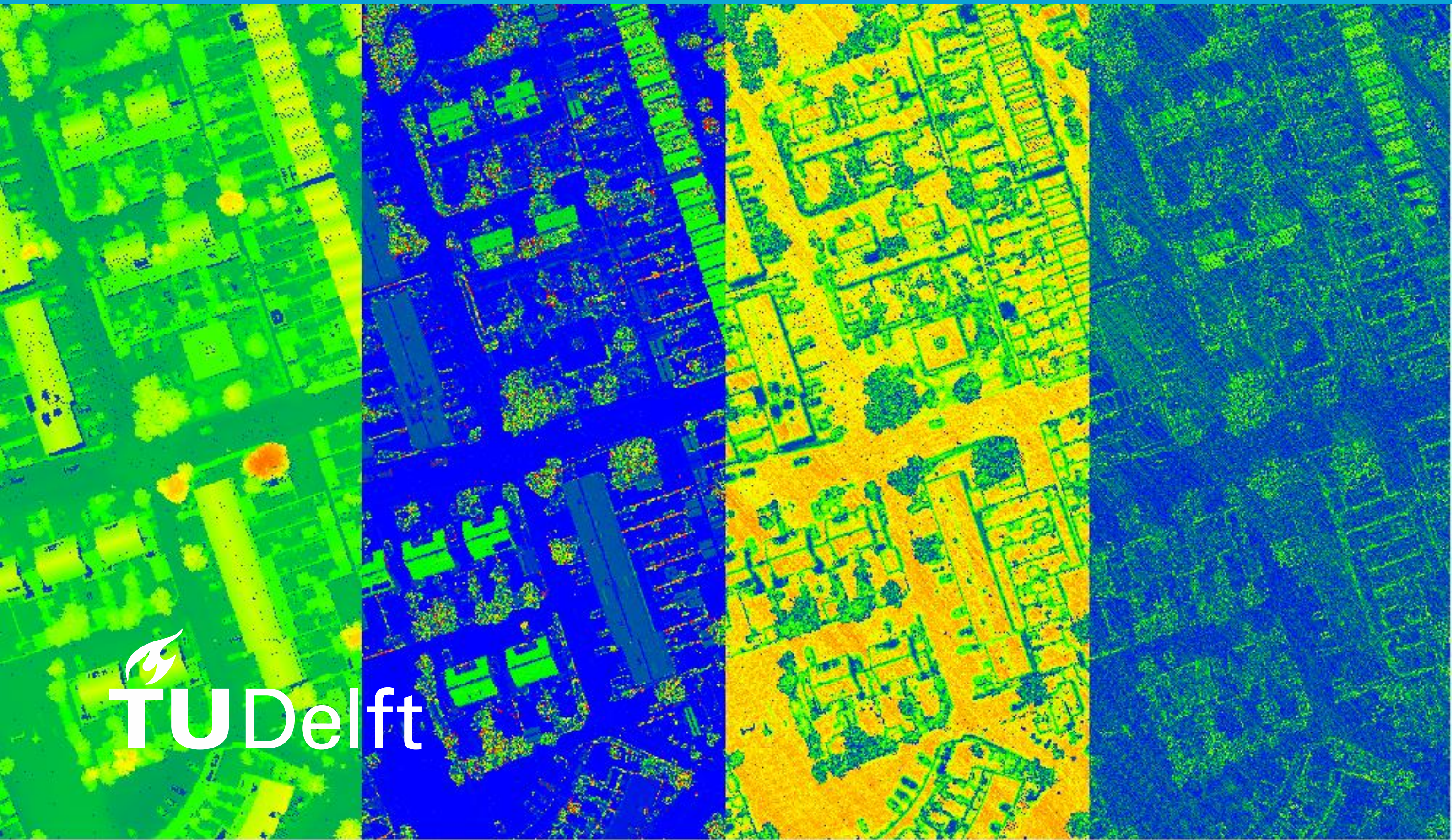
MSc thesis in Geomatics

Semantic segmentation of the AHN dataset with the Random Forest Classifier

Manos Papageorgiou
2021



MSc thesis in Geomatics

Semantic segmentation of the AHN dataset with the Random Forest Classifier

Manos Papageorgiou

June 2021

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:

3D geoinformation group
Delft University of Technology

Supervisors: Ravi Peters
Weixiao Gao
Co-reader: Zexin Yang

Abstract

Three-dimensional (3D) city models are of great significance and are high in demand. They can be used for various useful applications such as urban planning, visibility analysis and estimating the solar irradiation and energy demand of buildings throughout the day. Nowadays, Light Detection And Ranging (LiDAR) sensors are one of the most commonly used technologies for the acquisition of cheap, fast, dense, and reliable 3D point cloud datasets. At the same time, increasing attention has been focused lately on the utilization of these 3D point cloud datasets for the reconstruction of 3D city models. The 3D representation of a scene in the form of a 3D point cloud facilitates a variety of analysis tasks like object recognition, segmentation, and classification, which are an important prerequisite for many building reconstruction methods. Thus, having an accurate classifier that can automatically assign 3D points a respective semantic class label is of utmost importance as it can significantly reduce the time and cost required to analyse 3D scenes. Training machine learning and deep learning algorithms to perform this task has been the focus of many recent scientific works that provide promising results and insights for future work.

This thesis attempted to create an accurate Random Forest Classifier for LiDAR point cloud datasets using the Actueel Hoogtebestand Nederland 3 (AHN3) dataset as training data. The aim was to assist building reconstruction methods, and for this reason, only three semantic class labels are assigned to the points by the classifier, namely ground, building and other. Multiple experiments were conducted to test how large the training dataset needs to be, what features should be included and what point density the input point cloud needs to have for the classifier to perform well. Tests were also made using the DALES dataset to evaluate the performance of the classifier for different datasets and environments. Moreover, the time and memory required to train and test the various Random Forests models and the evaluation metrics of the results were stored as benchmarking research to provide insights and guide future work.

The classification approach that was used consists of five major steps. First, the input point cloud is uniformly sampled and then spherical local neighbourhoods of the points are computed in multiple scales. Height and eigen-based features are then extracted for each local neighbourhood, and the best subset of features is then used to train the Random Forest Classifier. Finally, each point of the input point cloud is assigned to the same class as its nearest neighbour in the sampled point cloud. A comparative analysis shows that the performance of our final Random Forest model stands in parallel with that of other available, more complex, deep learning algorithms if we take into consideration its simplicity and efficiency.

Acknowledgements

In this chapter, I would like to express my sincere gratitude to all those who helped me through these difficult, stressful, and critical times to complete my thesis project. This past year has been a challenge for everyone, as we are still very comforted by the restrictions and consequences of the COVID-19 pandemic. However, thanks to the help of my teachers, friends and family, I managed to continue to work and study and gradually learn to deal with the restrictions caused by the pandemic.

First of all, I would like to thank my two supervisors, Ravi Peters and Weixiao Gao, for their continuous and constant support, patience, friendliness, understanding and helpful suggestions during this graduation project. Thank you so much for your time and consideration. Without you, this graduation project would not have succeeded in its present form. Secondly, I would like to thank Zexin Yang, the co-reader of my thesis, and Yawei Chen, the Delegate of the Board of Examiners, for their constructive criticism and consideration.

Finally, I would like to thank my family and friends for always caring and being there for me and my sweet and kind classmates for their warm company and help. I wish you all the best. Thank you for all of the funny and happy moments we shared while studying at TU Delft.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

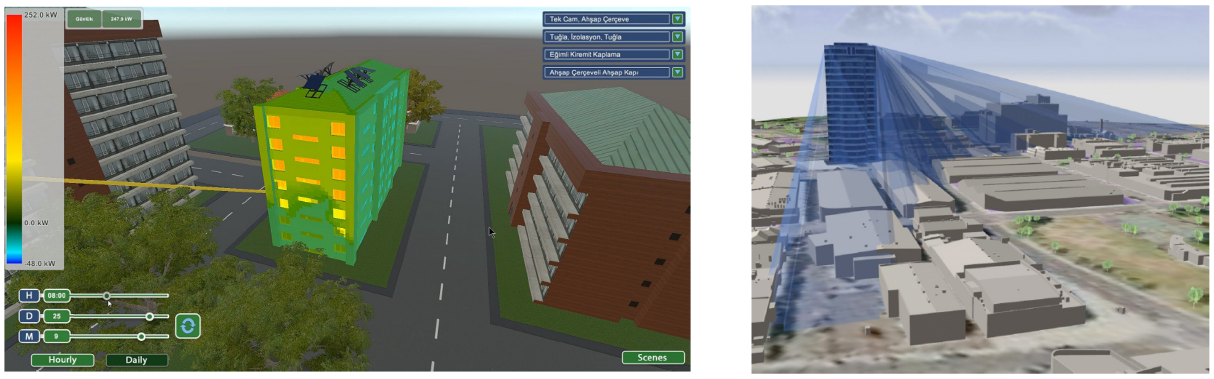
3D	Three-dimensional	v
2D	two-dimensional	xi
GIS	Geographical Information System	1
LODs	Levels of Detail	xi
LOD	Level of Detail	1
LOD1	Level of Detail 1	2
LiDAR	Light Detection And Ranging	v
AHN3	Actueel Hoogtebestand Nederland 3	v
AI	Artificial Intelligence	xi
DTM	Digital Terrain Model	2
DSM	Digital Surface Model	2
CNNs	Convolutional Neural Networks	xi
CNN	Convolutional Neural Network	5
IoU	Intersection Over Union	4
OA	Overall Accuracy	10
CCI	Class Consistency Index	30
MLP	Multi-layer Perceptron	14
RAM	random-access memory	4
GB	gigabytes	34
MB	megabytes	34
KB	kilobytes	61

1 Introduction

This chapter begins with a brief discussion on the motivation behind this graduation project, what scientific problem it attempts to solve, its importance and its relation to the field of Geomatics. The main research questions addressed by the project are then presented along with an overview of the obtained results. Finally, a general outline of the thesis is provided in the last section.

1.1 Motivation

The demand for 3D city models is higher than ever as more and more cities are adopting and embracing them. Nowadays, many 3D city models of major cities and regions can be found on the web as open datasets[1]. With a focus on buildings as prominent features, 3D city models represent urban environments using 3D geometry. Thus they are of greater value than 2D datasets and provide additional insights and applications (Figure 1.1) like the estimation of the solar irradiation, visibility analysis and 3D cadastre [Biljecki et al., 2015].



(a) (b)

Figure 1.1: Visualization of the estimated (a) solar irradiation of a building for a specific date and time and (b) shadow cast by a building for a few sun positions. (Images are courtesy of Argedor and CyberCity 3D as cited in [Biljecki et al., 2016]).

The quality of the 3D models and the amount of information that comes with them heavily depends on the acquisition technique used to collect the necessary data for their reconstruction. An essential concept in Geographical Information System (GIS) and 3D city modelling is the term LODs, which refers to the amount of information contained within a 3D city model, both in terms of its geometry and semantics. The desired Level of Detail (LOD) of a 3D city model determines the essential acquisition technologies and methods. For example, when dealing with LiDAR data, it indicates the minimum point cloud density needed and if other datasets are required [Biljecki, 2017; Kavisha, 2020; Tang et al., 2020]. In total, five different LODs are defined by the CityGML 2.0 standard of the Open Geospatial Consortium [Gröger et al., 2012]. The LODs are meant for describing several thematic classes of objects, but they primarily focus on buildings. The higher the LOD of a model, the more geometric and semantic information it contains (Figure 1.2).

[1]https://3d.bk.tudelft.nl/opendata/opencities/

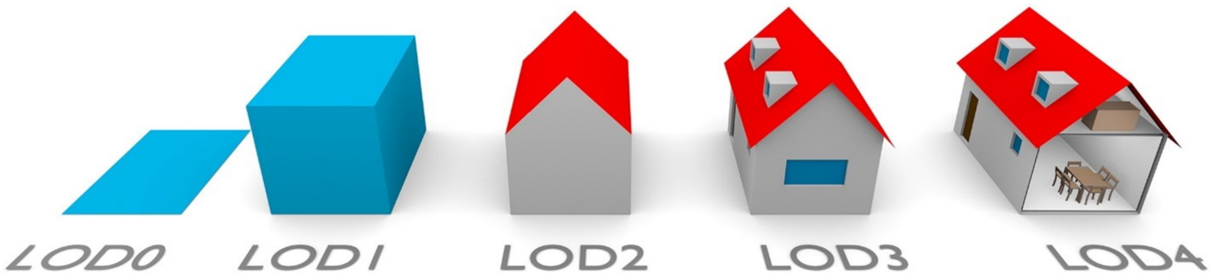


Figure 1.2: A visual example of the refined LODs for a residential building [Biljecki et al., 2016].

LiDAR data can be used for many applications such as forestry [Carson et al., 2004] and Digital Terrain Model (DTM)/Digital Surface Model (DSM) generation [Kraus and Pfeifer, 2001]. However, increasing attention has been focused recently on their utilization for reconstructing urban buildings due to their various useful applications. The development of acquisition technologies such as LiDAR and photogrammetric techniques greatly facilitated the acquisition of large point cloud datasets used for 3D city models reconstruction [Yi et al., 2017]. This development was encouraged by the lack of automatic, efficient and reliable acquisition techniques available in the market. Due to the high requirements of the different applications of 3D spatial data with respect to data amount and quality, this type of acquisition techniques eventually became essential [Kulawiak and Lubniewski, 2020].

In addition, much work has been done over the years to develop automatic building reconstruction methods [Zhou and Neumann, 2010, 2013; Yi et al., 2017; Gold et al., 2006]. For example, open-source tools like 3dfier[2] have been developed lately that can create 3D city models automatically. More specifically, this tool takes 2D topographic datasets and lifts every 2D polygon to the required height, obtained from a point cloud, to make them 3D and produce Level of Detail 1 (LOD1) 3D city models. However, practitioners engaged in developing and utilising 3D city models often come across several issues and limitations, especially when dealing with 3D city models with a level of detail higher than LOD1. One of these issues is the lack of datasets with the necessary accuracy to build such models. Potential errors and anomalies in point clouds and building footprints may yield erroneous results when attempting to estimate the height of the buildings. More importantly, datasets with the requirements to build such models are missing in many cases. For instance, accurately classified point clouds are difficult to be found or even non-existent for some areas. Therefore, efficient and accurate classifiers are needed to be developed for point cloud classification.

One possible solution is to train machine learning and deep learning algorithms to efficiently and accurately classify point clouds. Using multiple processing algorithms, the full potential of raw point clouds can be exploited by extracting all the necessary information from them to train such algorithms and assist in the reconstruction of 3D city models. Having a classified point cloud with at least three classes, ground, building and other points, can help in the extraction of the building footprints and to further segment the point cloud into individual building instances.

Both types of algorithms have their own strengths and weaknesses, making them worth comparing. Simple traditional machine learning methods mainly use handcrafted features or predefined classification rules with preprocessing/post-processing operations for point cloud classification and segmentation. Deep learning algorithms, on the other hand, provide a new approach to solve these two tasks because of their ability to self-learn features, which has been favoured in many fields [Jin et al., 2020]. A simple machine learning classifier like Random Forest can provide efficient and accurate results for such a task [Weinmann et al., 2015]. However, in recent years too much attention has been focused on using deep learning algorithms due to their flexibility and excellent performance when trained with a huge amount of data and the increasing availability of such data. Deep learning algorithms transformed the field of automated analysis of 3D data and greatly improved the performance for numerous point cloud analysis tasks, including classification, segmentation and registration, object detection and panoptic segmentation. But all that at the cost of being more computationally expensive than the other

[2]`https://github.com/tudelft3d/3dfier`

methods. This led to the development of a series of novel convolution methods, which present interest in terms of performance or versatility, that address the core problems of point cloud classification and segmentation. Some of these methods are data-driven deep networks like PointNet/PointNet++ [Qi et al., 2017a,b], KPConv [Thomas et al., 2019], ConvPoint [Boulch, 2020], SuperPoint [Landrieu and Simonovsky, 2018], PointCNN [Li et al., 2018] and ShellNet [Zhang et al., 2019]. These methods have been greatly pushed forward by the availability of open datasets, which can act as benchmarks for the objective comparison of algorithms and their performance [Hu et al., 2020]. Some of these 3D datasets can be broadly classified as object-level 3D models, such as ModelNet [Wu et al., 2015] and ShapeNet [Chang et al., 2015], and indoor scene-level 3D scans, such as S3DIS [Armeni et al., 2017] and ScanNet [Dai et al., 2017].

In this research, the performance of a simple machine learning algorithm for point cloud classification will be tested using the AHN3, AHN4, and DALES [Varney et al., 2020] datasets as training and testing data, and a comparison will be made with deep learning methods. Extensive experiments will be carried out with different data and parameters each time, and analysis will be made for their results. More specifically, this research will aim to address the problems faced when reconstructing 3D city models. Thus, only three classes will be considered, namely ground, building and other. A classified point cloud is particularly crucial for their reconstruction, and having a reliable and accurate classifier can help boost the performance and the flexibility of building reconstruction methods. An attempt will be made to produce a suitable classifier for this task and provide a list of python scripts that will serve as data preparation and classification tool for 3D point clouds of urban areas that can be used by an automatic building reconstruction method.

1.2 Research questions

1.2.1 Objectives

The main research question for this thesis is:
How well will existing machine learning algorithms perform when classifying a point cloud into three classes, namely ground, buildings and other, if we train and test them with the AHN3 dataset?

Based on this research question, the main goal of this research is to study the potential of machine learning and deep learning models for point cloud classification into three classes, which are ground, buildings and other, and to produce trained models that can perform this task accurately. To achieve all of this, the following sub-questions will be addressed:

1. What features of the 3D points should be included in the training and testing processes of the machine learning algorithms to improve their performance?
2. How accurately will the trained models perform when tested with datasets other than the AHN3 dataset?
3. How large does the training dataset needs to be for the models to perform well?
4. In case the trained model underperforms for LiDAR datasets with low point densities, what is the minimum point cloud density required for the trained model to perform well?
5. Do deep learning algorithms outperform profoundly simpler machine learning algorithms?

1.2.2 Scope of research and challenges

This thesis project focuses on point cloud classification with the Random Forest Classifier using the AHN3 dataset as training data. It attempted to produce a trained model that can accurately classify a point cloud into three classes, ground, buildings and other, and to assist in the automatic reconstruction of 3D city models from LiDAR point clouds. The processing of classified point clouds for the reconstruction of 3D city models was out of the scope of this research. Furthermore, this research focuses on benchmarking and comparing the performance of different models. Extensive experiments were conducted using different datasets, hyperparameters and handcrafted features.

Training, testing and validating machine learning algorithms is a very challenging task that requires a lot of time and computational power. Deep learning algorithms are expected to perform more accurately at the cost of being more computationally expensive. However, efficiency is also an important characteristic of a classifier, which is why a simpler machine learning algorithm was tested.

Classified LiDAR point clouds are not always available. In many cases, the available datasets have different accuracies and inconsistencies, making the reconstruction of 3D city models difficult or even impossible. Thus, reducing the minimum required data to produce a 3D city model to just an unclassified point cloud can significantly boost the efficiency and performance of building reconstruction methods.

The memory and time requirements, as well as the features and the scores of the trained models, were saved as benchmarking research to provide insights and guide future research. However, multiple challenges were needed to be tackled first to provide clear answers for all of the research questions and achieve worth mentioning results.

1.3 Overview of obtained results

Overall, multiple Random Forest models have been trained and tested with different data, different amounts of data and different handcrafted features combinations each time. Most of the handcrafted features used in the experiments were eigen-based features and features derived from the Z coordinate of the points. Visualization and analysis of the results at each step helped to identify the most prominent features and achieve better classification scores. The amount of memory required to train the models was reduced significantly over time by improving our python code. We also considered the efficiency of the proposed methodology, and thus, for many of our experiments, we kept track of how much time it took to train the models and how much random-access memory (RAM) was required. Details about the data, features and hyperparameters that were used are also provided. The final model trained with a subset of the AHN3 dataset was able to achieve an F_1 score of 0.90 and a mean Intersection Over Union (IoU) of 0.79 when tested with another larger subset of the AHN3 dataset. Using a simple machine learning algorithm like Random Forest, a classification model was achieved that, for some classes, has comparable performance with more complex deep learning models. Many observations were made, and conclusions were derived from the final results that answer many of the research questions addressed by the thesis project.

1.4 Thesis outline

The main content of this thesis is organised into five chapters. Chapter 2 is an elementary introduction to the theoretical background and scientific work related to this graduation project. Particularly artificial intelligence, machine learning and deep learning with an emphasis on point cloud classification algorithms.

In Chapter 3, the design of the experiments and the methodology and algorithms proposed to address the research questions of this project are described. Further details on the experiments and the implementation of the methodology that was developed are provided in Chapter 4.

In Chapter 5, the results of the experiments are presented from the application of our proposed methodology and algorithms to exemplary point cloud datasets. These results are then analysed and used to reflect on the methodology, first by examining them individually and then comparing them with the results from other methods proposed from related scientific work.

The thesis concludes in Chapter 6. A summary of the main results is given, followed by partial or complete answers to the research questions defined in the introduction. Finally, the limitations of our methodology are presented, which serve as the ground for the recommendation of some future work.

2 Related work

In this chapter, the scientific research related to this graduation project is reviewed. In the following sections, relevant basic knowledge is reviewed for AI, machine learning and deep learning, as well as information for machine learning algorithms and deep learning Convolutional Neural Network (CNN) architectures used for point cloud classification and segmentation. Lastly, some details about 3D scene analysis, in terms of automatically assigning 3D points a respective semantic class label, are provided.

2.1 Artificial intelligence, Machine Learning and Deep Learning

AI is a field that has progressed over the last decades and has many practical applications and active research topics. In general, it tries to simulate, expand, and extend human intelligence by developing techniques that enable machines to think without any human intervention and encompasses machine learning and deep learning concepts.

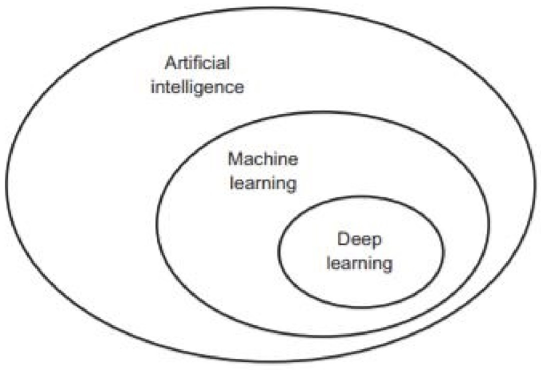


Figure 2.1: A Venn diagram of AI illustrating the link between AI, machine learning and deep learning, which is a kind of representation learning. Deep learning is a subset of machine learning, and machine learning is a subset of AI [Chollet, 2017].

2.1.1 Machine Learning

Machine learning is the most promising and applicable subset of AI concerned with statistical learning algorithms that can learn from data. In other words, it refers to different techniques that enable systems to adapt to new circumstances, recognize patterns in the data and improve their performance after making observations. Most machine learning algorithms have settings called hyperparameters that can be used to control their behaviour. These parameters are set before the learning process, and they can influence their performance. Moreover, the performance of machine learning algorithms heavily relies on having input handcrafted features that are informative, discriminating and independent and the accuracy of the labelled data used to train them [Goodfellow et al., 2016].

Generally, we can divide machine learning algorithms into supervised, unsupervised, semi-supervised and reinforcement learning algorithms. Supervised learning algorithms are provided with labelled datasets, which they use to evaluate their accuracy on the training data, and they are used for solving tasks like classification and regression. Unsupervised learning algorithms are fed with unlabelled datasets and attempt to learn useful properties of their structures and patterns. These algorithms are usually used for solving clustering and dimensionality reduction problems. Semi-supervised learning algorithms fall between the latter two categories as they combine labelled and unlabelled data during training. They are used in cases where labels for the majority of the observations are absent and make use of a small amount of labelled data to improve the learning accuracy. Finally, reinforcement learning algorithms can interact with an environment by creating a feedback loop between the learning system and its experiences, and this enables them to take actions that improve their behaviour and performance [Louridas and Ebert, 2016; Goodfellow et al., 2016; Sutton and Barto, 2018].

2.1.2 Deep Learning

On the other hand, deep learning is a subset of machine learning, and traditional machine learning algorithms strongly influenced its development. Specifically, deep learning algorithms are representation learning algorithms that can improve with experience and data. They focus on learning successive layers of increasingly meaningful data representations through a hierarchical learning process that utilizes artificial neural networks. In other words, deep learning is a multistage way of sufficiently scaled simple mechanisms to learn data representations. Deep learning algorithms can use handcrafted features but do not depend on them like simple machine learning algorithms. Instead, they manage to learn and extract important features of the data themselves. Each hidden layer of the neural network model they use stores weights that parameterize the transformation they implement on the input data. Through an iterative training process, these algorithms try to find the best set of values for the weights of all layers in their network to improve their performance. This is done using a loss function that computes a distance score between their predictions and the true labels of the training data. This score is then used as a feedback signal to adjust the values of the weights using an optimizer that implements a backpropagation algorithm. The initial values of the weights are random, and they are slowly adjusted with each iteration to minimize the loss function and yield better results. The performance of these algorithms heavily depends on the design of the neural network architecture they use and accuracy, and the size of the training data [Goodfellow et al., 2016; Chollet, 2017].

2.1.3 Machine Learning VS Deep Learning

Table 2.1 summarizes the similarities and differences between a simple machine learning algorithm and advanced deep learning convolution algorithms.

Table 2.1: Comparison between the Random Forest classifier and deep CNNs.

Characteristic	**Machine learning (Random Forests)**	**Deep learning (Deep CNNs)**
Data dependency	Can perform very well with a small or a medium size of data.	Require a large amount of data to perform well.
Computer specifications	Can be implemented on low-end machines with a CPU of fairly decent specifications	Implemented on high-end machines and usually require a GPU.
Computational cost & time	Depends on the amount of data. Training of a descent performing model does not necessarily last long.	Depends on the amount of data and the depth of the neural network. Training of a descent performing model last long.
Training process	Depends on labeled data and handcrafted features.	Depends on the labeled data. Extracts meaningful features from the data during the training process.
Accuracy	Depends on the hyperparameters, the accuracy of the labeled data and the quality of the handcrafted features.	Depends on the architecture of the neural network and the accuracy and the size of the labeled training data.
Interpretability	Easy to almost impossible.	Difficult to impossible. Black boxes.

2.2 3D point cloud analysis and classification with Machine Learning and Deep Learning algorithms

Nowadays, 3D point clouds are ubiquitous as LiDAR sensors, laser scanning systems and drones are increasingly used for cheap, fast, dense, and reliable 3D mapping. The 3D representation of a scene in the form of a 3D point cloud facilitates various analysis tasks like object recognition, segmentation and classification. Automatically assigning 3D points a respective semantic class label is an important prerequisite for further applications such as 3D city modelling and significantly reduces the time and cost required to analyse 3D scenes [Weinmann, 2016; Wang et al., 2019]. Machine learning and deep learning algorithms could be used to do this, some of which are mentioned in the following subsections. However, the processing workflow of a point cloud classification task consists of multiple steps that vary and are manually (machine learning) or automatically (deep learning) done depending on the classification algorithm and the approach that was used. In general, a typical approach for 3D point cloud classification involves (i) recovering the optimal local neighbourhoods for all points, (ii) extracting the geometric features of those neighbourhoods, (iii) identifying the most informative and discriminating features, and finally, (iv) classifying all points based on those features (Figure 2.2).

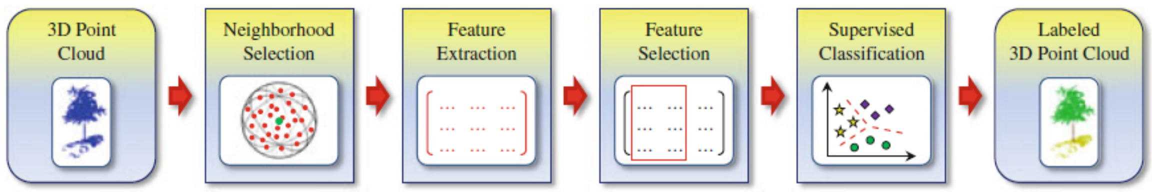


Figure 2.2: Framework presented by Weinmann [2016] for 3D point cloud classification. A respective generalized processing workflow consisting of four components for neighbourhood selection, feature extraction, feature selection, and classification.

In the following subsections, more details are provided for the crucial components of the processing workflow of a 3D point cloud classification task. General information is presented on how these processing steps are implemented when using machine learning and deep learning algorithms. Apart from the four major components we already mention, we also discuss point cloud sampling, an important processing step in the processing workflow of deep CNN architectures for point cloud classification that can also be used for machine learning algorithms.

2.2.1 Point cloud sampling

Point cloud sampling, also known as thinning, is the process of discarding a portion of the points to make a point cloud more manageable and quicker to visualise and process. This process is mostly done to tackle the major problem of high computational cost and time required for processing and visualising 3D point clouds. Also, in many cases, the point density of a point cloud may be higher than what is needed for a given application, and therefore redundant information has to be removed. In practice, commonly used sub-sampling methods for point cloud data thinning are the random, the nth-point, the minimal distance and the 2D or 3D grid methods. The first method randomly selects and discards a percentage of the points while the second method loops over them and keeps only the first point for every n points. The third method specifies a minimum distance that all points in the selected subset should have from each other, and the final method overlays a grid over the point cloud and keeps one point per grid cell. In this case, the number of points in the new sub-sampled point cloud depends on the specified size of the grid cells. The latter two methods can produce uniformly sampled point clouds as a result. In other words, they can achieve homogeneous spatial distribution of data points that is more suited for classification tasks [Ledoux et al., 2020; Poux, 2020].

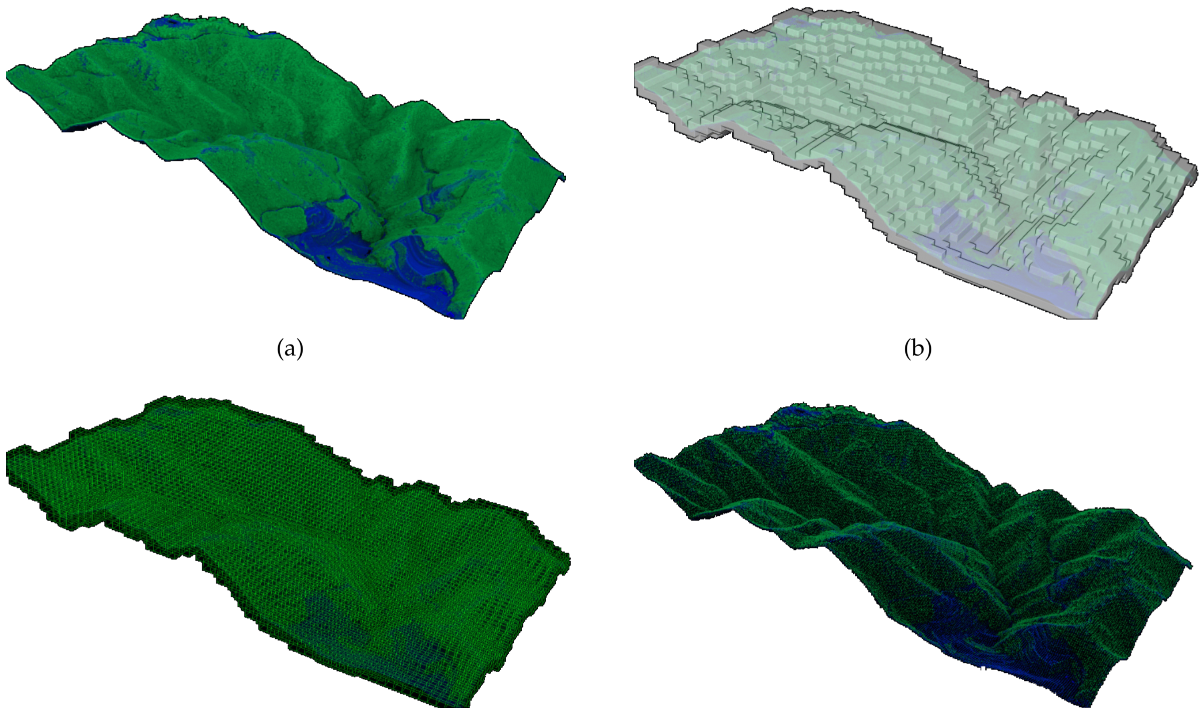

(a) (b) (c) (d)

Figure 2.3: Example of point cloud sampling using a 3D grid method. (a) Original point cloud, (b) voxelization, (c) voxel grid and (d) resulting uniformly sampled point cloud [Poux, 2020].

2.2.2 Neighbourhood selection

Neighbourhood selection is the first component in the processing workflow of a typical point cloud classification approach. It refers to the process of recovering the optimal local neighbourhoods for all points within a point cloud. In this case, a neighbourhood corresponds to a set of points in the vicinity of a point, for which different strategies may be applied to define it. Commonly applied strategies to define the neighbourhood of a point is using a sphere, a cylinder or its k-nearest neighbours. The spherical neighbourhood of a point is formed by all the points within a sphere that is centred at that point, with a fixed radius that must be specified. The cylindrical neighbourhood of a point is formed by all the points whose 2D projections onto a plane are within a circle centred at the projection of that point, with a fixed radius that must be specified. Lastly, the k-nearest neighbours neighbourhood of a point is formed by a fixed number of k closest points to that point according to a specified distance metric [Weinmann, 2016].

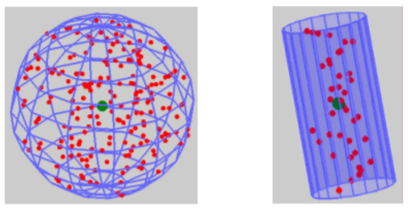

(a) (b)

Figure 2.4: Visualization of the (a) spherical neighbourhood, (b) cylindrical neighbourhood of a 3D point indicated by a green dot [Gross et al., 2007]. In the case of k-nearest neighbors using the Euclidean 3D distance as a distance metric and with the right specified number of k neighbors, the local neighborhood of the point would look the same as the spherical neighborhood.

These strategies of defining local point neighbourhoods are described as single-scale representations because they rely on using a constant scale parameter across all 3D points of a given point cloud. While efficient, these single-scale representation strategies do not consider the variations of the local properties of different neighbourhoods. Consequently, to increase the distinctiveness of the respectively derived geometric features for all local neighbourhoods, the scale parameters should not be the same for all 3D points. Information derived from multi-scale representations, which extract geometric features over multiple scales, facilitate the discrimination between different classes and improve classification tasks' results [Brodu and Lague, 2012; Niemeyer et al., 2014].

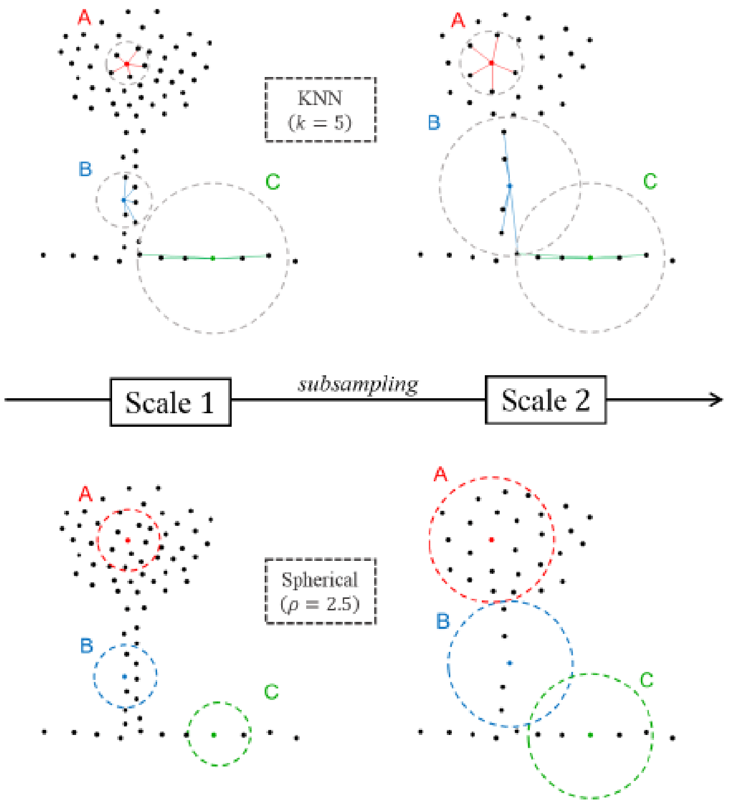


Figure 2.5: Behaviour of multi-scale neighbourhoods defined with k-nearest and spherical neighbourhoods [Thomas et al., 2018].

Thomas et al. [2018] highlighted some of the differences between multi-scale spherical and k-nearest neighbourhoods. As shown in Figure 2.5, spherical neighbourhoods can provide a more consistent geometrical meaning to the extracted features since they always correspond to a fixed part of the space, unlike k-nearest neighbourhoods. However, the number of points within those fixed area neighbourhoods can vary according to the point density of the input point cloud. If the input point cloud exhibits strong point density variations, then big scales and high point density areas will correspond to local neighbourhoods with too many points. At the same time, small scales and low point density areas will correspond to local neighbourhoods with too few points. Possible solutions for these two challenges is using a uniformly sub-sampled point cloud and multiple scales representations [Hackel et al., 2016].

