# Machine Learning-based Classification of Different 3D Point Cloud Data of Railway Environments using Random Forest and DGCNN


**BY:**

Anestoula Papalexiou


**UNDER:**

TU Delft CGI

# MACHINE LEARNING-BASED CLASSIFICATION OF DIFFERENT 3D POINT CLOUD DATA OF RAILWAY ENVIRONMENTS USING RANDOM FOREST AND DGCNN

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

Master of Science in Civil Engineering
Track: Geoscience and Remote Sensing

by

Anestoula Papalexiou

February 2021

ISBN 999–99–9999–999–9

The work in this thesis was made in the:

Department of Geoscience and Remote Sensing
Faculty of Civil Engineering and Geosciences
Delft University of Technology &
CGI

| | |
|---|---|
| Supervisors: | Dr. R.C. (Roderik) Lindenbergh |
| | Dr. F. (Franziska) Glassmeier |
| | Dr. A.A. (Alfredo) Núñez Vicencio |
| Company Supervisor: | R.L. (Robert) Voûte, CGI |

# ABSTRACT

Although monitoring and maintenance of railways is important to ensure safety and avoid delays and financial losses, it is still mainly based on human inspection. The complexity of a railway along with the large area it extends makes manual monitoring difficult and time-consuming. The increasing availability of 3D acquisition technologies has made point clouds a widely used 3D data form. Thus, identifying the key components of a railway and its environment using 3D point cloud data can be the first step for automating this procedure. The past years, machine learning has become the most popular subfield of artificial intelligence used for various different applications, with its subfield of deep learning evolving dramatically. Although deep learning has been vastly researched on 2D data, applying deep learning on 3D point clouds can be challenging due to the irregularity, unstructuredness and unorderedness of such data. To overcome those challenges, recent approaches use projection or voxelization, while the latest methods focus on working directly on raw point cloud data.

In the present research, 2 machine learning methods are implemented to classify 3D point cloud data of railway environments into 7 categories of rails, sleepers, track bed, masts, overhead wires, trees and other. To this end, the ensemble method of Random Forest, and the deep learning method of DGCNN (Dynamic Graph Convolutional Neural Network) are implemented. While Random Forest is a simple and handy ensemble algorithm used for a wide range of applications, DGCNN is a deep learning method based on PointNet, the pioneer method on raw point clouds, and graph CNNs. The methods are validated under 3 case studies, produced by structure from motion photogrammetry, airborne laser scanning and terrestrial laser scanning, and locating in 2 different areas within the Netherlands. For each method, 2 scenarios are developed for classifying colored and uncolored point clouds, respectively. Finally, the 2 methods are combined for the first scenario, as a first attempt to further improve the final results.

The obtained results show that, in this work, DGCNN performs better than Random Forest, and both methods approximate state-of-the-art performance. The contribution of colors is important to improve both the overall accuracy of the models, as well as the classification results of the individual classes. More specifically, Random Forest scenario 1, Random Forest scenario 2, DGCNN scenario 1, DGCNN scenario 2, and the combination of Random Forest and DGCNN scenario 1 result in an overall accuracy of 88.65%, 83.39%, 89.19%, 88.20% and 90.57%, respectively. The corresponding per class F1-scores for all methods and scenarios range between 43% and 94%. Both methods meet difficulties in generalizing on data from different sensor systems and different areas with very different point density and missing data. Nonetheless, the individual methods are already very promising, while they are able to achieve the required accuracy of more than 90% when combined.

# PREFACE

The present thesis is part of the Master program in Civil Engineering and the track of Geoscience and Remote Sensing at TU Delft, and was conducted in the Vital Civil Infrastructures department of CGI Nederland. The project lasted officially 9 months in total, from June 2020 to February 2021. Although thesis is an individual project, the contribution of supervisors, colleagues, friends and family in this work was valuable to me.

