improvements to start with and then more gradual improvement later on. However, even as the process ends, the rate of improvement implies better performance could be achieved on the class given more training time. The pattern seen for crossbeams is similar in that it starts out at very low IoU and takes a few epochs to begin showing improvement. Once the training model establishes the knowledge to determine which points belong to the crossbeam class, the rate of improvement increases, albeit in a more sporadic manner than what was the case for the wires class. This is with a high degree of certainty caused by the low number of training points belonging to the class and their geometrical characteristics. Here, a case for further training is the strongest due to the change in IoU continuing to fluctuate and the low IoU in absolute terms.

A final look at IoU metrics for each class is the noise classes, which is split in noise lower and noise upper, as seen in table 3.5. The noise is present due to atmospheric disturbances and the LiDAR equipment receiving impure returns, which is very common and an aspect of data collection that should be dealt with in all cases. This noise is usually not present in benchmark data. If it is present, the amount is marginal compared to the noise seen here. The lower noise is very consistent in its geometry while the upper noise is much more varied and mixes with the vegetation and terrain class in particular.

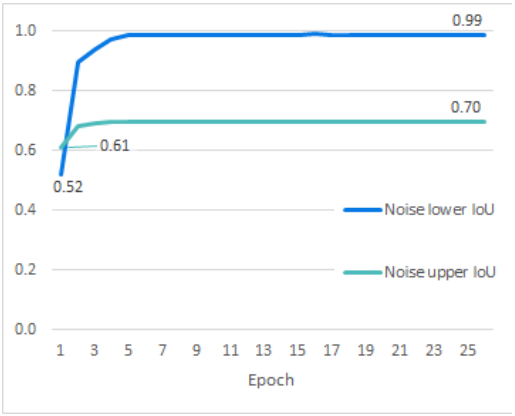


Figure 3.5: Pointnet++ performance on classes noise upper and lower

In the following, a visualisation of the PointNet++ segmentation performance will be presented.

Visualising the segmentation

Visualising the segmentation provides the most intuitive evaluation tool and highlights obvious areas which might need improvements, but it does not inform the user in detail regarding which classes are mistakenly given an incorrect label.

In figure 3.6, the result indicates that the segmentation has largely been successful in assigning the correct label to each point. The most obvious class that is wrongly classified is the crossbeam class (orange), which contains a fair proportion of wrongly classified wire points (yellow). The cloud above the terrain is the noise upper class (red), and the blue and green points are terrain and vegetation.

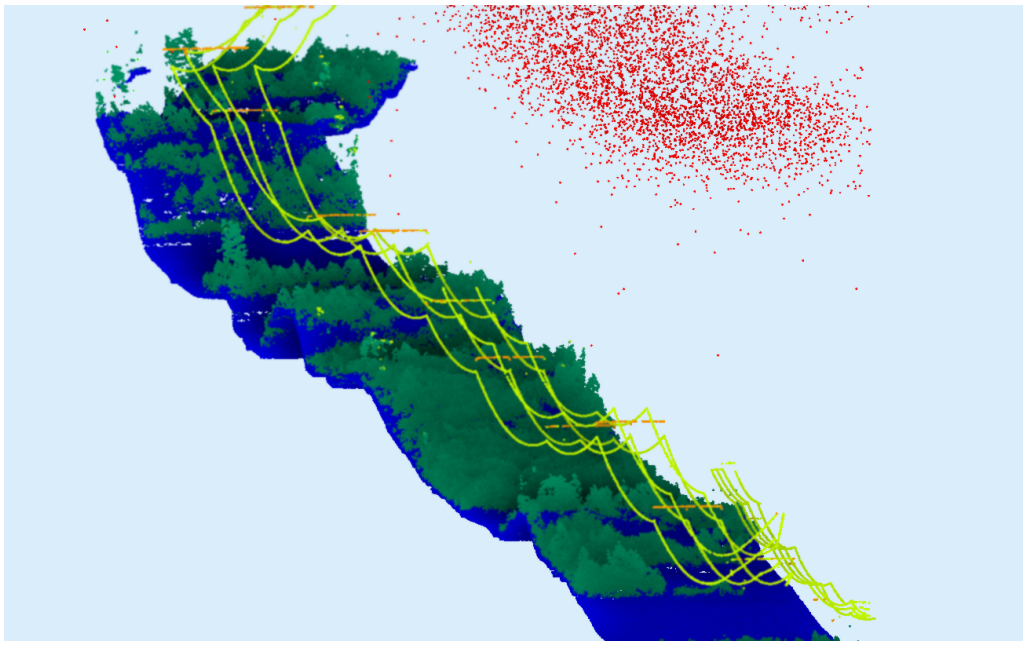


Figure 3.6: Pointnet++ segmentation of electrical wires and surrounding environment

Pointnet++ summarised

In summarising the performance of PointNet++ algorithms, a couple of metrics are relevant to highlight. These include accuracy, seen in table 3.1 ,and IoUs of the segmented classes, as seen in table 3.2. It should be pointed out that the point cloud tiles used for calculating the confusion matrix shown below differ from the data that was used for calculating the results shown in the charts above and will therefore not necessarily display the exact same accuracy and tendencies.

Table 3.1: Confusion matrix for Pointnet++ segmentation

	1	2	3	4	5	6		User acc
1 terrain	8184932	257268	0	3	7	166	8442376	0.970
2 vegetation	635827	8610971	0	1843	108	6437	9255186	0.930
3 noise lower	20	0	5833	0	0	227	6080	0.959
4 wires	0	111	0	209948	1178	18	211255	0.994
5 crossbeam	2	102	0	4061	6358	21	10544	0.603
6 noise upper	59825	87033	3065	728	30	355878	506559	0.703
	8880606	8955485	8898	216583	7681	362747	17373920	
Producer acc	0.922	0.962	0.656	0.969	0.828	0.981		18432000

The accuracies shown in the lowest row and right-most column in the confusion matrix are producer and user accuracies, respectively. These can be calculated due to the training data already having label information, which is withheld at testing time. The absolute number of points in each class is not equal to the number of points stored in each point cloud. Rather, each file is sampled multiple times to increase coverage and improve prediction quality. The accuracies shown still hold true due to all classes being sampled equally. The accuracies shown are promising in that a high degree of accuracy is reached for the most important classes, that is, vegetation, terrain and wires. Noise is more challenging for the

algorithm, which is most likely due to it being mixed with especially the vegetation class. This is particularly seen in the user accuracy for class 6 (noise upper). Furthermore, the crossbeam class is difficult to gain high accuracy on due to it being geometrically similar to the wire class and because only relatively few points are available for this class. These conclusions are echoed in table 3.2, which shows that the highest IoU is the wire class with an IoU of 96.4%.

Table 3.2: IOUs for Pointnet++ segmentation

IOU	
1 terrain	0.896
2 vegetation	0.897
3 noise lower	0.638
4 wires	0.964
5 crossbeam	0.536
6 noise upper	0.693

These metrics serve as good indicators of the accuracy that can be expected when performing segmentation using the PointNet++ algorithm in an outdoors environment captured with an aerial LiDAR.

The next section describes the Superpoint Graph algorithm and its performance in the training and testing stage.