2.2.3 Feature extraction and selection

Feature extraction and selection are the next components in the processing workflow of a typical point cloud classification approach. They refer to the process of extracting geometric features from the local neighbourhoods of the points and identifying which of them are the most informative and discriminating features.

As far as feature extraction is concerned, the basic geometric 3D properties and the local 3D shape of the 3D points within the local neighbourhoods could be used to directly extract a variety of 3D features. For example, valuable information about the points might be represented by their height values, which are derived by the Z coordinate of the points and by the vertical component of the normal vector of their neighbourhoods. Additionally, important features may be calculated using the normalized eigenvalues of the neighbourhoods, which encode information about the spatial distribution of their points.

Subsequently, during feature selection, it is considered that some of these features may be less relevant than others or even redundant. Thus, a suitable method must be followed to score or filter the features by evaluating the relations among them and between them and the classes. This is crucial since irrelevant information can influence the performance of a classifier, and a large number of features significantly increase the computational cost and time [Weinmann, 2016].

In general, we can group feature selection methods into three categories, namely filter-based, wrapper-based and embedded methods. Each category has its own advantages and disadvantages. Filter-based methods are fast, classifier-independent and could model feature dependencies, but they have no interaction with the classifier. On the other hand, wrapper-based methods interact with the classifier and could model feature dependencies as well. However, they are computationally intensive and depend on the classifier. Finally, embedded methods have similar pros and cons with wrapper-based methods, but they are faster and less computationally intensive [Saeys et al., 2007; Weinmann et al., 2015].

2.2.4 Classification with Machine Learning and Deep Learning algorithms

Nowadays, point clouds have become very popular and are used in several fields. Their increasing availability and the continuous decrease of the cost of LiDAR sensors enabled researchers to develop a series of novel deep learning algorithms to address the core problem of semantic segmentation of point clouds. Some of these algorithms focus on achieving more accuracy than others, while others strive for memory and computational efficiency. Each neural network architecture proposed over the years is a different approach to the problem and has its own strengths and weaknesses. Overall, 3D point cloud segmentation received a radical boost in performance due to the ongoing revolution of these data-driven deep neural networks.

With new methods being developed and new datasets becoming freely available, benchmarking is needed to identify which methods yield the best results for point cloud analysis tasks like classification. Multiple recent pieces of research implemented extensive experiments and provide benchmarks for comparing the performance of various machine learning and deep learning neural network architectures in the task of point cloud classification [Weinmann et al., 2015; Chaton et al., 2020; Hu et al., 2020; Varney et al., 2020]. Moreover, all methods discussed in this research provide their own benchmarks and comparisons with other state-of-the-art approaches. In most experiments, the KPConv [Thomas et al., 2019] neural architecture provides the best results in terms of Overall Accuracy (OA) and per-class accuracy. However, it does not outperform some of the other methods profoundly, and when tested with some datasets, it yields less accurate results. Furthermore, some of these deep learning methods and some machine learning algorithms might not be the most accurate ones, but they are a lot more efficient, and thus, they are still worth considering.

In the following subsections, more information is provided for some of these machine learning algorithms and deep learning architectures, focusing on the Random Forest classifier.

Random Forest Classifier

Decision Trees are a non-parametric supervised classification technique that learns simple decision rules inferred from the input data features and predicts the value of a target variable based on them [Wikipedia, 2021d]. Figure 2.6 shows how a decision tree makes a prediction and its main parts. It can be explained as a set of nodes and edges organized as a hierarchical structure. Each internal node contains a function that is used to test the input data and makes a decision. Each branch corresponds to the outcome of that test, and each terminal node contains a class label which is the prediction of the target variable. The uppermost node in the tree is called the root node. It can be interpreted as a hierarchical piecewise model that splits complex problems into a hierarchy of simpler ones [Criminisi et al., 2011].

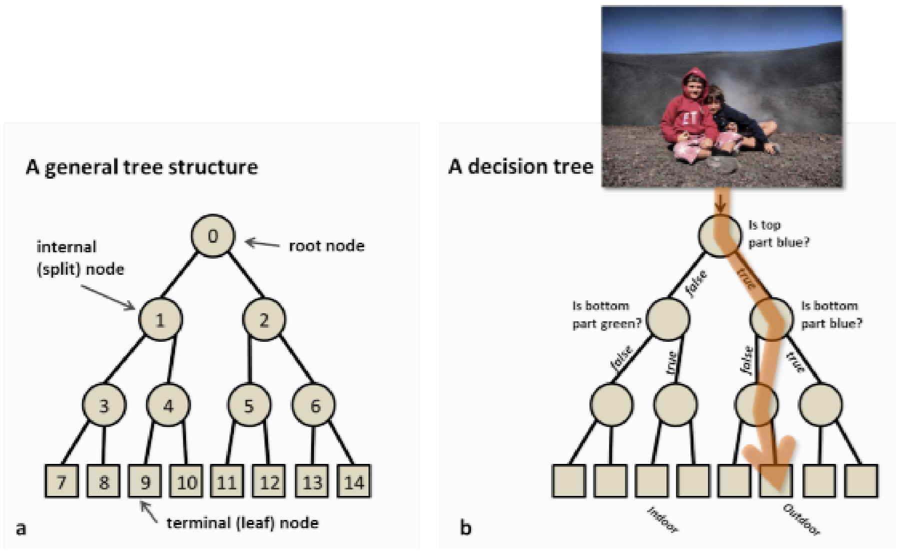


Figure 2.6: (a) The main parts of a decision tree. (b)Illustration of a decision tree that attempts to understand whether a photo represents an indoor or outdoor scene [Criminisi et al., 2011].

The Random Forest classifier is a fairly simple and relatively easy to understand machine learning method for solving classification problems. It is an ensemble method consisting of multiple smaller decision trees trained on a random subset of the training sample and produce their own predictions, which are then aggregated to produce a more accurate prediction. Random Forests can handle well large datasets with high dimensionality and heterogeneous feature types. Furthermore, unlike standard decision tree classifiers, they are not prone to overfitting, and they can provide estimates of what variables are important in the classification [Breiman, 2001]. Overall, the Random Forest classifier provides a good trade-off between accuracy and efficiency compared to other machine learning and deep learning algorithms [Weinmann et al., 2015].

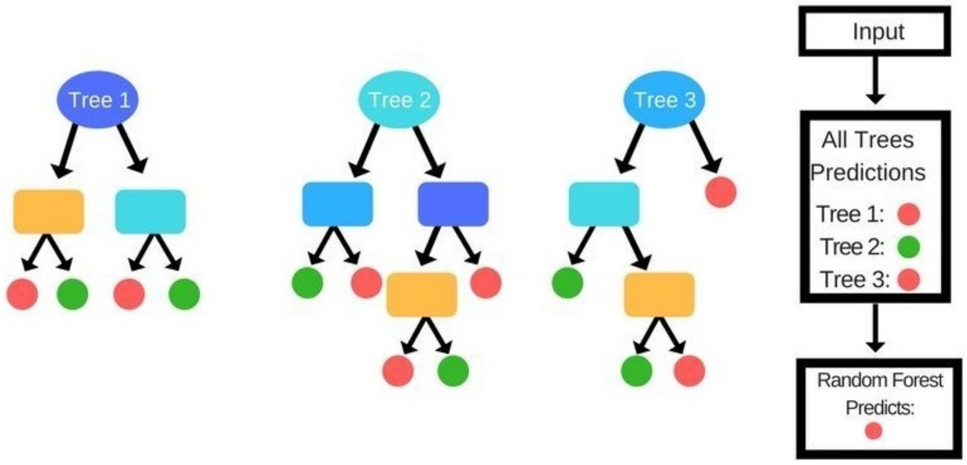


Figure 2.7: Diagram example of majority voting in a Random Forest Classifier [Polamuri, 2017].

Like many other simple machine learning algorithms, the accuracy of the Random Forest classifier depends on the quality of the input handcrafted features vector, and its hyperparameters greatly influence it. Having distinguishing features in our input vector can decrease the inhomogeneity of the labels at the nodes of the trees and make them highly uncorrelated, which leads to better splits and eventually better results. As for its hyperparameters, the Random Forest classifier has several ones that must be specified and should be optimized to improve its performance. Below we list and provide information for some of the most important hyperparameters of the Random Forest classifier:

- K number of features randomly selected at each node during the tree induction process

Bernard et al. [2009] explain that this number is an integer that takes values from 1 to M, with M being the number of features in the input handcrafted features vector. The smaller the value of K, the more the randomization in the feature induction process. If $K = 1$, then only one feature is randomly selected each time the tree structure is split. On the contrary, if $K = M$, then all the available features are used to grow each tree, and thus, no randomization is introduced. The value of the K parameter also influences the stability and accuracy of the trees. Low K values lead to less correlated trees that are more stable when aggregating and exploit better less significant features. However, they also lead to having trees that underperform on average since they are built based on suboptimal features [Probst et al., 2019]. Through experimentation, Geurts et al. [2006] have proven that when the K parameter is set equal to $\sqrt{M}$ is most of the times close to the optimal setting. In their experimental work, Bernard et al. [2007] introduced another value for the K parameter, $\log_2(M) + 1$, which was later used in further experiments [Bernard et al., 2009] together with $\sqrt{M}$ and were found to yield accurate results and sometimes exactly correspond to the best parametrization of K. However, these experiments also showed that the global relevancy of the features should also be taken into account when deciding what the value of the K parameter will be. In other words, if we have too many irrelevant, and not important features, the randomization will rapidly decrease the accuracy of the trees, while having too many features that provide important information facilitates the overfitting of trees and weakens the randomization effects in split selection. Finally, the experimental work of Wright and Ziegler [2015] illustrated that the computation time of a Random Forest classifier is reduced, almost in a linear way, with lower K values.

- N number of trees in the Random Forest

The research conducted by Oshiro et al. [2012] suggests, based on the findings of their analysis on different datasets, that the optimal number of trees that should be used within a Random Forest ranges between 64 and 128. Increasing the value of N more than that did not improve the performance significantly and only increased the computational cost. However, to find the best value for the hyperparameter N and obtain optimal performance, the dataset's characteristics should also be considered. Usually, the best performance can be achieved when growing the first 100 trees [Oshiro et al., 2012; Probst and Boulesteix, 2017]. Lastly, the experimental work of Wright and Ziegler [2015] illustrated that the computation time of a Random Forest classifier increases linearly with the number of trees.

- Node size

The node size hyperparameter controls the complexity of the trees by specifying the minimum number of observations contained in the terminal nodes for predictions [Probst et al., 2019]. The experimental works of Díaz-Uriarte and De Andres [2006] and Goldstein et al. [2011] have shown that the default value used for this hyperparameter, which is 1 for classification and 5 for regression, tends to produce good results. However, Segal [2004] illustrated that if the predictors of our data are noisy and if higher numbers of features considered for splitting a node are performing best, then optimal node size to improve performance is going to be higher. Moreover, increasing the node size means that the tree complexity is reduced, which usually decreases the computation time substantially.

- Splitting rule

The splitting rule of a Random Forest classifier is the function used to measure the quality of a split. It maximizes the impurity reduction introduced by a split. Every node in the tree is a condition on a single feature that splits the dataset so that similar response values end up in the same set. The term node impurity refers to how homogeneous are the labels of a node which can be calculated using measures like the Gini impurity and entropy [Nembrini et al., 2018; Ishwaran, 2015; Breiman, 2001]. As stated on Wikipedia [2021d], ”Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset." On the other hand, entropy is a measure of uncertainty or randomness that is higher for variables with more randomness and uniform distribution. Subsequently, impurity could also be used to calculate the importance of the features. For example, if a split has a large decrease of impurity, then the feature used for splitting is considered to be important. Based on this, the impurity importance of a feature can be computed as the mean and standard deviation of the accumulation of the impurity decrease of all nodes in the forest where a split has been made using that feature [Nembrini et al., 2018].

- Size of samples and replacement (bootstrap samples)

The sample size hyperparameter determines the number of data points used to train the trees. Low values tend to increase the randomness between the trees, however, they also reduce their prediction accuracy. The optimal value of the sample size is found where the best trade-off between stability and accuracy of the trees is achieved [Probst et al., 2019].

Bootstrapping is a random sampling method used to estimate statistics on a population by sampling a dataset with replacement [Wikipedia, 2021a]. Usually, each tree in a Random Forest is trained using the whole input dataset. When bootstrapping is enabled, the training samples of each tree are drawn with replacement. This means that each tree will be trained using different samples, the so-called bootstrap samples and that some samples from the input dataset will be used multiple times for their training. Out-of-bag (OOB) score is a metric used to validate a random forest when bootstrapping is used. During the training process of each tree, some of the samples are not included in their bootstrap training samples. These are the so-called out-of-bag samples, and the out-of-bag score can be calculated as the mean prediction error of the trees on these samples [Wikipedia, 2021g].

The work of Martínez-Muñoz and Suárez [2010] suggests that when the number of data points that are drawn from the input training dataset to train each tree is set optimally, there is no significant difference in the performance if sampling with replacement or without replacement is used. Moreover, their analysis concludes that the optimal value for this parameter is problem dependent and can be estimated with out-of-bag predictions. In most problems, they investigated that the performance was better when its value was set smaller than the default one, which is equal to the number of data points in the input training dataset. In addition, using smaller training samples reduces the runtime.

Multi-layer Perceptron

In machine learning, neurons are used to process weighted inputs by applying an activation function on them and return an output. The input values can be features extracted from the training data or the outputs of the neurons in the previous layer of the network. Different weights are applied to each input, and the activation function is applied to the sum of weighted inputs to predict the output. An additional constant, the bias, is attached to the neuron, and it is added to the weighted input before the activation function is applied to them.

As stated on Wikipedia [2021f], a perceptron or a single-layer neural network/perceptron is an algorithm for binary classification that uses a linear prediction function (activation function) and combines a set of weights with the input feature vector to decide whether or not it belongs to some specific class. Single-layer perceptrons do not contain any hidden layers, and they are the simplest feedforward neural networks, which means that information moves in only one direction within their network, from the input nodes to the output nodes. Figure 2.8 illustrates the procedures of a single-layer perceptron.

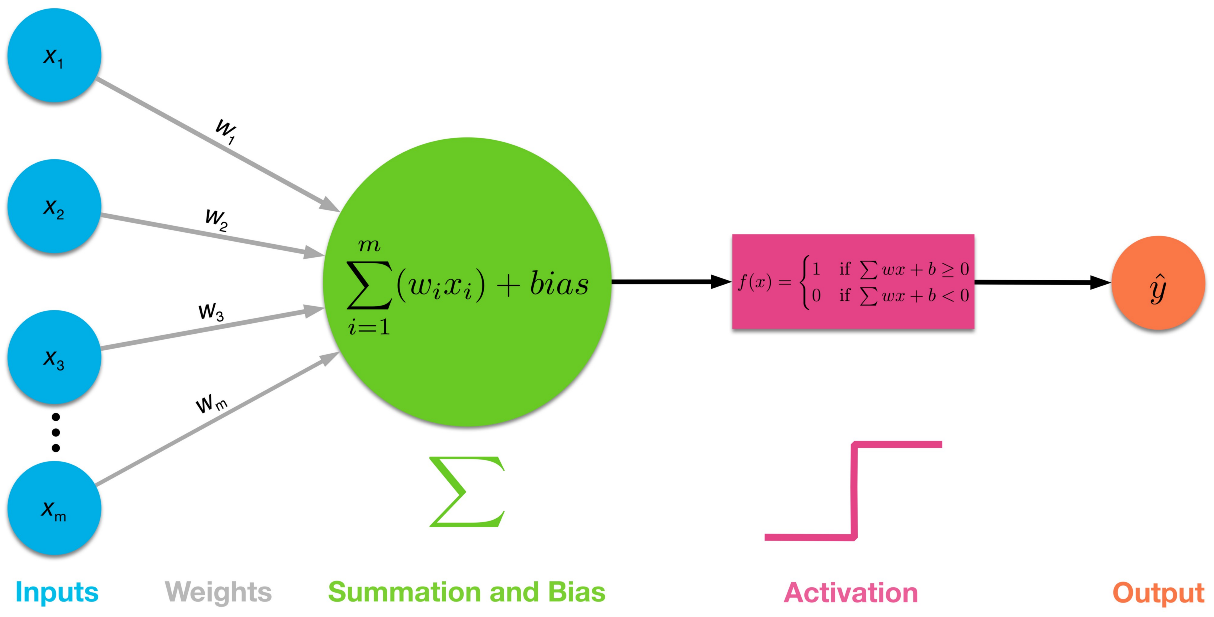

x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

Figure 2.8: Single-layer Perceptron using a threshold function [Kang, 2017].

A Multi-layer Perceptron (MLP) is a supervised learning algorithm and a class of feedforward artificial neural networks that consists of at least three layers of perceptrons, an input layer, a hidden layer, and an output layer. In other words, it is a cascade of single-layer perceptrons. What distinguishes MLPs from single-layer perceptrons is that the neurons of their hidden and output layers can also use non-linear activation functions. This enables them to model highly complex relationships between features and distinguish data that is not linearly separable. However, MLPs require hyperparameters tuning, they are sensitive to feature scaling and with different random weight initializations they can lead to different validation accuracy.

PointNet and PointNet++

According to Qi et al. [2017b], PointNet++ is a deep CNN with a hierarchical feature learning architecture that can process a set of points sampled in a metric space in a hierarchical fashion and is able to learn deep point set features efficiently and robustly. To do that, it carries out an iterative process that uses a distance metric to partition the set of points into overlapping local regions and extracts local features capturing fine geometric structures from small neighbourhoods. These features are then further grouped into larger units and processed to produce higher-level features. The process ends when the features for the whole point set are obtained.

PointNet [Qi et al., 2017a] is the predecessor of PoinNet++, which has proven to be efficient for processing 3D points and extracting semantics. However, it suffers from limitations when attempting to recognize fine-grained patterns and to generalize in complex scenes. To tackle these issues, PointNet++ applies PointNet recursively on a nested partitioning of the input set and uses it as its local feature learner to abstract sets of points or local features into higher-level representations.

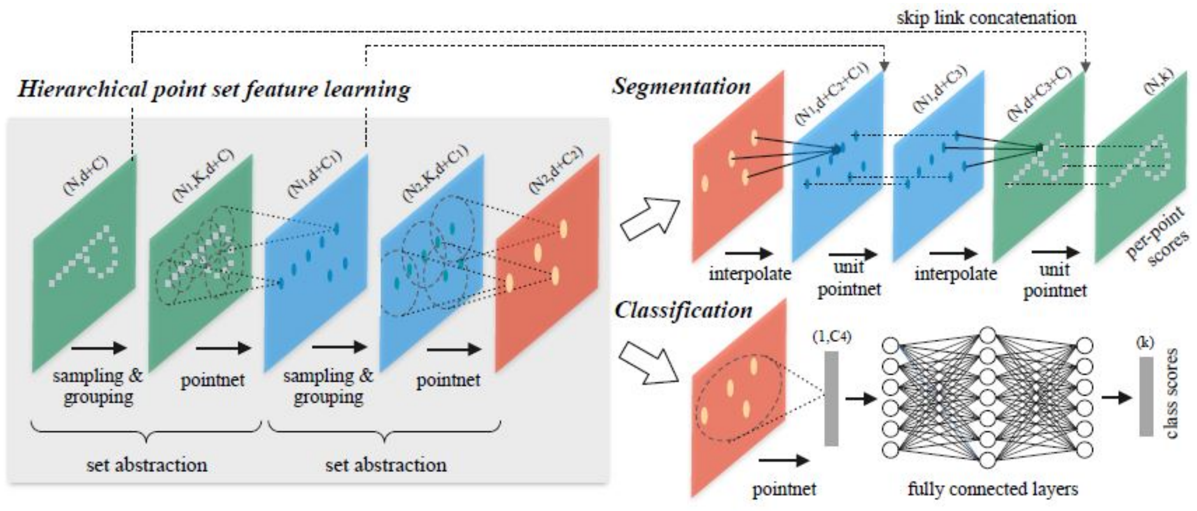


Figure 2.9: Illustration of the PointNet++ architecture and its application for set segmentation and classification using points in 2D Euclidean space as an example [Qi et al., 2017b].

What really makes PointNet++ special is its ability to achieve both robustness and detail capture by taking full advantage of neighbourhoods at multiple scales, and processing point sets efficiently and robustly, which was proven by experiments. With the help of random input dropouts during the training process, the network learns to adaptively weight patterns detected at different scales and combine multi-scale features according to the input data.

KPConv

The Kernel Point Convolution (KPConv) deep learning algorithm presented by Thomas et al. [2019] is a point convolution operator that consists of a set of local 3D filters. Similarly to image-based convolution, it uses a set of kernel points to define the areas where each kernel weight is applied. A correlation function defines the area of influence of the kernel points which carry the kernel weights. More importantly, the number of kernel points can vary, which increases the flexibility of the algorithm. Through the analysis of the results of experiments, KPConv was proven to build and train fast very deep architectures for classification and segmentation.

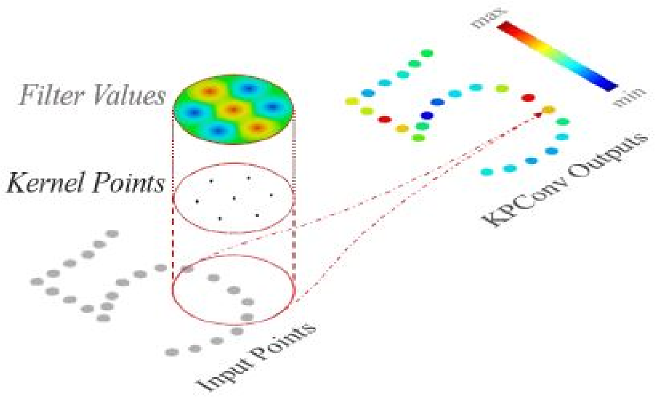


Figure 2.10: Rigid KPConv illustrated on 2D points [Thomas et al., 2019].

KPConv has a deformable version as well, which consists of learning local shifts applied to the kernel points and performs as good as its rigid version. Through this method, the algorithm manages to adapt the shape of its kernels for different regions of the input cloud by generating different shifts at each convolution location. However, flaws occur when kernel points are shifted away from the input points and to ensure that the deformed kernels fit the point cloud geometry and that we are not left with empty spaces, a regularization loss is needed to be applied.

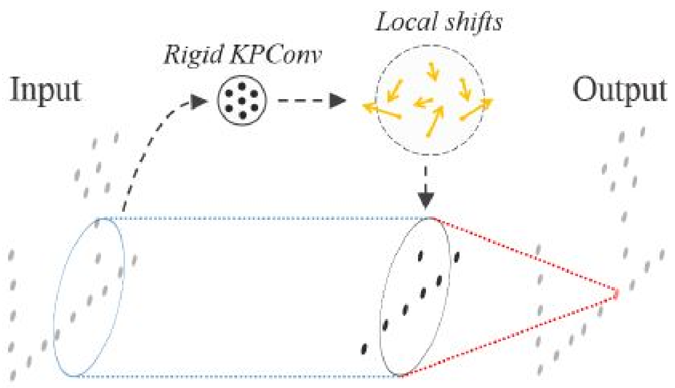


Figure 2.11: Deformable KPConv illustrated on 2D points [Thomas et al., 2019].

The two versions of KPConv have their own strengths. For example, the rigid version performs better on simpler tasks, like object classification or segmentation of small datasets, while the deformable one achieves high accuracy for complex tasks, like segmentation of large datasets with multiple varying object instances. Moreover, the deformable version is more robust to a lower number of kernel points, which implies a greater descriptive power, and overall it improves the algorithm's ability to adapt to the geometry of the scene objects. But whether someone uses the deformable version of KPConv for complex tasks or the rigid version of KPconv for simpler tasks, the benchmarking provided by Thomas et al. [2019] on several datasets indicates that KPConv will provide them with better results than several other state-of-the-art classification and segmentation approaches.

PointCNN

The PointCNN method proposed by Li et al. [2018] is a generalization of typical CNNs to feature learning from data represented in the point cloud. Operator $\mathcal{X}$-Conv, which is the core of PointCNN, weights and permutes the input points and features that are associated with them before they are processed by a typical convolution operator that applies on them element-wise product and sum operations. $\mathcal{X}$-convolutional layers in PointCNN work in a similar way as convolutional (Conv) layers in grid-based CNNs (Figure 2.12). The only difference they have is the way they use to extract local regions and learn their information.

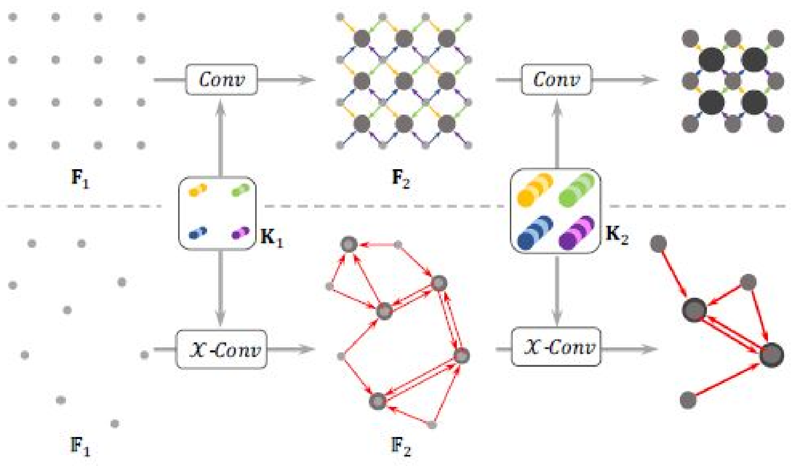


Figure 2.12: Illustration of a hierarchical convolution using Conv layers of typical CNNs on a regular grid and $\mathcal{X}$-Conv layers of PointCNN on a point cloud [Li et al., 2018].

While this method still has some room for improvement and open problems for future work, it manages to produce similar or even better results than other state-of-the-art methods, including PointNet++ [Qi et al., 2017b], when tested with multiple challenging benchmark datasets and tasks.

ConvPoint

The ConvPoint framework presented by Boulch [2020] is a generalization of discrete CNNs for point clouds that replaces discrete kernels with continuous ones. More specifically, it is a formulation of continuous convolutions that are simple and straightforward extensions of the discrete ones, designed for unstructured data, which can be used to build neural networks similarly to 2D CNNs. Figure 2.13 illustrates the continuous convolution formulation introduced in the work of Boulch [2020]. To establish a relation between the points $\{p\}$ of the input point cloud and the kernel elements $\{c\}$, a geometrical weighting function is applied that distributes them onto the kernel. The function is independently applied on each point to be invariant to their permutations and uses relative coordinates with respect to kernel elements to be invariant to global translations applied on the points and the kernel. The spatial part c and feature part $\{w\}$ of a kernel $\{(c, w)\}$ are processed separately. If the green arrows and boxes (spatial operations) are removed from Figure 2.13 then the convolution operation becomes discrete. This convolution operation is applied locally on a subset of the input point cloud.

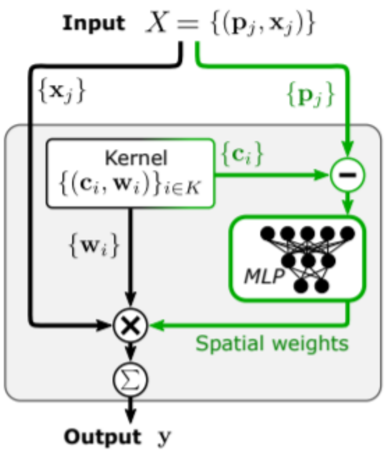


Figure 2.13: Convolution operation, black: features operations, green: spatial operations[Boulch, 2020].

Figure 2.14 represents the convolutional layer applied on an input point cloud P, and optionally $\{q\}$, that uses continuous convolutions. If not provided, $\{q\}$ can be selected by computing the local neighbourhoods of each q. For each point of $\{q\}$, local neighbourhoods in P are computed, and convolution operation is applied on them, creating the output features $\{y\}$. The result is $Q = \{(q, y)\}$, which is the union of the pairs.

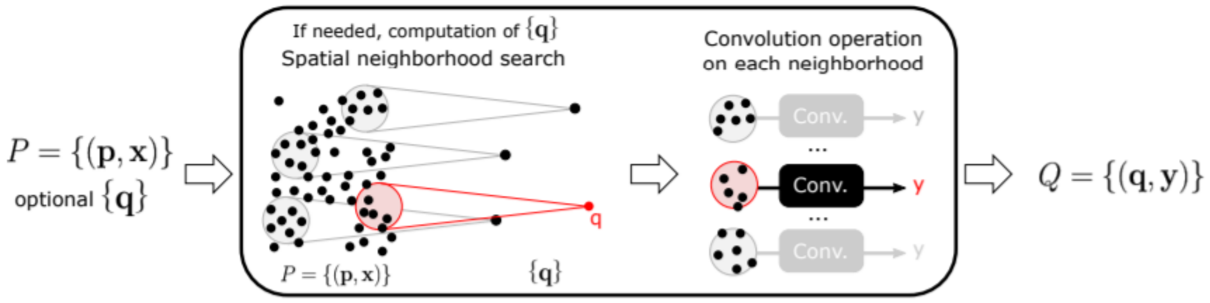

$P = \{(\mathbf{p}, \mathbf{x})\}$
optional $\{\mathbf{q}\}$
If needed, computation of $\{\mathbf{q}\}$
Spatial neighborhood search
$P = \{(\mathbf{p}, \mathbf{x})\}$
$\{\mathbf{q}\}$
q
Convolution operation on each neighborhood
Conv.
y
$Q = \{(\mathbf{q}, \mathbf{y})\}$

Figure 2.14: A convolutional layer consisting of two steps, spatial structure computation and applying the convolution operation, defined in Figure 2.13, on each neighborhood [Boulch, 2020].

ConvPoint can be applied to various classification and segmentation tasks, including large scale indoor and outdoor semantic segmentation. Experiments show that its performance for each task is comparable to other state-of-the-art methods.

SuperPoint graph representation

Landrieu and Simonovsky [2018] presented a deep learning framework to tackle the challenge of semantic segmentation of large-scale LiDAR data that consists of PointNets [Qi et al., 2017a] for superpoint embedding and graph convolutions for contextual segmentation. By using superpoint graph representations of point clouds, the proposed method enables the application of deep learning on large-scale point clouds without sacrificing fine details and with long-range interactions. Superpoint graph representations partition the point cloud data into geometrically homogeneous elements with rich edge features that efficiently encode the contextual relationship between object parts, which can then be used by deep learning algorithms based on graph convolutions to classify their nodes. Figure 2.15 illustrates the application of this framework on a toy scan of a table and a chair.

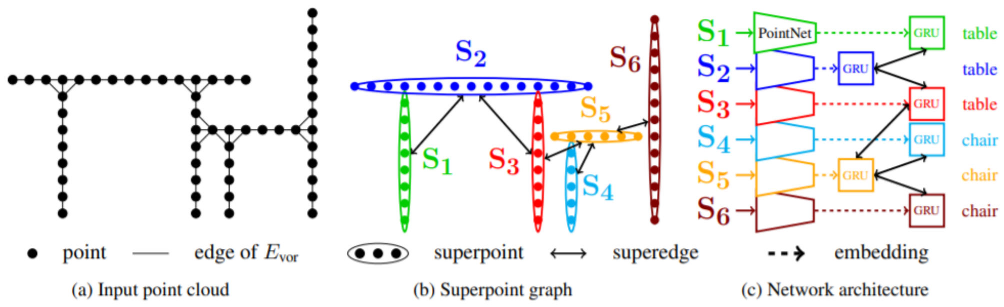


Figure 2.15: (a) Geometric partitioning of the input point cloud. (b) Superpoint graph representation of the input point cloud. (c) Superpoints embedding with PointNet networks. The embeddings are refined in Gated Recurrent Units by message passing along superedges to produce the final labelling [Landrieu and Simonovsky, 2018].

ShellNet

ShellNet is an efficient neural network introduced by Zhang et al. [2019] that directly consumes point clouds with large receptive fields while maintaining fewer layers. Its core operator is ShellConv, a simple and effective convolution operator that uses statistics from concentric spherical shells to define representative features. ShellConv is permutation invariant for input points, allowing traditional convolution to perform on such features. Figure 2.16 shows the architecture of the ShellNet for classification and segmentation. Typical 2D convolutional neural networks inspired the architecture but it uses ShellConv instead of 2D convolution. Three layers of ShellConv are applied on the input point cloud before the fully connected classifier in the case of classification, while a U-net architecture is followed in the case of segmentation.

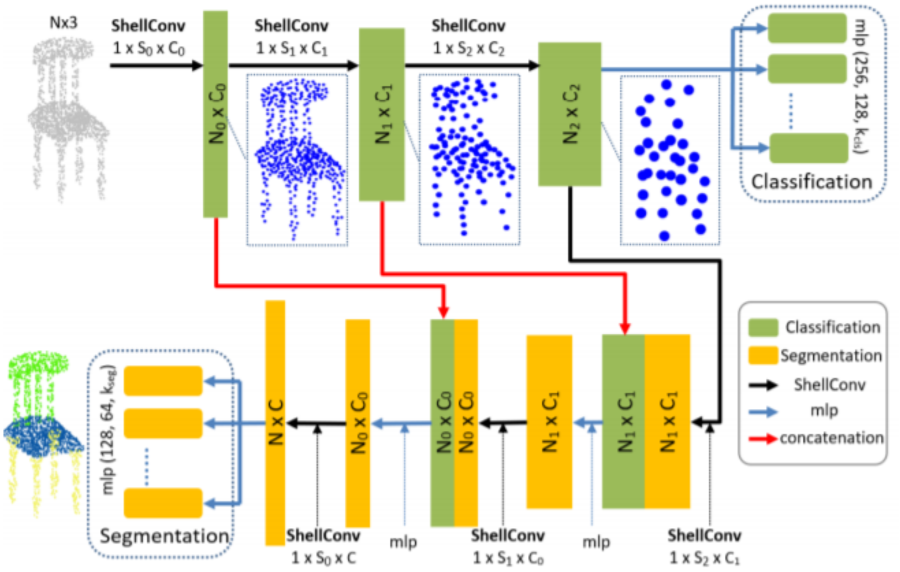


Figure 2.16: ShellNet architecture for classification and segmentation. The encoder is in green, and the decoder is in yellow. Point downsampling and upsampling are also included in the convolution, depending on its use. Zhang et al. [2019] explain that "$N_0 > N_1 > N_2$ denotes the number of points in the input and after being sampled in each convolution, and $C < C_0 < C_1 < C_2$ denotes the output feature channels at each point. $S_0 > S_1 > S_2$ denotes the number of shells in each ShellConv operator that is analogous to the convolution kernel size. Given a fixed shell size, the number of shells also decreases when the point cloud is downsampled. $1 \times S_0 \times C_0$ denotes a convolution that convolutes an input features using kernel $(1, S_0)$ and output C_0 feature channels."

ShellNet can be used for 3D scene understanding tasks such as object classification, object part segmentation, and semantic scene segmentation. Experiments show that it can achieve high accuracy that is comparable to other state-of-the-art methods when applied on publicly available datasets.

3 Methodology

Based on the research objectives defined in Chapter 1: Introduction, and the related scientific research and knowledge reviewed in Chapter 2: Related work, an analysis is provided in this Chapter of the methodology developed to address the research questions of this graduation project. Particular focus is given on describing the design of the implemented experiments and what mathematical equations and algorithms were used from a conceptual point of view. In the following sections, an overview of the developed methodology is presented, and information about the classification framework that was followed is provided. Lastly, the data processing methods used and the evaluation metrics calculated for the classification results are briefly described.

3.1 Overview

This section provides an overview of the classification approach that was developed and used to train and test our machine learning models. The methodology is broken down into several steps, which are explained in detail in the following sections.

The machine learning algorithm that was decided to be used for the experiments and to address the research questions of this graduation project is the Random Forest Classifier. More specifically, the implementation of the Random Forest algorithm via the scikit-learn Python library [Pedregosa et al., 2011] was used since all of our code was written in Python. What is different for this implementation[1] of the Random Forest algorithm compared to the one Breiman [2001] defined is that it combines classifiers by averaging their probabilistic prediction. As for our classification approach, we followed a similar processing workflow like the one proposed by Weinmann [2016].

Multiple changes have been made to our methodology over time, such as adding extra processing steps or using fewer features and multi-scale neighbourhoods, to tackle the memory bottlenecks we encountered. The final training and testing processing workflows of the classification approach developed for this graduation project, while taking into account the limitations faced during the initial experiments, consists of five major components and some smaller processes as depicted in Figures 3.1 and 3.2. The five major components that compose the 3D point cloud analysis framework are the following:

- **Uniform sampling** using a 3D voxel grid.
- **Neighborhood selection** using multi-scale spherical neighborhoods.
- **Height features extraction** using cylindrical neighbourhoods with large radii and the Z coordinate of the 3D points.
- **Eigen-based features extraction** using the normalized eigenvalues of the multi-scale spherical neighbourhoods.
- **Feature selection** by examining the importance of all features and filtering out the most insignificant ones.