First of all, I would like to express my gratitude to my daily supervisor and chairman of my committee, Roderik Lindenbergh, for his guidance, advice, suggestions and recommendations regarding the methodology and scientific writing of this research. Secondly, I am very much thankful to my company supervisor and vice president of CGI Nederland, Robert Voûte, for providing me with his supervision and giving me the opportunity to work within CGI and be part of their team. Also, I would like to thank my second and third supervisors, Franziska Glassmeier and Alfredo Núñez Vicencio, who were always willing to help during the project when necessary.

Furthermore, I wish to thank Kaixuan Zhou for his useful advice on the Random Forest implementation, Elyta Widyaningrum and Qian Bai for their help in the DGCNN implementation, and Adriaan van Natijne for giving me access to the HPC cluster of TU Delft. Additionally, I would like to thank all the Master students, PhD students and researchers in Roderik's team, as well as all the graduate students in Robert's team, for sharing their views and all the fun during team meetings.

Lastly, I am extremely grateful to my dear friends and family for standing by me during all these years and cheering me up through hard times. To conclude, my special thanks to everyone mentioned above for their support and encouragement in these unprecedented times for humanity. Working from home and trying to adapt to the new conditions of the pandemic was not easy, but thanks to all these people I finally managed to complete this work.

All the best to everyone and enjoy reading!

Anestoula Papalexiou
February 2021
Delft, Netherlands

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF EQUATIONS

# ACRONYMS

| | | |
|---|---|---|
| **1D** | 1-Dimensional | 17 |
| **2D** | 2-Dimensional | 2 |
| **3D** | 3-Dimensional | 1 |
| **AI** | Artificial Intelligence | 1 |
| **ALS** | Airborne Laser Scanning | 2 |
| **avPA** | average Per class Accuracy | 42 |
| **CAD** | Computer-Aided Design | 17 |
| **CNN** | Convolutional Neural Network | 3 |
| **CPU** | Central Processing Unit | 29 |
| **CRF** | Conditional Random Field | 19 |
| **CUDA** | Compute Unified Device Architecture | 29 |
| **DGCNN** | Dynamic Graph Convolutional Neural Network | 3 |
| **DL** | Deep Learning | 1 |
| **EdgeConv** | Edge Convolution | 3 |
| **FN** | False Negative | 41 |
| **FP** | False Positive | 41 |
| **GCP** | Ground Control Point | 23 |
| **GMM-EM** | Gaussian Mixture Model - Expectation Maximization | 19 |
| **GNSS** | Global Navigation Satellite System | 9 |
| **GPU** | Graphics Processing Unit | 29 |
| **HPC** | High Performance Computing | 28 |
| **IDE** | Integrated Development Environment | 28 |
| **IMU** | Inertial Measurement Unit | 9 |
| **IoU** | Intersection over Union | 42 |
| **k-NN** | k-Nearest Neighbors | 40 |
| **LiDAR** | Light Detection And Ranging | 1 |
| **mIoU** | mean Intersection over Union | 42 |
| **ML** | Machine Learning | 1 |
| **MLP** | Multi-Layer Perceptron | 17 |
| **MLS** | Mobile Laser Scanning | 9 |
| **NN** | Neural Network | 15 |
| **RG** | Region Growing | 19 |
| **OA** | Overall Accuracy | 17 |
| **OS** | Operating System | 29 |
| **PC** | Principal Component | 38 |
| **PCA** | Principal Component Analysis | 4 |
| **RANSAC** | Random Sample Consensus | 19 |
| **RD** | Rijksdriehoeks | 23 |
| **RGB** | Red Green Blue | 1 |

# 1 INTRODUCTION

Chapter 1 comprises the problem statement in Section 1.1, relevance in Section 1.2, research question in Section 1.3, as well as the research scope in Section 1.4.

## 1.1 PROBLEM STATEMENT

The operation and maintenance of railway infrastructures is of great importance, since rail transportation is one of the most popular means of traveling nowadays. The power transmission from overhead power lines to vehicles can be interrupted, and tracks can also be damaged mainly due to friction between their surfaces and the wheels of the trains. Trees around the railway can be blown down by the wind and block the rails, demanding monitoring not only of the railroad assets, but also of the surrounding environment. Failure events in essential parts of a railway, such as tracks or catenary system, can threaten the availability and security of railway services and result in disruption of services, significant financial losses and high safety risks.