3.1.2 Superpoint graph

The evaluation tools that are included in the Superpoint Graph repository [Landrieu, 2020] are not as extensive as is the case for PointNet++. However, there are still plenty of evaluation metrics included, and these will be covered in the following.

The loss and accuracy curve once again exhibit the same general behaviour as seen in the PointNet++ chart. In examining the chart, it seems that the performance on the training data is nearly flawless with a accuracy of 99.8% rounded to 100 for readability.

One should keep in mind that this degree of accuracy is only true for the loaded training data. In testing stages, a lower accuracy is expected. This behaviour exhibited by the metrics could also be a sign that overfitting might have occurred, meaning the model performs exceedingly well on training data and models testing data much worse. Whether or not overfitting occurs is examined through testing the model on data it has not yet been introduced to. Lastly, the loss function shown below has the exact same behaviour pattern implicitly as was shown in figure 3.1 with the decay and learning rate being adjusted and reset as necessary for the minimisation of loss.

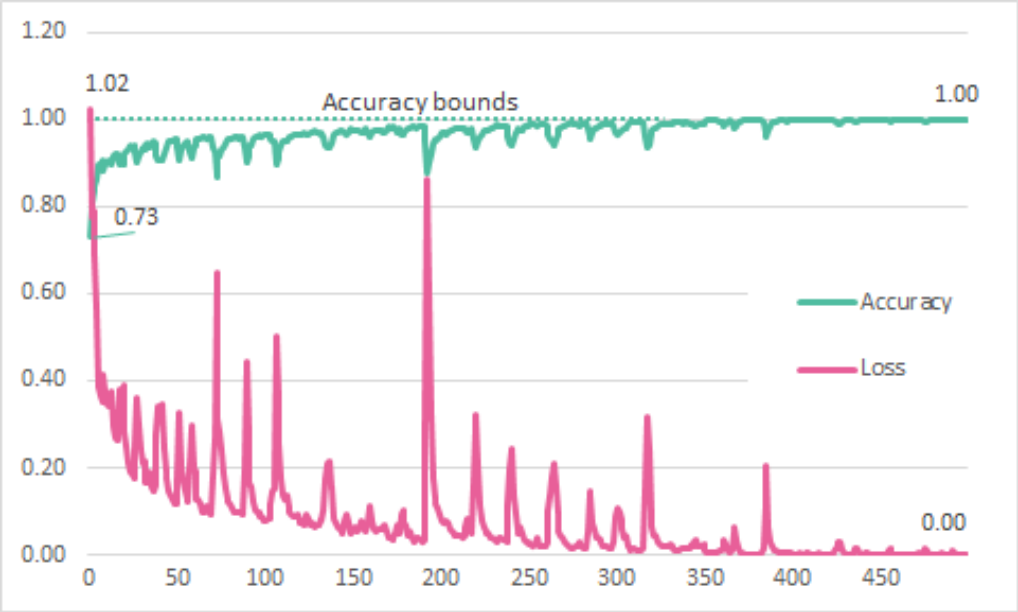


Figure 3.7: Superpoint Graph loss and accuracy metrics

Examining the IoU in the training stage as shown in figure 3.8, it becomes clear that both the average and best IoU quickly trends upwards with the average increasing and decreasing synchronously with the loss curve shown in figure 3.7.

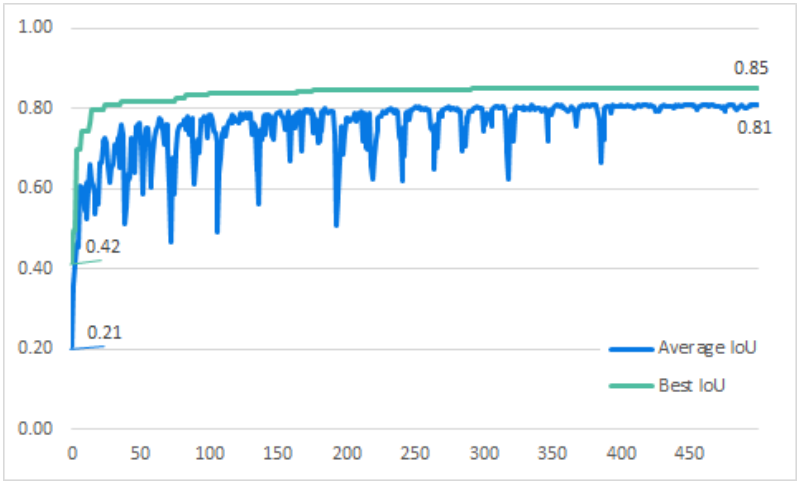


Figure 3.8: Superpoint Graph IoU, training stage

Furthermore, on examining the accuracy results in the testing stage, they indicate that accuracy can largely be preserved between training and testing data samples, which contradicts the overfitting hypothesis that was established earlier.

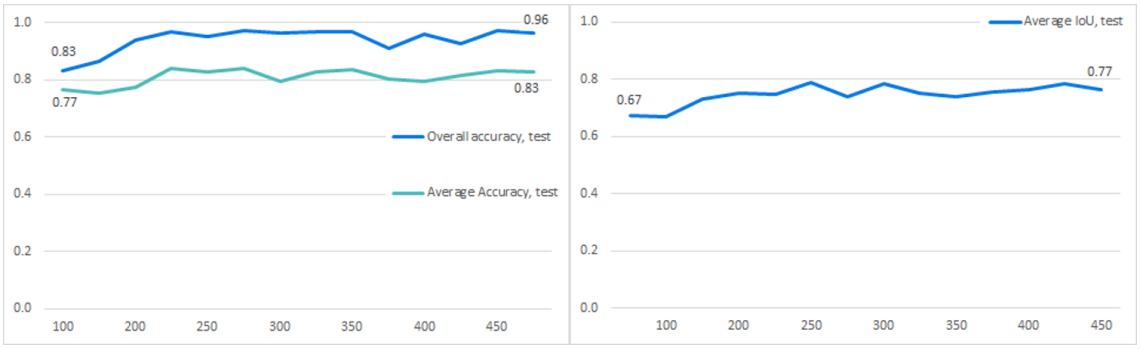


Figure 3.9: Superpoint Graph accuracy and IoU, testing stage

The calculated overall accuracy shown above is a metric that does not take class imbalance into account, while the average accuracy is a mean taken across accuracies on each class.

Visualising the segmentation

Seen from the same point of view as PointNet++, Superpoint Graph also performs well on the visual inspection, as shown in figure 3.10. The terrain and vegetation classes are generally classified well, and wires stand out clearly. Some crossbeams are cleanly labelled, while some are largely mislabelled, usually consisting of wire class points. The upper noise can also be seen as standing out quite clearly in the upper part of the scene, while not many noise points are seen near vegetation and terrain, even though it is almost certainly present when comparing the ground-truth labels with the predicted ones.