Our Python code was split into two separate scripts. One for training the Random Forest models (Figure 3.1) and one for testing the resulting trained models (Figure 3.2). The analysis for the training and testing data, however, is the same. Analyzing the data used for training and testing the Random Forest models is a very time-consuming task. Due to this, the training and testing data analysis were done separately to first make a visual inspection of the extracted features, decide which of them are the most

[1]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

important ones, and check if our code works well before testing the trained models. The first step in our methodology is to merge and relabel the input classified point cloud into three classes: ground, building, and other. After that, we continue with our 3D point cloud analysis framework. A few extra processing steps were added at the end of our testing process to assign a class label to all input point cloud points and calculate the classification evaluation metrics.

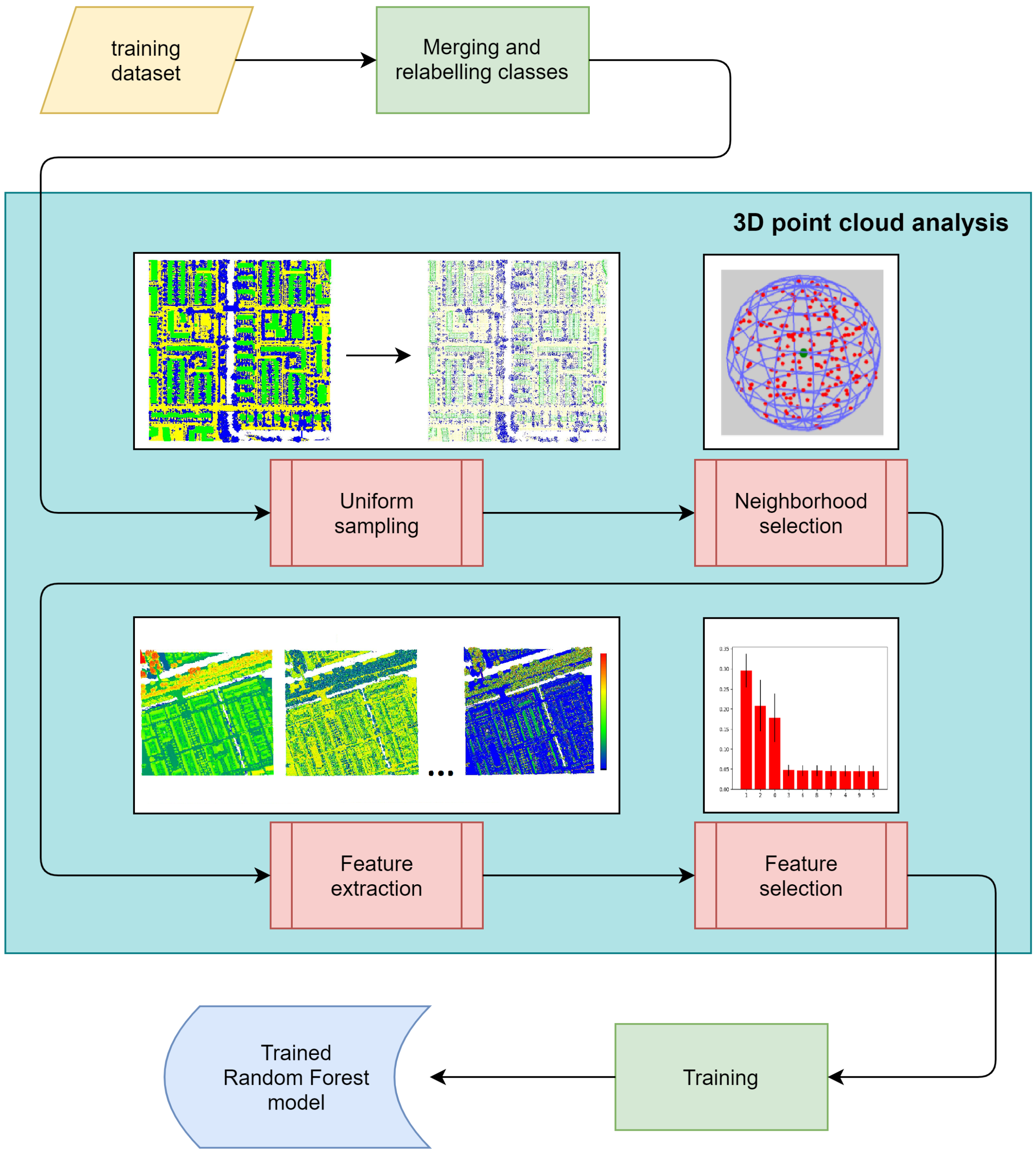


Figure 3.1: Flowchart diagram of the methodology followed to train different Random Forest classification models.

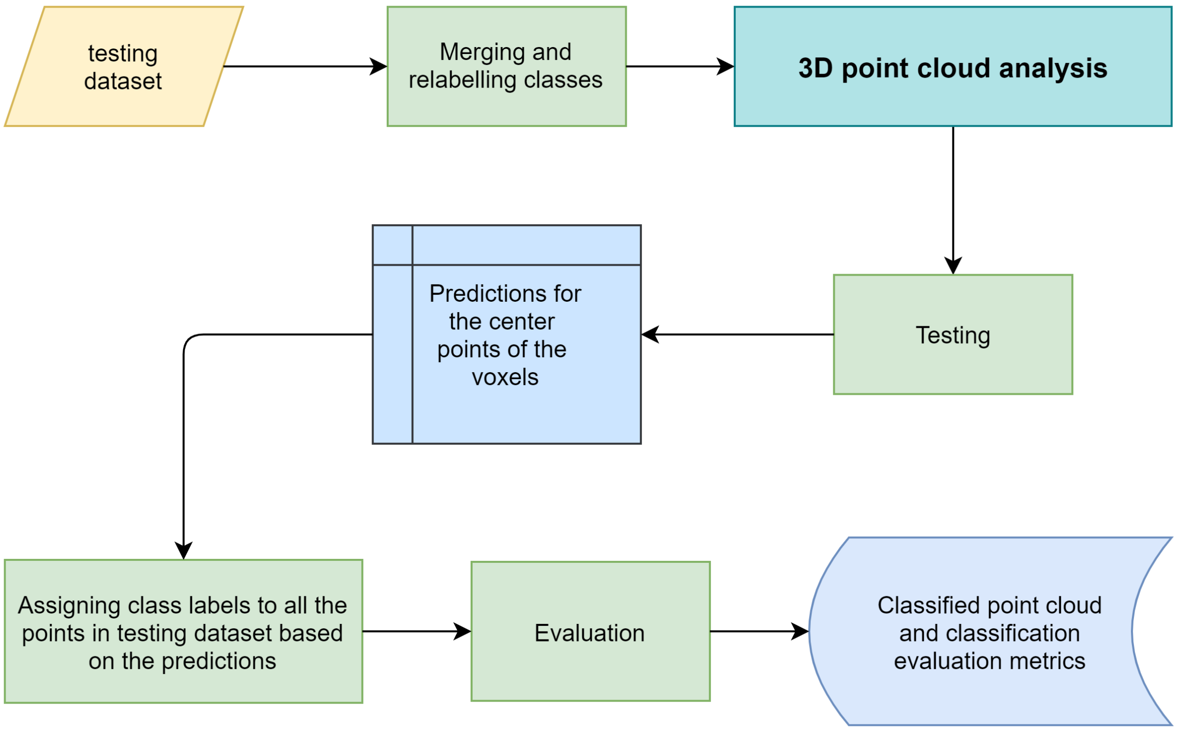

testing dataset
Merging and relabelling classes
3D point cloud analysis
Testing
Predictions for the center points of the voxels
Assigning class labels to all the points in testing dataset based on the predictions
Evaluation
Classified point cloud and classification evaluation metrics

Figure 3.2: Flowchart diagram of the methodology followed to test the Random Forest classification models. The "3D point cloud analysis" step refers to the same series of processing steps as the ones noted in Figure 3.1.

3.2 Data diversification

The quality of the training and testing data plays an important role in achieving optimal model performance. The right data can provide a model with more information during the learning process, and the learned model can be a better fit for the data. Therefore, the first thing that we did was decide which point cloud tiles should be used to train and test our models. Data diversification is a method used to improve the performance of a model by using more informative and less redundant samples of data with enough information to train the model. Simply put, the increase of diversity in training data aims to maximize the information contained in the data [Gong et al., 2019; Shi et al., 2021]. Based on this method, we compared the feature diversities of our point cloud tiles to identify which are the most suitable ones to be used as training and testing data. To compute the feature diversity F_m of a tile m the following equation was used:

$$F_m = \frac{\sum_{i=1}^{N_f} \sigma_i^2}{N_f} \tag{3.1}$$

where σ_i^2 represents the variances of the handcrafted features of the N_f dimensional feature vector.

Before we compute the variances of all features within a tile, we first apply a min-max normalization on the values of each feature. For every feature, its minimum value is transformed into a 0, the maximum value is transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. The formula of the min-max normalization is as follows:

$$value_{normalized} = \frac{value - min}{max - min} \tag{3.2}$$

In addition, to decide which tiles are the best, we also considered the number of classes each tile has and calculated the percentages of points that belong to each class. Tiles with points from all three classes were considered more important than others despite having a smaller feature diversity score. Finally, we visually inspected all tiles and discarded some of them because the environment they represent, i.e. flat areas and construction sites with cranes, led to false high and low Z features values.

3.3 Point cloud classification framework

In this section, we briefly explain the ideas behind the processing steps of our classification methodology. The five major components of our 3D point cloud analysis framework are individually addressed. We discuss their purpose, what mathematical equations were used and what experiments we have done to help us decide which available approaches yield the best results.

3.3.1 Merging classes and relabelling

Before we begin analysing the training and testing data, we first relabel the input AHN classified point cloud into three classes, namely ground, building and other, since we are not interested in other classes for building reconstruction. In our initial experiments, we kept the original building and ground classes of the AHN dataset and merged all other classes into a new class, the other class. Our initial results showed that our classification model could not distinguish the water points and bridges from the ground. Thus, we decided to merge these two classes, 9 and 26, with the ground class since we mainly focus on assisting building reconstruction methods, and this decision does not affect our scope. However, this decision comes with a risk because class 26 contains other man-made structures, which could cause our classification model to underperform for the building and other classes. Thankfully, only a few of these man-made structures can be found in our data, and no significant drop has been noted for the ground and other class evaluation metrics after this change.

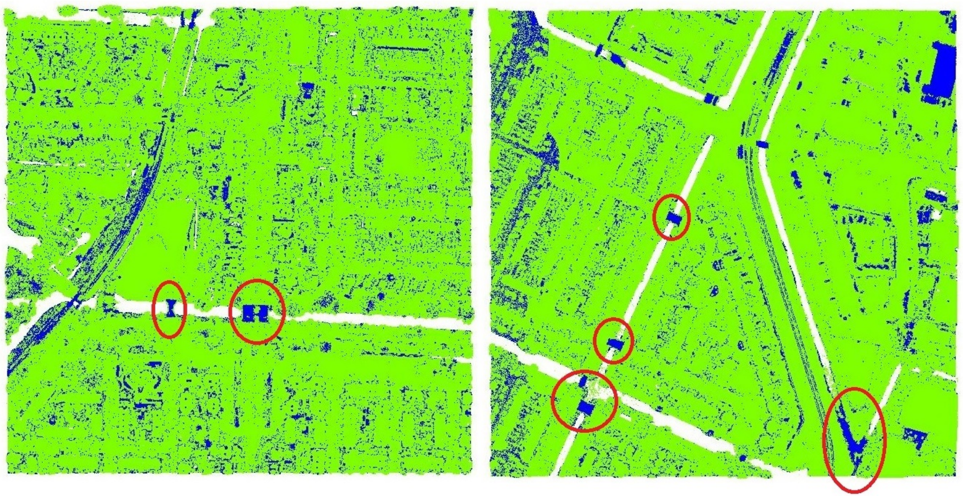


Figure 3.3: Error maps of initial experiments. Bridges and water points in the red circles are misclassified as ground. Blue = misclassified points, Green = correctly classified points.

Table 3.1: Class labels that are merged and relabelled into our new ground, building, and other classes.

Merging classes and relabelling			
Class	Class label	Corresponding AHN classes	Corresponding DALES classes
other	0	all classes except ground (2), building (6), water (9) and special infrastructural work (26)	vegetation (2), cars (3), trucks (4), power lines (5), fences (6), poles (7)
building	1	building (6)	building (8)
ground	2	ground (2), water (9), special infrastructural work (26)	ground (1)

3.3.2 Uniform sampling

To reduce the computational cost and time required to train and test our Random Forest models, the input point cloud datasets are first uniformly sampled using a 3D grid method described in the online article of Poux [2020]. Using the original input point clouds that contain millions of points slowed down the whole process when attempting to loop over the points and calculate their local neighbourhoods. The uniformly subsampled point clouds with fewer points helped speed up the training and testing process and, consequently, have a more efficient classifier.

The sampling process is simple and straightforward. It can be broken down into six steps. First, the bounding box of the input point cloud is found using the minimum and maximum values of the X, Y, Z coordinates of the 3D points. Secondly, the bounding box is discretized into voxels by dividing the bounding box dimensions with the user-specified voxel size. Thirdly, we identify which voxels contain points and separate them from the empty ones. Then we calculate the number of points and the indexes of those points in all non-empty voxels. Finally, an iterative procedure begins where we loop over all non-empty voxels and find which points are closer to their centres. Those points and their labels are stored and appended in the new subsampled point cloud. The computational complexity of the process is $O(n)$ since it uses a single for loop. The overview of the uniform sampling process is provided in Algorithm 3.1.

Algorithm 3.1: UNIFORM SAMPLING (*pcd*, *vs*) [Poux, 2020]

```
Input: A 3D point cloud $pcd$, and the voxel size of the 3D grid $vs$
Output: $pcd_s$: a uniformly sampled version of $pcd$
1  find the 3D bounding box of the $pcd$
2  discretize the bounding box into voxels of size $vs$
3  $v_{keys} \leftarrow$ find the indexes of non-emtpy voxels
4  $ids \leftarrow$ calculate the sorted list of the indexes of the points in the non-empty voxels
5  $n \leftarrow$ calculate the number of points in each voxel
6  $last\ seen = 0$
7  $pcd_s = nil$
8  for $idx, vox$ in $enumerate(v_{keys})$ do
9      $pts \leftarrow$ get the $last\ seen : n_{idx}$ point indexes from the $ids$ list
10     $p \leftarrow$ get the closest point to the center of $vox$
11     $pcd_s \leftarrow$ append $p$
12     $last\ seen += n_{idx}$
13 return $pcd_s$
```

3.3.3 Neighbourhood selection

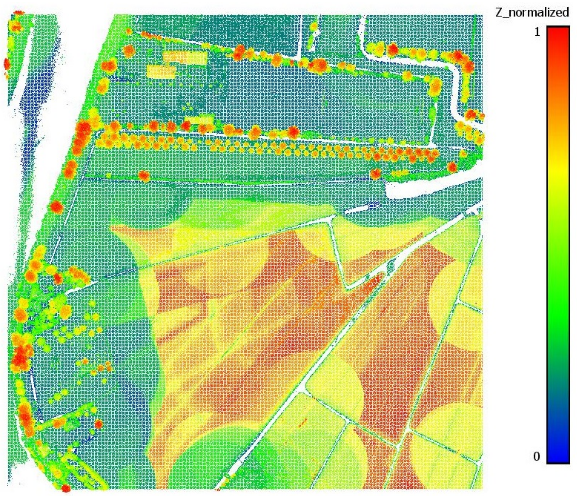
For this graduation project, we decided to use two of the three 3D point neighbourhood definitions mentioned by Weinmann [2016] to extract meaningful handcrafted features from our data. More specifically, for our eigen features and point density feature, we used multi-scale spherical neighbourhoods, and for our height features, we used a cylindrical neighbourhood with a large radius.

An alternative choice for our eigen features is the k-nearest neighbours definition, but unlike the spherical neighbourhood definition, it does not correspond to a fixed part of space. A point cloud with strong point density variations can lead to local spherical neighbourhoods with too many or too few points and local k-nearest neighbourhoods covering a small or a huge part of space. However, we solve this problem with our proposed methodology by applying a uniform sampling algorithm on the input point cloud before we extract the local neighbourhoods of the 3D points and using multi-scale representation.

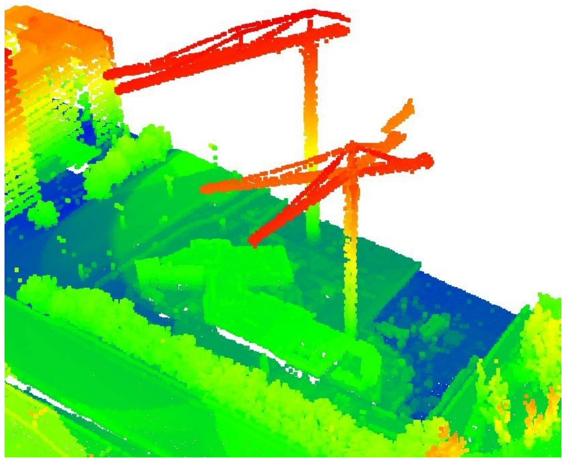
The addition of the uniform sampling algorithm in our 3D point cloud analysis process ensures that our spherical neighbourhoods will have the same number of points in most cases. This means that using the k-nearest neighbours definition with the Euclidean 3D distance as a distance metric will yield the same results with the right number of k neighbours. Because of this, we decided to focus on using the spherical neighbourhood definition for our experiments since it can provide a more consistent geometrical meaning to the extracted features.

On the other hand, the cylindrical neighbourhood definition without a height parameter that sets a distance limit for the points included in the neighbourhoods is not suited for describing the local geometry of 3D points, and due to this, it was not considered. For our height features, however, we used the cylindrical neighbourhood definition since they are more meaningful and discriminating when the variance of the Z coordinates of the points inside the local neighbourhoods is large. To make sure that for every point a high (building roof) and low (ground) Z coordinate value could be found within their neighbourhood large cylindrical neighbourhoods with a radius of 50 meters were used.

A larger radius is better. However, in our case, using a radius larger than 50 meters significantly increased the execution time of our python scripts. We avoided using all the points within each tile to calculate each point's height features to help our classifier better distinguish buildings and other objects from the ground, even in cases where mountains or very tall objects like cranes could be found in the data. We are aware, however, that this method we decided to follow suffers from some limitations. The radius we use can yield false $Z_{normalized}$ values in some cases, as shown in Figure 3.4. We tried to minimize such errors by carefully selecting our training and testing data during the data diversification process.



(a)



(b)

Figure 3.4: (a) False high $Z_{normalized}$ values for plain fields caused by the luck of tall objects within the 50 meters cylindrical neighbourhoods of the points. In this case, a larger radius for the cylindrical neighbourhoods would yield better results. (b) False low $Z_{normalized}$ values for objects close to a crane. In this case, a smaller radius for the cylindrical neighbourhoods would yield better results.

3.3.4 Feature extraction

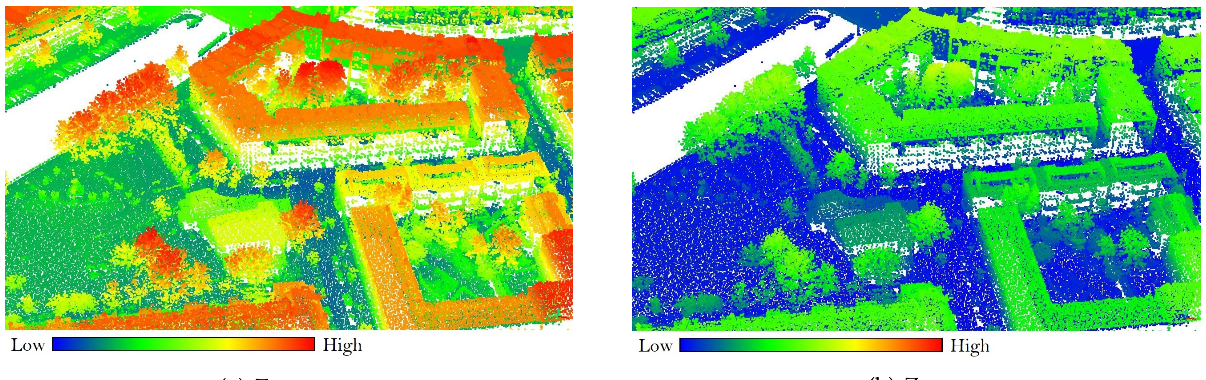
Height features

The heigh features of every 3D point in the input point cloud were calculated using the Z coordinates of all the points within their cylindrical local neighbourhood. The range of values of the Z coordinates differs depending on the location of the input point cloud. To tackle that, the normalized Z coordinates of the points were used instead, which were calculated as follows:

$$Z_{normalized} = \sqrt{\frac{(Z_i - Z_{min})}{(Z_{max} - Z_{min})}} \tag{3.3}$$

In addition, we also take into account the difference between their Z coordinate value and the minimum Z coordinate value found within their local neighbourhood.

$$Z_{below} = Z_i - Z_{min} \tag{3.4}$$



(a) $Z_{normalized}$ (b) Z_{below}

Figure 3.5: Visualization of the two height features.

Eigen-based features

For every 3D point in the input point cloud and the points inside its spherical local neighbourhood, the respective derived normalized eigenvalues λ_i with $i \in \{1, 2, 3\}$ and $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ were exploited to obtain a set of local 3D shape covariance features [Hackel et al., 2016; Weinmann et al., 2015; Thomas et al., 2018]. The eigenvalues and eigenvectors of every local neighborhood were extracted from their covariance matrices defined by:

$$con(N) = \frac{1}{N} \sum_{p \in N} (p - \overline{p})(p - \overline{p})^T \tag{3.5}$$

Where p is the centroid of the neighbourhood N. The feature set calculated for this research using the normalized eigenvalues and eigenvectors encapsulates omnivariance, eigenentropy, anisotropy, planarity, linearity, surface variation, sphericity and verticality according to the equations of Table 3.2 and the number of points inside the spherical neighbourhoods.

Table 3.2: Covariance geometric features calculated from the normalized eigenvalues of the local neighbourhoods of the 3D points [Hackel et al., 2016].

Feature	Equation
Omnivariance	$(\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
Anisotropy	$(\lambda_1 - \lambda_3)/\lambda_1$
Planarity	$(\lambda_2 - \lambda_3)/\lambda_1$
Linearity	$(\lambda_1 - \lambda_2)/\lambda_1$
Surface Variation	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
Sphericity	λ_3/λ_1
Verticality	$1 - \lvert n_z \rvert$

In the following paragraphs, we briefly describe each eigen feature:

- **Omnivariance:** a measure that describes the volumetric point distribution of a neighbourhood. In other words, how inhomogeneously the points of a neighbourhood are spread across a 3D volume [Waldhauser et al., 2014]. High values occur for lower point densities. In our results, we can see that trees have higher values.

- **Anisotropy:** a measure which is higher for local point neighbourhoods with eigenvectors that differ a lot and oriented objects, which enables anisotropic operations to discriminate between orientated and non-orientated objects [Elberink and Maas, 2000]. When visually inspecting our results, we see that trees have lower values of anisotropy.

- **Planarity:** a feature used to describe the smoothness of a surface [Waldhauser et al., 2014] and can discriminate planar structures. Planar objects such as the ground and building roofs have high planarity values, while trees have low planarity values.

- **Linearity:** a feature used to detect line structures. Higher values occur for sets of points that can be modelled by a 3D line [Waldhauser et al., 2014] like powerlines and the edges of buildings.

- **Surface variation:** or curvature, "is the amount by which a curve deviates from being a straight line, or a surface deviates from being a plane" as stated by Wikipedia [2021c]. Our results show that higher values occur for tree points and the edges of buildings.

- **Sphericity:** a feature used to measure how closely a structure resembles that of a perfect sphere. Vegetation has higher sphericity values.

- **Verticality:** a feature used to detect vertical structures. Building facades have significantly higher verticality values than other objects and the ground.

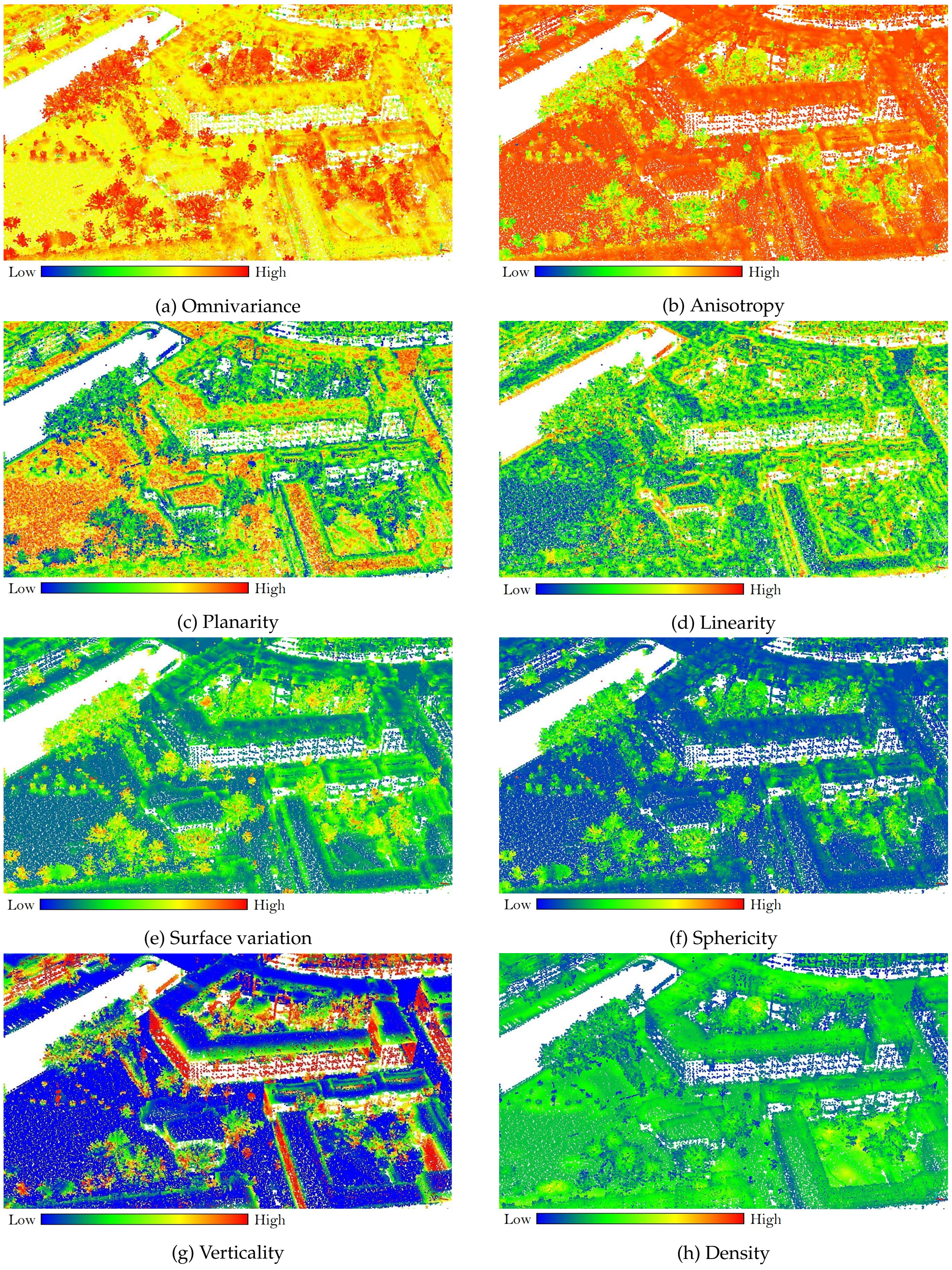


Figure 3.6: Visualization of the seven eigen features and point density values of the local spherical neighbourhoods of the points.

3.3.5 Feature selection

Considering that some of the handcrafted features that we extracted from the local neighbourhoods of the points might be less significant than others, and also due to the memory bottlenecks that we encountered during our initial experiments, feature extraction was a necessary and important step in our final processing workflow. Overall, we used four different methods to analyse which subset of the handcrafted features is the best one:

1. Visual inspection of the handcrafted features to check the variance and the range of the values and if some objects are highlighted.
2. Training and testing Random Forest models with a different number of handcrafted features.
3. Training and testing Random Forest models with different combinations of handcrafted features.
4. Calculating the feature importances based on the mean decrease in impurity. More specifically, compute them as the mean and standard deviation of accumulation of the impurity decrease within each tree.

3.3.6 Assigning class labels to the testing data

Our classification model predicts class labels only for the candidate centre points of the voxels in the voxel grid used for our uniform sampling algorithm. In other words, it only makes predictions for the subsampled point cloud tiles and not for the whole testing dataset. As far as the candidate centre points of the voxels are concerned, it was decided to keep for each voxel the coordinates and the class label of the closest point from the input point cloud to their centre. Alternatively, the barycenters of all the points inside each voxel could have been used. However, this method was not used for efficiency reasons since new points, and class labels must be calculated.

After our model makes predictions for the points of the subsampled point cloud tiles, the nearest neighbour approach is followed to assign a class label to all the points of the input point cloud testing tiles. More specifically, a k-dimensional tree is built using the points of the subsampled point cloud tiles, and then the nearest neighbour for each point from the input point cloud tiles is found using the tree. The points are then assigned to the same class label as the one predicted for their nearest neighbour.

3.4 Evaluation metrics

Following the examples of previous work [Chaton et al., 2020; Hu et al., 2020; Varney et al., 2020], the evaluation metrics that will be used for benchmarking the deep learning and machine learning algorithms are the mean IoU, per class IoU, Class Consistency Index (CCI), OA, F_1 score and the confusion matrix. IoU is simply a ratio calculated by dividing the intersection with the union of the predicted and ground-truth values. In total, there will be three classes for which their IoU will be calculated as follows:

$$IoU_i = \frac{c_{ii}}{c_{ii} + \sum_{j \neq i} c_{ij} + \sum_{k \neq i} c_{ki}} \tag{3.6}$$

The term c_{ii} is the number of points that have the same predicted and ground-truth labels (True Positives), while the terms c_{ij} and c_{ki} stand for the number of points with wrong predicted labels (False Positives and False Negatives) for a certain class. The mean IoU is simply the mean across all three classes, and it is defined as:

$$\overline{IoU} = \frac{\sum_{i=1}^{N} IoU_i}{N} \tag{3.7}$$

OA is the ratio of correct predictions to total predictions made. It is calculated for most benchmarking researches, however, it can be a misleading and unreliable metric for imbalanced datasets, such as a point cloud with a significant disparity between classes. The OA suffers from this problem which can hide details about the performance of a model. For example, if a high OA score is achieved for an imbalanced dataset, we are not sure if that is because all classes are being predicted equally well or whether some of the classes are being neglected by the model. A high OA score can be achieved if we always predict the most common class value. The OA can be computed as:

$$OA = \frac{\sum_{i=1}^{N} c_{ii}}{\sum_{j=1}^{N} \sum_{k=1}^{N} c_{jk}} \tag{3.8}$$

High OA does not necessarily mean a good classification but a high mean IoU and CCI indicate that an algorithm is robust and has high performance that is also uniform across each class. The CCI can be calculated with the following equation, where σ^2 is the variance of the per-class IoU values:

$$CCI = 1 - \frac{\sigma^2}{|\overline{IoU}|} \tag{3.9}$$

The confusion matrix is a table used to describe the performance of a classification model. The significance of this table is that it shows in which class some points are incorrectly classified. This information helps to understand which features should be included, excluded or recalculated to improve the performance of a classification model.

Table 3.3: Confusion matrix.

	Predicted: Positives	**Predicted: Negatives**
Actual: Positives	True Positives (TP)	False Negatives (FN)
Actual: Negatives	False Positives (FP)	True Negatives (TN)

Precision is the fraction of relevant instances among the retrieved instances and it can be seen as a measure of quality. High precision means that an algorithm returns more relevant results than irrelevant ones [Wikipedia, 2021h]. Based on the confusion matrix, precision is defined as:

$$precision = \frac{TP}{TP + FP} \tag{3.10}$$

Recall is the fraction of relevant instances retrieved and can be seen as a measure of quantity. High recall means that an algorithm returns most of the relevant results [Wikipedia, 2021h]. Based on the confusion matrix, recall is defined as:

$$recall = \frac{TP}{TP + FN} \tag{3.11}$$

Finally, the F_1 score is a metric that measures the accuracy of a model and conveys the balance between the precision and the recall [Wikipedia, 2021e]. The formula of the F_1 score is the harmonic mean of the precision and recall, and it is calculated as:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3.12}$$

4 Implementation details / Experiments

In chapter 4, the implementation details of our experiments and methodology are provided. We explain what tools and datasets were used, how our training and testing data were created and what parameter values were used in our experiments. We also discuss what Python packages and libraries were used to implement our 3D point cloud analysis tasks and what limitations we faced during our experiments.

4.1 Tools

To meet the objectives of the graduation project, several tools were needed to process and visualize the LiDAR point cloud datasets. Table 4.1 lists all the tools and software that we used and for what purpose.

Table 4.1: Tools and software used to meet the objectives of the graduation project.

Tool/software	**Usage description**
QGIS	to make shapefiles that will assist in the creation of the training and testing datasets
FME	to create the training and testing datasets
Python coding	to process the training and testing datasets and to train and test the Random Forest models
Cloud Compare	to visually inspect the extracted features and the results of the classification
LAStools	to visually inspect the training and testing datasets
TU Delft's HPC cluster	to run python scripts that require large computing resources and a lot of processing time

4.1.1 TU Delft's HPC cluster

The HPC cluster of TU Delft is a collection of large computing resources, namely processor (CPUs), Graphics cards (GPUs), memory and storage that are shared with others to use. In the cluster, jobs can be scheduled, and a lot of computational work can be done. Users are able to utilize multiple computers together, use a lot of threads, run long computations and compute on big datasets like point clouds. Training and testing machine learning models requires a lot of computational power, memory and time. Due to this, it was essential to make use of the HPC cluster as standard computers do not usually have the required specifications.

When a job is submitted in the HPC cluster, the scheduler allocates resources for it based on the partition and Quality of Service (QoS) that the user specified. These two variables also determine the CPU, GPU and memory limits for that specific job. Details about the partitions, Quality of Services and resources limits are provided in the Table 4.2.

Table 4.2: Characteristics and resources limits of the partitions and QoS in the HPC cluster.

Partition	QoS	Priority	Maximum run time	CPU limits		Memory limits	
				Per QoS	Per user	Per QoS	Per user
general	interactive	high	1 hour	-	2	-	16 GB
	short	normal	4 hours	2884	1696	17689 GB	10405 GB
	long	low	7 days	2544	678	15608 GB	4162 GB
	infinite	none	infinite	32	-	250 GB	-
stud-ewi	stud-ewi	low	1 hour	256	2	1625 GB	12 GB

Running python scripts with Slurm

The HPC cluster uses the Slurm[1] scheduler to efficiently manage workloads. Slurm is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters that requires no kernel modifications for its operation and is relatively self-contained. To run python scripts that utilize the computing resources of the HPC cluster, batch scripts had to be created and submitted to Slurm that specify which python scripts should be executed and what resources should be allocated to them. The essential options and commands that must be specified in a batch script to run a script are the following ones:

- **#!/bin/sh** she-bang line that must be the first line in the batch script tells the system that the batch script contains a set of commands to be fed to the command interpreter indicated, in this case, the Bourne shell or a compatible shell, with path /bin/sh.
- **#SBATCH –partition=general** option sets the partition of the job, in this case general.
- **#SBATCH –qos=short** option sets the Quality of Service of the job, in this case short (maximum run time: 4 hours).
- **#SBATCH –time=00:01:00** option determines for how long the job will run and use the allocated resources, in this case, 1 minute.
- **#SBATCH –ntasks=1** option determines the number of parallel tasks for the job, in this case, 1.
- **#SBATCH –cpus-per-task=1** option determines the number of allocated CPUs per task, in this case, 1.
- **#SBATCH –mem=1024** option determines how much RAM in megabytes (MB) is allocated for the job, in this case, 1 gigabytes (GB).
- **#SBATCH –mail-type=END** option sends a mail notification when the job is done.
- **srun python scriptname.py** command specifies which python script will be executed.

4.2 Datasets

The focus of this graduation project is studying how accurately a simple machine learning algorithm like the Random Forest Classifier will perform compared to more complex deep learning models if we train it with an accurate LiDAR dataset like the AHN3. Subsequently, an attempt is made to create an efficient and accurate point cloud classifier used by automatic building reconstruction methods. Lately, large LiDAR datasets of different cities worldwide can be found on the internet as open data, which can serve as training and testing data for machine learning and deep learning models. This thesis made an effort to exploit some of those large and accurate datasets and train, test, and compare machine learning models. Specifically, we focused on using the AHN3 dataset[2], the AHN4 dataset and the DALES dataset[3].

[1]https://slurm.schedmd.com/overview.html
[2]https://downloads.pdok.nl/ahn3-downloadpage/
[3]https://docs.google.com/forms/d/e/1FAIpQLSe3IaTxCS7wKH01SHn_o7U86ToIw9K26vc0bkwiELn6wwh8gg/viewform

These three datasets have their own characteristics and cover different environments that we discuss in the following sections.