The prevention of failures and their repercussions can be achieved by regular inspection and maintenance of railroad environments. However, inspection heavily relies on human factors and visual interpretation. While manual monitoring is time-consuming, dangerous and inefficient, it also increases the operational costs, and inserts human errors. The complexity of a railroad, with the multiple components it consists of and the large area it extends, creates difficulties in monitoring continuously the railway and detecting accurately and promptly the damages. Thus, identifying the key components of a railway and its environment can be the first step for automating this procedure. Nonetheless, since rail infrastructures need to be measured periodically, because of a changing environment but also of the trains that block the infrastructure for other uses, it is hard to find the best way to do so. Drones can offer a fast way for capturing large-scale railroads. According to the Dutch government, the use of drones for commercial purposes must comply with the following rules: Rules for the commercial use of drones.

ProRail is a governmental organization which is responsible for the maintenance and extension of the Dutch railway network infrastructure, for allocating rail capacity, and for traffic control. Currently, ProRail uses multiple different methods to monitor railways, such as aerial Light Detection And Ranging (LiDAR) and terrestrial and train-based laser scanners to collect 3-Dimensional (3D) point clouds, and drones to collect multiple photographic images. The images are further processed using photogrammetric methods to extract positions of the points in 3D space (3D point clouds). The increasing availability of 3D acquisition technologies has made point clouds a widely used 3D data form. Point clouds are 3D data points in space. They contain at least the $XYZ$ geometric coordinates of the points, as well as other extra attributes, such as Red Green Blue (RGB) values and intensity, and they are usually generated using remote sensing and photogrammetric techniques.

Machine Learning (ML) is a field at the intersection of statistics, Artificial Intelligence (AI), and computer science, about programming computers to learn from data, Müller et al. [2016]; Russell [2018]. Since ML started flourishing, it quickly became the most popular and successful subfield of AI, driven by the availability of faster hardware and larger datasets. A subfield of machine learning is Deep Learning (DL), which is about learning representations from data, while emphasising on learning suc-

cessive layers of increasingly meaningful representations, Chollet [2017]. Unlike DL, other ML approaches tend to focus on learning only 1 or 2 layers of representations of the data (shallow learning). Although DL was first theorized years before, it only became useful the recent years due to the need of large amount of labeled data and high computational power for training. Labeled data are data annotated with one or multiple labels, mainly used to train ML models. In fact, part of the labeled data is used for training a ML model and part of it for testing the model. The process of training a ML model includes using a ML algorithm (the learning algorithm) with training data to learn from. The ML model refers to the resulting model from the training process. The test data are then used to evaluate the performance of the model.

In the past few years, deep learning has been vastly researched on 2-Dimensional (2D) data (mainly images), and achieved great progress in classification, segmentation and object detection tasks. The rapid development of 3D devices (such as LiDAR), the last few years, made DL on 3D point cloud data gain attention and the use of DL for processing them feasible. However, when applying DL on 3D point clouds many challenges occur, Bello et al. [2020]. Some of them are associated with occlusion, missing data, noise or outliers and points misalignment. But the most pronounced challenges come from the nature of point clouds and their properties of irregularity, unstructuredness and unorderedness. Point clouds are irregular, so the points are not evenly sampled across different regions of a scene, meaning that there could be regions covered by dense points and others by sparse points. In contrast to images, point clouds are unstructured, so the data are not on a regular grid, but the points are scanned independently and the distance between neighboring points varies in an indefinite way (no explicit neighbourhood relations). Moreover, a point cloud is an unordered set of points (usually $XYZ$), where the order of points stored as a list in a file does not change the scene represented by the points.