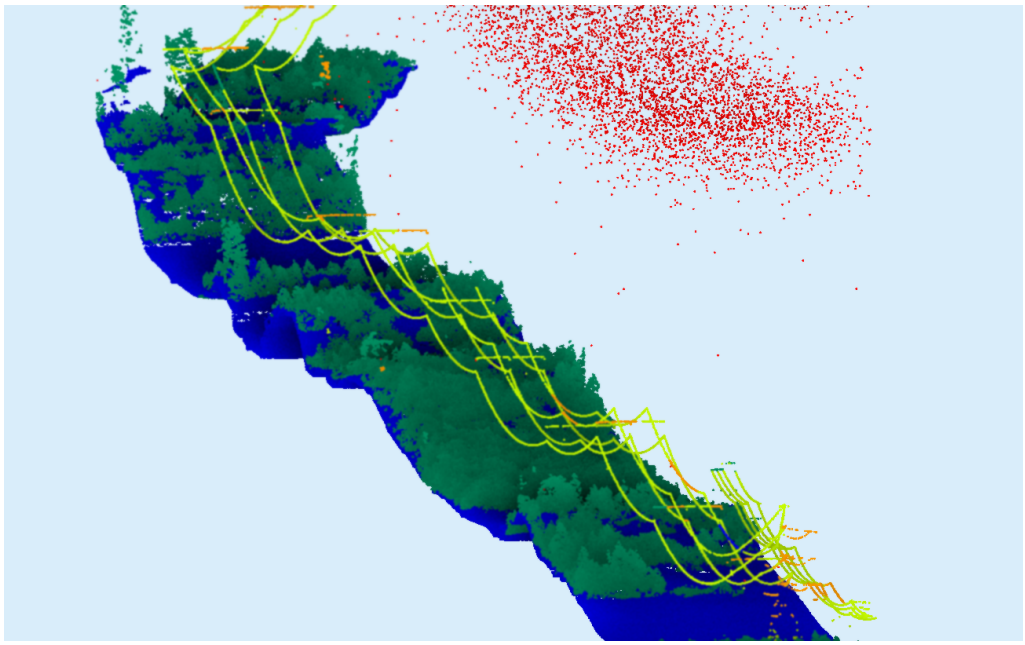


Figure 3.10: Superpoint Graph segmentation of electrical wires and surrounding environment

Overall, the visual inspection shows that Superpoint Graph performs well on both training and testing data. However, it has a more difficult time classifying the crossbeam class and differentiating it from the wire class.

Superpoint Graph summarised

Overall, Superpoint Graph performs well on most classes as shown in the confusion matrix below, table 3.3.

Table 3.3: Confusion matrix for Superpoint Graph segmentation, point count downscaled by 10^{16}

	1	2	3	4	5	6		User acc
1 terrain	2116103	61640	0	2	0	4	2177749	0.972
2 vegetation	40665	956845	0	797	726	177	999210	0.958
3 noise lower	3	0	582	0	0	0	585	0.995
4 wires	82	172	0	23908	817	0	24979	0.957
5 crossbeam	0	92	0	578	495	3	1168	0.424
6 noise upper	5004	9233	0	144	1	29283	43665	0.671
	2161857	1027982	582	25429	2039	29467	3127216	
Producer acc	0.979	0.931	1.000	0.940	0.243	0.994		184320

The crossbeam and noise upper class exhibit inaccuracies, but overall, the algorithm maintains high accuracy for the majority of the classes. It therefore shows promise when applied to a scene such as the one being segmented in this case.

3.1.3 PointCNN

PointCNN is the last algorithms that will be examined. As with the two prior algorithms, this also begins with loading the prepared training data and applying a loss minimisation function, which means that the loss curve quickly drops from 2.7 to below 1.0 and settles around 0.1 as seen in figure 3.11. From this follows that accuracy and Average accuracy/class increases and accuracy especially approaches 1.0 shortly after the segmentation is begun while the average accuracy settles at a lower level marginally above 0.8.

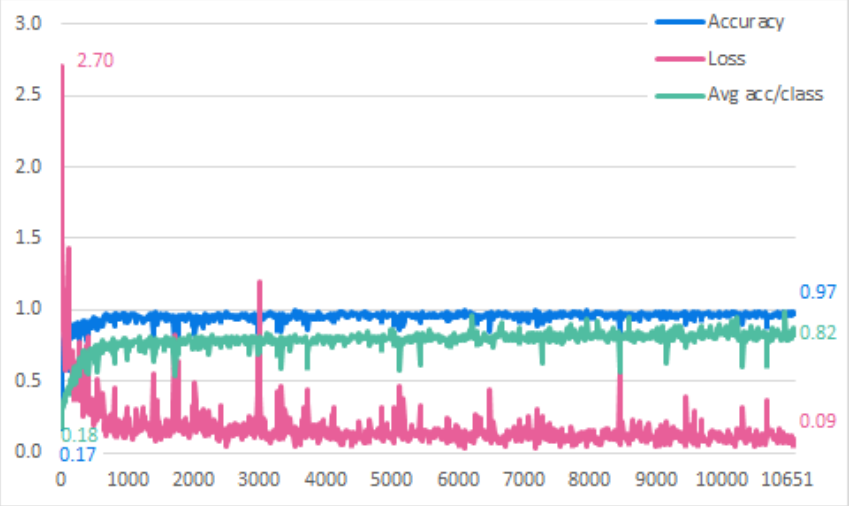


Figure 3.11: PointCNN accuracy and loss, training stage

The first test of the model performance happens at iteration 500, which is the reason for the loss shown below in figure 3.12 being relatively low compared with the values shown in the training stage above. As the iterations increase, a clearer picture emerges than what was apparent by examining figure 3.11 with a gradual, but nonetheless noticeable, improvement that especially becomes clear by looking at the average class accuracy.

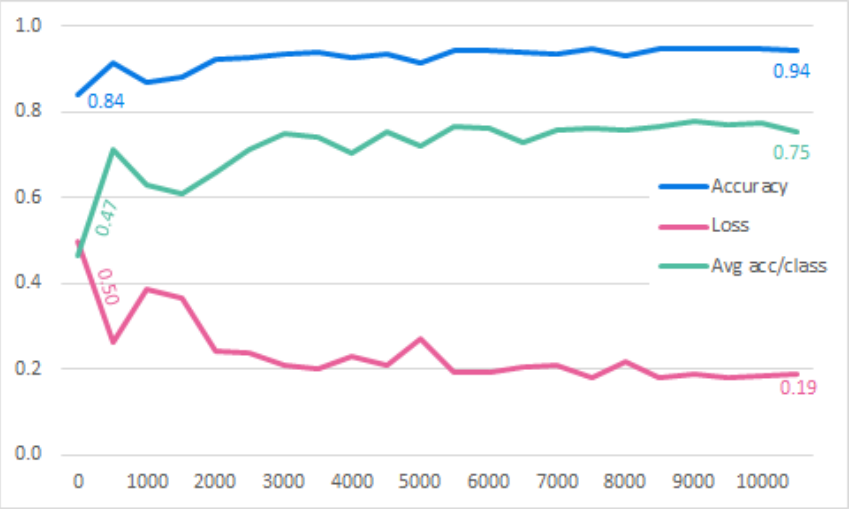


Figure 3.12: PointCNN accuracy and loss, testing stage

One consideration would be to apply a dynamic learning rate as the training progresses, since this might lead to lower loss values and therefore higher accuracy. This was the case for both PointNet++ and Superpoint Graph, and it could therefore possibly be beneficial to apply it in a similar manner in this case.

Visualising the segmentation effort

In this case, the visualisation is not as intuitive to look at as for the two previous algorithms. The clouds shown below in figure 3.13 have not had their labels written back onto the original and georeferenced points. For that reason, it is in the form the algorithms receives it as, meaning it has been scaled and transformed. With this, all XYZ coordinates have therefore been given new values so that it is a normally distributed dataset.