4.2.1 AHN3 and AHN4 datasets

The national Dutch AHN3 LiDAR dataset is a freely available point cloud dataset with detailed and precise height data for the whole Netherlands. The measured points have a height accuracy with a systematic and stochastic error of five centimetres and a planimetric accuracy with eight centimetres systematic error and five centimetres stochastic error. The dataset is disseminated in the LAZ format and uses the LAS classification codes. A custom code is also used for an extra class that includes only specific built structures such as bridges and viaducts. Besides urban and rural areas, wooded areas are classified as well due to the multiple reflections of the LiDAR systems used for the collection of the data that provide enough ground-level points. The point density of the AHN3 dataset is on average between 6 and 10 points per square meter [AHN, 2020]. As far as the AHN4 dataset is concerned, it has similar characteristics with AHN3, but it is still a work in progress. It uses the same classification codes and is divided into tiles of the same size as the tiles of AHN3 dataset. Its point density, however, is a lot larger than the AHN3 dataset.

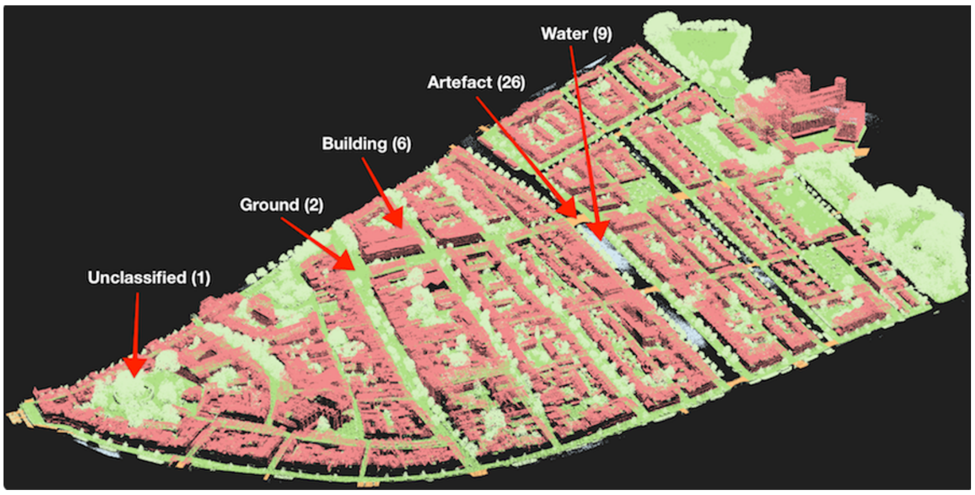
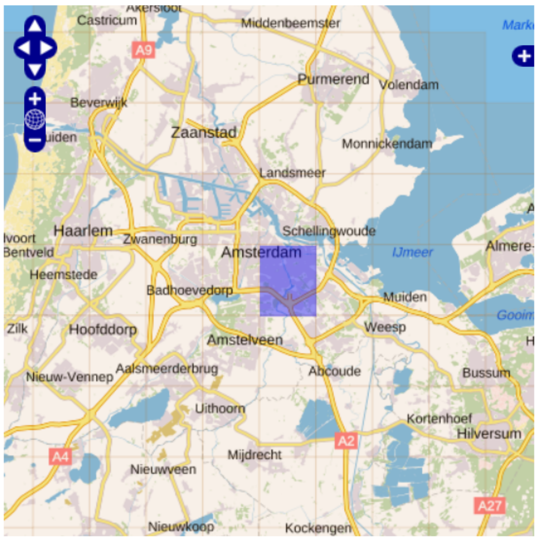


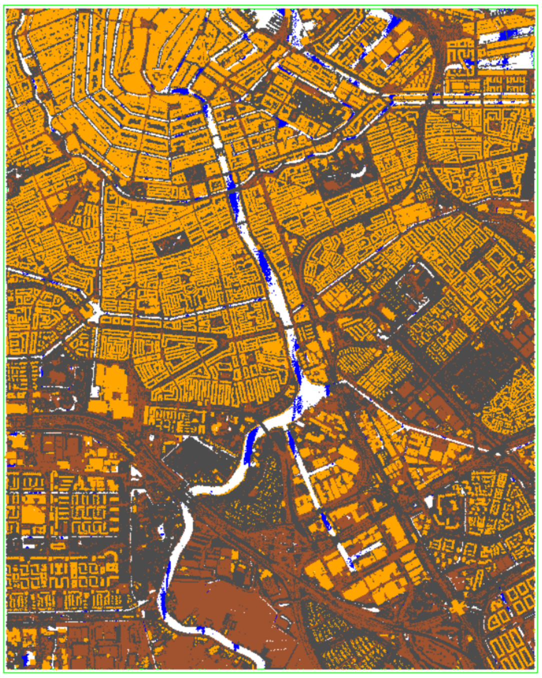
Figure 4.1: Classification codes used in the AHN3 dataset [Ledoux et al., 2020].

The AHN3 and AHN4 datasets were used to train and test Random Forest models. Because the whole size of these two datasets is enormous, it was very difficult for us to train and test Random Forest models with them. Thus, only two tiles were used from these two datasets. Tile 25GN1 was used from both datasets to create training and testing data covering the same environment but with different characteristics. Also, these two tiles were chosen because our goal was to create a classifier to assist automatic building reconstruction methods. They cover a dense urban area (city centre of Amsterdam) with a large percentage of building points and many different types of buildings (apartment buildings, churches, small houses etc.) which can increase the diversity of our features and subsequently improve the performance of the classifier.

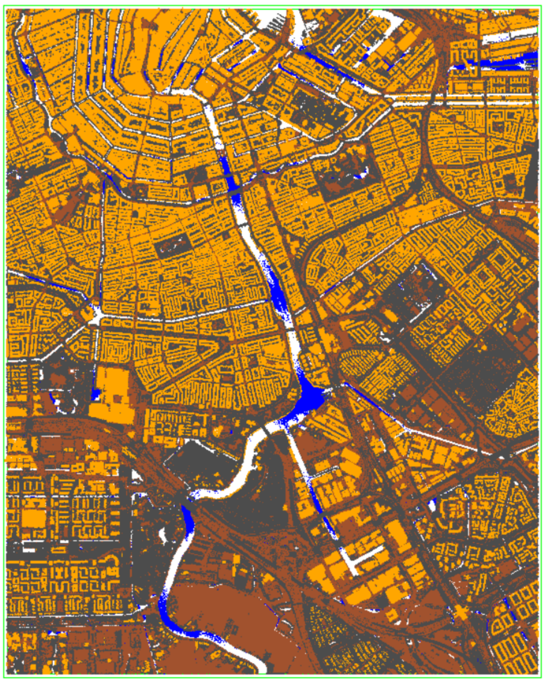


(a)

Figure 4.2: Location of the AHN3 and AHN4 tiles used for training and testing the Random Forest models.



(a)



(b)

Figure 4.3: Visualization of the two tiles used for training and testing Random Forest models. (a) AHN3 and (b) AHN4 tile. The two tiles cover the same area. It is visible from the water points that the AHN4 tile has a lot more points.

4.2.2 DALES dataset

The DALES dataset is a dense aerial LiDAR point cloud of the city of Surrey in British Columbia, Canada, distributed in the LAS format. It consists of 40 tiles, 0.5 squared kilometres each, that are split into 29 training files and 11 testing files. The projection of the data is UTM zone 10N, the horizontal datum is NAD83, and the vertical datum is CGVD28, using the metro Vancouver Geoid. Easting, northing, Z, and object category is provided for every point in their original UTM Zone 10N coordinates. In total, there are 8 categories in the dataset: ground (1), vegetation (2), cars (3), trucks (4), power lines (5), fences (6), poles (7) and buildings (8). The mean error for hard surface vertical accuracy was determined to be $\pm$ 8.5 cm at 95% confidence. When all returns are included, and after an initial noise filtering, the point cloud's measured density is on average 50 points per meter [Varney et al., 2020]. The DALES dataset was used to train models, compare their performance with previous works, and test the models trained with AHN3 and AHN4 datasets to examine their performance with different datasets.

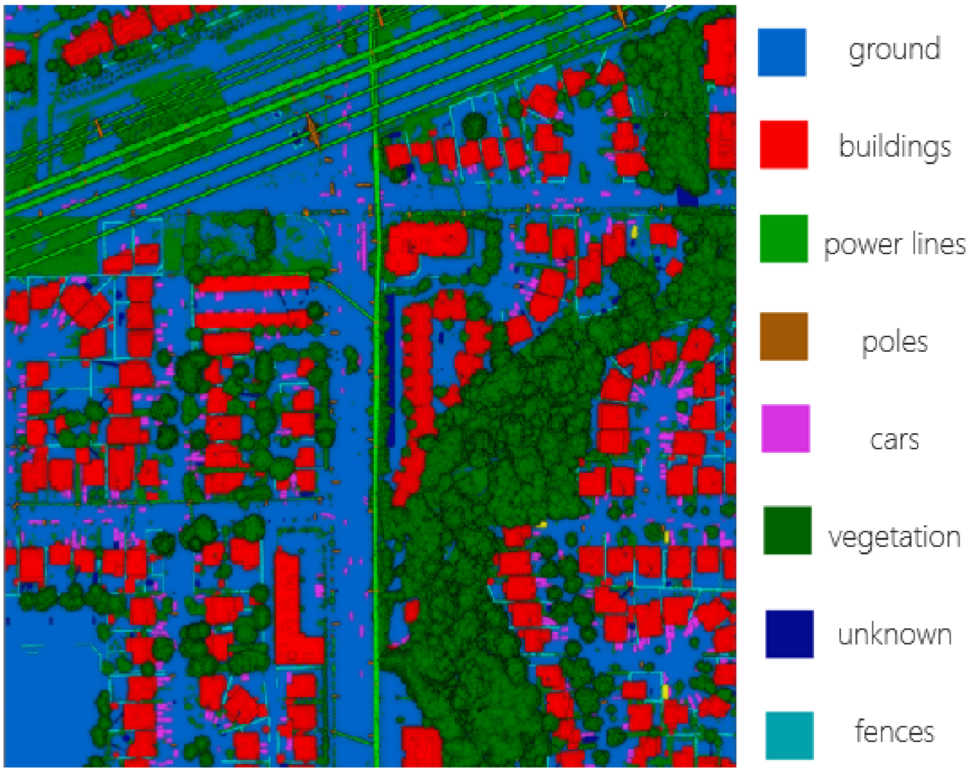


Figure 4.4: A training tile of the DALES dataset and its 8 classes [Varney et al., 2020].

4.3 AHN3 and AHN4 training and testing data

Using Python coding, the two AHN3 and AHN4 tiles were split into 120 tiles of equal size, as shown in Figure 4.5a. Each tile is a square with a perimeter of 2 kilometres and an area of 0.25 squared kilometres (500X500 meters). An analysis was made for these tiles to identify which ones are the best to be used as training and testing data for our experiments and our final model. For each tile, we computed its feature diversity score, the number of classes and the percentages of points that belong to each class. The best 40 tiles (Figure 4.5b) out of the 120 tiles have been chosen based on these three characteristics. Moreover, we visually inspected them to discard potentially unwanted tiles (Figure 4.6). This analysis was made for all the 240 tiles derived from the two AHN3 and AHN4 tiles, but our selection of the best tiles was focused on the statistics of the AHN3 tiles. For many tiles, their statistics were similar for both datasets. However, there are a few differences since the AHN4 dataset was collected more recently and has a higher point density. To better compare the results of our experiments, we decided to use the same tiles for both datasets (see Section 5.2).

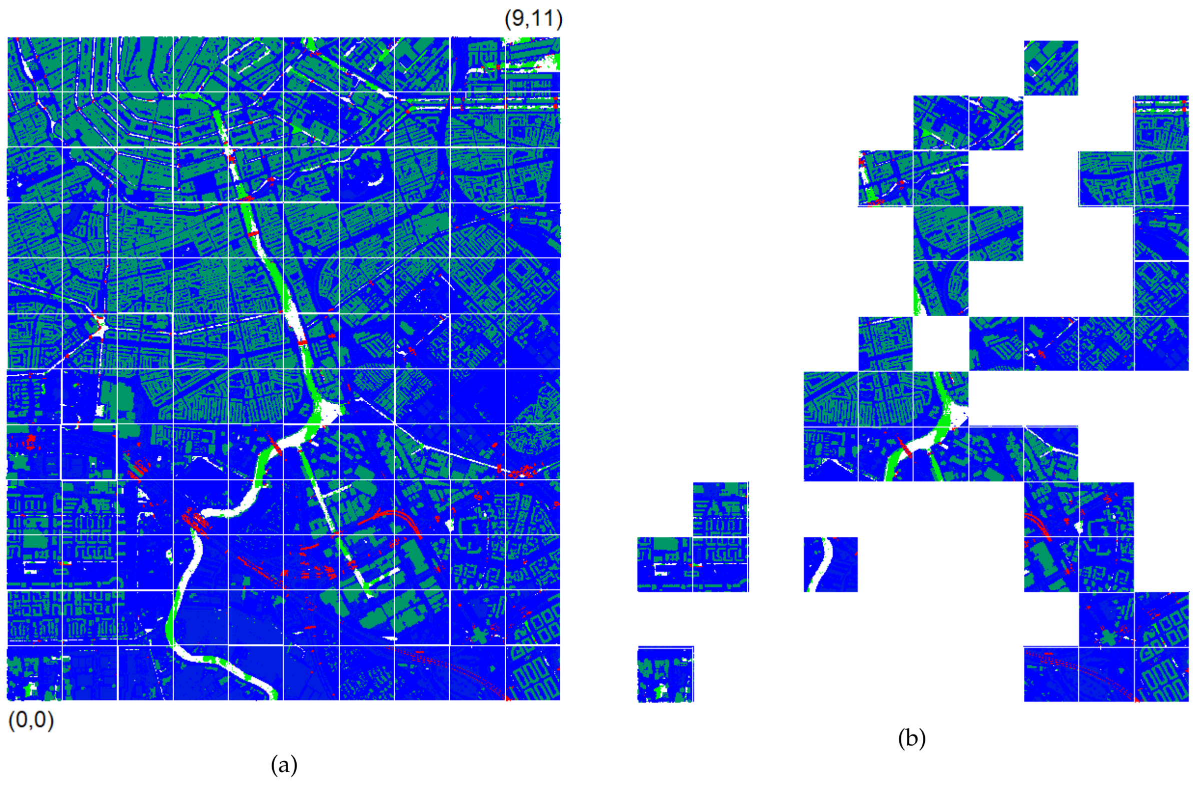


Figure 4.5: (a) The 120 tiles clipped from the 25GN1 tile of the AHN3 and AHN4 datasets. Tiles grid: bottom left (0,0), top right (9,11). (b) The best 40 tiles.

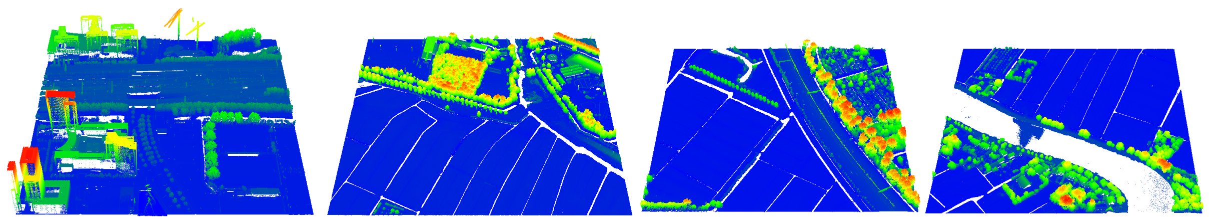


Figure 4.6: Tiles with high feature diversity that were discarded because they represent environments (very flat areas) or include objects (cranes) that we did not want them to be used during the training process of our classification model.

On average, our best 40 tiles from the AHN3 dataset have a point density of 18 $\pm$ 12 points per squared meter, while the corresponding tiles from the AHN4 dataset have a point density of 42 $\pm$ 25 points per squared meter.

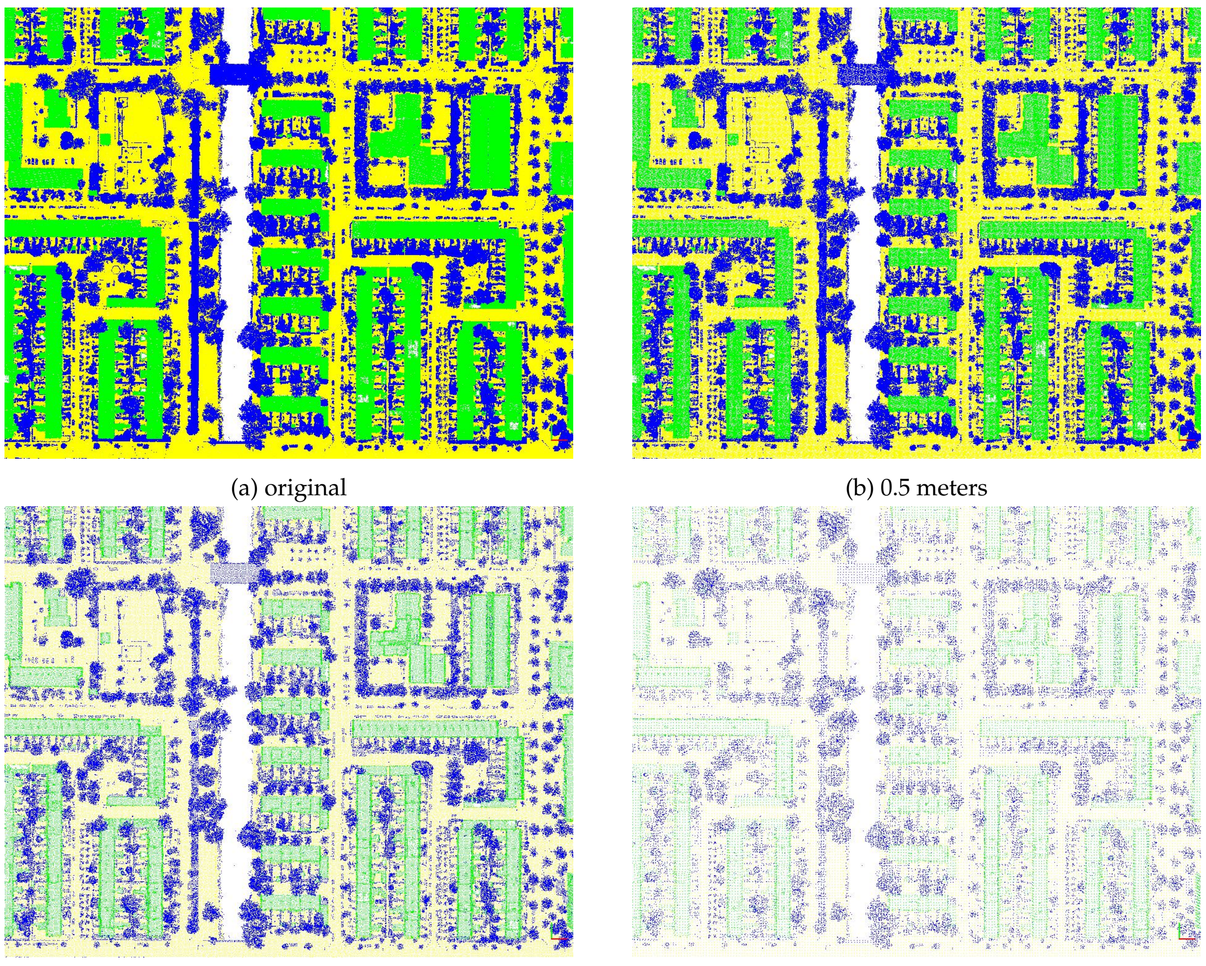
4.4 3D point cloud analysis and classification

4.4.1 Programming specifics

For the purposes of this project, the methodology was implemented in Python. Data analysis tasks, model training and testing, and evaluation of the results were done using methods from the NumPy [Harris et al., 2020], SciPy [Virtanen et al., 2020], Laspy, pandas [Wes McKinney, 2010] and scikit-learn [Pedregosa et al., 2011] Python libraries. The classification metrics of the main experiments were visualised using methods from the matplotlib Python library.

4.4.2 Uniform sampling experiments

To decide which voxel size is the best for our uniform sampling algorithm, models that use different voxel sizes were trained and tested to check the effect that the voxel size parameter can have on the performance of our classifier. In total, four different models were trained, tested and evaluated. The first model was trained and tested without applying a uniform sampling algorithm on the input data, while for the rest, different voxel sizes were used for their uniform sampling, namely 0.5, 1, 1.5 and 2 meters. Testing voxel sizes larger than 2 meters was thought to be unnecessary and impractical because they reduce the number of points a lot. It also means that points with a distance of up to 2 meters between them will be assigned to the same class since we later use the predicted class labels of the centre points of the voxels to manually assign predicted class labels to all the points based on the nearest neighbour approach. The results of these experiments can be found in Section 5.4.



(a) original

(b) 0.5 meters

(c) 1 meter

(d) 2 meters

Figure 4.7: Example results of the uniform point cloud sampling algorithm. (a) Original point cloud, subsampled point cloud with (b) 0.5, (c) 1 and (d) 2 meters voxel size.

4.4.3 Feature selection experiments

The first strategy we followed to identify which features are the most informative and discriminating ones was a visual inspection to check the variance and the range of values of the extracted features and see if some objects of interest are highlighted. To do this, the values of the extracted features were stored in CSV format and visualized through the Cloud Compare software.

The second strategy we followed was to test the performance of Random forest models trained with a different number of features. First, we trained the model using only the Z features and the eigenvalues of the points. Then we added the normal vectors of the points, and later we started adding one by one the eigen-based features to the feature vectors of the points. We observed each time how the performance of the classification model had changed.

Our third strategy was to test the performance of Random forest models trained with different combinations of features. As we already mentioned in our Chapter 3: Methodology, we focused on experimenting with the spherical neighbourhood definition. Specifically, we conducted experiments to test the effect that single-scale and multi-scale spherical neighbourhoods of various radii can have on the performance of our classifier. We trained multiple models each time using features derived from 1, 2 or 3 different spherical neighbourhoods at the same time. Tests using features derived from more than 3 neighbourhoods or with very large radii have not been made because, in these cases, the computational time increases considerably, making our model inefficient. These tests were mainly made to figure out the optimal multi-scale representations that should be used to train our final model.

The fourth and final strategy we used was to compute the importances of the features of our Random Forest model as the mean and standard deviation of the impurity within each tree. To compute these importances, a method from the scikit-learn [Pedregosa et al., 2011] Python library was used. The results of these experiments can be found in Section 5.6.

4.4.4 Training and testing data experiments

To test the effect that the size of training data can have on the performance of our classification model, we trained and tested the model with different amounts of training data. Specifically, since our AHN3 training data were split into tiles of equal size, we trained seven different models using 1, 2, 3, 5, 10, 15 and 20 tiles during their training process and tested them with the same testing data. The testing data were the tiles ranked 20th to 30th according to their feature diversity score. A comparison was made between the evaluation classification metrics of their results (see Section 5.5).

To test the effect that different training and testing datasets can have on our classification model's performance, we trained and tested models using our three different datasets, namely AHN3, AHN4 and DALES. Specifically, three different Random Forest models were trained using the same model hyperparameters, multi-scale spherical neighbourhoods, handcrafted features and parameters for our 3D point cloud analysis algorithms. Each model used one of the three datasets as training data. Their performance was evaluated on all three datasets, and a comparison was made between their classification evaluation metrics (see Section 5.7).

4.4.5 Hyperparameter tuning

Hyperparameter optimization is an important and required step to get the most out of a machine learning model. To improve the classification accuracy of our Random Forest model, we tried to identify the best set of hyperparameters to configure our model. One of the most common methods used to optimize the hyperparameters of a machine learning model is grid search, which defines a search space as a grid of hyperparameter values and evaluates every position in the grid. In our case, a small grid search was implemented to test the effect of seven different hyperparameters on the performance of our Random Forest model. The results of these experiments can be found in Section 5.3. The eight hyperparameters that we tested are:

1. **bootstrap:** that refers to whether bootstrap samples are used when building trees. If not, the whole dataset is used to build each tree.
2. **max_features :** the maximum number of features to consider when looking for the best split.
3. **min_samples_leaf:** the minimum number of samples, in our case points, from the input data required to be at a leaf node.

4. **min_samples_split:** the minimum number of samples, in our case points, from the input data that must be placed in an internal node before that node is split.
5. **max_depth:** the maximum depth of the tree.
6. **criterion:** that refers to the function used to measure the quality of a split. In our case, Gini impurity or entropy.
7. **n_estimators:** that corresponds to the number of trees in the forest.
8. **oob_score:** that determines whether out-of-bag samples are used to estimate the generalization score and is only available if $bootstrap = True$.

K-Fold Cross-Validation

For every combination of hyperparameters that was tested, a K-Fold Cross-Validation resampling procedure, with $\kappa = 3$, was used to evaluate the Random Forest models. K-Fold Cross-Validation has only one parameter, the κ parameter that stands for a number of equal size groups that the data will be randomly partitioned into. From those κ groups of data, one is used to validate the model while the others are used as training data. The process is then repeated κ times to evaluate all possible combinations of training and validation data, and their results are averaged to produce a single estimation [Wikipedia, 2021b].

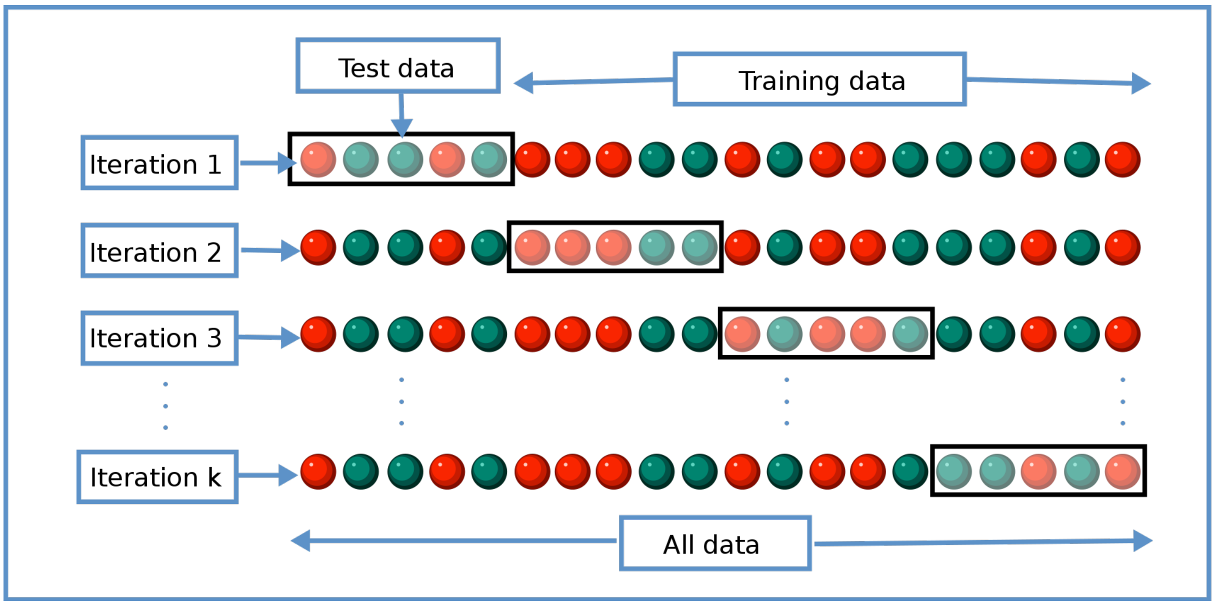


Figure 4.8: K-Fold Cross-Validation [Wikipedia, 2021b].

5 Results and discussion

Chapter 5 provides the results of the experiments of the study. More specifically, in the following sections, we analyse and present the obtained results from the experiments we conducted to understand how different local neighbourhood definitions, features, number of features, hyperparameters, data sizes, and data environments can influence the performance of our classifier. We discuss the limitation we faced and which approaches and parameters we believe are the best. Finally, comparisons are made between our results and the results of previous works, and an application in presented.

5.1 Overview

Overall, the evaluation metrics of our classification results for some of our experiments are lower than the results of previous works. This is mainly because of the high computational cost and time required to analyse the point cloud datasets and prepare them for training and testing our Random Forest models. To tackle this problem, we used large voxel sizes for our uniform sampling algorithm and small training data for some of our experiments to speed up the process. This led to having models that underperform. Consequently, we focused more on comparing the results and presenting the difference between their evaluation metrics. However, in some cases, we also provide the actual scores of their evaluation metrics. In each experiment, we tried to use the best possible parameters that we could. Our final model was trained and tested with the optimal parameters, hyperparameters and training and testing data that provide a good trade-off between accuracy and efficiency. For the rest of our experiments, however, these choices were made based on findings of previous works since we were still trying to figure out the optimal ones through them. With the results of our experiments, we can provide answers to all of the research questions addressed by this graduation project. It is also worth mentioning that there were many failed attempts while trying to implement these experiments that really slowed down the progress of the graduation project, especially while trying to analyse the training and testing data with optimal parameters.

5.2 Feature diversity of the training and testing data

Before we began our experiments with the Random Forest models, we had to select the right data to be used as training and testing data from the AHN3 and AHN4 datasets. As we discussed in the previous chapters, to do that, we split the 25GN1 tile of the AHN3 and AHN4 datasets into 120 equally sized tiles and followed a data diversification strategy to compute important information about them that helped us decide which ones are the best. Table 5.1 provides a summary of the feature diversity scores F_m, the number of classes, the percentages of points that belong to each class, and the number of points for the first best 40 tiles out of the 120 tiles that were clipped from the 25GN1 tile of the AHN3 while table 5.2 for the equivalent tiles from the 25GN1 tile of the AHN4 dataset. The features used to calculate the feature diversity scores of the tiles were derived from 3 different spherical neighbourhoods with radii 2, 3 and 4 meters. In total, the variances of 24 features have been used for each tile. A uniform sampling algorithm with a voxel size of 1 meter has been applied to the tiles before extracting their features.

Table 5.1: Statistics of the first best 40 tiles clipped from the AHN3 dataset.

no.	**tile**	F_m	**no. of classes**	**other (%)**	**building (%)**	**ground (%)**	**no. of points**
1	(7,3)	0.0433	3	53.2	12.8	34.1	4,563,251
2	(5,5)	0.043	3	39.5	33.7	26.8	3,017,251
3	(9,0)	0.0428	3	48.1	17.8	34.1	4,114,615
4	(7,2)	0.0428	3	46.5	19.4	34.2	4,519,839
5	(5,4)	0.0427	3	62	10.3	27.7	3,596,646
6	(3,2)	0.0425	3	70.9	0.5	28.6	4,681,849
7	(9,1)	0.0421	3	55.7	11.9	32.4	4,513,323
8	(8,0)	0.042	3	58.7	0.5	40.8	4,452,822
9	(9,9)	0.0419	3	59.3	20.7	20	4,856,019
10	(8,3)	0.0418	3	59	11.9	29.1	3,828,687
11	(1,2)	0.0418	3	60.3	12.7	27.1	5,332,243
12	(7,11)	0.0416	3	31.7	39.6	28.7	3,447,268
13	(4,4)	0.0414	3	70.6	3.7	25.7	4,375,794
14	(0,0)	0.0413	3	54.5	15.5	29.9	4,418,212
15	(6,6)	0.0413	3	51.4	18.1	30.4	4,204,809
16	(9,10)	0.0411	3	39.7	35.3	25	3,990,567
17	(7,4)	0.0411	3	43	25.2	31.8	3,734,563
18	(0,2)	0.0411	3	55.7	17.4	26.9	4,566,563
19	(6,10)	0.041	3	57.1	22.3	20.6	4,415,975
20	(9,8)	0.0409	3	46.1	16	37.9	3,860,100
21	(8,2)	0.0409	3	67.9	8.6	23.5	4,748,568
22	(4,6)	0.0408	3	43.4	31.8	24.8	3,808,372
23	(7,0)	0.0406	3	59.7	1.7	38.6	4,333,094
24	(6,8)	0.0405	3	47.2	31.6	21.2	4,720,536
25	(9,6)	0.0405	3	67.5	7.4	25.2	4,837,172
26	(9,7)	0.0405	3	57.4	18.4	24.2	4,421,793
27	(3,5)	0.0404	3	39.7	35.3	25	5,261,248
28	(6,4)	0.0404	3	40.6	19.5	39.9	3,634,286
29	(5,8)	0.0404	3	35.3	41.6	23.1	4,011,017
30	(8,1)	0.0404	3	62.2	6.6	31.2	3,975,009
31	(5,9)	0.0403	3	47.8	30.3	21.9	4,105,674
32	(5,7)	0.0403	3	44.5	29.7	25.8	3,592,834
33	(7,6)	0.0403	3	71.2	5.4	23.4	4,833,607
34	(1,3)	0.0403	3	37.4	29	33.6	4,717,004
35	(8,9)	0.0402	3	38.8	34	27.2	3,993,220
36	(3,4)	0.0402	3	68.7	9.5	21.8	6,029,536
37	(8,6)	0.0402	3	55.5	18.2	26.3	4,522,453
38	(4,9)	0.0401	3	39.2	34	26.8	3,140,716
39	(4,5)	0.0401	3	43.6	33.2	23.2	4,090,139
40	(5,10)	0.04	3	55.4	24.1	20.4	4,090,587

Table 5.2: Statistics of the corresponding tiles clipped from the AHN4 dataset.

no.	**tile**	F_m	**no. of classes**	**other (%)**	**building (%)**	**ground (%)**	**no. of points**
1	(7,3)	0.0404	3	50.1	12.7	37.2	9,111,285
2	(5,5)	0.0413	3	35.7	30.8	33.5	7,805,095
3	(9,0)	0.0416	3	46.6	20	33.3	10,592,425
4	(7,2)	0.0385	3	44.5	18.5	37	9,981,479
5	(5,4)	0.0433	3	57.4	8.1	34.4	8,981,406
6	(3,2)	0.0405	3	68.2	0.4	31.4	10,298,226
7	(9,1)	0.0396	3	56.1	9.9	34	10,638,417
8	(8,0)	0.0393	3	56.8	0.5	42.7	8,990,897
9	(9,9)	0.0401	3	61.2	17.4	21.4	10,541,974
10	(8,3)	0.0406	3	54.5	15.7	29.8	9,755,432
11	(1,2)	0.0411	3	70.4	9.6	20	12,783,974
12	(7,11)	0.0376	3	34.2	36.8	29	7,731,449
13	(4,4)	0.0407	3	72.6	3.9	23.4	10,990,093
14	(0,0)	0.0406	3	61.8	10.9	27.2	9,351,733
15	(6,6)	0.0407	3	43.7	22.9	33.4	10,560,537
16	(9,10)	0.0401	3	37.3	40.6	22.1	10,489,806
17	(7,4)	0.0403	3	42.5	22.1	35.4	9,655,165
18	(0,2)	0.0414	3	68.3	13.5	18.3	12,015,153
19	(6,10)	0.0400	3	53.6	25.6	20.8	10,891,964
20	(9,8)	0.0386	3	44.5	20.8	34.7	10,971,698
21	(8,2)	0.0394	3	65.9	11.1	23.1	10,980,502
22	(4,6)	0.0405	3	41.9	35.1	23.1	10,801,701
23	(7,0)	0.0393	3	58.1	1.3	40.6	9,164,434
24	(6,8)	0.0401	3	42.6	38.6	18.8	12,174,899
25	(9,6)	0.0399	3	64	7.2	28.9	12,275,039
26	(9,7)	0.0393	3	59.6	16.9	23.6	9,885,090
27	(3,5)	0.0380	3	44.1	32.6	23.4	9,001,933
28	(6,4)	0.0384	3	28.2	29.9	41.8	9,747,800
29	(5,8)	0.0405	3	34.5	45.7	19.8	11,088,512
30	(8,1)	0.0383	3	60.1	6.7	33.3	9,082,421
31	(5,9)	0.0392	3	46.3	29.1	24.6	8,734,156
32	(5,7)	0.0395	3	45.9	25.2	28.9	9,435,695
33	(7,6)	0.0394	3	70.8	5.4	23.8	11,359,030
34	(1,3)	0.0406	3	43.4	25	31.6	9,574,920
35	(8,9)	0.0393	3	38	33.8	28.2	8,872,447
36	(3,4)	0.0392	3	68.3	9.4	22.4	12,349,350
37	(8,6)	0.0397	3	49.1	21.4	29.5	10,638,670
38	(4,9)	0.0385	3	34.6	34.8	30.6	7,382,931
39	(4,5)	0.0391	3	45.6	31.8	22.6	9,715,158
40	(5,10)	0.0393	3	53.3	26	20.7	11,331,886

5.3 Hyperparameters

For our experiments with the hyperparameters of our Random Forest classifier, the first 20 best tiles from the AHN3 dataset, as shown in table 5.1, were used as training and testing data. A uniform sampling algorithm with a voxel size of 2 meters has been applied on them, and features have been extracted from 3 different spherical neighbourhoods with radii 3, 4 and 5 meters. Since we had to train and test many models for these experiments, a larger voxel size was used for our uniform sampling algorithm from the data diversification experiments. As a result, the radii of the spherical neighbourhoods are larger as well. We used the default hyperparameters provided by the scikit-learn [Pedregosa et al., 2011] Python library, and each time we changed the values for one of them to test the effect they have on the accuracy of our model. Table 5.3 provides the default values for the eight hyperparameters that we tested. The evaluation metric that we used to compare all classification results is the F_1 score. The results of our experiments indicate that the default hyperparameters are, in most cases, the optimal ones.