The present research consists of a project from the global information technology consulting and systems integration company, CGI. The project explores 2 different approaches, a more simple ensemble-based approach (Random Forest) and a more complicated deep learning-based approach (Dynamic Graph Convolutional Neural Network, DGCNN). The objective of this work is to perform automatic pointwise classification (semantic segmentation) of point cloud data of railway scenes into 7 categories of rails, sleepers, track bed, masts, overhead wires, trees and other, using Random Forest and DGCNN. It is also investigated how these algorithms perform on data acquired from different sensor systems and different areas within the Netherlands. For this purpose, the methods are validated under 3 different types of data, produced by Structure from Motion (SfM) photogrammetry, Airborne Laser Scanning (ALS) and Terrestrial Laser Scanning (TLS). More specifically, the 3 different datasets employed are: an Unmanned Aerial Vehicle (UAV)-SfM dataset from Groningen with $XYZ$ Cartesian coordinates and RGB values, an UAV-LiDAR dataset from Stroe with 3D Cartesian coordinates and color values as well, and a TLS dataset from Stroe only with $XYZ$ Cartesian coordinates. Hence, for each method, 2 scenarios are developed for classifying colored and uncolored railway point clouds, respectively. In the end, the 2 methods are combined only for the first scenario, as a first attempt to further improve the final results.

## 1.2 RELEVANCE

The autonomous railway monitoring and maintenance is important to ensure travel safety, speed up the inspection processes and reduce human involvement and the possibility of human error. So far little attention has been paid to mapping the whole railway environment automatically from point clouds, while mainly focusing on specific parts of it. Previous research, including 3D point cloud data or ML techniques on railways, mostly focus either on rail tracks or catenary systems. The combination of

image processing and ML techniques has also been used for computer vision problems related to identifying rail tracks and separating the components of catenary systems.

However, as stated, point clouds are by nature irregular, unordered and unstructured. These properties of point clouds make the application of deep learning on point clouds very challenging, especially when it comes to Convolutional Neural Networks (CNNs). CNNs are based on convolution operation using data that are regular, ordered and on a structured grid. To overcome those challenges, recent approaches convert point cloud data into a structured grid using projection or voxelization, and then apply 2D or 3D convolution to classify the point clouds. Lately, methods have also been developed, taking directly as input raw point cloud data and avoiding the conversion into a structured grid. PointNet, Qi et al. [2017], is a pioneer in deep learning on raw point clouds. The network maintains permutation invariance of points and provides a unified architecture for object detection, part segmentation and scene semantic parsing (semantic segmentation) applications. Although the semantic segmentation application of PointNet was developed based on benchmarks that represent indoor scenes, it has also been tested on outdoor environments by other researchers later on.

After PointNet, many more deep learning methods have been proposed exclusively for point clouds, that proved to be faster and more accurate than PointNet. Some of them are based on PointNet, including some additions and modifications for further improvement of the results. Dynamic Graph Convolutional Neural Network (DGCNN), Wang et al. [2019b], is one of those DL methods consisting of an extension of PointNet. Edge Convolution (EdgeConv) is an operation that captures local geometric structure while maintaining permutation invariance. It generates edge features describing the relationships between a point and its neighborhood, and is invariant to the ordering of neighbors (permutation invariant). EdgeConv has been integrated into the basic version of PointNet, while excluding feature transformation, and the method is named DGCNN. Later, other studies extended also the semantic segmentation application of DGCNN from indoor scenes to outdoor environments, particularly using aerial point cloud data to classify urban environments.

On the other hand, Random Forest is a type of ML algorithm belonging to ensemble methods, that is ML techniques combining of several base models to produce the final optimal predictive model. Typically, it is used for classification and regression problems, while in classification tasks, it is used to categorize data based on their features. It is actually a collection of decision trees generated using a random subset of data, with decision tree the process of deciding which class the data belong to. Eventually, Random Forest performs majority voting among all its trees to predict the output classes, that is the classes the majority of trees have voted for. It is a simple and handy ML algorithm that can be used for several different applications and perform fairly good at all of them, although it might not be as efficient as specialized algorithms for specific tasks. Random Forest has already been applied for classification tasks in various different fields, from remote sensing to finance and healthcare sectors, and many more, but not on point cloud data from railways yet.