If this was to be implemented, a crucial step is to be able to write the predicted point labels back onto the corresponding point in the original cloud. Unfortunately, due to the tools available in the PointCNN toolbox, this has not been possible since the original indices are lost during processing.

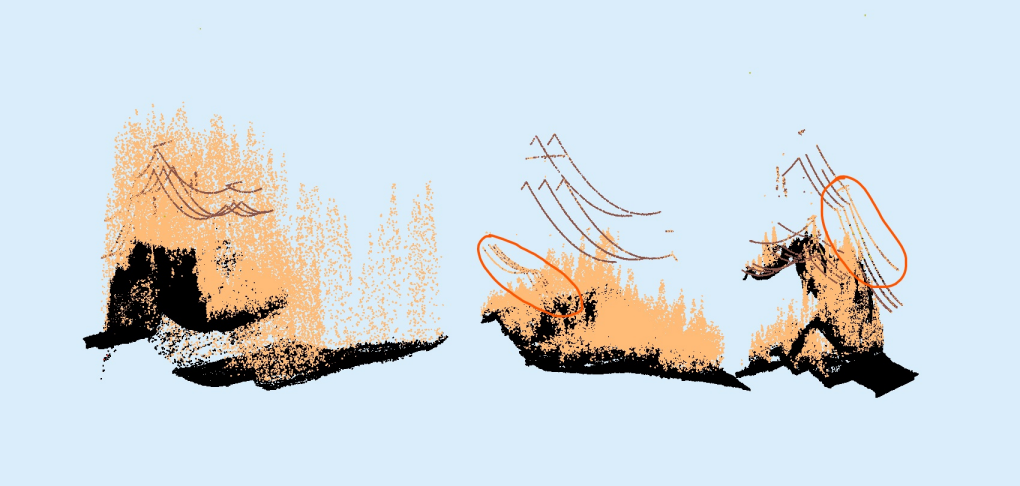


Figure 3.13: PointCNN segmentation of electrical wires and surrounding environment

For the blocks themselves, the segmentation looks promising for the terrain and vegetation classes. However, none of the crossbeams received the correct labels, and the wires are also not a clearly labelled as for the PointNet++ and Superpoint Graph algorithms, with problem areas circled in orange.

PointCNN summarised

Besides the visual inspection and metrics provided during training and testing, a confusion matrix provides additional insight into what classes are often mislabelled and confused for certain other classes, as seen in table 3.4.

There is low accuracy on the crossbeam class but high accuracy in finding noise points with especially the accuracy of the noise upper class, which was lower in the previous algorithms. The overall accuracy calculated from the confusion matrix matches the one provided by the evaluation tools included by the PointCNN tool-set, which both come to 0.95.

Table 3.4: Confusion matrix for PointCNN segmentation, point count downscaled by 10^3

	1	2	3	4	5	6		User acc
1 terrain	37030.73	1463.96	0.53	11.38	0.01	91.81	38598.41	0.96
2 vegetation	1219.63	16107.18	0.17	28.09	2.87	54.56	17412.49	0.93
3 noise lower	13.63	158.02	1501.87	3.71	0.02	53.88	1731.12	0.87
4 wires	8.38	151.48	13.29	1536.78	8.03	48.37	1766.32	0.87
5 crossbeam	0.16	30.79	1.28	82.71	21.74	5.53	142.20	0.15
6 noise upper	82.87	728.82	207.40	21.69	1.24	28514.79	29556.79	0.96
	38355.39	18640.24	1724.52	1684.35	33.90	28768.94	84713.08	
Producer acc	0.97	0.86	0.87	0.91	0.64	0.99		89207.34

3.1.4 Resources required for implementation

A brief investigation into the complexity and operation of the algorithms has been carried out to gain insight into how much work is required in order to implement the algorithms. Time is the main metric used in this case, since it is assumed that virtual machines will be used. Because of this, time taken translates directly to monetary cost.

The overall training time for PointNet++ was 20 hours. In a scenario where the algorithm is going to be implemented, this should be longer and include more training data as additional accuracy could be gained this way. A well-trained model would also be desirable due to the time it is expected to be used for segmentation, which would be considerably longer. The time spent classifying the testing batches of roughly 1,000,000 points were 60 seconds. The time spent on training and inference will be used at a later point to calculate the investment necessary to use virtual machines for large scale segmentation tasks.

In examining how much time is needed for the training stage, the chart seen in figure 3.14 was produced. It shows the time each step of the training process took. It is not straightforward to explain the reason why some steps take tens or hundreds of times longer to process. Nonetheless, a general drop is seen about halfway into training as well as a stabilisation of time per step, which could be caused by the "landscape of loss" being simpler or smoother to navigate for the loss minimisation function.

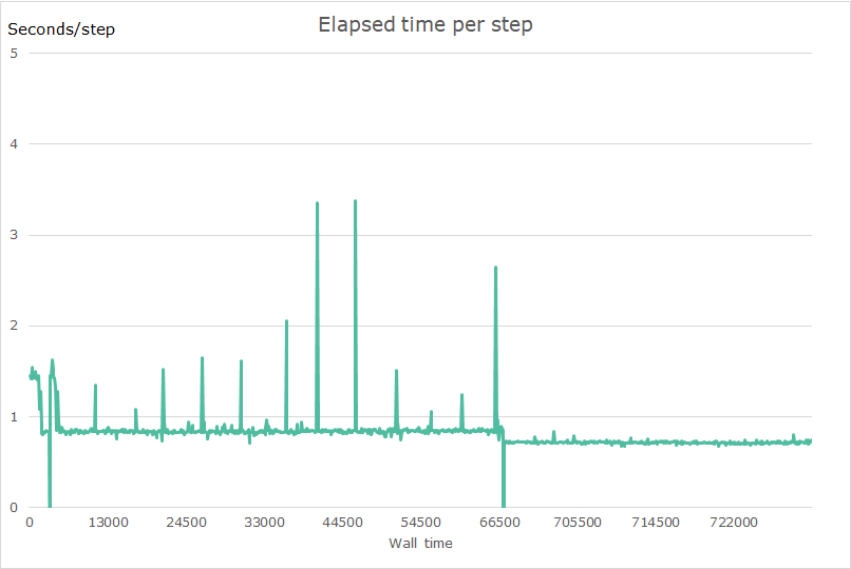


Figure 3.14: Time spent on each step in training by the Pointnet++ algorithm

Having examined the two other algorithms, very similar inference times were observed and it is therefore not clear that one algorithm outperforms the others markedly in terms of speed.

Knowing the time taken for inference and the hourly cost of running a virtual machine, such as the one outlined in section 2.2.1, we can calculate an hourly cost. The total time spent spent using the virtual machine in May and the cost results in an hourly cost of DKK 14. In that time, 60 million points could be segmented, assuming no pause and no time spent developing or training with an estimated 1,000,000 points/minute.

3.2 Comparing classes to other segmentation methods

This section will evaluate the prediction on the dataset using a methodology that differs from the previous section. The evaluation method will be used to assess the accuracy of the terrain class, specifically to understand the degree of compliance between the classified terrain surfaces. In order to do this, raster analysis will be utilised to understand absolute coverage and accuracy in the generated surfaces. The raster derived from the original terrain class in the ground truth data will be used as a reference.