Table 5.3: Default values of the hyperparameters of the Random Forest Classifier.

Hyperparameter	**Value**
n_estimators	100
criterion	Gini
max_depth	None
min_samples_split	2
min_samples_leaf	1
max_features	sqrt
bootstrap	True
oob_score	False

5.3.1 Number of trees

The hyperparameter *n_estimators* refers to the number of trees in the forest. Usually, the higher the number of trees, the better the algorithm fits and learns from the data. A very small number may lead to overfitting, but adding many trees can slow down the training process considerably and does not significantly improve the performance. Thus, a balance must be found between accuracy and efficiency. For this parameter, the values we tested are 1, 32, 64, 100, 128, 200, 250, 300 and 500 trees. Figure 5.1 illustrates the F_1 score values for the models trained and tested with different numbers of trees.

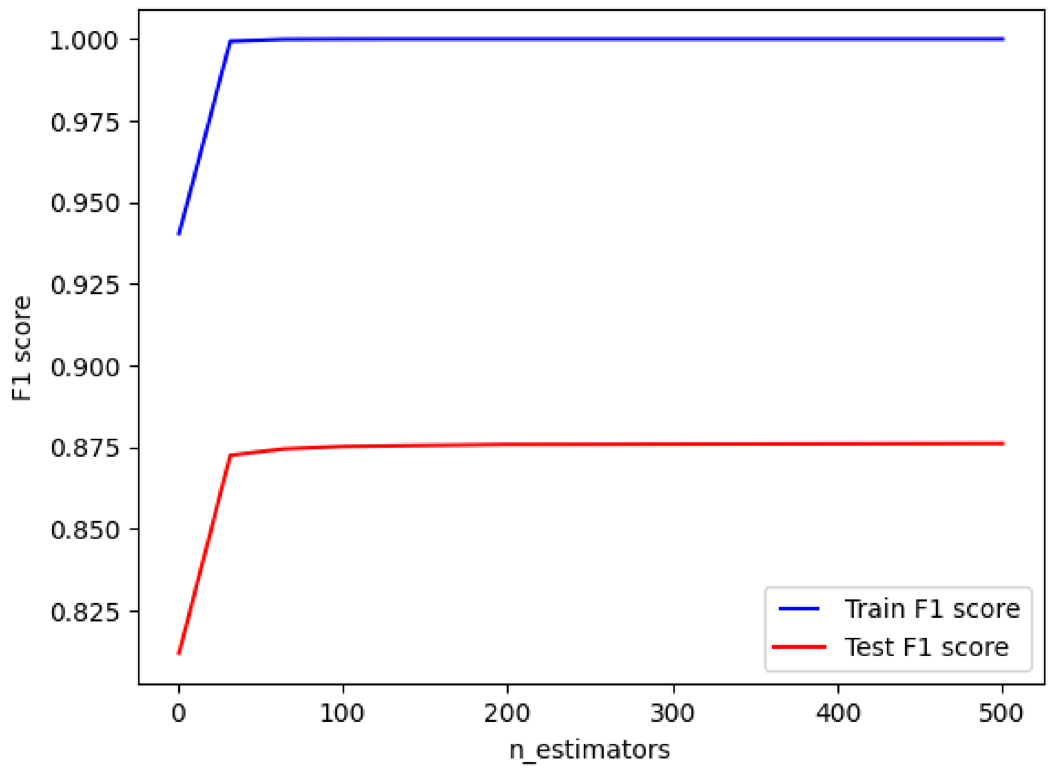


Figure 5.1: F_1 score vs the number of trees in the forest.

As we can see in Figure 5.1, our experiments agree with the literature. The best performance is achieved when the number of trees in the forest is close to 100. After 100 trees, the F_1 score does not increase significantly.

5.3.2 Depth of each tree in the forest

The parameter *max_depth* refers to the depth of each tree in the forest. The higher the value, the deeper the tree and the more splits it has. For this parameter, the values we tested are ranging from 1 to 32. Figure 5.2 illustrates the F_1 score values for the models trained and tested with different numbers of tree depth.

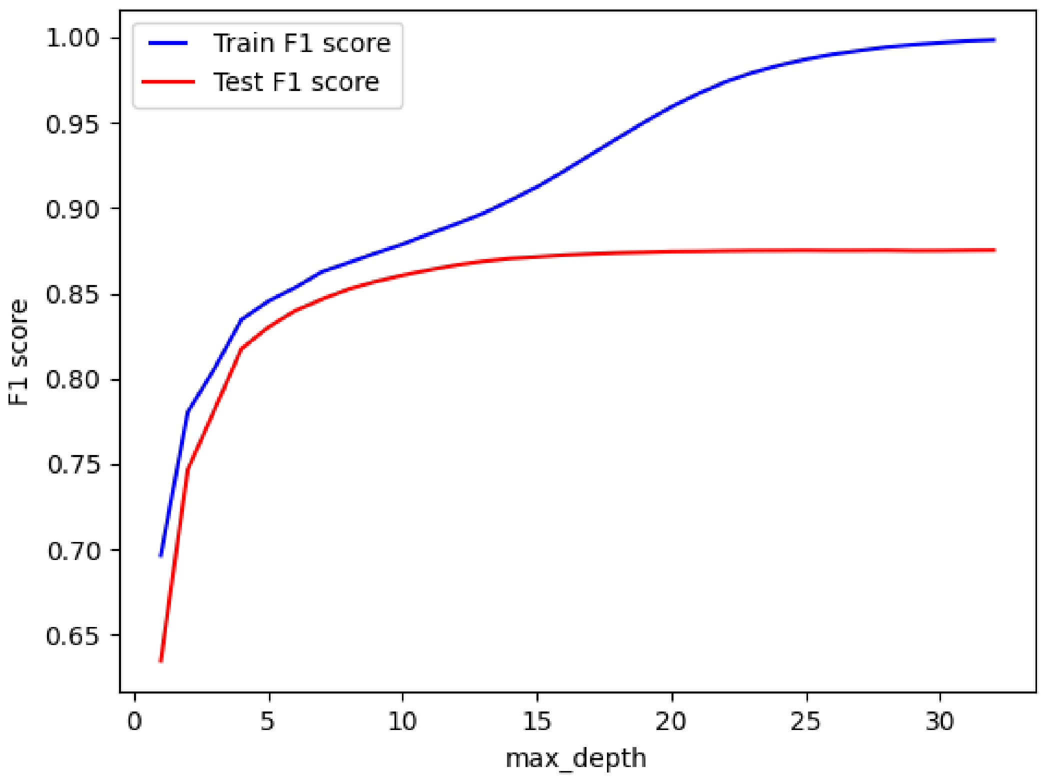


Figure 5.2: F_1 score vs the depth of each tree in the forest.

Looking at Figure 5.2 we see that the best performance is achieved when the depth of each tree in the forest is close to 15. After 15, the F_1 score stops increasing. We also notice that the train and test curves have different trends. The larger the depth, the higher the F_1 score for the training dataset. Eventually, at some point, the F_1 score becomes 1, and this happens because when the depth of each tree is very large, the model overfits the training data.

5.3.3 Minimum number of samples required to split an internal node

The hyperparameter *min_samples_split* refers to the minimum number of points (samples) from the original point cloud required to split an internal node. Its value can range from 1 to all of the samples at each node. We tested its effect on the model for values from 10% to 100% of the samples. Figure 5.3 illustrates the F_1 score values for the models trained and tested with different values for the *min_samples_split* hyperparameter.

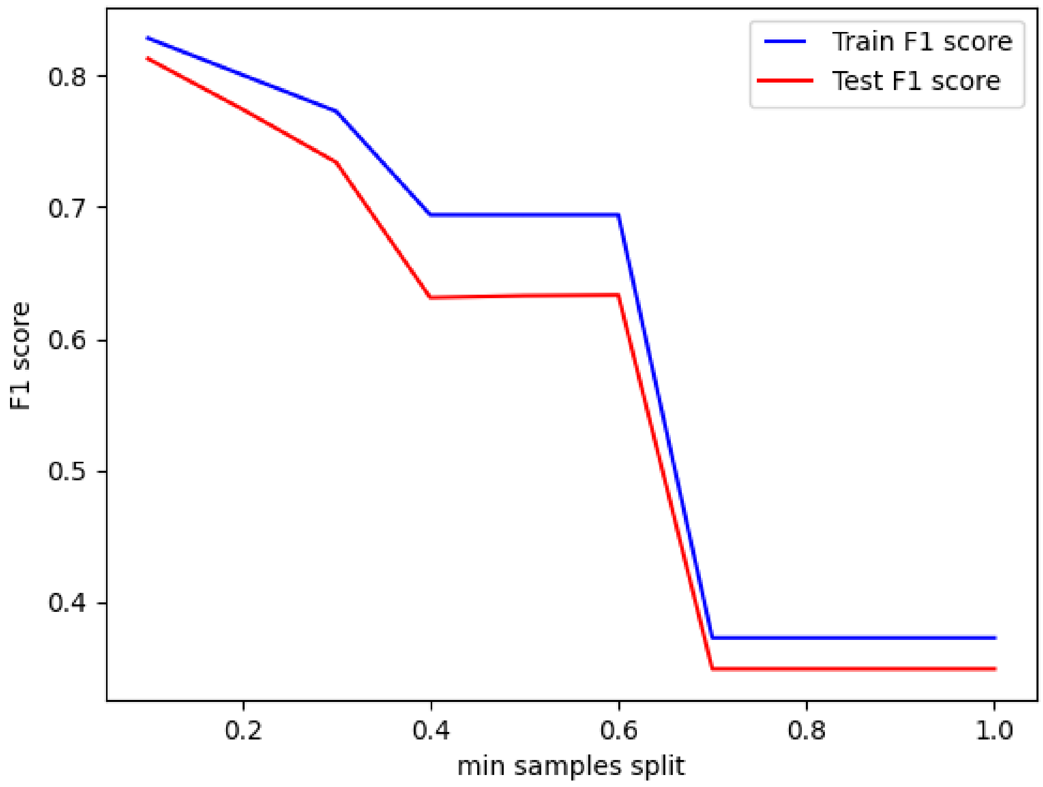


Figure 5.3: F_1 score vs the minimum number of samples required to split an internal node.

The results of Figure 5.3 illustrate that out of all the percentages of samples that we tested, the best performance is achieved when the minimum number of samples required to split an internal node is 10%. After 10%, the F_1 score value starts to decrease considerably, levelling out close to 0 when the value for $min\_samples\_split$ is equal to or larger than 70%. The smaller the value of $min\_samples\_split$, the better the performance of the model. The smallest value that it can get is 2 data points, which is the default value.

5.3.4 Minimum number of samples required to be at a leaf node

The hyperparameter $min\_samples\_leaf$ refers to the minimum number of points (samples) from the original point cloud required to be at a leaf node. It is similar to the $min\_samples\_split$ hyperparameter. However, it concerns the leaves, in other words, the base of the tree. The value of this hyperparameter can range from 1 to all of the samples at each node. We tested its effect on the model for values from 10% to 50% of the samples. Figure 5.4 illustrates the F_1 score values for the models trained and tested with different values for the $min\_samples\_leaf$ hyperparameter.

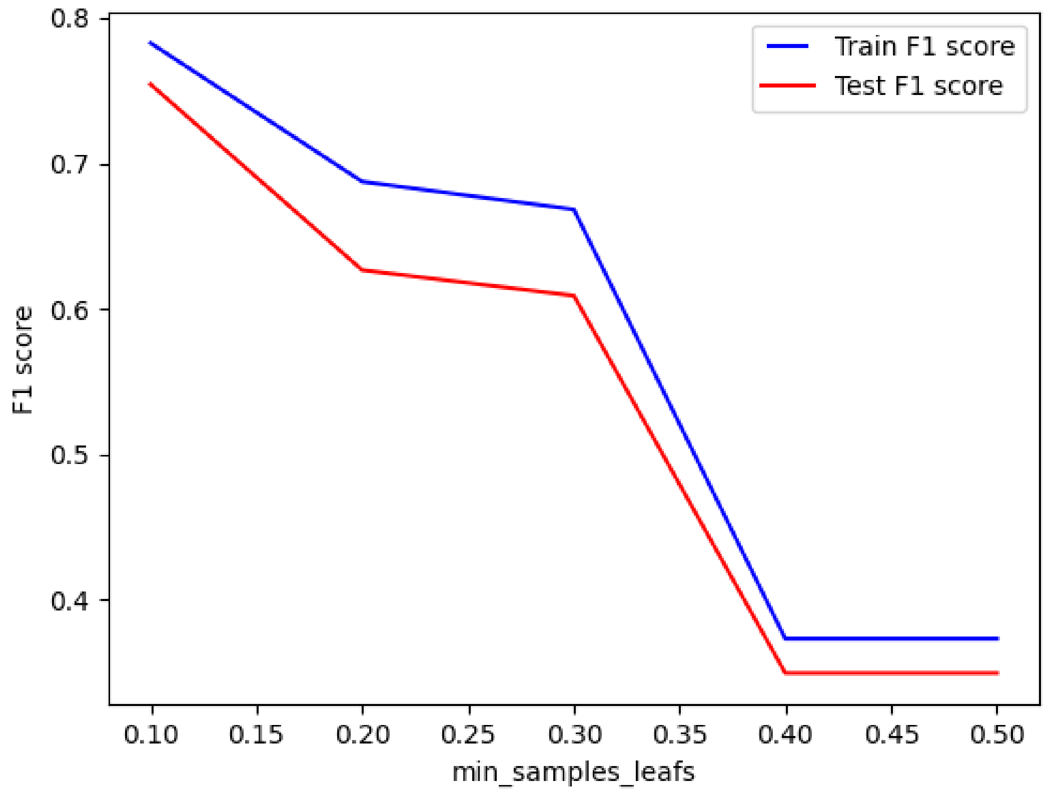


Figure 5.4: F_1 score vs the minimum number of samples required to be at a leaf node.

The results of Figure 5.4 illustrate that out of all the percentages of samples that we tested, the best performance is achieved when the minimum number of samples required to be at a leaf node is 10%. After 10%, the F_1 score value starts to decrease considerably, following a downward trend as the percentage of samples used is increased. The line graph levels out close to 0 when the value of $min\_samples\_leaf$ is equal to or larger than 40%. The smaller the value of $min\_samples\_leaf$, the better the performance of the model. The smallest value that it can get is 1 data point, which is the default value.

5.3.5 Number of features to consider when looking for the best split

The parameter $max\_features$ refers to the number of features to consider when looking for the best split. For this hyperparameter, the values we tested are 1, 3, 5, 8, 11, 14, 17, 20 and 23 features. In total, we used 24 features to train and test the models. Figure 5.5 illustrates the F_1 score values for the models trained and tested with different values for the hyperparameter $max\_features$.

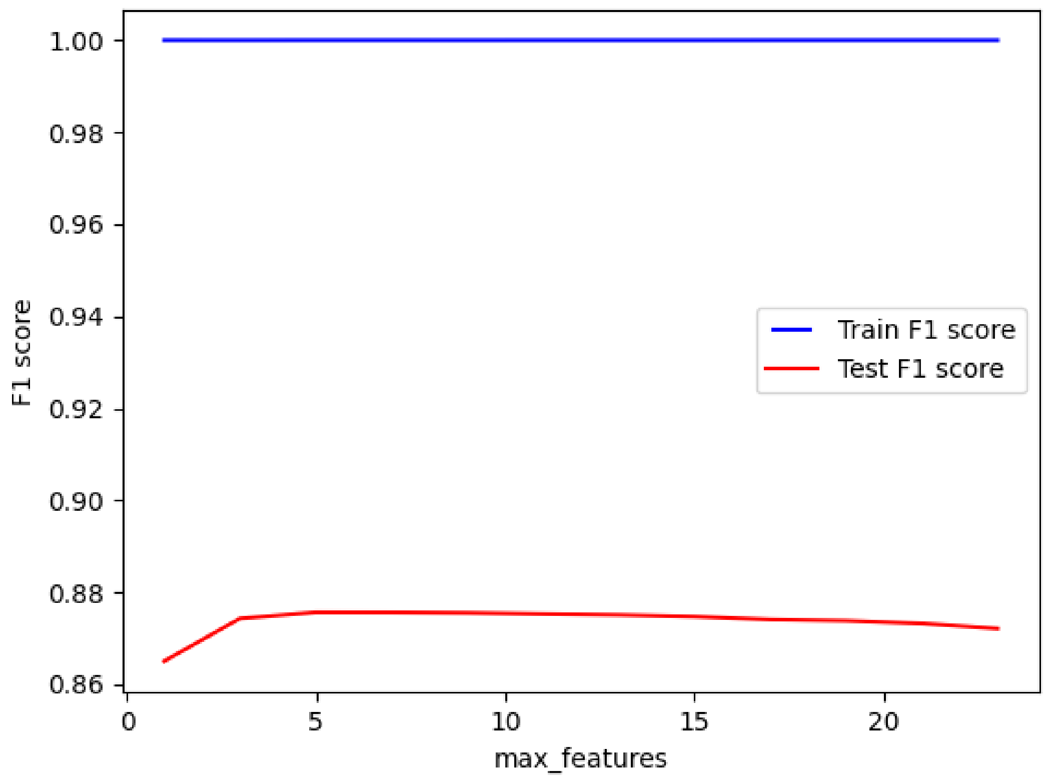


Figure 5.5: F_1 score vs the number of features to consider when looking for the best split.

As we can see in Figure 5.5, our experiments agree with the literature. The best performance is achieved when the number of features to consider when looking for the best split is close to the squared root of the number of features used. The F_1 score starts to decrease when using more than 5 features.

5.3.6 Splitting rule, bootstrap and out-of-bag samples

The bootstrap hyperparameter refers to whether bootstrap samples are used or not when building the trees, and its value can be False or True. The criterion hyperparameter refers to the function used to measure the quality of the splits, and its value can be Gini or entropy. For all the tests that we made, the results for both values of these two hyperparameters were very similar. Because of this, we decided to stick with the default values, which are bootstrap = True and criterion = Gini. Based on the literature, the fact that the model's accuracy was similar with and without bootstrap samples indicates that the parameter that determines the sample size used to train each tree is set optimally. As to whether out-of-bag samples should be used to estimate the generalization score, setting this hyperparameter to True for our final model (see Section 5.8) slightly increased the accuracy of our model and notably reduced its training time.

5.4 Density of the training data

Table 5.4 illustrates how much the overall accuracy, per class IoU, mean IoU, CCI, F_1 score, training time, RAM required, and the size of our classification model can be influenced by the voxel size of the uniform sampling algorithm. Five different models have been trained using a different voxel size for the uniform sampling algorithm each time. All of them were trained with the same 10 tiles corresponding to the first 10 best tiles, as shown in the table 5.1. The testing data that we used for these experiments are tiles 11 to 20. The default hyperparameters were used to train the models (see Table 5.3). Eigen features from 2 different spherical neighbourhoods with radii 3 and 4 meters have been extracted.

Table 5.4: Comparison of the performance of Random Forest models that use different voxel size values for the uniform sampling algorithm. The models were trained with the AHN3 dataset.

		IoU (%)									
voxel size (m)	OA (%)	mean	other	building	ground	CCI (%)	F_1 score	no. of points in the training data	training time (minutes)	RAM (GB)	model size (KB)
0.5	89.70	79.07	78.63	72.07	86.51	99.56	0.90	19,654,135	187.79	33.7	14,581,904
1	89.04	78.03	77.32	71.46	85.31	99.59	0.89	6,644,360	73.41	8.3	5,638,778
1.5	87.45	75.07	82.09	63.39	79.73	99.08	0.87	3,034,839	33.85	4.6	3,299,808
2	87.41	74.42	73.29	65.21	84.75	99.14	0.87	1,648,188	14.94	2.7	1,946,693
-	86.41	73.64	76.66	62.46	81.81	99.09	0.86	42,144,302	460.34	59.9	24,501,065

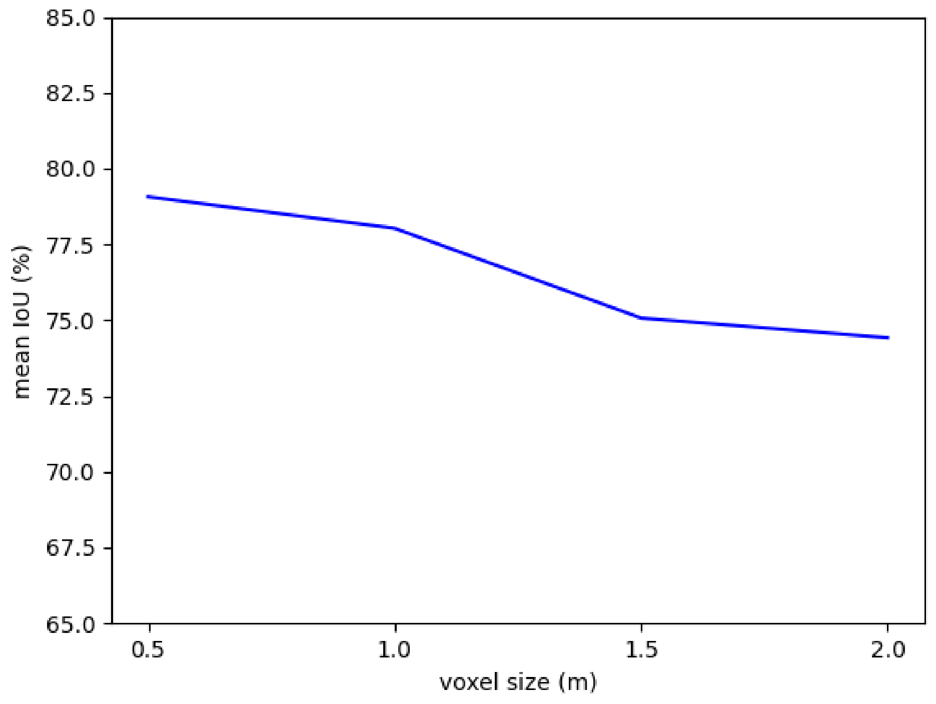


Figure 5.6: Mean IoU vs the voxel size of the uniform sampling algorithm.

These results show that our classifier becomes more efficient if our training data have a smaller point density. Not only is the runtime reduced but also the required amount of RAM and the size of the resulting trained model. Using a 1 meter voxel size did not reduce the accuracy of the classifier significantly. It is important to note that the model trained without applying a uniform sampling algorithm on the input point cloud (voxel size = -) performed worse than the rest. This highlights the importance of the uniform sampling algorithm and having a point cloud with evenly distributed points. The raw AHN3 point cloud does not have a consistent point density. Thus, some of the local spherical neighbourhoods have too many or too few points in them, resulting in features that are not discriminating and informative enough. However, it goes without saying that a higher point density yields better results, which is proven by our results. The smaller the voxel size, the more points are used to train the model, and the better the accuracy of our model. Overall, these results show that, in our case, using a 1 meter voxel size for the uniform sampling algorithm provides a good trade-off between accuracy and efficiency.

5.5 Size of the training data

Table 5.5 illustrates how much the overall accuracy, per class IoU, mean IoU, CCI, F_1 score, training time, RAM required and the size of our classification model can be influenced by the size of the training data. Seven different models have been trained using a different number of training tiles from the AHN3 dataset, 1, 2, 3, 5, 10, 15 and 20 tiles. These correspond to the first 20 best tiles, as shown in table 5.1. The testing data that we used for these experiments are tiles 21 to 30. The default hyperparameters were used to train the models (see Table 5.3). A uniform sampling algorithm with 1 meter voxel size has been applied to the tiles before extracting their eigen features for 3 different spherical neighbourhoods with radii 2, 3 and 4 meters.

Table 5.5: Comparison of the performance of Random Forest models trained using a different number of training tiles from the AHN3 dataset.

		IoU (%)									
no. of training tiles	OA (%)	mean	other	building	ground	CCI (%)	F_1 score	no. of points in the training data	training time (minutes)	RAM (GB)	model size (KB)
1	87.21	75.26	76.63	66.93	82.21	99.47	0.87	662,595	6.74	1.1	569,095
2	88.42	77.93	78.39	72.47	82.93	99.77	0.88	1,178,285	15.48	1.9	1,154,913
3	88.82	78.59	78.67	73.53	83.57	99.79	0.89	1,777,401	25.82	2.9	1,618,543
5	88.54	78.05	78.71	72.28	83.17	99.74	0.89	2,976,008	30.15	4.4	2,668,248
10	89.27	79.17	78.99	73.92	84.58	99.76	0.89	6,644,360	80.04	9	5,496,317
15	89.34	79.41	79.02	74.56	84.64	99.79	0.89	10,159,167	96.38	13.8	8,378,902
20	89.27	79.36	79.24	74.54	84.32	99.80	0.89	13,290,986	129	24.4	11,704,444

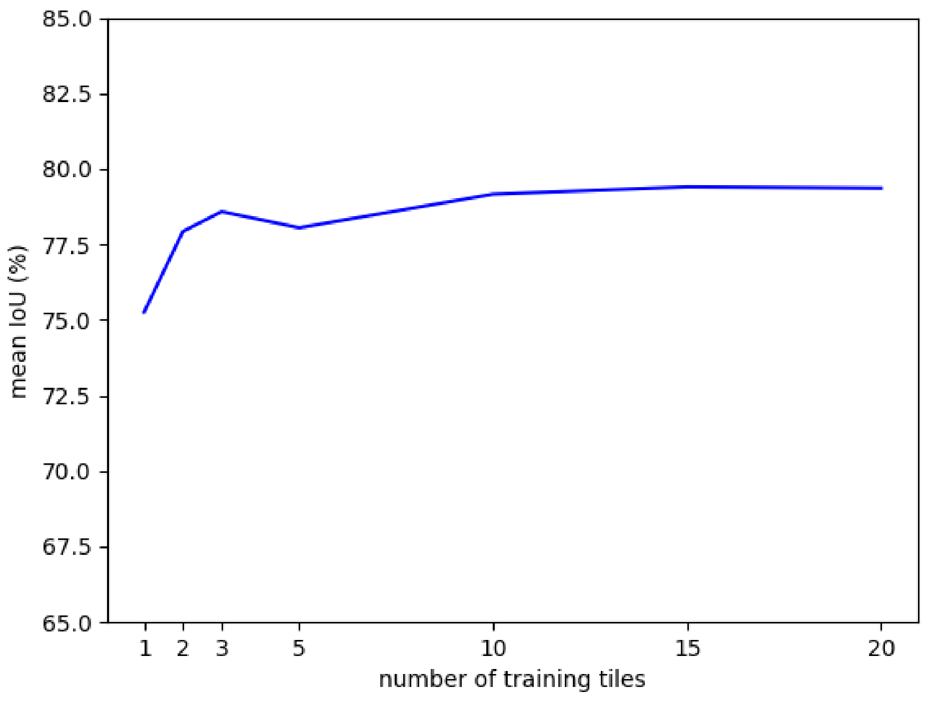


Figure 5.7: Mean IoU vs the number of tiles used during the training process.

We observe that the larger the size of the training data, the better the performance of the classification model. However, after 2 tiles, the difference is not significant. We also notice that the model trained with 5 tiles performed worse than the model trained with 3 tiles and the model trained with 20 tiles from the one trained with 15 tiles. This also shows that the quality of the training data is crucial since the feature diversity score of the tiles used to train the model was slowly decreasing. We started from the best and added the next best each time. We also observe that the less training data, the less the training time, the required amount of RAM and the size of the resulting trained model. From the analysis of these results, we can say that, in our case, using only 3 training tiles that are equal to an area of 0.75 squared kilometres is enough to provide us with a decent performing model, and it is more efficient.

5.6 Handcrafted features

For all of the models that we tested, their calculated feature importances, based on the mean decrease in impurity, show that with a great difference, the most important features are the two height features, $Z_{normalized}$ and Z_{below}. We trained and tested models using both of them or only one of them. Based on these models' results, we decided that it is best to use both during the training process. As for the eigen-based features, their importance can vary depending on the sizes of the radii of the local spherical neighbourhoods from which we extract them. For example, omnvivariance is a significant feature when spherical neighbourhoods with large radii are used. In the case of spherical neighbourhoods with small radii, however, it can reduce the performance of a model if it is included in the feature vector of the points. It is also worth mentioning that another eigen-based feature was computed during our initial experiments. This feature is eigentropy, but we decided to discard it since it always lowered the performance of our model when we included it.

This section presents the results of our experiments with the handcrafted features and local neighbourhoods of the 3D points. For these experiments, the models were trained using only the first two tiles from table 5.1. The testing data were tiles 11 to 20. The default hyperparameters were used to train the models (see Table 5.3). A uniform sampling algorithm with 1 meter voxel size has been applied to the tiles before extracting their eigen features from 3 different spherical neighbourhoods with radii 2, 3 and 4 meters. Table 5.6 shows the classification results of our experiments with the models we trained with a different number of features and multi-scale representations. Table 5.7 shows the classification results of our experiments with the models we trained with different numbers of features that were gradually increased. We started from a base model trained only with 3 features, the 2 height features and density, and each time one of the eigen features was added to the training process to test its influence. For these experiments, the features that we used were extracted from the spherical neighbourhoods of the 3D points with radius 2 meters.

Table 5.6: Comparison of the performance of models trained with different number of features derived from different spherical neighborhoods. To train and test all seven models only 8.4 GB of RAM were required.

			IoU (%)						
no. of features	radii of spherical neighborhoods	OA (%)	mean	other	building	ground	CCI (%)	F_1 score	training time (minutes)
10	2	86.34	72.74	72.49	61.95	83.77	98.91	0.86	7.17
10	3	86.91	74.39	73.19	66.92	83.06	99.41	0.87	5.62
10	4	87.16	75.29	72.85	70.28	82.72	99.62	0.87	5.41
17	2, 3	88.91	77.701	76.69	70.64	85.78	99.50	0.89	7.54
17	2, 4	89.68	79.29	78.13	73.50	86.23	99.65	0.90	7.46
17	3, 4	89.13	78.45	76.47	73.40	85.49	99.66	0.89	7.31
24	2, 3, 4	90.16	80.12	78.69	74.71	86.96	99.68	0.90	7.75

Looking at table,5.6 we notice that the more features we use, the higher the accuracy of our model. Also, the largest radius provided the most informative and discriminating features since it has the best results. However, the difference with the other radii is not very significant. Multi-scale representations performed better than single-scale representations, and their training time was not considerably higher.

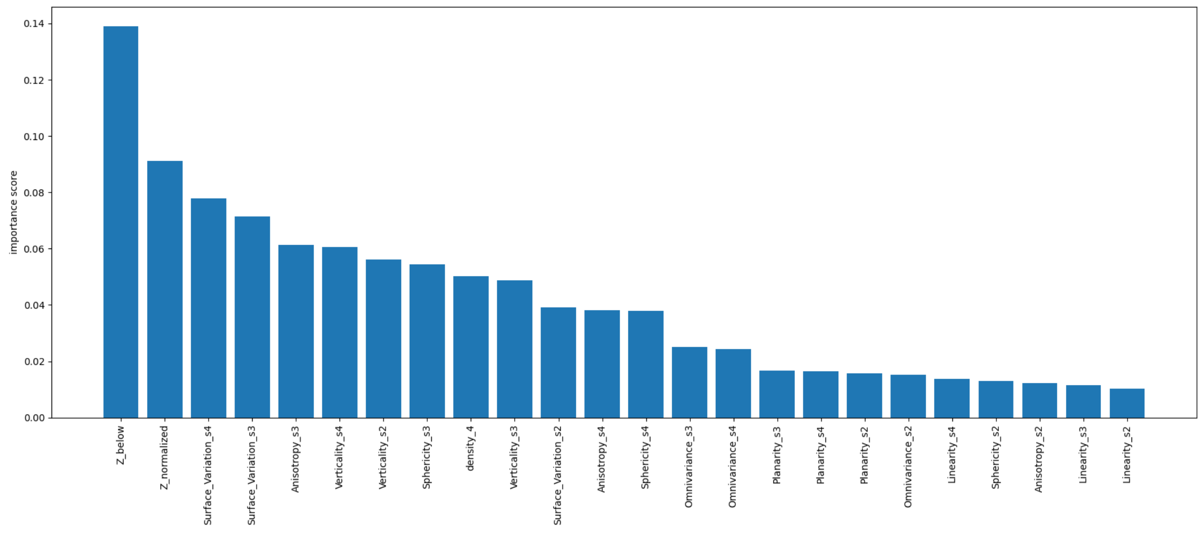


Figure 5.8: Feature importances calculated based on the mean decrease in impurity.

Figure 5.8 presents the feature importances of the model trained with 24 features, the 2 height features, density and the seven eigen features from 3 different spherical neighbourhoods of radii 2, 3 and 4. The importances are calculated based on the mean decrease in impurity within each tree. Feature Z_{below} has the highest score followed by $Z_{normalized}$ and Surface Variation for the spherical neighbourhood with a radius of 4 meters. Linearity has the lowest scores.

Table 5.7: Comparison of the performance of models trained with a different number of features derived from a spherical neighbourhood of 2 meters. The first model was trained using only 3 features, the 2 height features and density, and each time one of the eigen features was added to the calculation. We removed one of the 2 height features for the last two models to check if it is worth keeping both.

			IoU (%)					
no. of features	**features used for training**	**OA (%)**	**mean**	**other**	**building**	**ground**	**CCI (%)**	**F1 score**
3	$Z_{normalized}$, Z_{below}, Density	75.58	56.97	54.17	43.04	73.67	97.18	0.76
4	$Z_{normalized}$, Z_{below}, Density, Omnivariance	80.68	64.23	63.49	51.72	77.49	98.27	0.81
5	$Z_{normalized}$, Z_{below}, Density, Omnivariance, Anisotropy	82.78	67.45	66.93	55.90	79.51	98.62	0.83
6	$Z_{normalized}$, Z_{below}, Density, Omnivariance, Anisotropy, Planarity	84.96	70.84	70.49	60.11	81.93	98.88	0.85
7	$Z_{normalized}$, Z_{below}, Density, Omnivariance, Anisotropy, Planarity, Linearity	85.26	71.33	70.73	60.92	82.35	98.92	0.85
8	$Z_{normalized}$, Z_{below}, Density, Omnivariance, Anisotropy, Planarity, Linearity, Surface Variation	85.49	71.72	70.87	61.61	82.67	98.96	0.85
9	$Z_{normalized}$, Z_{below}, Density, Omnivariance, Anisotropy, Planarity, Linearity, Surface Variation, Sphericity	85.06	71.00	70.58	60.36	82.08	98.89	0.85
10	$Z_{normalized}$, Z_{below}, Density, Omnivariance, Anisotropy, Planarity, Linearity, Surface Variation, Sphericity, Verticality	86.29	72.64	72.48	61.75	83.69	98.90	0.86
9	Z_{below}, Density, Omnivariance, Anisotropy, Planarity, Linearity, Surface Variation, Sphericity, Verticality	86.04	72.04	71.62	60.79	83.72	98.78	0.86
9	$Z_{normalized}$, Density, Omnivariance, Anisotropy, Planarity, Linearity, Surface Variation, Sphericity, Verticality	86.32	72.74	72.25	62.24	83.73	98.94	0.86

Looking at table 5.7, we notice that the more features we use, the higher the accuracy of our model. All eigen features have important information to offer and increased the performance of the model. The results of the last 2 models show that keeping both height features does not increase the accuracy of our model. The performance of the model that used only the $Z_{normalized}$ feature was a bit better.

5.7 Testing with other datasets

In the following subsections, we presented the classification results of the three different Random Forest models that have been trained and tested using the AHN3, AHN4 and DALES datasets as training and testing data. All possible combinations of training and testing datasets have been implemented to test the effect that different datasets can have on our classification model. As far as the parameters of our 3D point cloud analysis algorithms are concerned, we used a 1 meter voxel size for our uniform sampling algorithm, and features have been extracted from three different spherical neighbourhoods with radii 2, 3 and 4 meters. For the models trained with the AHN3 and AHN4 datasets, only the first two tiles from tables 5.1 and 5.2. Tiles 21-30 were used as testing data. The default hyperparameters were used to train the models (see Table 5.3). We calculated all the classification metrics mentioned in the Chapter 3 Methodology as well as the confusion matrices.

Overall, we observe that our models perform worse when tested with different datasets than those used to train them. The model trained with the DALES dataset provided the best classification results. However, the training data of the DALES dataset is larger, and that is something that can influence the accuracy of a model. If we look at the results individually, it performed worse when tested with the other two datasets. The two models trained with the same AHN3 and AHN4 datasets have almost the same results. This shows that the performance of a model is influenced by the differences between the environments of the training and testing data.