## 1.3 RESEARCH QUESTION

The main research question of this research states:

***To what extent can a railway infrastructure be mapped in its asset components using 3D point cloud data and machine learning?***

To the best knowledge of the author, there are no attempts so far to fully map the whole railway environment, using the combination of point cloud data and DL techniques. Furthermore, the interesting aspect of investigating how a classical ensemble method, like Random Forest, performs on classifying point clouds from railway scenes compared to a more sophisticated DL approach, is incorporated to the study. An ad-

ditional challenge of the project is to test the applicability of the methods on data derived from different sources and different areas within the Netherlands. Finally, it is in the interest of this work to explore the contribution of color values to the classification of point clouds.

The main research question creates several subquestions which help structuring the project. The subquestions can be divided into 3 main categories: the mapping requirements, the machine learning approaches, and the project possibilities.

- **Mapping requirements:**
  - What types of components can be detected in a railway infrastructure?
  - What are the accuracy requirements when classifying objects in a railway infrastructure?
- **Machine learning approaches:**
  - What is a suitable training strategy?
  - How to evaluate the performance of the models?
  - How to investigate the uncertainty of the models on classifying railway components?
- **Project possibilities:**
  - Is it possible to achieve state-of-the-art performance when classifying point cloud data from the entire railway environment at the same time?
  - What are the pros and cons of a DL method compared to a classical ensemble method for classification tasks?
  - How well do the models generalize so they can be used for data from different sensor systems and different areas?
  - What is the impact of color values in classification of point clouds?

## 1.4 RESEARCH SCOPE

The current research follows the following methodology. Random Forest and DGCNN are used to perform pointwise classification of railway point clouds into rails, sleepers, track bed, masts, overhead wires, trees and other. First of all, the UAV-SfM dataset from Groningen is used for training and testing both the ensemble (Random Forest) and the DL (DGCNN) models. The training and test sets come from 2 different areas within the dataset, with a distance of around 25 meters from each other, which guarantees a more reliable evaluation of the performance of the models. The training data are created manually, since 2D representation of the scenes is not available.

Random Forest is trained and tested once (scenario 1) using RGB values and geometric features generated from Principal Component Analysis (PCA). The resulting model is also validated visually on the unlabeled UAV-LiDAR dataset from Stroe, which also contains color values. The second time (scenario 2), Random Forest is trained and tested using only the geometric features from PCA, and the second model is also validated visually on the unlabeled TLS dataset from Stroe, which does not have color values. The procedure followed for DGCNN is similar to that of Random Forest. DGCNN is trained and tested using the 3D Cartesian coordinates of the points, their RGB values and their normalized 3D coordinates (scenario 1). The trained model is again validated visually on the UAV-LiDAR dataset from Stroe. The second time (scenario 2), DGCNN is trained and tested excluding the RGB values, and the second model is also validated visually on the TLS dataset from Stroe.

The classification accuracy and other classification metrics are computed for both methods based on the test set from UAV-SfM, and used to quantitatively assess the performance of the models. Moreover, the predicted class probabilities are calculated

for all predicted data, aiming to investigate the uncertainty of the models in predicting the different components on railways. Finally, the 2 methods are combined using the prediction results and predicted probabilities, in order to exploit the probabilities for further improvement of the final results. The combination is applied only to the first scenario, as a first experiment. After the combination, the predicted labels of the test set are those resulted either from Random Forest or DGCNN, depending on which method provided the prediction for each point with higher probability.

The present study has multiple various goals at the same time. First of all, investigating if state-of-the-art performance can be achieved when classifying points cloud data from the entire railway environment at one go. Secondly, comparing how well can a simple ensemble method perform on the classification of point clouds of railways compared to a more complicated DL method. Thirdly, exploring the importance of RGB values in classification tasks, and the impact on the quality of the results when omitting them. Finally, testing the performance of the methods on data obtained from different sensors and different areas.