Firstly, it should be noted that the majority of the errors related to the terrain class, and, by extension, the raster surface, come from the confusion between the terrain and vegetation classes in the predicted datasets.

Three raster datasets were generated from the point clouds containing the terrain points from PointNet++, Superpoint Graph and the ground truth data. These can be seen below in figure 3.15. They are generated in order to better understand the completeness of the terrain class.

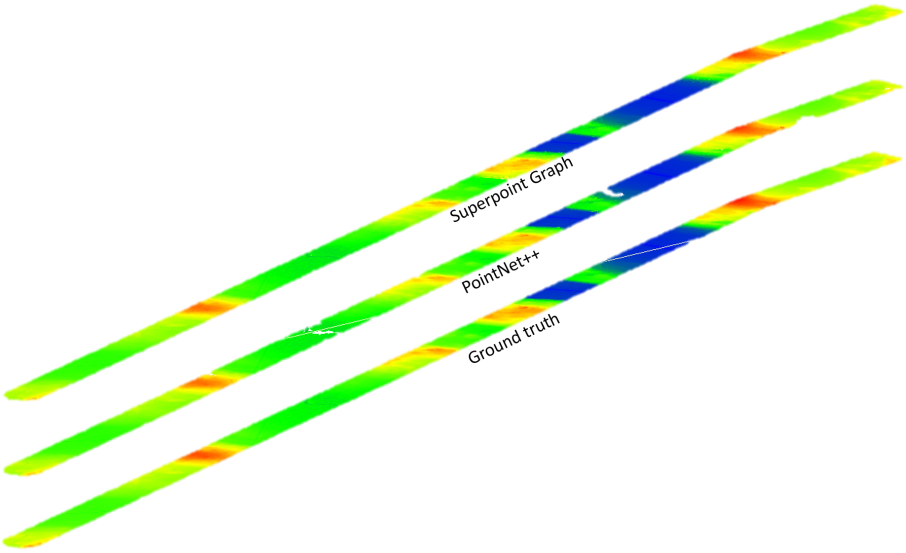


Figure 3.15: Raster layer generated from the terrain class as predicted by algorithms and ground truth

The three rasters seen above are then stacked to view how they compare with respect to completeness, which is seen in figure 3.16. The first raster strip consists of a combination of Pointnet++ and ground truth raster. The second strip consist of the Superpoint Graph raster and ground truth raster. The last strip is the ground truth raster shown on its own for reference. Areas where the red ground truth raster can be seen through the predicted raster strips are areas with poor coverage. It is only visible in certain areas and the coverage is generally acceptable. It is interesting that the areas with poor coverage are as dissimilar as is the case. This would stem from the fact that the two algorithms have ended up with a dissimilar understanding of what constitutes terrain and vegetation.

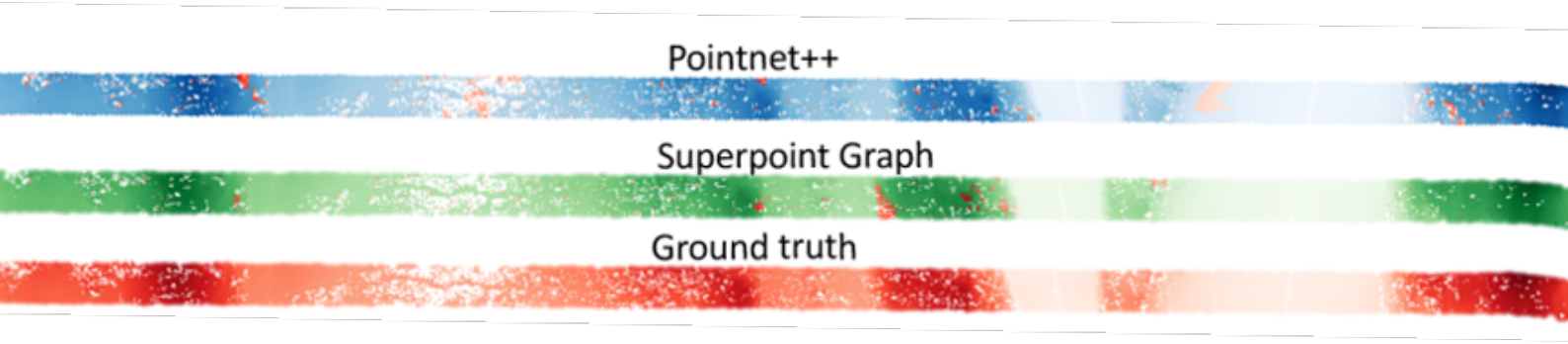


Figure 3.16: Missing terrain information from Superpoint Graph and Pointnet++ scene segmentation compared to ground truth raster (bottom)

Furthermore, the algorithm tends to mislabel points in groups, which can be observed in the raster surface not exhibiting many, smaller isolated areas, but rather showing a few, larger regions with poor coverage. A downside to this would be that many smaller areas could more easily be interpolated to increase completeness, but extrapolating data to larger areas quickly becomes inaccurate.

It is also worth mentioning that the reference data shown in figure 3.16 is not complete; it is actually missing terrain points in areas where the algorithms have classified points so a surface can be generated.

Having gained an understanding of the coverage, an inquiry into the height differences calculated by comparing ground truth and predicted surfaces is worthwhile.

There is generally a high degree of agreement between the ground truth raster and the surfaces generated by the two algorithms. Figure 3.17 shows a zoomed-in view of the height differences between the two raster surfaces in the areas they overlap with the ground truth data. This area was selected for further examination due to a few zones, shown in black, which are the areas that differ the most from the raster ground truth surface.