5.7.1 Model trained with the AHN3 dataset

The following figures and table provide the classification results for the model trained the with AHN3 dataset. The mean IoU is almost the same when tested with the AHN3 and AHN4 datasets which is a good sign that indicates that our model is stable. The model performed worse for the DALES dataset. The building and ground class IoU dropped considerably in that case. This is probably because the DALES dataset represents a different environment with buildings that can not be found in the AHN3 and AHN4 datasets. Looking at the confusion matrix, we notice that most misclassified building points were classified as other points. The F_1 scores of the three results, however, are comparable.

Table 5.8: Evaluation metrics of the Random Forest model trained with the AHN3 dataset and tested with all other datasets.

		IoU (%)					
Testing dataset	OA (%)	mean	other	building	ground	CCI (%)	F_1 score
AHN3	88.42	77.93	78.39	72.47	82.93	99.77	0.88
AHN4	88.16	77.27	72.42	75.44	83.94	99.69	0.88
DALES	80.31	64.51	70.58	51.43	71.53	98.67	0.81

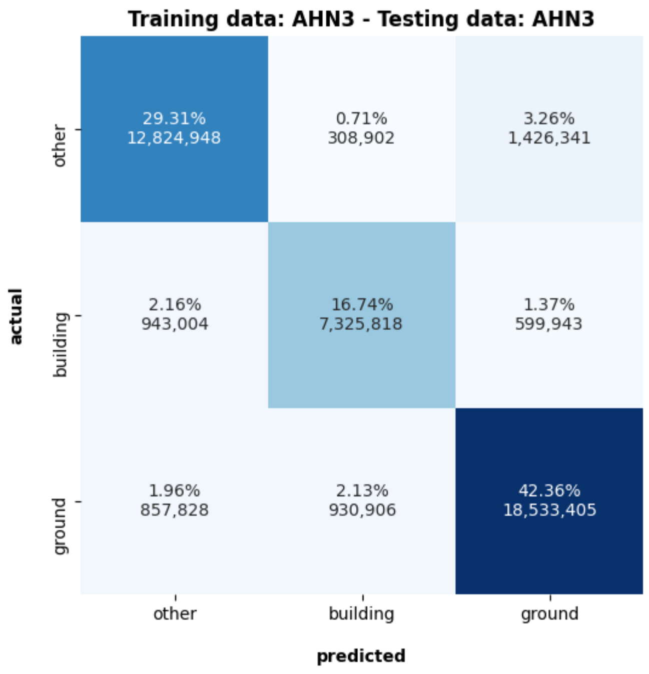


Figure 5.9: Confusion matrix of the classification result of the Random Forest model that was trained with the AHN3 dataset and tested with the AHN3 dataset.

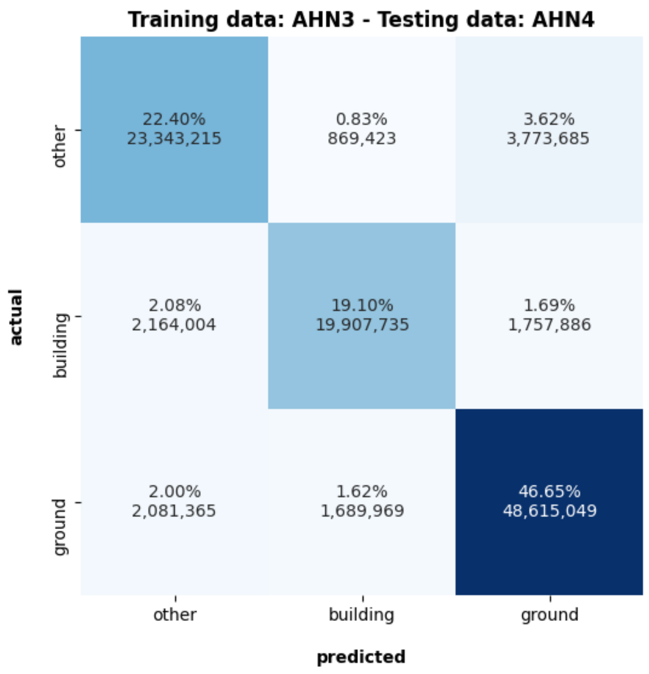


Figure 5.10: Confusion matrix of the classification result of the Random Forest model that was trained with the AHN3 dataset and tested with the AHN4 dataset.

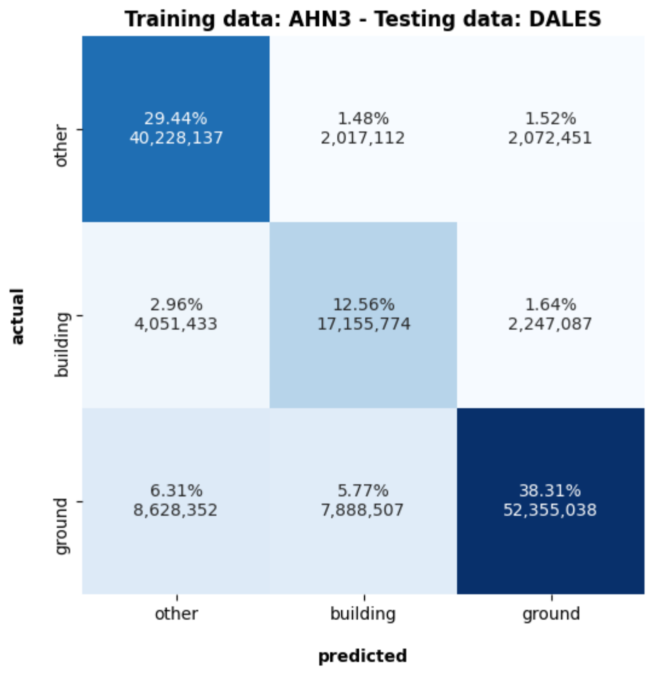


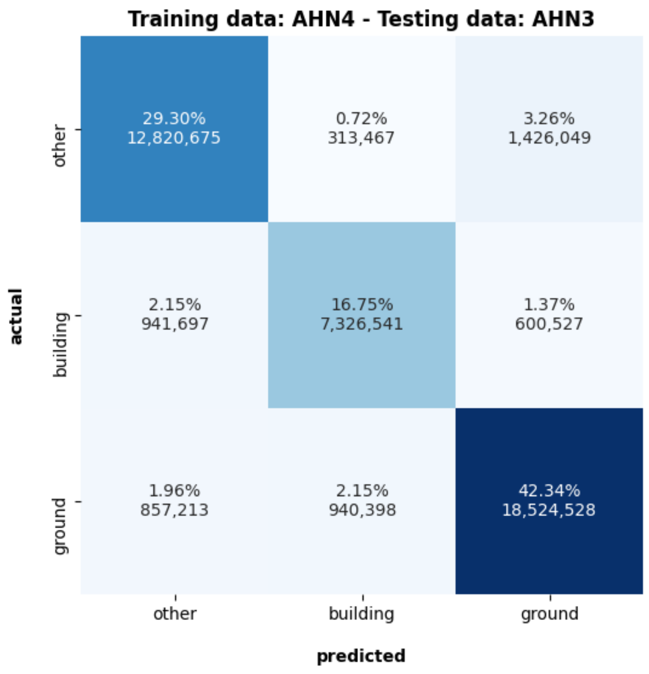
Figure 5.11: Confusion matrix of the classification result of the Random Forest model that was trained with the AHN3 dataset and tested with the DALES dataset.

5.7.2 Model trained with the AHN4 dataset

The following figures and table provide the classification results for the model trained with the AHN4 dataset. Similarly to the model trained with the AHN3 dataset, the mean IoU is almost the same when tested with the AHN3 and AHN4 datasets, which is another good sign that our model is stable. The model performed worse for the DALES dataset. The building and ground class IoU dropped considerably in that case, probably for the same reason we mentioned for the model tested with the AHN3 dataset since the two datasets are almost the same. The confusion matrix shows that most of the misclassified building points were classified as other. The F_1 scores of the three results are comparable, in this case as well.

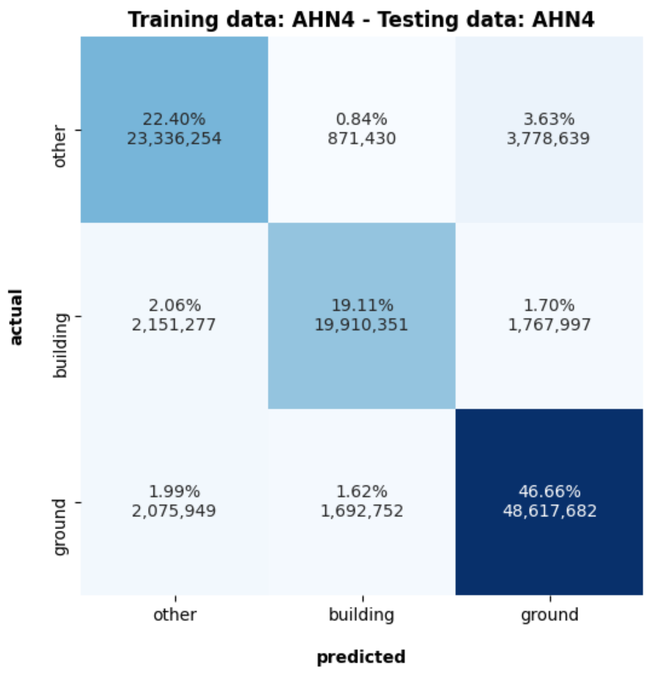
Table 5.9: Evaluation metrics of the Random Forest model trained with the AHN4 dataset and tested with all other datasets.

		IoU (%)					
Testing dataset	OA (%)	mean	other	building	ground	CCI (%)	F_1 score
AHN3	88.39	77.88	78.37	72.38	82.89	99.76	0.88
AHN4	88.16	77.27	72.44	75.44	83.92	99.69	0.88
DALES	80.38	64.61	70.59	51.59	71.65	98.69	0.81



actual \ predicted	other	building	ground
other	29.30% 12,820,675	0.72% 313,467	3.26% 1,426,049
building	2.15% 941,697	16.75% 7,326,541	1.37% 600,527
ground	1.96% 857,213	2.15% 940,398	42.34% 18,524,528

Figure 5.12: Confusion matrix of the classification result of the Random Forest model that was trained with the AHN4 dataset and tested with the AHN3 dataset.



actual \ predicted	other	building	ground
other	22.40% 23,336,254	0.84% 871,430	3.63% 3,778,639
building	2.06% 2,151,277	19.11% 19,910,351	1.70% 1,767,997
ground	1.99% 2,075,949	1.62% 1,692,752	46.66% 48,617,682

Figure 5.13: Confusion matrix of the classification result of the Random Forest model that was trained with the AHN4 dataset and tested with the AHN4 dataset.

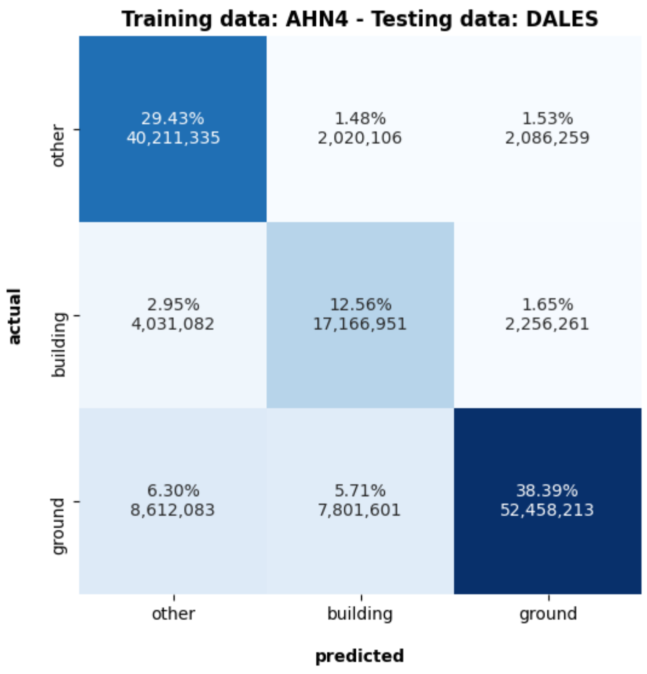


Figure 5.14: Confusion matrix of the classification result of the Random Forest model that was trained with the AHN4 dataset and tested with the DALES dataset.

5.7.3 Model trained with the DALES dataset

The following figures and table provide the classification results for the model trained with the DALES dataset. The mean IoU, in this case, is similar for all testing data. The model performed better when tested with the DALES dataset. As for the other two datasets, their mean IoU dropped slightly compared to the previous two cases. The confusion matrices show that the model mostly misclassifies other points as ground. The F_1 scores of the three results are very close. A possible reason why this model performed better than the other two is that the DALES training dataset is larger. However, the other two models were a lot more efficient since they were trained with a point cloud covering an area of 0.5 squared kilometres. As for the DALES dataset, each tile is equal to an area of 0.5 squared kilometres, and it has 29 training tiles in total. The previous experiments we made showed us how much the size of the training data could influence the performance of our model. It is a good trade-off between accuracy, stability and efficiency.

Table 5.10: Evaluation metrics of the Random Forest model that was trained with the DALES dataset and tested with all other datasets.

		IoU (%)					
Testing dataset	OA (%)	mean	other	building	ground	CCI (%)	F_1 score
AHN3	86.58	74.53	75.93	67.12	80.56	99.58	0.86
AHN4	86.11	73.49	69.80	69.37	81.29	99.59	0.86
DALES	89.75	79.41	79.62	73.07	85.55	99.67	0.90

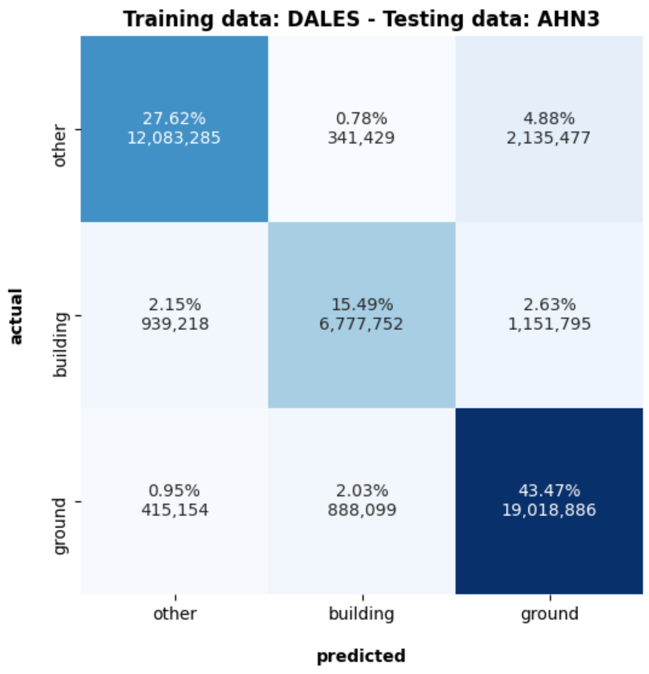


Figure 5.15: Confusion matrix of the classification result of the Random Forest model that was trained with the DALES dataset and tested with the AHN3 dataset.

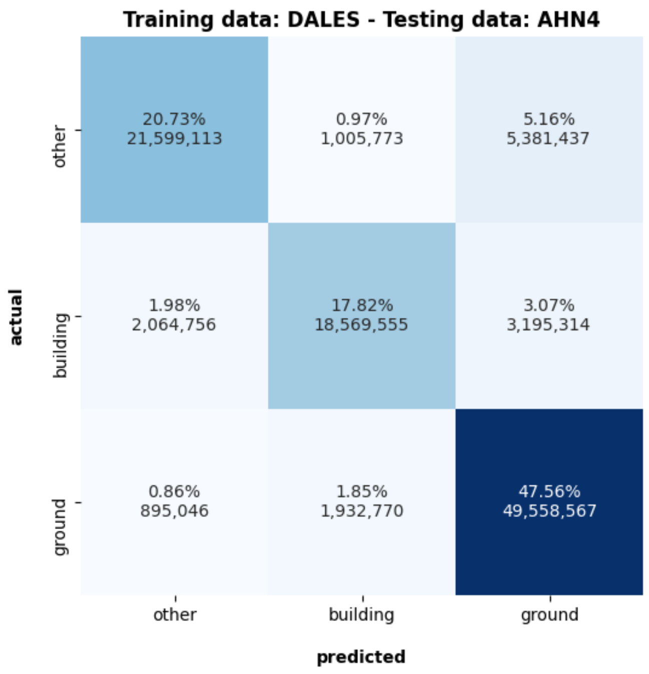


Figure 5.16: Confusion matrix of the classification result of the Random Forest model that was trained with the DALES dataset and tested with the AHN4 dataset.

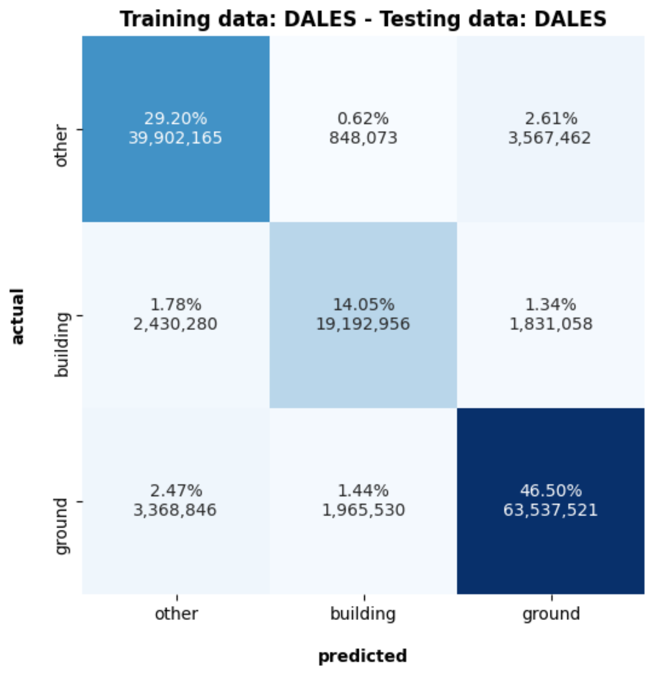


Figure 5.17: Confusion matrix of the classification result of the Random Forest model that was trained with the DALES dataset and tested with the DALES dataset.

5.8 Final model

Based on the observation we made from all of the experiments that we conducted, our final model was trained using **13 features**, the two **height features**, $Z_{normalized}$ and Z_{below}, **density**, Verticality, Surface Variation and Planarity from three different spherical neighbourhoods with radii 2, 3 and 4 metes and Linearity from a spherical neighbourhood with radius 4 meters. In some of the previously mentioned experiments, the $Z_{normalized}$ feature seems better than the Z_{below} feature, while in others, the opposite. In the end, we decided to keep both since it does not increase the training time and the required amount of RAM considerably. Moreover, we also trained models without one of the 2 features, and their results were worse than the model that both of them. The **hyperparameters** that were used to train the model are:$n\_estimators = 100$, $criterion = Gini$, $max\_depth = 15$, $min\_samples\_split = 2$, $min\_samples\_leaf = 1$, $max\_features = sqrt$, $bootstrap = True$ and $oob\_score = True$. A uniform sampling algorithm with a **voxel size of 1 meter** was applied to the training and testing data. The best three tiles from tables 5.1 were used as training data while the rest, 4 to 40, were used as testing data. Training of model lasted **12.45 minutes** and required **1.1 GB of RAM**. The final model has a **size of 229,494 kilobytes (KB)**. However, we understand that these numbers can vary depending on the implementation. In our case, parallel computing was not implemented to speed up the data analysis process due to some difficulties and limitations we faced with some Python packages and the HPC cluster of TU Delft.

Our final model was first trained using 24 different features. These are the eigen-based features we previously mentioned in Chapter 3 Methodology from three different spherical neighbourhoods with radii 2, 3 and 4 metes, the two height features, $Z_{normalized}$ and Z_{below}, and density. However, we decided that it was best to test each feature's effect on our final model now that different training and testing data are used and the hyperparameters were optimized. The mean permutation importances for 10 permutations were calculated for all 24 features as shown in Figure 5.18 while Table 5.11 provides the evaluation metrics of the models trained and tested using different numbers of features. Each time the feature with the lowest permutation importance score was excluded from the feature vector of the points, and the model was trained and tested again. Permutation importances are more costly to calculate. However, they overcome limitations of the impurity-based feature importances, and thus, they are more reliable. This approach is an iterative process for every feature used to train the model. To put it simply, for each feature, it is observed how random re-shuffling (permutations) of its values influences the performance of the model. From the results of Table 5.11 we see that our model actually performs better when features with low permutation importances are discarded. The model using only 13 features yield the highest mean IoU and building IoU. The performance of our model started to decrease when less than 13 features were used to train it.

Overall, our final model managed to achieve an F_1 score of **0.90** and a mean IoU score of **0.79** across all testing tiles. Table 5.12 provides the overall evaluation metrics of the final model when tested for all the tiles of the AHN3 dataset that we clipped, and figure 5.19 shows the confusion matrix of the classification result for all points in all the 37 tiles. Figure 5.20 presents the 3 tiles used to train our model, tiles (7,3), (5,5) and (9,0). These 3 tiles cover an area of 0.75 squared kilometres in total, which was enough to classify accurately the 37 tiles, which are equal to an area of 9.25 squared kilometres. Figure 5.21 presents the ground truth class labels and predictions for 3 of the tiles with the lowest evaluation metrics, tiles (4,9), (6,4) and (8,0). Figures 5.22, 5.23 and 5.24 show a closer look at the problematic areas of those 3 tiles, while Figure 5.25 shows some of the most common problems related to buildings that can be found in the predictions of all of our testing tiles.

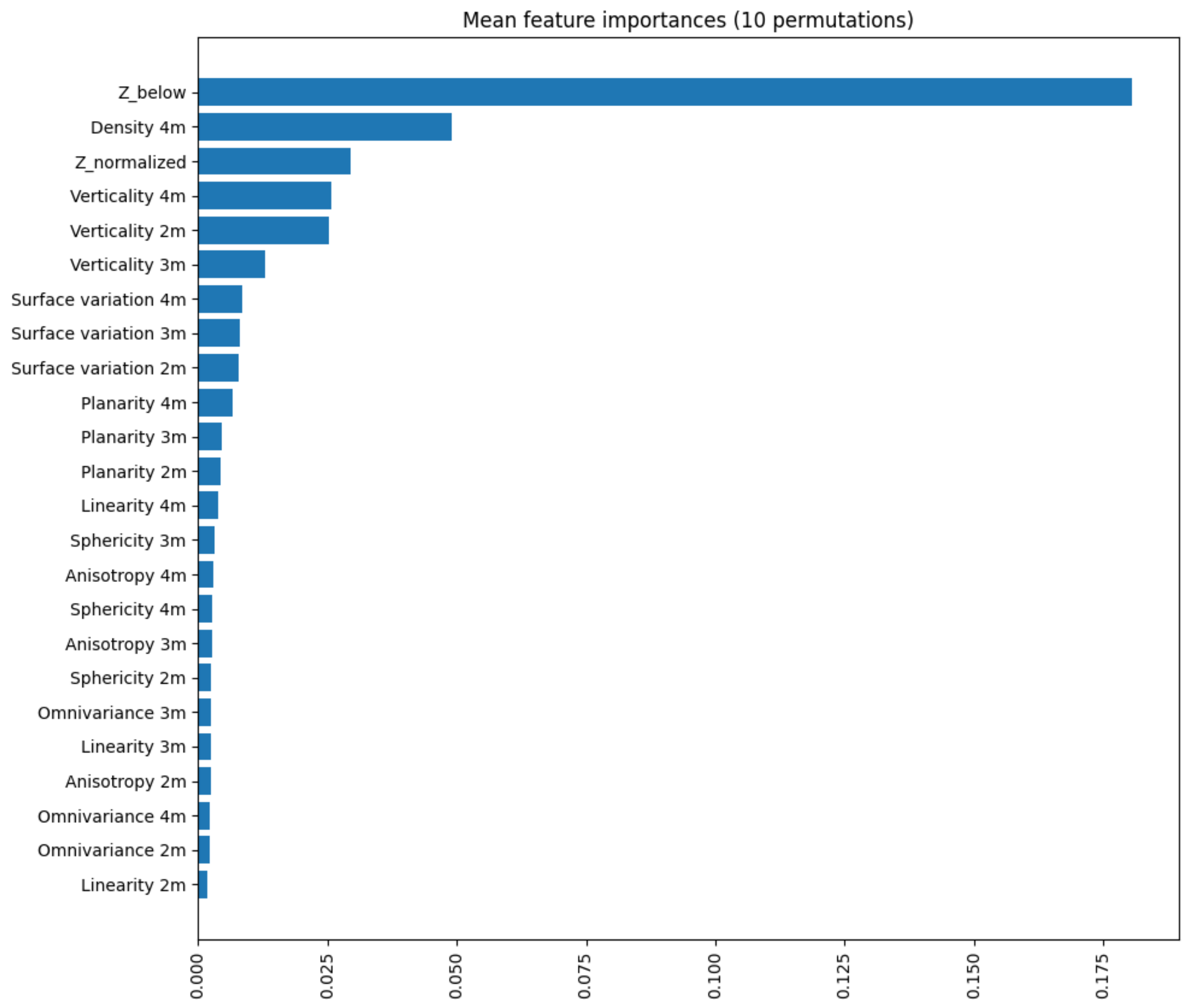


Figure 5.18: Permutation importances of the 24 features used to train our final model. Each feature was permuted 10 times.

Table 5.11: Evaluation metrics of the models trained with different numbers of features and the optimal hyperparameters. All 24 features were used to train the first model, and each time, the feature with the lowest importance score, based on Figure 5.18, was removed and the model was trained and evaluated again.

			IoU (%)					
no. of features	feature removed	OA (%)	mean	other	building	ground	CCI (%)	F1 score
24	-	89.72	79.50	79.88	72.98	85.65	99.66	0.90
23	Linearity (2m)	89.73	79.52	79.91	73.01	85.64	99.67	0.90
22	Omnivariance (2m)	89.75	79.57	79.94	73.09	85.67	99.67	0.90
21	Omnivariance (4m)	89.76	79.60	79.94	73.18	85.67	99.67	0.90
20	Anisotropy (2m)	89.79	79.65	80.01	73.26	85.69	99.68	0.90
19	Linearity (3m)	89.78	79.65	80.01	73.27	85.68	99.68	0.90
18	Omnivariance (3m)	89.80	79.69	80.03	73.36	85.69	99.68	0.90
17	Sphericity (2m)	89.81	79.72	80.08	73.40	85.69	99.68	0.90
16	Anisotropy (3m)	89.82	79.73	**80.11**	73.41	85.69	99.68	0.90
15	Sphericity (4m)	**89.84**	79.74	80.08	73.40	**85.75**	99.68	0.90
14	Anisotropy (4m)	89.81	79.72	80.02	73.40	85.73	99.68	0.90
13	Sphericity (3m)	89.83	**79.75**	80.08	**73.44**	85.72	**99.68**	0.90
12	Linearity (4m)	89.78	79.68	80.08	73.34	85.62	99.68	0.90
11	Planarity (2m)	89.74	79.61	79.99	73.26	85.58	99.68	0.90
10	Planarity (3m)	89.64	79.43	79.87	73.00	85.42	99.68	0.90
9	Planarity (4m)	89.26	78.69	79.22	71.79	85.07	99.62	0.89
8	Surface Variation (2m)	89.16	78.49	78.58	71.68	85.20	99.61	0.89
7	Surface Variation (3m)	88.80	77.78	77.79	70.61	84.93	99.56	0.89
6	Surface Variation (4m)	86.71	73.51	73.27	62.90	84.37	98.96	0.86
5	Verticality (3m)	85.93	72.11	72.18	60.62	83.54	98.79	0.86
4	Verticality (2m)	82.87	67.21	67.08	54.30	80.25	98.33	0.86

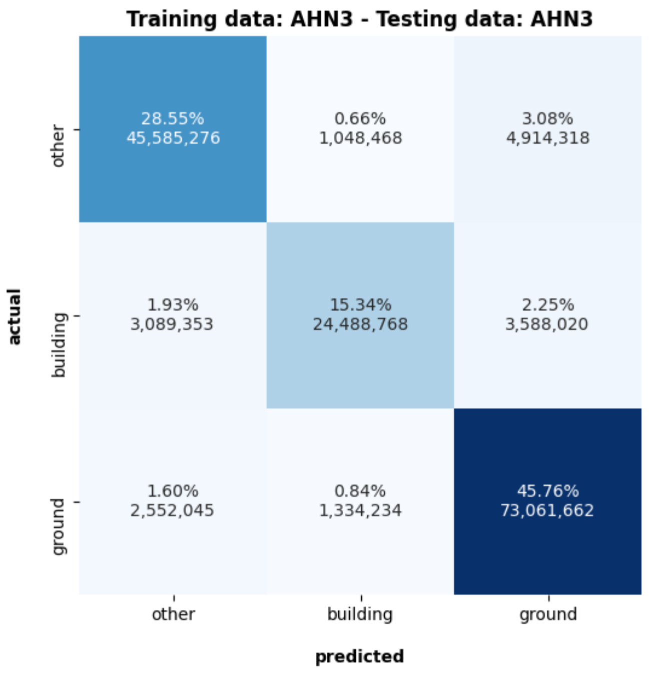
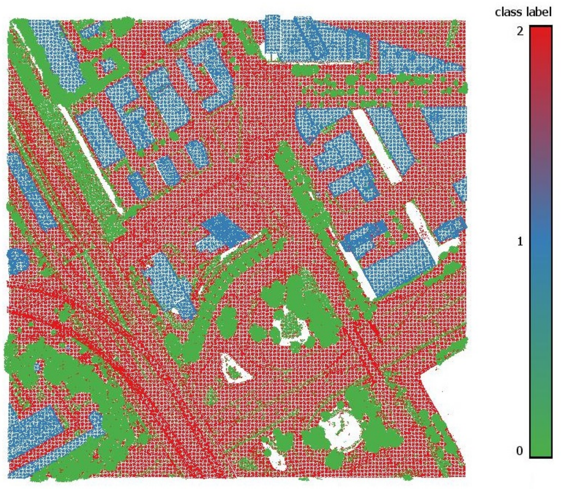


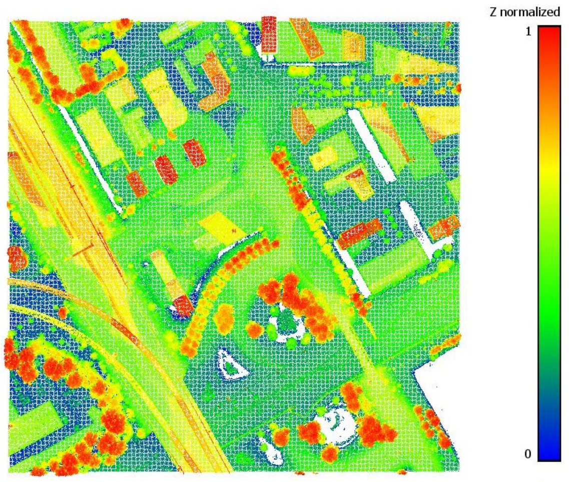
Figure 5.19: Confusion matrix of the classification result of our final Random Forest model.

Table 5.12: Overview of the evaluation metrics of our final model for all AHN3 tiles.

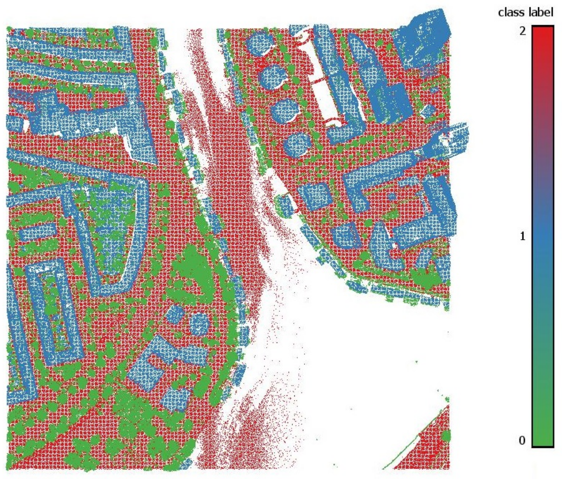
			IoU (%)					
no.	**tile**	**OA (%)**	**mean**	**other**	**building**	**ground**	**CCI (%)**	**F1 score**
1	(7,3)	-	-	-	-	-	-	-
2	(5,5)	-	-	-	-	-	-	-
3	(9,0)	-	-	-	-	-	-	-
4	(7,2)	87.67	75.96	76.19	69.56	82.14	99.65	0.88
5	(5,4)	92.18	81.34	85.52	68.56	89.95	98.96	0.92
6	(3,2)	92.15	61.55	87.35	9.60	87.70	78.07	0.93
7	(9,1)	90.93	81.04	81.78	75.08	86.27	99.74	0.91
8	(8,0)	87.80	56.43	81.16	5.13	82.99	76.67	0.90
9	(9,9)	92.32	85.44	83.87	84.07	88.38	99.95	0.92
10	(8,3)	89.85	76.78	84.82	59.86	85.66	98.14	0.90
11	(1,2)	94.40	87.31	83.91	84.83	93.18	99.80	0.94
12	(7,11)	89.92	78.90	66.31	81.13	89.26	98.86	0.90
13	(4,4)	92.92	74.37	88.92	44.43	89.75	93.97	0.93
14	(0,0)	90.43	78.88	76.89	71.87	87.88	99.43	0.90
15	(6,6)	90.78	80.79	80.46	74.22	87.68	99.63	0.91
16	(9,10)	88.96	79.19	72.12	81.87	83.59	99.68	0.89
17	(7,4)	86.74	74.65	75.94	65.98	82.03	99.41	0.86
18	(0,2)	88.06	75.13	81.17	60.11	84.12	98.48	0.88
19	(6,10)	89.00	79.62	78.76	77.01	83.11	99.92	0.89
20	(9,8)	86.62	71.96	71.34	60.45	84.08	98.70	0.87
21	(8,2)	91.49	78.96	88.04	62.98	85.86	98.37	0.91
22	(4,6)	90.03	81.19	75.79	81.35	86.42	99.77	0.90
23	(7,0)	85.52	55.49	77.27	8.24	80.97	79.84	0.87
24	(6,8)	90.68	82.80	78.94	83.03	86.43	99.89	0.91
25	(9,6)	93.73	84.39	89.47	73.77	89.93	99.33	0.94
26	(9,7)	90.89	82.65	81.35	80.32	86.27	99.92	0.91
27	(3,5)	90.16	81.13	72.31	84.67	86.40	99.52	0.90
28	(6,4)	84.33	67.92	67.72	54.20	81.85	98.12	0.84
29	(5,8)	89.30	79.41	70.02	82.66	85.55	99.43	0.89
30	(8,1)	81.56	61.39	72.92	38.34	72.92	95.67	0.82
31	(5,9)	90.97	83.15	78.67	83.33	87.44	99.85	0.91
32	(5,7)	89.30	79.50	74.22	77.20	87.10	99.62	0.89
33	(7,6)	90.18	68.58	87.25	33.76	84.73	91.14	0.90
34	(1,3)	91.03	80.41	69.43	83.49	88.32	99.20	0.91
35	(8,9)	91.96	83.75	74.10	87.89	89.25	99.44	0.92
36	(3,4)	91.95	81.61	86.28	71.13	87.43	99.32	0.92
37	(8,6)	88.40	76.06	83.13	62.50	82.54	98.79	0.88
38	(4,9)	85.56	73.40	66.07	72.73	81.42	99.46	0.86
39	(4,5)	89.66	80.72	76.24	80.17	85.76	99.81	0.90
40	(5,10)	89.92	81.06	81.10	76.52	85.54	99.83	0.90
		89.82	**79.75**	**80.09**	**73.47**	**85.69**	**99.69**	**0.90**



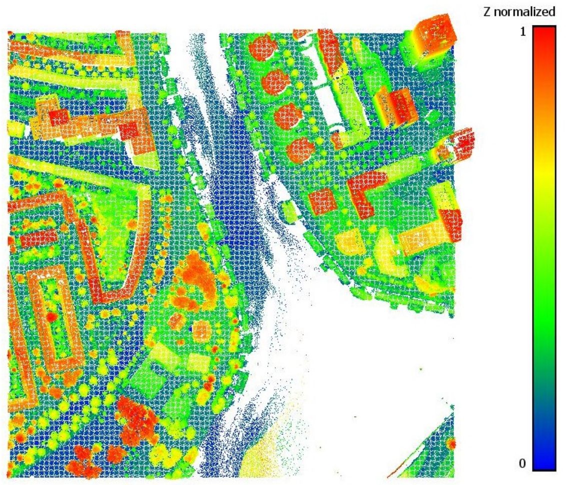
(a) tile (7,3) - class labels



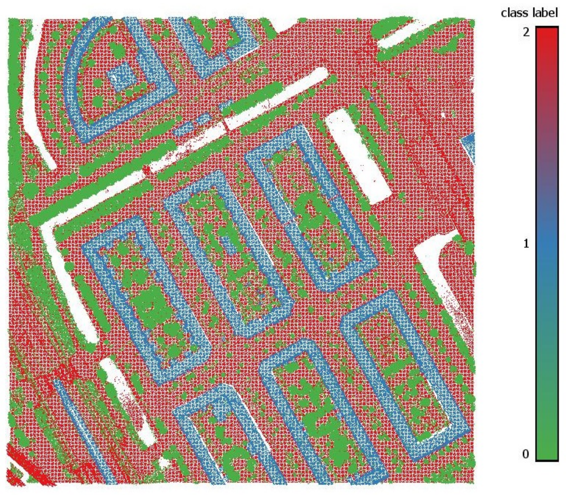
(b) tile (7,3) - $Z_{normalized}$



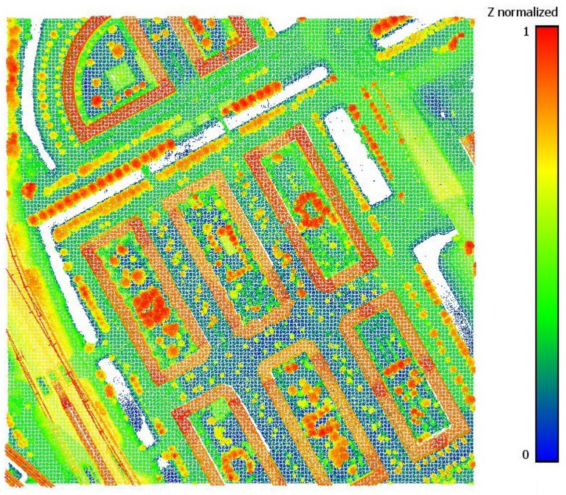
(c) tile (5,5) - class labels



(d) tile (5,5) - $Z_{normalized}$

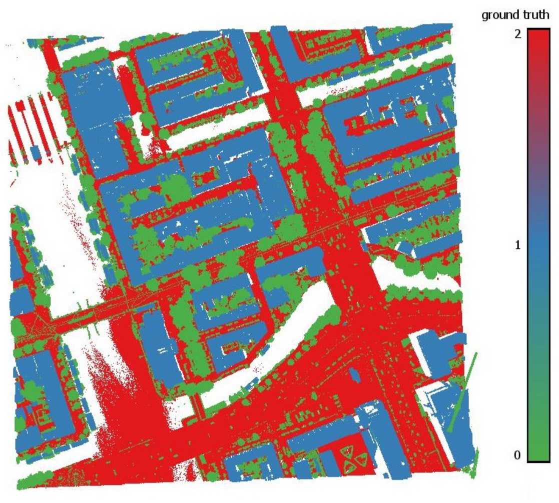


(e) tile (9,0) - class labels

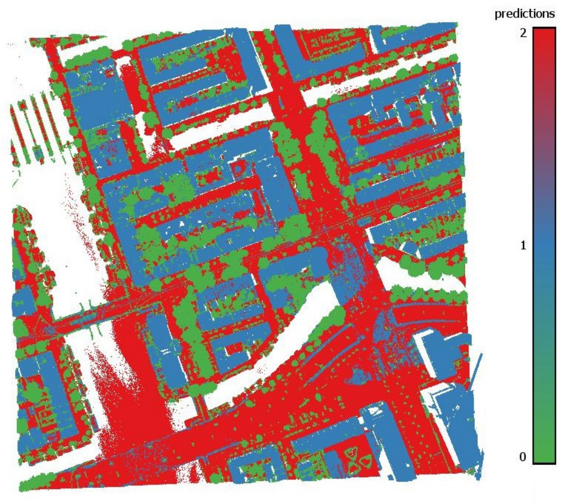


(f) tile (9,0) - $Z_{normalized}$

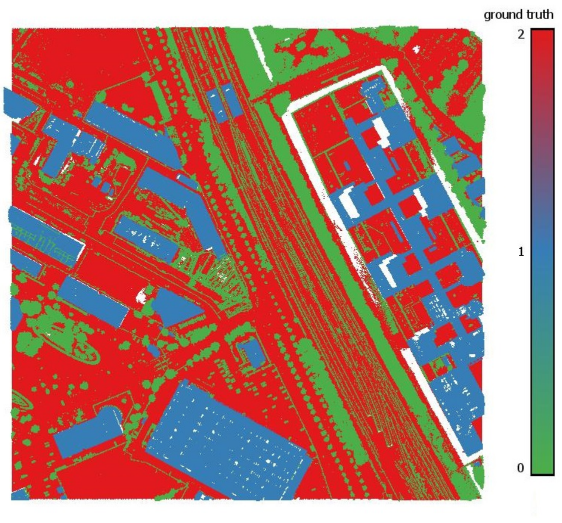
Figure 5.20: Visualization of the class labels and $Z_{normalized}$ features of the 3 best tiles used as training data for our final model.