In order to be more instructive concerning this work, it is important to mention also what is beyond the scope of this research. Regarding the Random Forest implementation, the default parameters are used, and no optimization or parameter tuning is implemented. Instead, only different features have been explored prior to concluding to the final models. For DGCNN, most of the parameters used are the ones suggested as default, but some of them are adjusted to fit the type of data in this study. The final models are the best occured after training for 50 epochs. Moreover, only parts of the datasets are used to indicate the performance of the methods, aiming to save computational time and power. Also, the UAV-LiDAR and TLS datasets have not been annotated to avoid extra manual labor.

The present thesis consists of 7 chapters: Chapter 1 contains an introduction to this work, Chapter 2 the theoretical background related to this research, Chapter 3 the data and tools used to accomplish this study, Chapter 4 the methodology followed during the project, and Chapter 5 the results produced. Finally, Chapter 6 includes a discussion about the methodology and results, and Chapter 7 the conclusion along with recommendations for future work.

# 2 THEORETICAL BACKGROUND

Chapter 2 contains a short overview of the main railway components in Section 2.1, and point clouds and acquisition techniques in Section 2.2. Section 2.3 refers briefly to machine learning, and Section 2.4 to the ensemble method used in this work, Random Forest. In Section 2.5, the general principals of a subfield of machine learning, deep learning, are presented, and Section 2.6 is dedicated to the deep learning method used, Dynamic Graph Convolutional Neural Network (DGCNN). Finally, Section 2.7 is related to the classification of railway point clouds, and Section 2.8 includes a summary of this chapter, as well as a brief introduction to the upcoming chapter.

## 2.1 MONITORING RAILWAYS

Railways constitute a significant part of transport infrastructure around the world, used to transport passengers or cargo. They are subject to regular inspection and maintenance, mainly executed manually by humans. However, monitoring railroad environments is not trivial, since they are complex, consisting of multiple components, and extend over large areas. The main components of a railroad infrastructure are the rails, sleepers, track bed, masts and overhead wires, Arastounia [2015]; Cozza [2005]. Figure 2.1 illustrates these components.

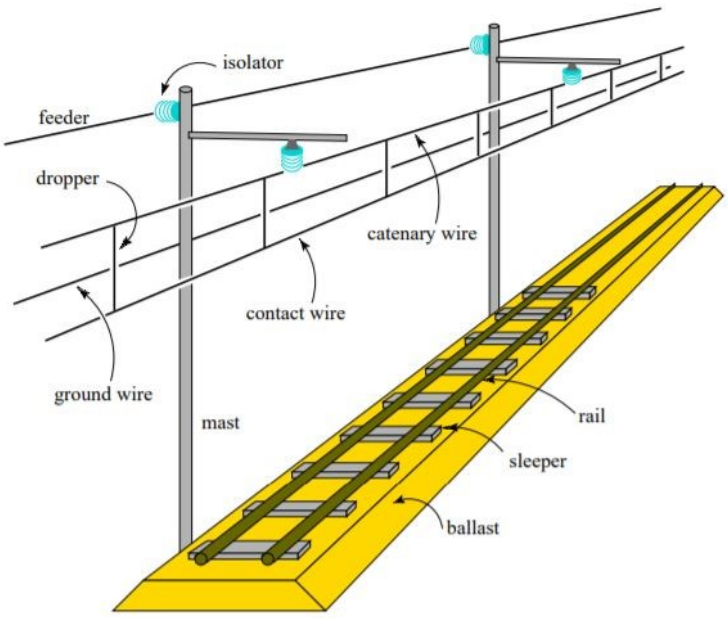



**Figure 2.1:** Main railway components: rails, sleepers, track bed (or ballast), masts (including isolators), overhead wires (here: catenary wires, contact wires, droppers, ground wires and feeders). (Adapted from: Cozza [2005])

- **Rails** (or rail tracks) are made of rolled steel and placed in 2 parallel lines of defined distance, where trains can move. They can be straight, or comprise switches, crossings and insulated joints.

- **Sleepers** (or ties) are rectangular supports for the rail tracks, typically laid perpendicular to the tracks. Traditionally, they are wooden, while nowadays they are usually made of reinforced concrete.