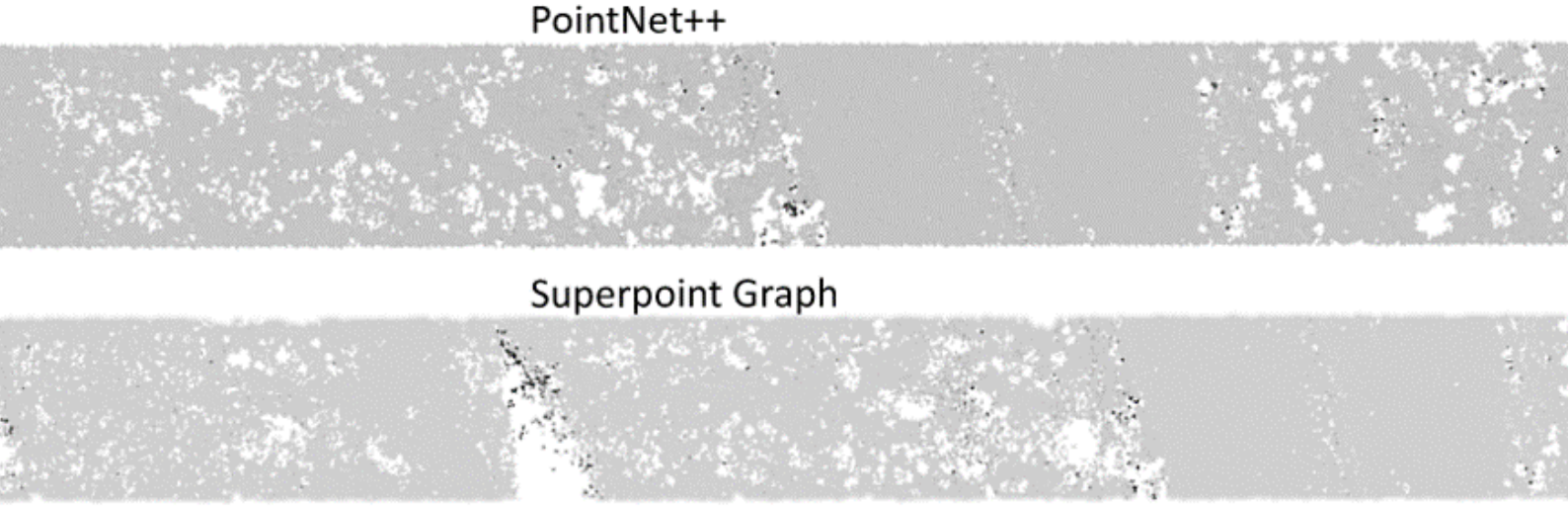


Figure 3.17: Disagreement (in black) between ground truth and the examined algorithms

Both algorithms exhibit the most inaccurate segmentation in this area, which could be caused by a number of different factors. However, it seems to stem from low vegetation mistakenly being labelled as terrain points, which could be because of their geometry being relatively planar when captured using aerial LiDAR as in this case.

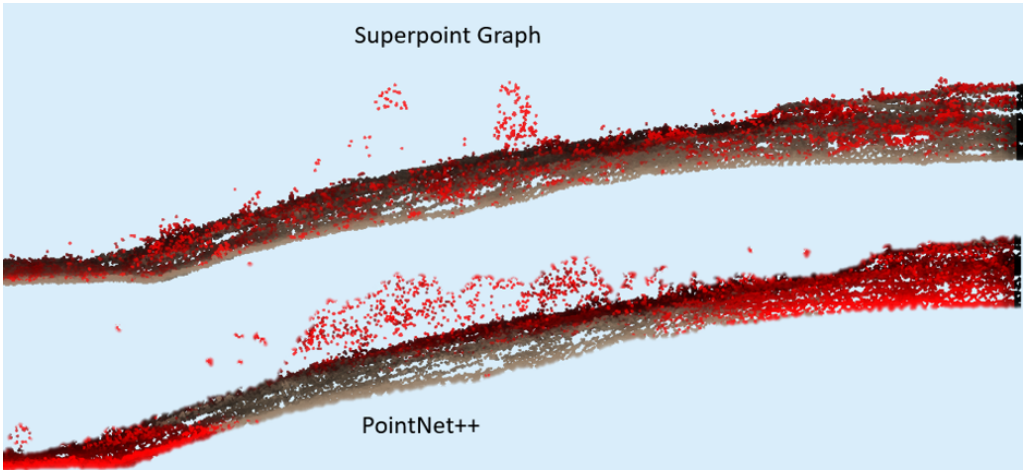


Figure 3.18: Viewing the points used to generate the raster surface from the predicted data (red) and ground truth (brown). Note the error in the center

Figure 3.18 shows the area with high error. It is possible to make out the vegetation in the points labelled in red above the actual terrain surface, shown in brown. Besides the error-ridden area, there is high compliance between the two surfaces, which can be seen on either side of the wrongly labelled terrain.

3.3 Concluding on the results

Some general conclusions can be drawn from the results presented above with regard to both accuracy and estimated cost of implementation given virtual machines services such as Microsoft Azure, Google Compute Engine or Amazon Web Services.

Firstly, with regards to accuracy, PointNet++ performs the best on the prepared non-benchmark data. It manages to create clearly defined clusters of correctly labelled points, even under difficult circumstances such as the crossbeam class given its low point count. Superpoint Graph manages comparable performance and generally provides a simpler workflow for implementation and more intuitive evaluation tools. PointCNN also achieves high accuracy; however, it is lacking compared to the other two algorithms, especially when examining the wire and crossbeam class.

With respect to time and costs related to hardware, these can be seen as marginal in a case of implementation. The largest cost lies in skilled labour needed for development and implementation.

Discussion 4

This chapter will discuss the applicability and options for further development related to point cloud classification and segmentation in general with regard to the algorithms utilised in this work. Closing the chapter will be thoughts on reproducibility and data availability.

The algorithms presented above show a lot of promise for improving efficiency in classification and segmentation workflows. Classification is usually carried out manually, and having these algorithms perform the classification automatically could save a lot of man-hours as was also pointed out in an article published by GeoAI, Medium [Dmitry Kudinov, 2019].

It is not unreasonable to argue that, within a short time span, algorithms could markedly reduce the need for manual classification and segmentation work. This is especially the case when considering the classified data could undergo considerable post-processing routines, which would be able to further assure the quality of the point cloud data.

Had this thesis focused on just a single algorithm as opposed to three, questions regarding the maximum bounds of accuracy could have been answered with more confidence. This is because sacrifices had to be made with regards to development time, preprocessing, cleaning clouds and training periods in order to test all three algorithms. An interesting project could be testing how far one of these algorithms could be taken by optimising as many of the relevant steps as possible.

The "one-step approach" proposed in this report is naive in a sense, since simpler and proven methods are available for a number of the tasks I attempt to solve in this case. Taking the terrain class as an example, one might point out that terrain can already be classified with high accuracy using different techniques, the simplest being based on the observation that the last return from each laser pulse is an expression of the pulse striking the terrain. Another alternative include "Cloth Simulation Filter" [Zhang et al., 2016] which in simple terms define the terrain by simulating a cloth surface being draped along the bottom of the point cloud and thereby describing the terrains surface. It could be argued that these two examples constitute a more intuitive approach which are more easily assured for quality and are more transparent in many respects.

Depending on the specific application of a project, the quality of the input data and the questions that need answering, these algorithms could be accurate enough at this point in time for a large scale implementation, taking the accuracies presented in the *Results* chapter into account.

Comparing the benchmark and non-benchmark results

Having made the distinction between the benchmark and non-benchmark throughout this thesis, it is deemed relevant to compare the algorithm performance in the training stage for each of these algorithms. Table 4.1 shows the accuracy and mIoU on for each of the algorithms as applied to two different types of data. The metrics for the Semantic3D dataset is gathered from results as reported by the algorithm authors [Qi et al., 2017][Li et al., 2018][Landrieu and Simonovsky, 2018].

Table 4.1: Comparing performance on benchmark and non-benchmark data, *mean accuracy per class, not mIoU

	PointNet++		**Superpoint G.**		**PointCNN**	
	Semantic3D	Non-B.	Semantic3D	Non-B.	Semantic3D	Non-B.
Accuracy	0.86	0.87	0.94	0.99	0.99	0.94
mIoU	0.63	0.73	0.73	0.81	n/a	0.75*

Even though no mIoU values were available for PointCNN as applied to Semantic3D nor the non-benchmark data, some general conclusions can be drawn from the results in the table above. The accuracy remains comparable between the dataset types, while mIoU is higher for the non-benchmark data for both PointNet++ and Superpoint Graph. Finally, one thing to keep in mind is that even though PointNet++ seems to perform the poorest in the visual and analytical assessment of the algorithm, performance on the test data does seem to indicate a model with a good fit on the training and testing data. Meanwhile, Superpoint Graph and PointCNN indicate overfitting based on their near flawless performance on training data.