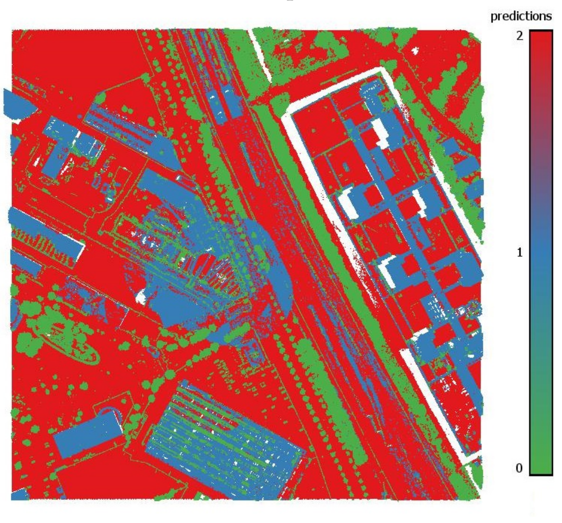
(a) tile (4,9) ground truth



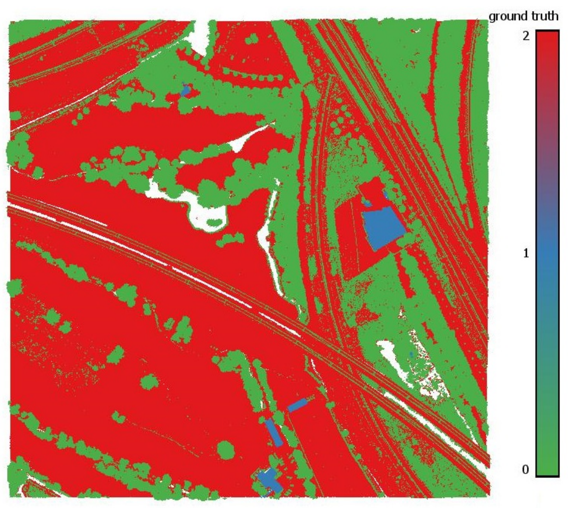
(b) tile (4,9) predictions



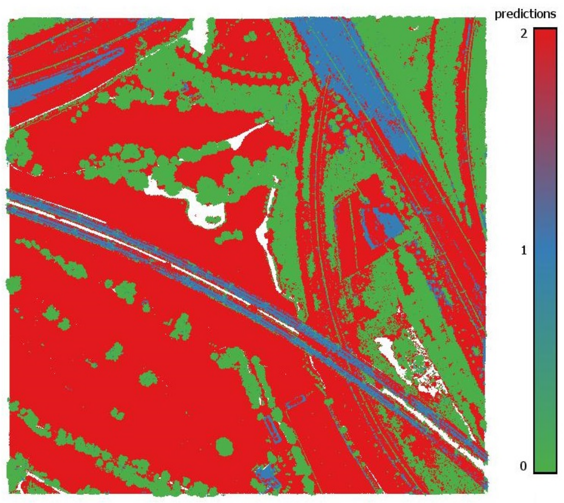
(c) tile (6,4) ground truth



(d) tile (6,4) predictions



(e) tile (8,0) ground truth



(f) tile (8,0) predictions

Figure 5.21: Visualization of the ground truth values and predictions for some of the tiles with the lowest evaluation metrics.

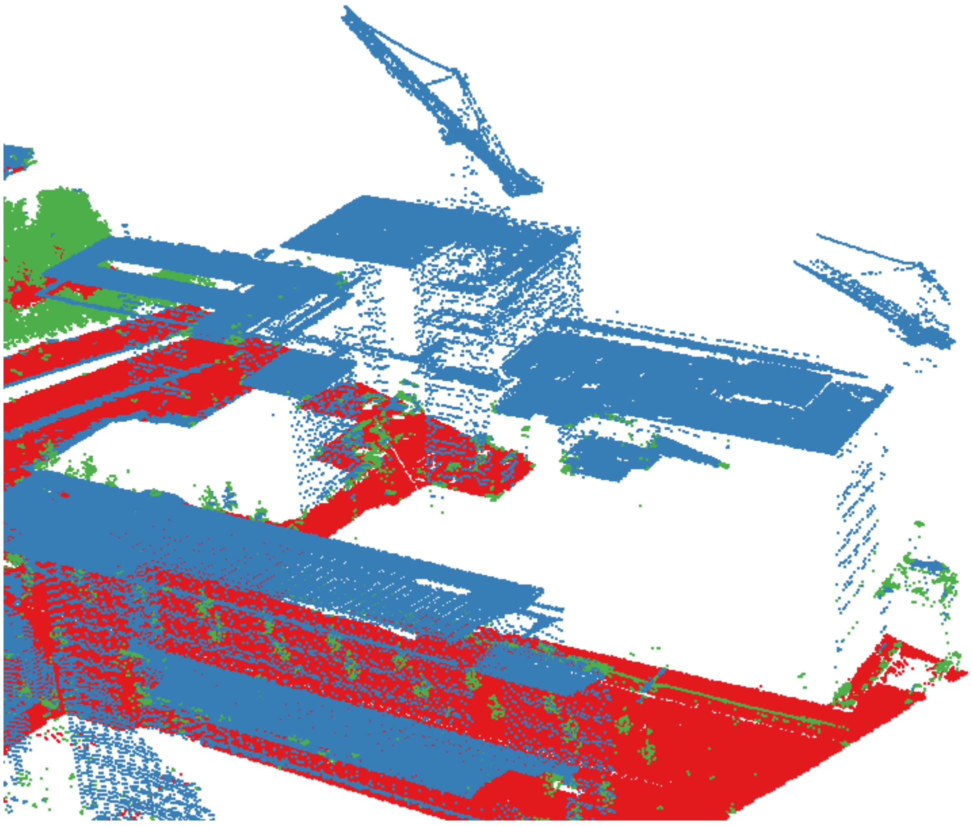


Figure 5.22: Zoom in at predictions of tile (4,9).

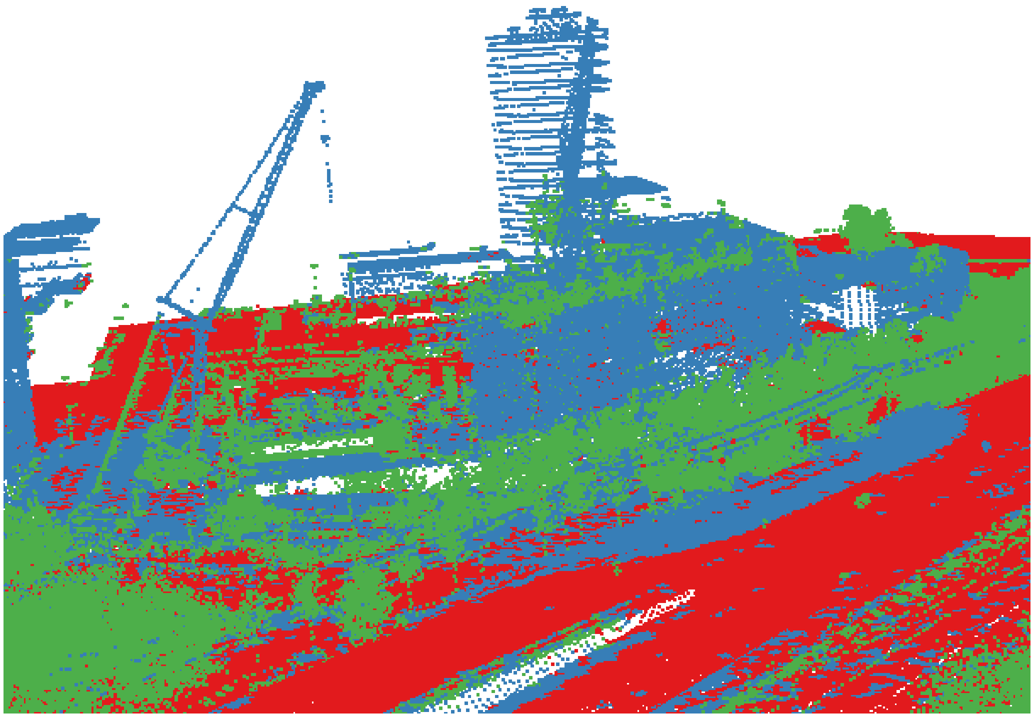


Figure 5.23: Zoom in at predictions of tile (6,4).

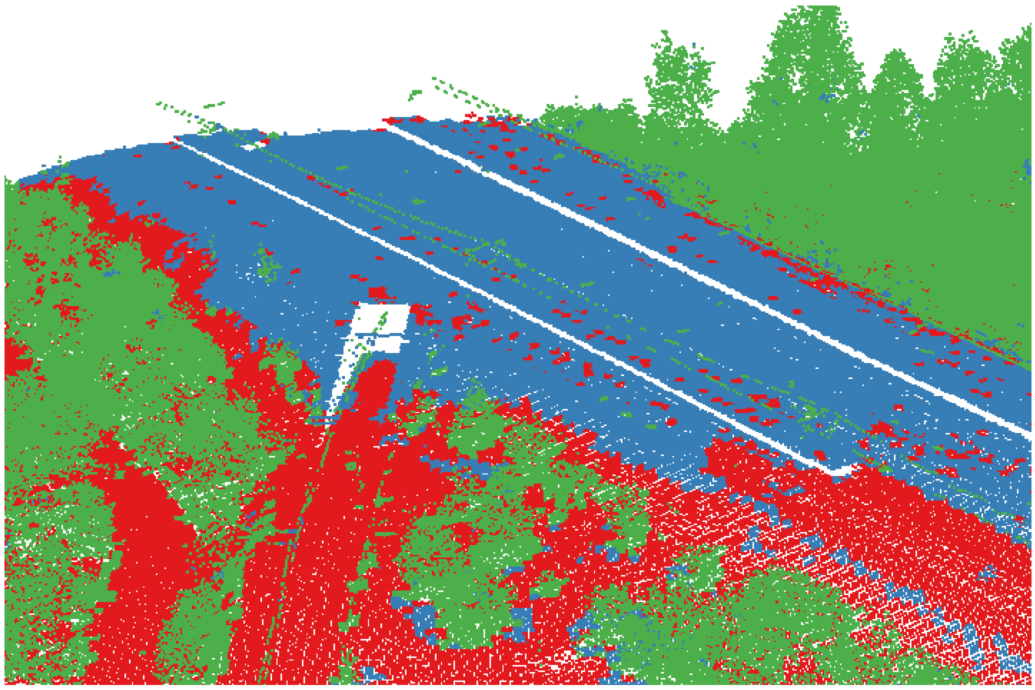
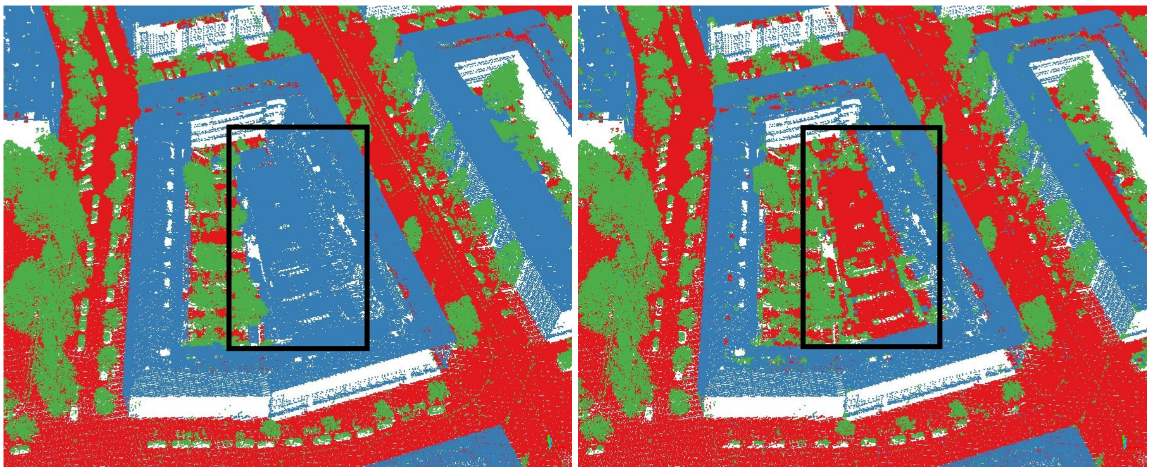
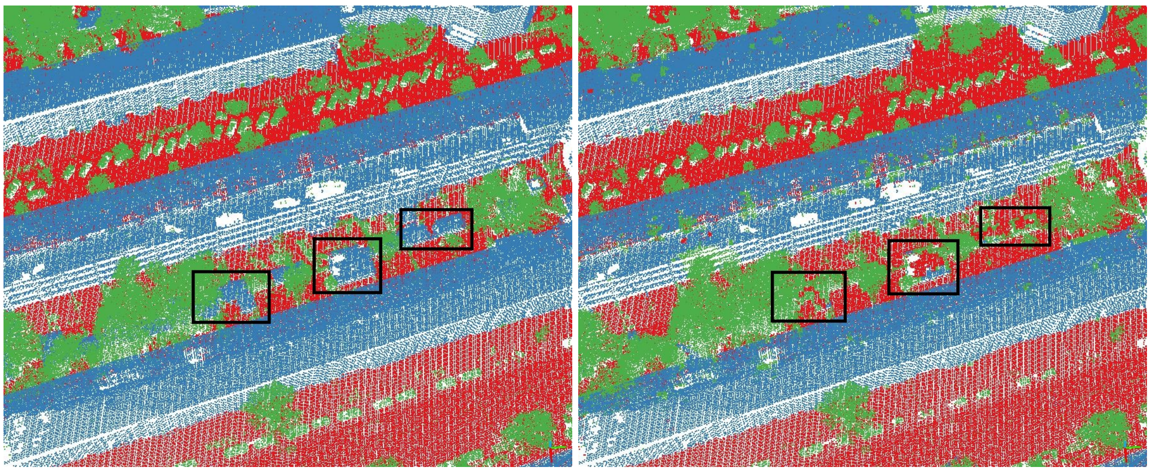


Figure 5.24: Zoom in at predictions of tile (8,0).

The problematic areas of the tiles (4,9), (6,4) and (8,0) that are shown in the figures 5.22, 5.23 and 5.24 seem to be caused by the wrong values of the height features of their points. The heigh features are the most important features for our model, and they influence its performance a lot. We observe that tall objects like cranes are misclassified as buildings. The model is not able to distinguish between them. Not only that, they influence the height features of their surrounding points as well. For example, in tile (4,9) and (6,4), cranes are misclassified as buildings. Moreover, a circle of misclassified ground points as buildings is visible around the crane in tile (6,4). It is clear that this has to do something with the cylindrical neighbourhood of 50 meters used to extract the heigh features of the points. We suspect that because points with very high Z values are close to those ground points, their Z features have higher values than the other ground points around them. Thus, the model decides to classify them as buildings since this is the main characteristic that distinguishes the ground from the buildings. As for the tile (8,0), we see that the ground changes to higher altitude sharply, and for similar reasons, some ground points are misclassified as buildings. Using a higher radius for our cylindrical neighbourhoods or additional features and training data that would help the model distinguish buildings from the ground better could probably yield better results for such problematic areas.

Figure 5.25 presents three more cases where buildings were misclassified as ground and other. As far as the problems related to the other class are concerned, the objects that mostly cause errors to the predictions of our model are trees. In case 1, we see that very short parts of buildings are misclassified as ground. In case 2, we observe how very small buildings that are very close to trees are misclassified as other. Finally, the problem shown in case 3 seems to be related to the height, surface variation and planarity values of the points. The flat roof of a building is misclassified as ground even though its corners are correctly classified as building.


(a) Case 1: ground truth (left) and predicted (right) labels. Low parts of a tall building misclassified as ground and other.


(b) Case 2: ground truth (left) and predicted (right) labels. Small building parts under vegetation, that are separate from the main tall building part, misclassified as ground and other.

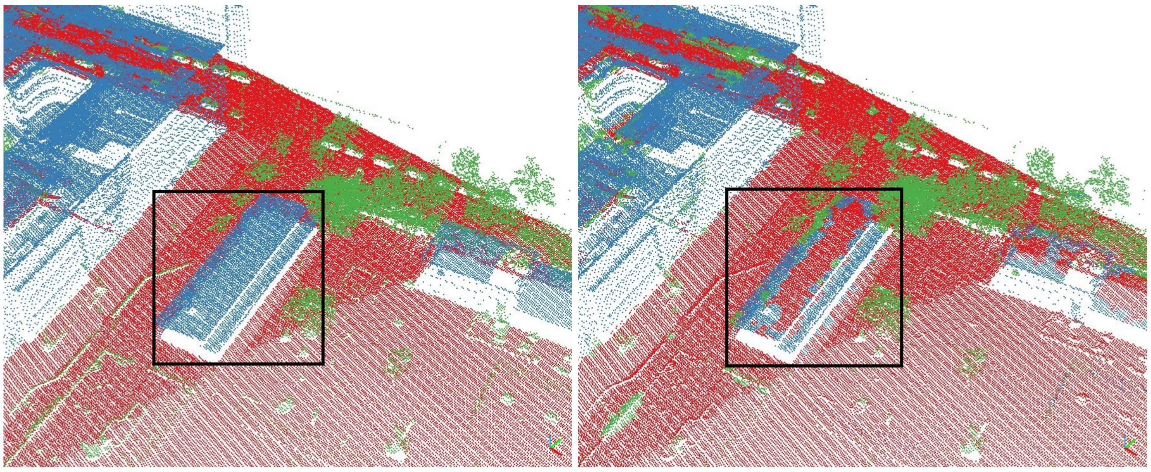

(c) Case 3: ground truth (left) and predicted (right) labels. Flat roof surface of a building misclassified as ground even though the edges of the building are correctly classified.

Figure 5.25: Problems related to classifying buildings.

5.9 Comparison with Deep Learning algorithms on the DALES dataset

To compare the performance of our proposed methodology with other deep learning models, we trained and tested a Random Forest model with the DALES dataset using all the available classes. The comparison was made based on the results provided in the work of Varney et al. [2020]. Table 5.13 provides the OA, mean IoU, per class IoU and CCI for the KPConv [Thomas et al., 2019], PointNet++ [Qi et al., 2017b], ConvPoint [Boulch, 2020], SuperPoint [Landrieu and Simonovsky, 2018], PointCNN [Li et al., 2018] and ShellNet [Zhang et al., 2019] deep learning algorithms and for our simpler Random Forest machine learning algorithm. The same hyperparameters, features and voxel size for our uniform sampling algorithm were used with our final model. Figure 5.26 provides an Algorithm Performance Map across the six tested benchmark deep learning algorithms we previously mentioned and our Random Forest Classifier.

Table 5.13: Overview of different deep learning methods on the DALES dataset as presented by Varney et al. [2020] compared to our Random Forest Classifier. The overall accuracy, mean IoU, per class IoU and CCI for each category are reported. KPConv outperforms all other methods on the DALES dataset. Our proposed method has the lowest mean IoU score.

		IoU									
Method	OA	mean	ground	buildings	cars	trucks	poles	power lines	fences	vegetation	CCI
KPConv	**0.978**	**0.811**	0.971	**0.966**	**0.853**	**0.419**	**0.75**	**0.955**	**0.635**	**0.941**	**0.951**
PointNet++	0.957	0.683	0.941	0.891	0.754	0.303	0.4	0.799	0.462	0.912	0.905
ConvPoint	0.972	0.674	0.969	0.963	0.755	0.217	0.403	0.867	0.296	0.919	0.852
SuperPoint	0.955	0.606	0.947	0.934	0.629	0.187	0.285	0.652	0.336	0.879	0.846
PointCNN	0.972	0.584	**0.975**	0.957	0.406	0.048	0.576	0.267	0.526	0.917	0.798
ShellNet	0.964	0.574	0.96	0.954	0.322	0.396	0.2	0.274	0.6	0.884	0.822
RFC	0.890	0.451	0.860	0.720	0.059	0.000	0.202	0.817	0.149	0.799	0.721

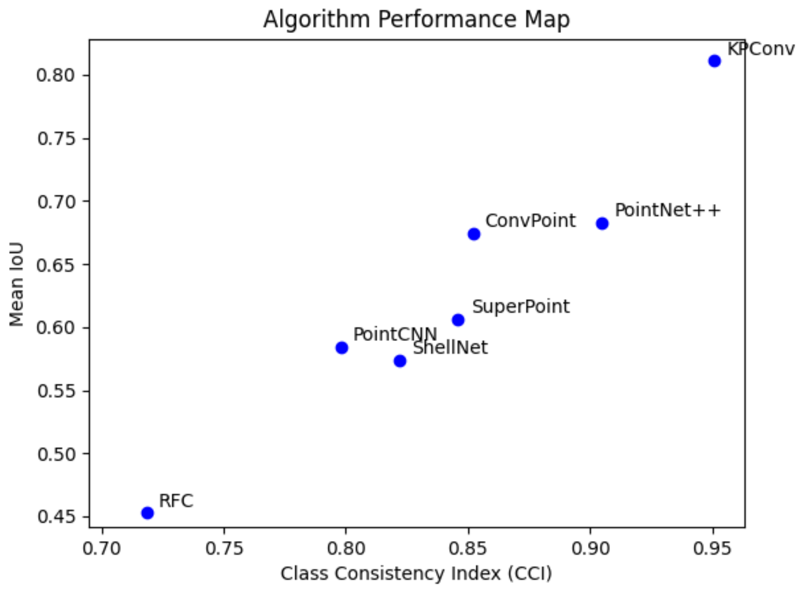


Figure 5.26: Algorithm Performance Map across six tested benchmark algorithms and our Random Forest Classifier. Axis CCI represents the uniformity of the algorithms while axis Mean IoU their correctness.

The results show that deep learning methods perform better than our model. The only benefit of our approach is that it is simpler and easier to understand, and possibly more efficient. It required 14.5 GB of RAM to train the model, and the training process lasted 121.84 minutes. The final model has a size of 2,278,256 KB. It is also worth noting that the difference is not that significant for some of the classes compared to some of the deep learning methods. In fact, our model performed better than ShellNet, PoinCNN, SuperPoint, and PointNet++ for the power lines. In general, the IoU for all classes besides ground, building and vegetation is very low. Our model was unable to distinguish mostly trucks, cars and fences. The difference in OA is less than 10% compared to KPConv that outperformed all the other algorithms. However, the mean IoU is a lot less, almost half of KPConv.

5.10 Additional experiments with the MLP algorithm on the AHN3 dataset

The scikit-learn Python library [Pedregosa et al., 2011] provides modules that include multiple other machine learning and deep learning algorithms for classification tasks, such as the MLP supervised learning algorithm. Adjusting our code to use the MLP algorithm instead of the Random Forest classifier was very easy and simple, and thus, some additional smaller experiments were made to test its performance on the AHN3 dataset. The same features and training and testing data were used for these experiments as the ones used for our final Random Forest model (see section 5.8). As far as the hyperparameters of the MLP algorithm are concerned, we experimented with seven of them and tried to identify their optimal values in order to improve the performance of the algorithm. These seven hyperparameters are the learning rate schedule for weight updates, the solver for weight optimization, the activation function for the hidden layers, the maximum number of iterations the solver can do if convergence is not reached, the number of neurons in the hidden layers, the size of minibatches for stochastic optimizers and the number of hidden layers. The following figures present the results of the grid search that was conducted on these seven hyperparameters. Multiple models were trained and tested for each hyperparameter, changing their values each time while using the default values for the rest as indicated in Table 5.14.

Table 5.14: Default values of the hyperparameters of the MLP algorithm.

Hyperparameter	**Value**
learning_rate	constant
solver	adam
activation	relu
max_iter	200
hidden_layer_sizes	100
batch_size	auto
no. of hidden layers	1

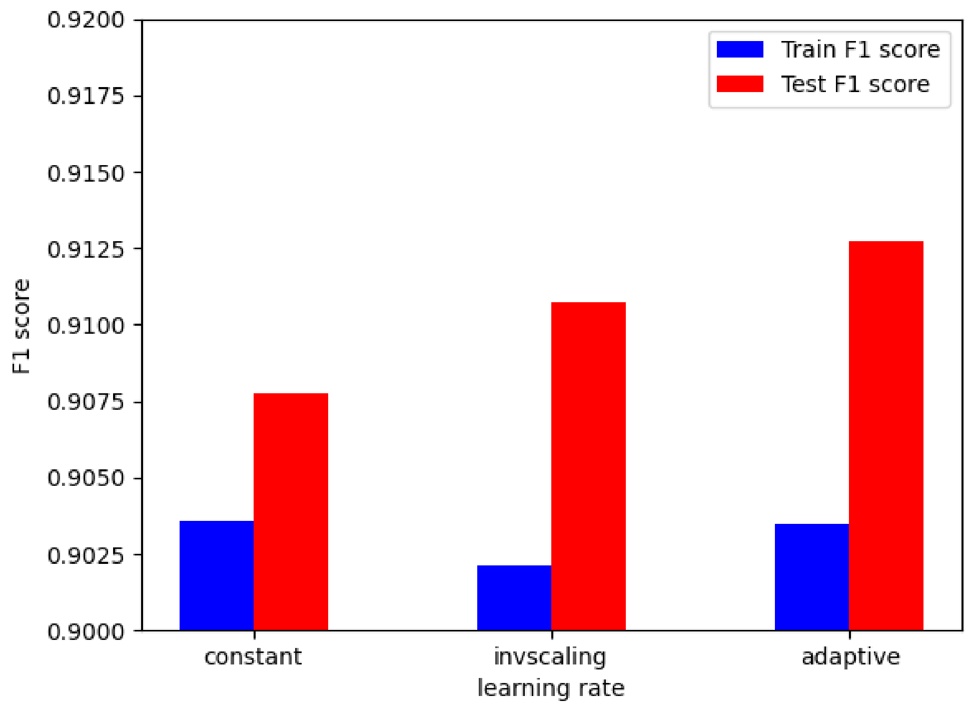


Figure 5.27: F_1 score vs the learning rate specification.

Figure 5.27 shows that the model performs better when an adaptive learning rate is used. In this case, adaptive means that if the training loss does not decrease, the learning rate is reduced.

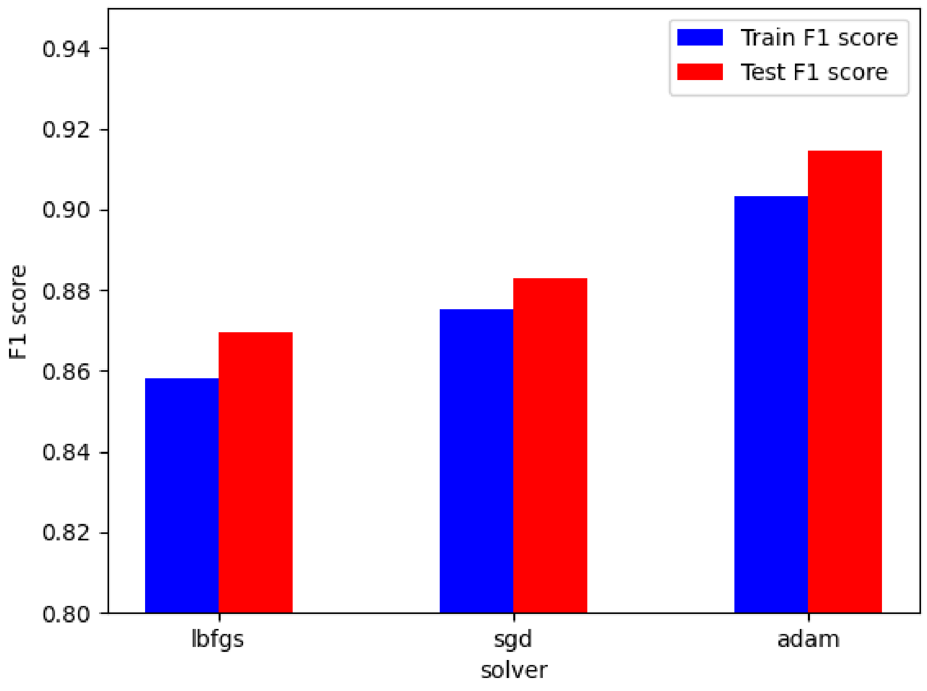


Figure 5.28: F_1 score vs the solver function.

Figure 5.28 shows that the model performs better when adam is used as the solver for weight optimization. Adam refers to a stochastic gradient-based optimizer proposed by Kingma and Ba [2014].

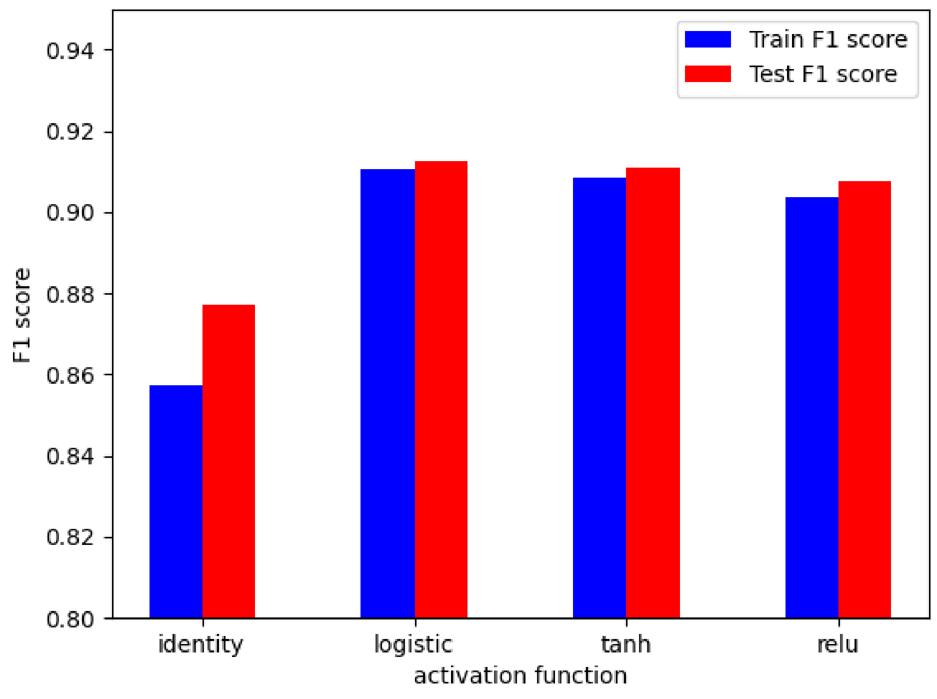


Figure 5.29: F_1 score vs the activation function.

Figure 5.29 shows that the model performs better when the logistic sigmoid activation function is used that returns $f(x) = 1/(1 + exp(-x))$.

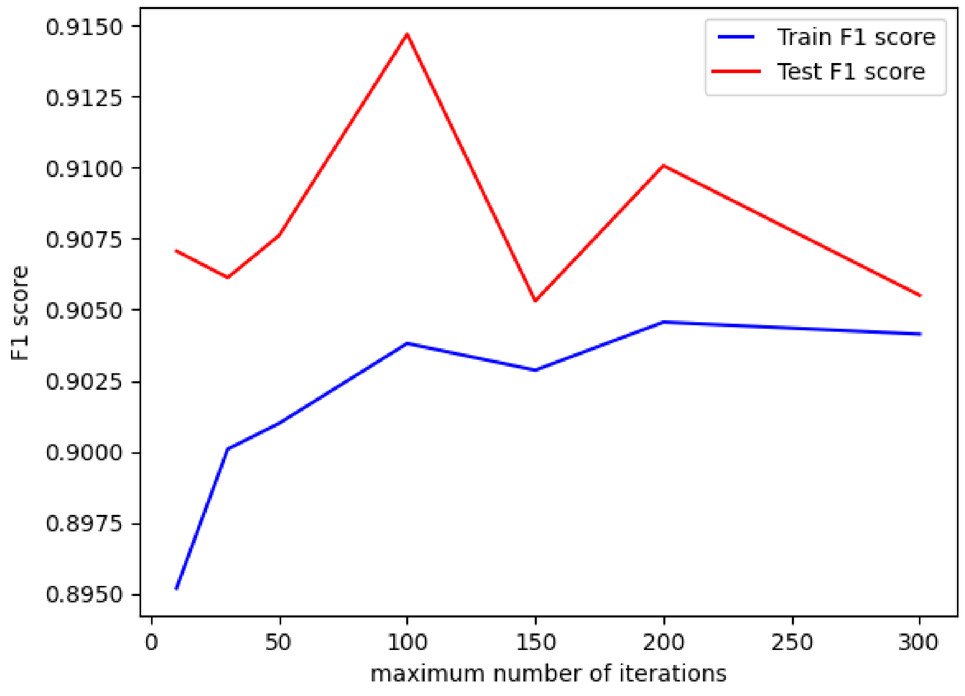


Figure 5.30: F_1 score vs the maximum number of iterations.