- **Track bed** (or ballast) is the surface underlying the rails and sleepers, and is actually placed between, below and around them. It is used to bear the load from sleepers and keep rails in place, facilitate water drainage, and resist vegetation growth that might interfere with the railroad structure. Normally, it is made up of crushed stone, hence the term "ballast".

- **Overhead wires** are overhead power cables that transmit electricity to trains. It is a multiconductor structure, consisting of catenary wires, contact wires, droppers and other types of wires, such as ground wires, feed wires and return wires.

- **Masts** are, mainly cylindrical, poles placed at regular spatial intervals close to the rails. They are used to support the suspended overhead wires. The isolators (or insulators) are used to attach power cables to masts and provide electrical insulation.

Regularly, there is a large amount of trees lining the railways, that can cause problems. Falling trees might block rails or damage power cables, leading to delays in the railway system. Therefore, it would be of great interest to not only identify the basic components of a railway infrastructure, but also the surrounding trees, as well as other nearby objects. Table 2.1 summarizes the key components in a railway environment, as considered in this work, that would be useful to detect.

**Table 2.1:** Main components in railway environments, along with a short description of them.

| Component | Description |
|---|---|
| Rails | Parallel lines of defined distance, where trains can move. |
| Sleepers | Rectangular supports for the rail tracks, typically laid perpendicular to the tracks. |
| Track bed | Surface underlying the rails and sleepers used to keep rails and sleepers in place, facilitate water drainage and resist vegetation growth. |
| Masts | Poles placed at regular spatial intervals close to the rails to support the suspended power cables. |
| Overhead wires | Overhead power cables used to transmit electricity to trains. |
| Trees | Trees (and vegetation) lining the railways. |
| Other | Remaining objects on a railway. |

## 2.2 POINT CLOUDS AND ACQUISITION TECHNIQUES

A point cloud is a set of 3-Dimensional (3D) data points in space, representing objects or scenes. In its simplest form, a 3D point cloud is represented by the $XYZ$ geometric coordinates of the points. However, additional features can also be included, such as intensity or reflection, Red Green Blue (RGB) values or color, and surface normals. A point cloud is very convenient for representing the 3D world, and it is preferred for many scene understanding related applications in various fields, such as autonomous driving, robotics, augmented and virtual reality, etc.

Most commonly, 3D point clouds are generated by remote sensing or photogrammetric methods. Remote sensing mainly refers to 3D laser scanners and Light Detection And Ranging (LiDAR) techniques in order to collect point cloud data, Dong and Chen [2017]. LiDAR is a measurement technique that deploys laser to gauge precise distances. Using ultraviolet, visible or near-infrared light, LiDAR measures the time it takes for signals to bounce off objects and return to the scanner in order to map spatial relationships and shapes. LiDAR scanners (or else laser scanners) can capture detailed measurements from whole rooms to entire scenes.

There are 3 different techniques of laser scanning: Terrestrial Laser Scanning (TLS), Mobile Laser Scanning (MLS) and Airborne Laser Scanning (ALS), depending on the platform the laser scanner is mounted on. In TLS, the scanner is placed on a tripod. For scanning larger areas, MLS is used, where the scanner is mounted on a moving vehicle, while for even larger areas, ALS technique is used by mounting the scanner on a manned airplane. In ALS, the scanner can also be mounted on an Unmanned Aerial Vehicle (UAV), that is a class of aircrafts that can operate without the onboard presence of a pilot, such as drones, but the areas scanned are smaller than those in MLS.

**Terrestrial laser scanners** are contact-free measuring devices providing detailed and highly accurate 3D point cloud data of objects quickly and efficiently (see Figure 2.2), Oguchi et al. [2011]. A terrestrial laser scanner emits laser pulses towards numerous points in space, and measures the distance between the device and the target to calculate the 3D coordinates of the points, along with their reflectance, RGB values and other attributes. Each individual scan is made up of a great volume of individual measurements, but it is also line-of-sight. This means that, usually, multiple scans are necessary to provide a complete 3D view of an object. A modern TLS device is able to measure from $10^4$ to $10^6$ points per second with an accuracy of $0.1 - 1$ cm. The TLS devices are usually devided into short-, medium- and long-range scanners depending on the maximum distance the laser light can travel to record a point in the field. Longer-range scanners are made to measure spatially larger areas, while those with shorter range are used to map spatially smaller areas with higher accuracy and more detail. This indicates a trade-off between the pulse rate and the laser light energy of a TLS device. In short-range scanners, the interval between adjacent measured points can reach up to 1 mm for relatively small areas. Appropriate software packages are generally used to manage and analyze the acquired large amount of data.