Further development

As pointed out in section 2.5, there are possibilities for including additional information in the training of the networks that constitute the algorithms. The original data used in this work contains a lot more information than what was used in this segmentation effort, which only trained on the geometry of the objects in question. However, it would be an avenue for further development to have attributes like intensity, RGB, return number and number of returns be included in the training of the model. As it stands currently, these attributes can be used either before or after training on the data for further refinement, but they are not taken into account in any meaningful way during training.

A final note with regard to the further development within this field would be that these algorithms are still immature. One of the most obvious signs that these algorithms still have growth potential lies in the nearly non-existent implementation in commercial products that visualise and process point cloud data. One could imagine how these algorithms could expand the point cloud processing capabilities of products like ArcGIS Pro [ESRI, 2019].

Reproducibility

The reproducibility of this study is by its circumstances limited and depends on a number of factors being kept constant.

The first issue with reproducing this work exactly lies in the dataset not being publicly available. Another large scale outdoors dataset of a similar nature could be expected to reach similar performance, as long as the targeted classes are also kept similar as different classes with limited points available for training should be expected to be difficult to label correctly.

Secondly, if comparable results are desired, the algorithm versions and related software environment should be matched as outlined in the respective repositories found on GitHub at [Jensen, 2020], with the original algorithms found on GitHub at [Yangyan Li and Chen, 2020] (PointCNN), [Landrieu, 2020] (Superpoint Graph) and [ISL, 2019].

Conclusion 5

The conclusion of this study will aim at answering the research questions posed in the introductory chapter of the text.

Which machine-learning algorithms are suitable for performing the semantic segmentation task?

All the algorithms tested perform the task with satisfactory accuracy and quickly labels large volumes of points even with moderate computing power available. The largest difference between the algorithms are their ease-of-use where both PointNet++ and Superpoint Graph outperform PointCNN, with Superpoint Graph being the most intuitive and along with competitive performance, this algorithm is deemed suitable for this type of task with only a small barrier to entry.

What accuracy can be expected when performing automatic semantic segmentation in large-scale outdoor point cloud scenes using ML algorithms?

As described in the *Results* chapter, high accuracies can be expected across all classes with the exception of classes containing limited information such as the crossbeam class or the noise class. These do not exhibit consistent patterns that the algorithms can learn to recognise. Depending on the algorithm selected, good accuracy can be expected even for classes like the crossbeam class, since both PointNet++ and Superpoint Graph managed to correctly label the crossbeam point clusters. All algorithms achieved above 85% overall accuracy, with the wire class being almost flawlessly classified, in some cases reaching +95% accuracy. While the results are promising, one should generally be thorough when evaluating the quality of classification and segmentation. This is because factors such as class imbalance and the nature of the used metrics might paint a misleading picture of the performance achieved.

What are the costs related to implementing automatic semantic segmentation costs measured in time taken and price?

The expenses related directly to the hardware required for an accurate segmentation are low compared to the potential gain that might be achieved by implementing an automatic segmentation algorithm. Taking the cost of saved man-hours into account, the case for shifting towards automated processes is only strengthened. The largest cost with regards to implementation lies in employing skilled labour for development, implementation and maintenance of the systems.

Bibliography

Abadi et al., **2015**. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. [ONLINE] Available at: `https://www.tensorflow.org/`. Software available from tensorflow.org.

Aggarwal, **2004**. Shefali Aggarwal. *Principles of remote sensing.* Satellite remote sensing and GIS applications in agricultural meteorology, pages 23–38.

Alpaydin, **2020**. Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2020.

Archish Rai Kapil., **2018**. Archish Rai Kapil. *Holdout Cross-Validation.* [ONLINE] Available at: `https://www.datavedas.com/holdout-cross-validation/`. Accessed: 11-04-2020.

Armeni et al., **2017**. Iro Armeni, Sasha Sax, Amir R Zamir and Silvio Savarese. *Joint 2d-3d-semantic data for indoor scene understanding.* arXiv preprint arXiv:1702.01105.

Bowman, **2008**. J. Bowman. *Differences between Euclidean and non-Eucl. geometries.* `https://en.wikipedia.org/wiki/File:Noneuclid.svg`, 2008.

Bronstein et al., **2017**. Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam and Pierre Vandergheynst. *Geometric deep learning: going beyond euclidean data.* IEEE Signal Processing Magazine, 34(4), 18–42.

Campbell and Wynne, **2011**. James B Campbell and Randolph H Wynne. *Introduction to remote sensing.* Guilford Press, 2011.

Chang et al., **2015**. Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su et al. *Shapenet: An information-rich 3d model repository.* arXiv preprint arXiv:1512.03012.

Chi-Feng Wang, **2018**. Chi-Feng Wang. *A Newbies Guide to Stochastic Gradient Descent With Restarts.* [ONLINE] Available at: `https://towardsdatascience.com/https-medium-com-reina-wang-tw-stochastic-gradient-descent-with-restarts-5f511975163` Accessed: 11-04-2020.

Contributors, **November 2018**. PDAL Contributors. *PDAL Point Data Abstraction Library*, 2018. [ONLINE] Available at: `https://doi.org/10.5281/zenodo.2556738`.

Daina Taimina, **2017**. David W. Henderson Daina Taimina. *Non-Euclidean geometry.* Encyclopædia Britannica, inc.

Defferrard et al., **2016**. Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst and Xavier Bresson. *FMA: A Dataset For Music Analysis*, 2016.

Deng, **2012**. Li Deng. *The mnist database of handwritten digit images for machine learning research [best of the web].* IEEE Signal Processing Magazine, 29(6), 141–142.

Dmitry Kudinov, **2019**. Dmitry Kudinov. *PointCNN: replacing 50,000 man hours with AI.* [ONLINE] Available at: `https://medium.com/geoai/pointcnn-replacing-50-000-man-hours-with-ai-d7397c1e7ffe`. Accessed: 29-05-2020.

Eitel et al., **2016**. Jan UH Eitel, Bernhard Höfle, Lee A Vierling, Antonio Abellán, Gregory P Asner, Jeffrey S Deems, Craig L Glennie, Philip C Joerg, Adam L LeWinter, Troy S Magney et al. *Beyond 3-D: The new spectrum of lidar applications for earth and ecological sciences.* Remote Sensing of Environment, 186, 372–392.

ESRI, **2019**. ESRI. *Deep learning in ArcGIS Pro.* [ONLINE] Available at: `https://pro.arcgis.com/en/pro-app/tool-reference/image-analyst/an-overview-of-the-deep-learning-toolset-in-image-analyst.htm`.

Esri, **2012**. Esri. *Types of lidar.* [ONLINE] Available at: `https://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/types-of-lidar.htm`. Accessed: 11-04-2020.

Flawnson Tong, **2019**. Flawnson Tong. *What is Geometric Deep Learning?* [ONLINE] Available at: `https://medium.com/@flawnsontong1/what-is-geometric-deep-learning-b2adb662d91d`. Accessed: 24-09-2019.

Foundation, **2020**. Python Software Foundation. *Python Language Reference, version 3.7.* `https://www.python.org`, 2020.