In Figure 5.30 we observe that the model performs better when the maximum number of iterations the solver iterates if convergence is not reached, is 100.

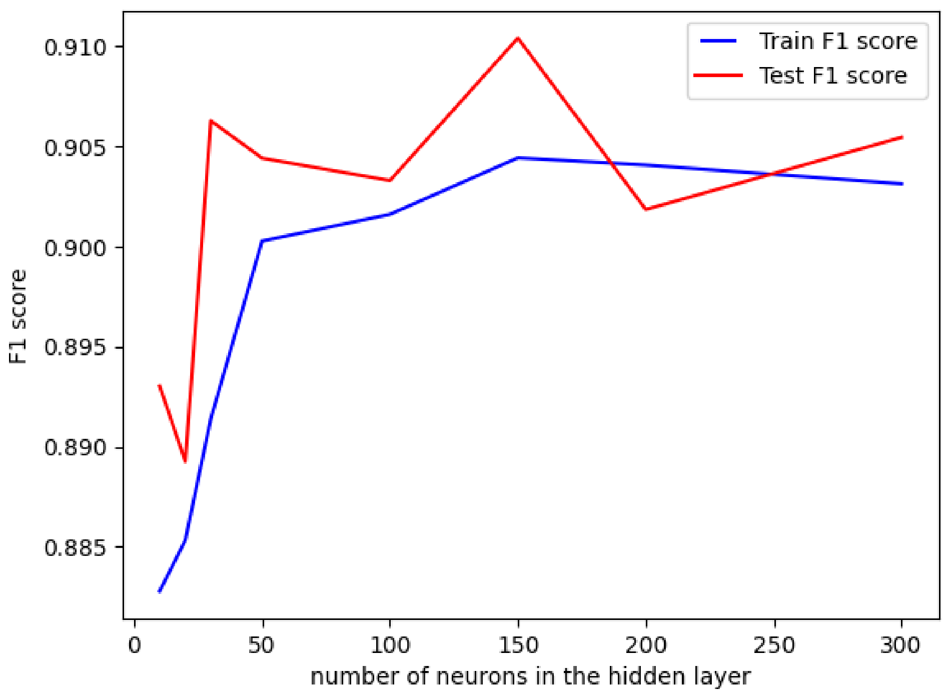


Figure 5.31: F_1 score vs the number of neurons in the hidden layer.

In Figure 5.31 we observe that the model performs better when the number of neurons in the hidden layer is 150.

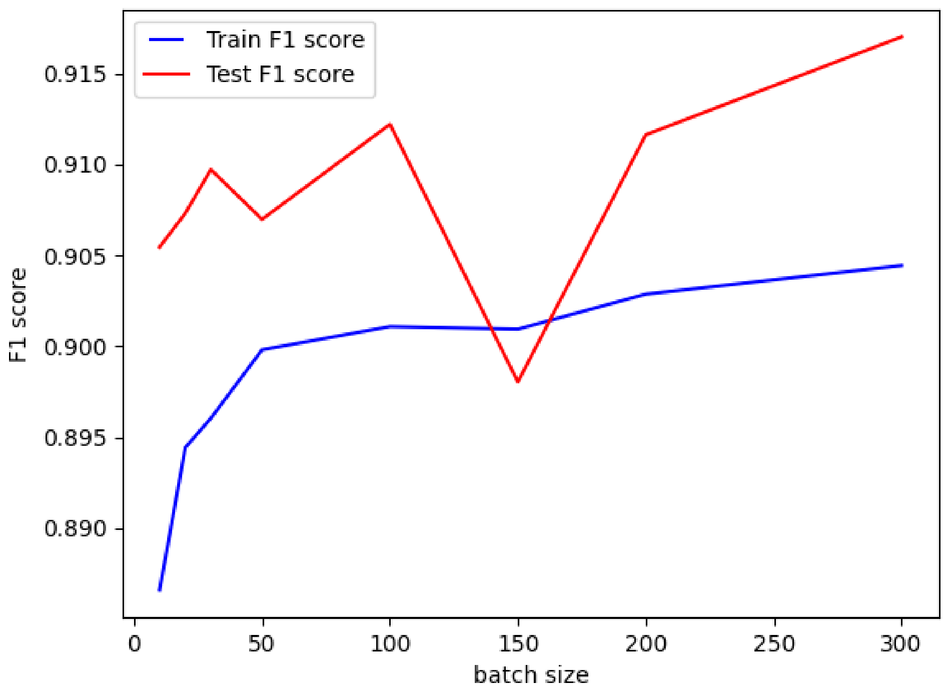


Figure 5.32: F_1 score vs the batch size.

In Figure 5.32 we observe that the optimal batch size is 100.

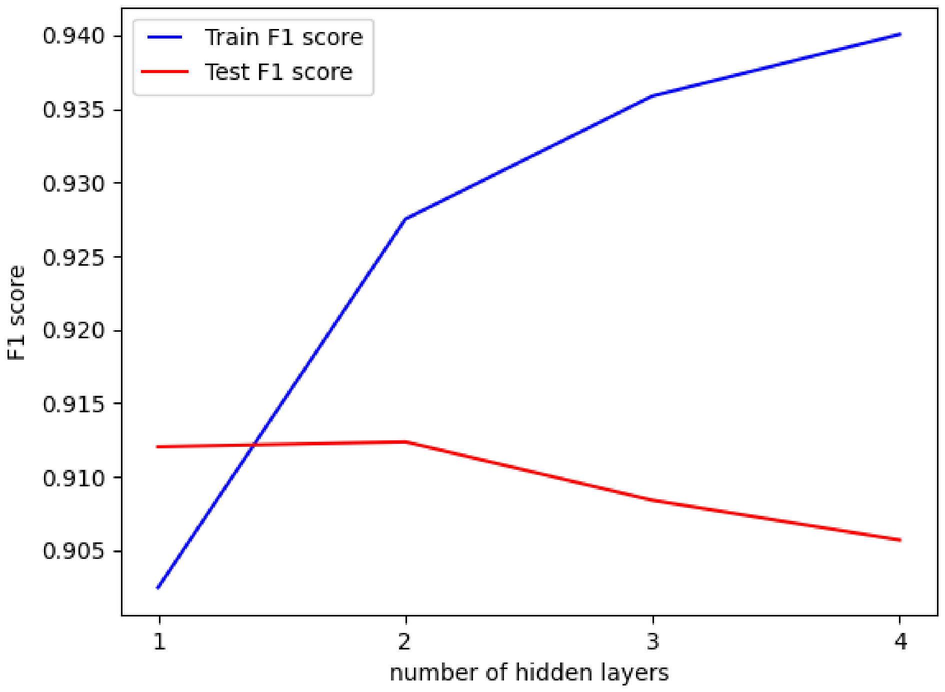


Figure 5.33: F_1 score vs the number of hidden layers in the MLP.

In Figure 5.33 we observe that the optimal number of hidden layers to have is 2. Having more than that leads to overfitting. The F_1 score of training data increases considerably while the F_1 score of the testing data starts to decrease.

Table 5.15: Overview of the Random Forest classifier and MLP on the AHN3 dataset.

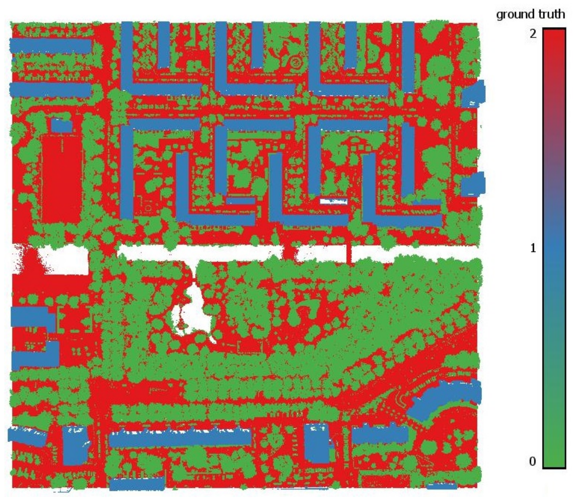
		IoU (%)								
method	**OA (%)**	**mean**	**other**	**building**	**ground**	**CCI (%)**	**F1 score**	**training time (minutes)**	**RAM (GB)**	**model size (KB)**
RFC	89.82	79.75	80.09	73.47	85.69	99.69	0.90	12.45	1.1	229,494
MLP	88.25	76.79	79.58	67.70	83.10	99.44	0.88	72.26	0.79	598

Table 5.15 presents the evaluation metrics of the final Random Forest and MLP models when trained and tested with the best 40 AHN3 tiles and with the optimal hyperparameter values. The results provided in Table 5.15 indicate that the Random Forest classifier performed slightly better than the MLP algorithm. The per-class IoU is similar for both methods. The IoU of the ground class was the highest, followed by the IoU of the other class and finally the IoU of the building class. What is interesting is the difference between their computation costs and time and the size of their resulting trained models. The training time of the Random Forest classifier was significantly lower than the MLP. However, the MLP algorithm required less GB of RAM to train the model, and the size of its resulting trained model is much smaller. Overall, the Random Forest classifier performed better and was more efficient in terms of computation time, while the MLP was more efficient in terms of computation cost.

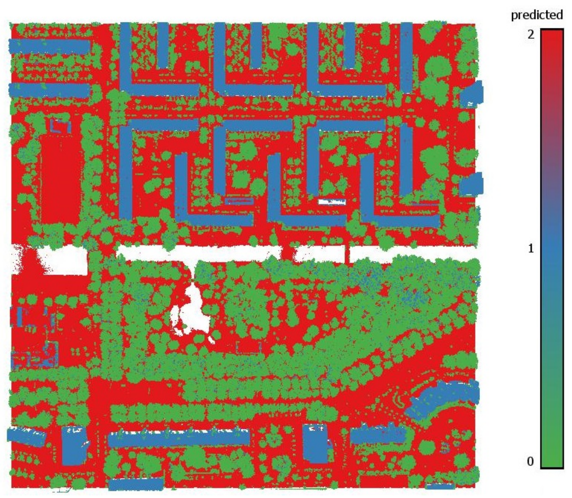
5.11 Potential application

Our final model has classified some of our testing tiles very accurately. Tile (1,2) is one of those tiles with an F_1 score of 0.94 (see section 5.8 Table 5.12 no. 11). In this section, we illustrate the LOD1 3D city models that were created for tile (1,2) (Figure 5.34) using both the ground truth and predicted class labels of its points, and we compare them. To create the models of the buildings, their polygon footprints were used. They were downloaded via a WFS service from the web service of 3D BAG[1] by the 3D geoinformation research group of TU Delft. Polygons that intersect the boundaries of the tile were not used. The ground surface was created using only the ground points from the classified point cloud. A uniform sampling algorithm was applied on the first with a voxel size of 5 meters. Table 5.16 provides an overview for the roof and ground surface heights that were calculated from the ground truth and predicted classified point clouds for 40 buildings found in tile (1,2) as well as for those extracted from the polygons derived from the web service of 3D BAG.

Looking at the height values of Table 5.16 we see that the values that we calculated for the roof surfaces of the buildings from our predicted classified point cloud are, in most cases, almost the same as the other two. As for the heights calculated for the ground surfaces of the buildings, we notice a few centimetres difference between the three values. This is also because we used a different method to extract those heights. In our case, we considered ground points as well because, in many cases, our model classifies low building facade points as ground. Consequently, considering only the building points resulted into very high ground surface values. Finally, looking at Figures 5.34a and 5.34b we see that the misclassified ground points create a lot of unwanted spikes on our ground surface.



(a) tile (1,2) - ground truth



(b) tile (1,2) - predicted

Figure 5.34: Visualization of the ground truth and predicted class labels of the testing tile (1,2).

[1] https://3dbag.nl/en/viewer

Table 5.16: Comparison of the ground truth and predicted building roof and ground surface heights derived from the AHN3 point cloud with the heights stored in the attributes of the polygons derived from the web service of 3D BAG by the 3D geoinformation research group of TU Delft. h_dak_50p and h_dak_min are elevations above sea level (NAP) at roof level, calculated as the median and minimum of all elevation points on the corresponding roof part. Ground truth and predicted building heights were calculated as the median of all building elevation points inside the corresponding polygon. Ground truth and predicted ground heights were calculated as the 10th percentile of all ground elevation points inside the buffered area by 0.5 meters of the corresponding polygon.

no.	**building roof surface**				**building ground surface**		
	ground truth	**predicted**	**h_dak_50p**		**ground truth**	**predicted**	**h_dak_min**
1	10.782	10.783	10.784		-0.654	-0.661	-0.640
2	10.744	10.744	10.742		-0.798	-0.796	-0.822
3	10.813	10.813	10.816		-0.757	-0.762	-0.735
4	10.888	10.887	10.888		-0.729	-0.697	-0.728
5	1.854	1.851	1.858		-0.752	-0.701	-0.817
6	24.627	24.636	24.632		-0.780	-0.739	-0.788
7	24.738	24.738	24.745		-0.670	-0.656	-0.680
8	1.744	1.745	1.743		-0.823	-0.807	1.681
9	10.813	10.813	10.820		-0.771	-0.774	-0.789
10	1.824	1.821	1.801		-0.714	-0.713	-0.700
11	10.869	10.868	10.871		-0.905	-0.899	-0.898
12	42.518	43.919	44.106		-0.915	-0.913	-0.964
13	1.861	1.854	1.862		-0.693	-0.672	1.798
14	10.895	10.895	10.894		-0.729	-0.731	-0.741
15	1.846	1.840	1.845		-0.697	-0.636	-0.675
16	1.809	1.814	1.798		-0.711	-0.711	-0.626
17	10.866	10.866	10.868		-0.621	-0.618	-0.610
18	10.747	10.747	10.748		-0.690	-0.685	-0.655
19	13.686	13.683	13.691		-0.581	-0.578	-0.550
20	1.845	1.842	1.846		-0.684	-0.622	-0.664
21	24.609	24.612	24.614		-0.853	-0.786	-0.823
22	10.808	10.807	10.807		-0.868	-0.868	-0.955
23	45.030	45.033	45.048		-0.634	-0.634	-0.633
24	10.746	10.746	10.748		-0.675	-0.651	-0.645
25	10.909	10.909	10.908		-0.667	-0.641	-0.669
26	1.401	1.525	1.408		-0.788	-0.719	-0.843
27	10.918	10.917	10.918		-0.809	-0.809	-0.743
28	1.849	1.885	1.820		-0.714	-0.707	-0.707
29	1.733	1.733	1.733		-0.888	-0.876	-0.865
30	10.889	10.889	10.893		-0.705	-0.674	-0.779
31	45.053	45.065	45.054		-0.630	-0.630	-0.584
32	10.888	10.887	10.884		-0.591	-0.589	10.793
33	10.818	10.818	10.818		-0.898	-0.887	-0.906
34	10.768	10.768	10.768		-0.799	-0.795	-0.839
35	1.745	1.748	1.745		-0.818	-0.800	-0.785
36	10.779	10.779	10.782		-0.713	-0.714	-0.716
37	1.746	1.746	1.748		-0.821	-0.797	1.685
38	10.764	10.765	10.767		-0.682	-0.679	-0.680
39	10.808	10.808	10.808		-0.906	-0.894	-0.877
40	1.842	1.846	1.843		-0.686	-0.642	-0.616

Figures 5.34a and 5.34b present the resulting LOD1 3D city models. The two models were stored in CityJSON format and their geometry structure was validated using the val3dity[2] and the "validate" function of the cjio[3] Python package.

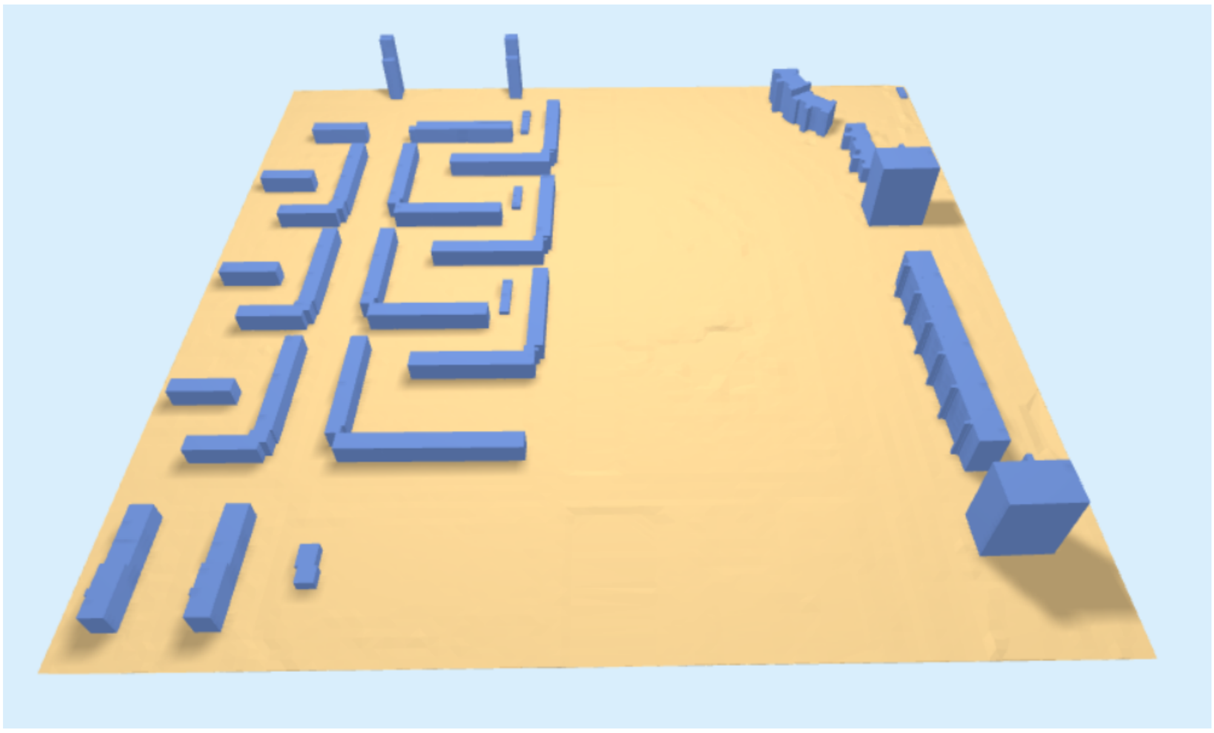


Figure 5.35: LOD1 3D city model derived from the ground truth classified point cloud.

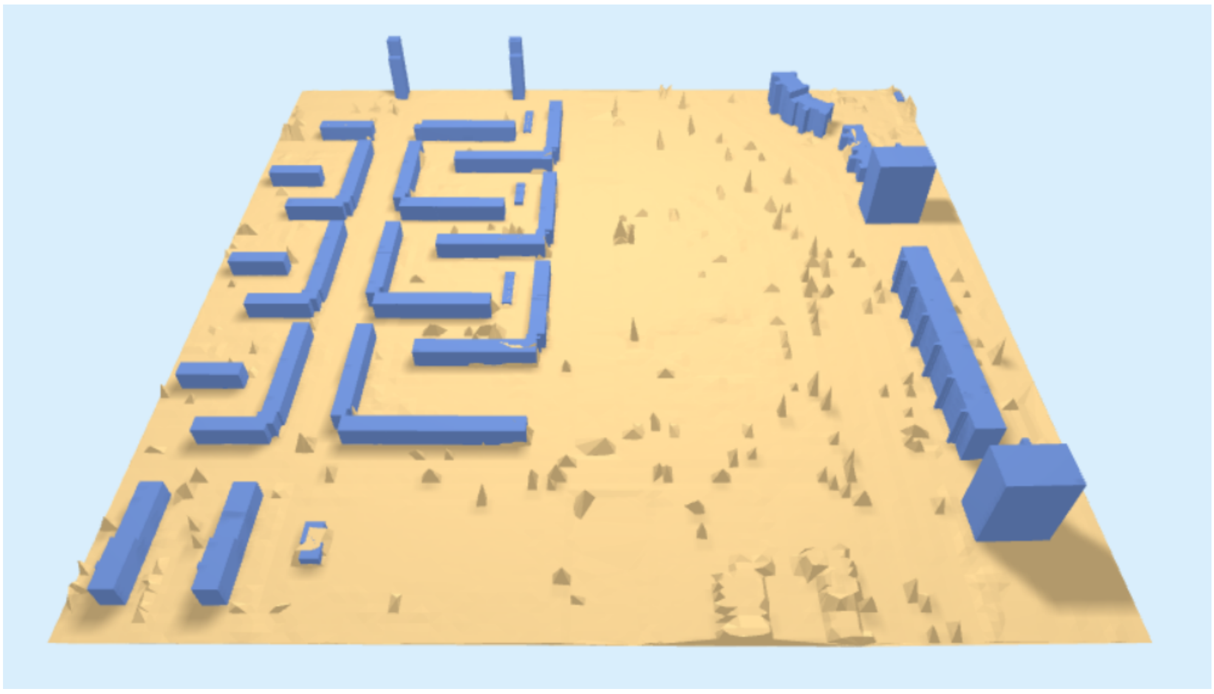


Figure 5.36: LOD1 3D city model derived from the predicted classified point cloud.

[2]https://github.com/tudelft3d/val3dity
[3]https://github.com/cityjson/cjio

6 Conclusions

In this final chapter, we review the proposed methodology and the research questions of this graduation project to assess the degree in which they have been addressed and make some conclusions. We also recommend some future work with respect to the limitations of our proposed methodology and discuss a potential application of our work.

6.1 Research overview

This thesis study proposes an automatic approach to accurately assign a class label to the 3D points of point cloud datasets. With our proposed methodology, we managed to train a Random Forest classification model that classifies points into three classes, namely other, building and ground, and that achieved a mean IoU of 0.79 and an F_1 score of 0.90 when train and tested with the AHN3 dataset. Our model performed worse on the DALES dataset than other deep learning models we mentioned in the previous chapter. However, its results are comparable, considering its simplicity. Moreover, in some cases, the IoU for the poles and power lines classes was higher than for some of the deep learning methods that we compared our model with. Finally, our proposed methodology mostly focused on separating building, ground and other (mostly vegetation) points, and for those classes, it performed decently. The python scripts of this project can be found in a GitHub repository[1].

In Chapter 1: Introduction, six research questions were stated. For each question, a short answer is provided, based on the degree the question was finally addressed, and they are supported by evidence, which has been presented in previous chapters.

How well will existing machine learning algorithms perform when classifying a point cloud into three classes, namely ground, buildings and other, if we train and test them with the AHN3 dataset?

As we have seen from our experiments, with the proper point cloud analysis, the right parameters, handcrafted features and training data as described in our proposed methodology, an existing machine learning algorithm like the Random Forest Classifier can achieve an F_1 score of 0.9 when trained and tested with the AHN3 dataset. However, our method suffers from some limitations. In our initial experiments, our method could not distinguish water points and bridges from the ground. Thus, we decided to include them in the ground class. Moving forward, additional features may be needed to help the algorithm distinguish better between these classes of points.

What features of the 3D points should be included in the training and testing processes of the machine learning algorithms to improve their performance?

To achieve models with great performance, one should definitely include features related to the Z coordinate of the 3D points. Both height features that we used in the research were proven important. Including both of them in our final model increased its accuracy. The eigen-based features we mention in this thesis can provide important discriminating information about points if calculated for multi-scale representations. From our experiments, we found that the most important ones were surface variation, anisotropy and verticality.

How accurately will the trained models perform when tested with datasets other than the AHN3 dataset?

The results of our experiments show that if models are tested with datasets that represent environments other than the ones they were trained with, their accuracy will decrease. In our experiments, three different datasets were tested, the AHN3, AHN4 and DALES datasets. Since the AHN3 and AHN4 training

[1] https://github.com/MPa-TUDelft/Semantic-segmentation-of-the-AHN-dataset-with-the-Random-Forest-Classifier

and testing tiles we used are very similar, the models we trained with one of these datasets did not underperform when tested with the other. Their evaluation metrics were almost the same. When tested with a different dataset like the DALES dataset, however, we see from the results of our experiments that their accuracy dropped a bit. The quality of the training data plays an important role in the stability of the model. For a model to provide good results, it needs to be applied to a dataset of similar characteristics.

How large does the training dataset needs to be for the models to perform well?

Using a very large training dataset does not necessarily mean more accuracy. This is what our experiments have proven. At some point, the accuracy does not increase considerably. The quality of the training data is also important. The tiles with the highest feature diversity scores provided our model with enough information to classify the rest accurately. Furthermore, using more training data significantly increases the time and memory required to train a model. From our final results, we can see that a training dataset equal to an area of .75 squared kilometres was enough to produce a model that can achieve an F_1 score of 0.9 for a testing dataset equal to an area of 9.25 squared kilometres.

In case the trained models underperform for LiDAR datasets with low point densities, what is the minimum point cloud density required for the trained models to perform well?

The larger the point density of the training data, the better the performance of a classifier will be. However, this increases the time and memory required to train a model. From our experiments, we found that a point cloud with a point density of 1 point per 1 cubic meter can produce efficiently a classification model that can achieve an F_1 score above 0.85 for any input testing dataset with point density equal or larger to that.

Do deep learning algorithms outperform profoundly simpler machine learning algorithms?

Looking at the results of previous works, deep learning algorithms definitely outperform simpler machine learning algorithms like the Random Forest Classifier. From our results and the findings of previous works, we can say that if trained with the right data and handcrafted features, that difference can be smaller than 10% in terms of overall accuracy.

6.2 Future work

Based on our proposed methodology's limitations and new questions raised during our experiments, we would like to recommend some future work in this final section of this thesis. These recommendations aim to improve our proposed methodology further and incorporate it into applications such as automatic building reconstruction methods. In the future, we would like to explore and recommend continuing this research in the following directions:

- **Machine Learning algorithms:** For this graduation project, we only tested one machine learning algorithm, the Random Forest Classifier. Other machine learning algorithms, such as Support Vector Machines and Adaboost, could be tested to compare their performance and efficiency. Also, we would like to implement an automatic method that selects which features are the best to train these machine learning models.
- **Deep Learning algorithms:** Nowadays, new deep learning methods for point cloud classification are constantly being developed, surpassing each time some of their predecessors in efficiency and accuracy. In our future work, we would mainly focus on implementing and testing these algorithms. Some attempts have already been made to apply some of these deep learning algorithms on our datasets, but they were unsuccessful due to their complexity and requirements (i.e. powerful GPU, Linux operating system).
- **Point cloud classification labels:** As of right now, our method only classifies a point cloud in three different categories of points, namely ground, building and other. In the future, we would like to test the ability of our method to semantically segment a point cloud into additional classes like vehicles, high vegetation, low vegetation and others, that can be found in an urban environment.

- **Point cloud analysis:** An additional processing step for further automatization would make our method much more user-friendly and allow its incorporation in applications, such as urban scene reconstruction. For example, separating the building points from the rest of the point cloud and analysing them to further segment them into individual building points would help reconstruct 3D city models. We will consider involving that in our future work.

6.2.1 Further improvement of our proposed methodology

A processing step that could be added to our proposed methodology to potentially improve the performance of our resulting classification models is to oversegment our input point cloud into superpoints. Superpoints are geometrically simple and meaningful shapes found in the input point cloud and consist of an informative and diverse set of points that are assumed to be semantically homogeneous. Oversegmentation of point clouds is the 3D equivalent of oversegmenting images into superpixels, which have been successfully used to develop supervised superpixels oversegmentation approaches. Superpoints are exploited by some deep learning architectures to semantically segment point clouds with high accuracy [Shi et al., 2021; Landrieu and Simonovsky, 2018; Landrieu and Boussaha, 2019].

Bibliography

AHN (2020). Kwaliteitsbeschrijving. https://www.ahn.nl/kwaliteitsbeschrijving/. Accessed: 2021-03-20.

Armeni, I., Sax, S., Zamir, A. R., and Savarese, S. (2017). Joint 2d-3d-semantic data for indoor scene understanding. *CoRR*, abs/1702.01105.

Bernard, S., Heutte, L., and Adam, S. (2007). Using random forests for handwritten digit recognition. volume 2, pages 1043–1047.

Bernard, S., Heutte, L., and Adam, S. (2009). Influence of hyperparameters on random forest accuracy. In *International workshop on multiple classifier systems*, pages 171–180. Springer.

Biljecki, F. (2017). *Level of detail in 3D city models*. PhD thesis.

Biljecki, F., Ledoux, H., and Stoter, J. (2016). An improved lod specification for 3d building models. *Computers Environment and Urban Systems*, 59:25–37.

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Coltekin, A. (2015). Applications of 3d City Models: State of the Art Review. *ISPRS International Journal of Geo-Information*, 4:2842–2889.

Boulch, A. (2020). Convpoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24–34.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Brodu, N. and Lague, D. (2012). 3d terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology. *ISPRS Journal of Photogrammetry and Remote Sensing*, 68:121–134.

Carson, W. W., Andersen, H.-E., Reutebuch, S. E., and McGaughey, R. J. (2004). Lidar applications in forestry–an overview. In *Proceedings of the ASPRS Annual Conference*, pages 1–9.

Chang, A. X., Funkhouser, T. A., Guibas, L. J., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012.

Chaton, T., Nicolas, C., Horache, S., and Landrieu, L. (2020). Torch-points3d: A modular multi-task frameworkfor reproducible deep learning on 3d point clouds. In *2020 International Conference on 3D Vision (3DV)*. IEEE.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Co., USA, 1st edition.

Criminisi, A., Konukoglu, E., and Shotton, J. (2011). Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Technical Report MSR-TR-2011-114.

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T. A., and Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CoRR*, abs/1702.04405.

Díaz-Uriarte, R. and De Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):1–13.

Elberink, S. O. and Maas, H.-G. (2000). The use of anisotropic height texture measures for the segmentation of airborne laser scanner data. *International archives of photogrammetry and remote sensing*, 33(B3/2; PART 3):678–684.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63:3–42.

Gold, C., Tse, R., and Ledoux, H. (2006). Building reconstruction — outside and in. pages 355–369.

Goldstein, B. A., Polley, E. C., and Briggs, F. B. (2011). Random forests for genetic association studies. *Statistical applications in genetics and molecular biology*, 10(1).

Gong, Z., Zhong, P., and Hu, W. (2019). Diversity in machine learning. *IEEE Access*, 7:64323–64350.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Gröger, G., Kolbe, T. H., Nagel, C., and Häfele, K.-H. (2012). OpenGIS City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0. *OGC Document No. 12-019*, page 344.

Gross, H., Jutzi, B., and Thoennessen, U. (2007). Segmentation of tree regions using data of a full-waveform laser. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36.

Hackel, T., Wegner, J., and Schindler, K. (2016). Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-3:177–184.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585:357–362.

Hu, Q., Yang, B., Khalid, S., Xiao, W., Trigoni, N., and Markham, A. (2020). Towards semantic segmentation of urban-scale 3d point clouds: A dataset, benchmarks and challenges. *arXiv preprint arXiv:2009.03137*.

Ishwaran, H. (2015). The effect of splitting on random forests. *Machine learning*, 99(1):75–118.

Jin, S., Su, Y., Zhao, X., Hu, T., and Guo, Q. (2020). A point-based fully convolutional neural network for airborne lidar ground point filtering in forested environments. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:3958–3974.

Kang, N. (2017). Multi-layer neural networks with sigmoid function- deep learning for rookies (2).

Kavisha, K. (2020). *Modelling and managing massive 3D data of the built environment*. PhD thesis.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kraus, K. and Pfeifer, N. (2001). Advanced dtm generation from lidar data. volume 34, pages 23–30. Citeseer.

Kulawiak, M. and Lubniewski, Z. (2020). Improving the accuracy of automatic reconstruction of 3d complex buildings models from airborne lidar point clouds. *Remote Sensing*, 12:1643.

Landrieu, L. and Boussaha, M. (2019). Point cloud oversegmentation with graph-structured deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7440–7449.

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

Ledoux, H., Arroyo Ohori, K., and Peters, R. (2020). *Computational modelling of terrains*. v0.7.2 edition.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. (2018). Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31:820–830.

Louridas, P. and Ebert, C. (2016). Machine learning. *IEEE Software*, 33:110–115.

Martínez-Muñoz, G. and Suárez, A. (2010). Out-of-bag estimation of the optimal sample size in bagging. *Pattern Recognition*, 43(1):143–152.

Nembrini, S., König, I. R., and Wright, M. N. (2018). The revival of the gini importance? *Bioinformatics*, 34(21):3711–3718.

Niemeyer, J., Rottensteiner, F., and Soergel, U. (2014). Contextual classification of lidar data and building object detection in urban areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 87:152–165.

Oshiro, T. M., Perez, P. S., and Baranauskas, J. A. (2012). How many trees in a random forest? In Perner, P., editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 154–168, Berlin, Heidelberg. Springer Berlin Heidelberg.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Polamuri, S. (2017). How the random forest algorithm works in machine learning. https://dataaspirant.com/random-forest-algorithm-machine-learing/. Accessed: 2021-04-12.

Poux, F. (2020). How to automate lidar point cloud processing with python. *Medium*.

Probst, P. and Boulesteix, A.-L. (2017). To tune or not to tune the number of trees in random forest. *J. Mach. Learn. Res.*, 18(1):6673–6690.

Probst, P., Wright, M. N., and Boulesteix, A.-L. (2019). Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):e1301.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.

Saeys, Y., Inza, I., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517.

Segal, M. R. (2004). Machine learning benchmarks and random forest regression. UCSF: Center for Bioinformatics and Molecular Biostatistics. Retrieved from https://escholarship.org/uc/item/35x3v9t4.

Shi, X., Xu, X., Chen, K., Cai, L., Foo, C.-S., and Jia, K. (2021). Label-efficient point cloud semantic segmentation: An active learning approach. *ArXiv*, abs/2101.06931.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Tang, L., Ying, S., Li, L., Biljecki, F., Zhu, H., Zhu, Y., Yang, F., and Su, F. (2020). An application-driven lod modeling paradigm for 3d building models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 161:194–207.

Thomas, H., Goulette, F., Deschaud, J.-E., Marcotegui, B., and LeGall, Y. (2018). Semantic classification of 3d point clouds with multiscale spherical neighborhoods. In *2018 International conference on 3D vision (3DV)*, pages 390–398. IEEE.

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420.

Varney, N., Asari, V. K., and Graehling, Q. (2020). Dales: A large-scale aerial lidar data set for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 186–187.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Waldhauser, C., Hochreiter, R., Otepka, J., Pfeifer, N., Ghuffar, S., Korzeniowska, K., and Wagner, G. (2014). Automated classification of airborne laser scanning point clouds. In *Solving Computationally Expensive Engineering Problems*, pages 269–292. Springer.

Wang, L., Meng, W., Xi, R., Zhang, Y., Ma, C., Lu, L., and Zhang, X. (2019). 3d point cloud analysis and classification in large-scale scene based on deep learning. *IEEE Access*, 7:55649–55658.

Weinmann, M. (2016). *Reconstruction and Analysis of 3D Scenes - From Irregularly Distributed 3D Points to Object Classes*. Springer.

Weinmann, M., Jutzi, B., Hinz, S., and Mallet, C. (2015). Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105:286–304.

Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Wikipedia (2021a). Bootstrapping (statistics) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Bootstrapping%20(statistics)&oldid=1029162403. [Online; accessed 07-July-2021].

Wikipedia (2021b). Cross-validation (statistics) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Cross-validation%20(statistics)&oldid=1026642057. [Online; accessed 05-June-2021].

Wikipedia (2021c). Curvature — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Curvature&oldid=1024397727. [Online; accessed 06-June-2021].

Wikipedia (2021d). Decision tree learning — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Decision%20tree%20learning&oldid=1026589978. [Online; accessed 04-June-2021].

Wikipedia (2021e). F-score — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=F-score&oldid=1023569737. [Online; accessed 05-June-2021].

Wikipedia (2021f). Feedforward neural network — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Feedforward%20neural%20network&oldid=1021889859. [Online; accessed 29-June-2021].

Wikipedia (2021g). Out-of-bag error — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Out-of-bag%20error&oldid=1023317101. [Online; accessed 07-July-2021].

Wikipedia (2021h). Precision and recall — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Precision%20and%20recall&oldid=1025561211. [Online; accessed 05-June-2021].

Wright, M. and Ziegler, A. (2015). ranger: A fast implementation of random forests for high dimensional data in c++ and r. *Journal of Statistical Software*, 77.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yi, C., Zhang, Y., Wu, Q., Xu, Y., Oussama, R., Wei, M., and Wang, J. (2017). Urban building reconstruction from raw lidar point data. *Computer-Aided Design*, 93.

Zhang, Z., Hua, B.-S., and Yeung, S.-K. (2019). Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1607–1616.

Zhou, Q.-Y. and Neumann, U. (2010). 2.5d dual contouring: A robust approach to creating building models from aerial lidar point clouds. pages 115–128.

Zhou, Q.-Y. and Neumann, U. (2013). Complete residential urban area reconstruction from dense aerial lidar point clouds. *Graphical Models*, 75:118–125.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.