Andreas Geiger, Philip Lenz and Raquel Urtasun, *2012.* Andreas Geiger, Philip Lenz and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Grilli et al., **2017**. E Grilli, F Menna and F Remondino. *A review of point clouds segmentation and classification algorithms.* The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 42, 339.

Guo et al., **2017**. Shuxia Guo, Thomas Bocklitz, Ute Neugebauer and Jürgen Popp. *Common mistakes in cross-validating classification models.* Analytical methods, 9(30), 4410–4417.

Timo Hackel, N. Savinov, L. Ladicky, Jan D. Wegner, K. Schindler and M. Pollefeys, *2017.* Timo Hackel, N. Savinov, L. Ladicky, Jan D. Wegner, K. Schindler and M. Pollefeys. SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume IV-1-W1, pages 91–98, 2017.

Honsberger, **1995**. Ross Honsberger. *Episodes in nineteenth and twentieth century Euclidean geometry*, volume 37. Cambridge University Press, 1995.

Isenburg, **2020**. M. Isenburg. *LAStools - efficient LiDAR processing software*. `http://rapidlasso.com/LAStools`, 2020.

ISL, **2019**. Intel ISL. *Semantic3D semantic segmentation with Open3D and PointNet++*. `https://github.com/intel-isl/Open3D-PointNet2-Semantic3D`, 2019.

Jensen, **2020**. Tobias Hjelmbjerg Jensen. *Custom algorithm implementations*. `https://github.com/Tojens?tab=repositories`, 2020.

Kingma and Ba, **2014**. Diederik P Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

Philipp Koehn, *2005*. Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86. Citeseer, 2005.

Krause, **1986**. Eugene F Krause. *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation, 1986.

SPT Krishnan and Jose L Ugia Gonzalez. Google compute engine. In *Building your next big thing with Google cloud platform*, pages 53–81. Springer, 2015.

Krizhevsky et al., **2009**. Alex Krizhevsky, Geoffrey Hinton et al. *Learning multiple layers of features from tiny images*.

Landrieu, **2020**. Loic Landrieu. *Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs*. `https://github.com/loicland/superpoint_graph`, 2020.

Loic Landrieu and Martin Simonovsky, *2018*. Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di and Baoquan Chen, *2018*. Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in neural information processing systems*, pages 820–830, 2018.

Jonathan Masci, Emanuele Rodolà, Davide Boscaini, Michael M Bronstein and Hao Li. Geometric deep learning. In *SIGGRAPH ASIA 2016 Courses*, pages 1–50. 2016.

Daniel Maturana and Sebastian Scherer, *2015*. Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda and Michael M Bronstein, *2017*. Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

NOAA, **2020**. NOAA. *What is LiDAR*. [ONLINE] Available at: https://oceanservice.noaa.gov/facts/lidar.html. Accessed: 11-04-2020.

Guan Pang and Ulrich Neumann, 2016. Guan Pang and Ulrich Neumann. 3D point cloud object detection with multi-view convolutional neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 585–590. IEEE, 2016.

Paszke et al., **2017**. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga and Adam Lerer. *Automatic differentiation in PyTorch*.

Pier Paolo Ippolito, **2019**. Pier Paolo Ippolito. *Hyperparameters Optimization*. [ONLINE] Available at: https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d. Accessed: 11-04-2020.

Qi et al., **2016**. Charles R Qi, Hao Su, Kaichun Mo and Leonidas J Guibas. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. arXiv preprint arXiv:1612.00593.

Qi et al., **2017**. Charles R Qi, Li Yi, Hao Su and Leonidas J Guibas. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. arXiv preprint arXiv:1706.02413.

Reback et al., **February 2020**. Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Adam Klein, Matthew Roeschke, Simon Hawkins, Jeff Tratner, Chang She, William Ayd, Terji Petersen, Marc Garcia, Jeremy Schendel, Andy Hayden, MomIsBestFriend, Vytautas Jancauskas, Pietro Battiston, Skipper Seabold, chris b1, h vetinari, Stephan Hoyer, Wouter Overmeire, alimcmaster1, Kaiqi Dong, Christopher Whelan and Mortada Mehyar. *pandas-dev/pandas: Pandas*, 2020. [ONLINE] Available at: https://doi.org/10.5281/zenodo.3509134.

Remesan and Mathew, **2016**. Renji Remesan and Jimson Mathew. *Hydrological data driven modelling*. Springer, 2016.

Rusu, **2010**. Radu Bogdan Rusu. *Semantic 3d object maps for everyday manipulation in human living environments*. KI-Künstliche Intelligenz, 24(4), 345–348.

Radu Bogdan Rusu and Steve Cousins, 2011. Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.

Safe software, **2015**. Safe software. *Point Clouds*. [ONLINE] Available at: https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Workbench/!FME_Geometry/point_cloud_geometry.htm. Accessed: 11-04-2020.

Andre Samberg, 2007. Andre Samberg. An implementation of the ASPRS LAS standard. In *ISPRS Workshop on Laser Scanning and SilviLaser*, pages 363–372, 2007.

Scikit-learn, **2019**. Scikit-learn. *Cross-validation: evaluating estimator performance.* [ONLINE] Available at: `https://scikit-learn.org/stable/modules/cross_validation.html`. Accessed: 14-05-2019.

software, **2020**. GPL software. *CloudCompare (version 2.10)*, 2020. [ONLINE] Available at: `https://www.cloudcompare.org/`.

Taha and Hanbury, **2015**. Abdel Aziz Taha and Allan Hanbury. *Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool.* BMC medical imaging, 15 (1), 29.

Team, **2020**. QGIS Development Team. *QGIS Geographic Information System.* `http://qgis.osgeo.org`, 2020.

R. V. Ussyshkin, L. Theriault, M. Sitar and T. Kou, *2011.* R. V. Ussyshkin, L. Theriault, M. Sitar and T. Kou. Advantages of Airborne Lidar Technology in Power Line Asset Management. In *2011 International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping*, pages 1–5, 2011.

Weitkamp, **2006**. Claus Weitkamp. *Lidar: range-resolved optical remote sensing of the atmosphere*, volume 102. Springer Science & Business, 2006.

Xu et al., **2008**. Zujian Xu, Feng Yang, Yong Huang, Zizheng Wang and Yanjing Liu. *Lidar applications in the electrical power industry.* Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci, 36, 137–140.

Yangyan Li and Chen, **2020**. Mingchao Sun Wei Wu Xinhan Di Yangyan Li, Rui Bu and Baoquan Chen. *PointCNN: Convolution On X-Transformed Points.* `https://github.com/yangyanli/PointCNN`, 2020.

Yastikli and Cetin, **2016**. N Yastikli and Z Cetin. *Classification of LiDAR data with point based classification methods.* Int. Arch. Photogr. Remote Sens. Spat. Inf. Sci, pages 441–445.

Zhang et al., **2016**. Wuming Zhang, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang and Guangjian Yan. *An easy-to-use airborne LiDAR data filtering method based on cloth simulation.* Remote Sensing, 8(6), 501.

Zhu et al., **2017**. Xiao Xiang Zhu, Devis Tuia, Lichao Mou, Gui-Song Xia, Liangpei Zhang, Feng Xu and Friedrich Fraundorfer. *Deep learning in remote sensing: A comprehensive review and list of resources.* IEEE Geoscience and Remote Sensing Magazine, 5(4), 8–36.