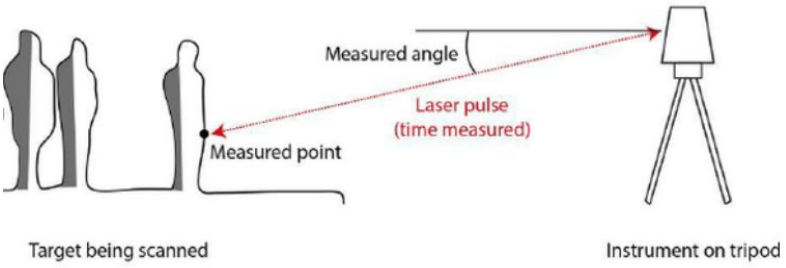



**Figure 2.2:** Terrestrial laser scanner: A laser scanner placed on a tripod, that emits laser pulses towards numerous points in space and measures the distance between the device and the target to calculate the $XYZ$ coordinates of the points, along with additional attributes. (Adapted from: Virtanen et al. [2014])

**Mobile laser scanning** systems are LiDAR systems used for generating highly accurate 3D point clouds (see Figure 2.3), Wang et al. [2019a]. They are mounted on moving platforms (e.g. vehicles, boats, robots), and collect 3D information along the driving paths. An MLS system consists of a 3D laser scanner, a Global Navigation Satellite System (GNSS) receiver for determining the precise location of the scanner, an Inertial Measurement Unit (IMU) for determining the precise orientation of the scanner, and a camera for capturing 2-Dimensional (2D) photos of the scenes during

scanning. Since MLS provides short measure range and flexibility in data acquisition, it is able to obtain highly accurate point clouds (millimeter-level) of high point density (up to a few thousand $points/m^2$).

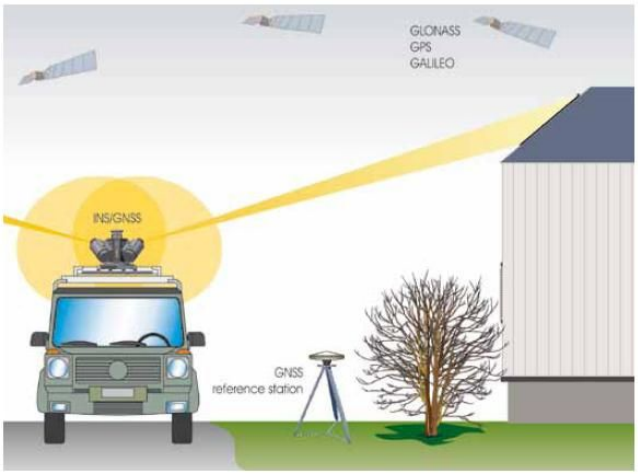



**Figure 2.3:** Mobile laser scanning system: It consists of a laser scanner, a GNSS receiver, an IMU and a camera mounted on a moving vehicle. The *laser scanner* emits laser pulses around the driving path, and measures the time it takes for the signal to return to the scanner to determine the position of the targets in 3D space. The *GNSS* is used to define the location of the scanner, the *IMU* is used to define the orientation of the scanner, and the *camera* is used to capture 2D photos of the scenes during scanning. (Image source: Wang et al. [2012])

**Airborne laser scanning** systems are LiDAR systems mounted on aerial vehicles, such as airplanes and UAVs (see Figure 2.4). LiDAR works in the same way as in TLS and MLS and allows 3D mapping of larger surface areas faster, but with lower resolution. Similar to MLS, the system includes a 3D laser scanner, a GNSS receiver, an IMU and a camera.

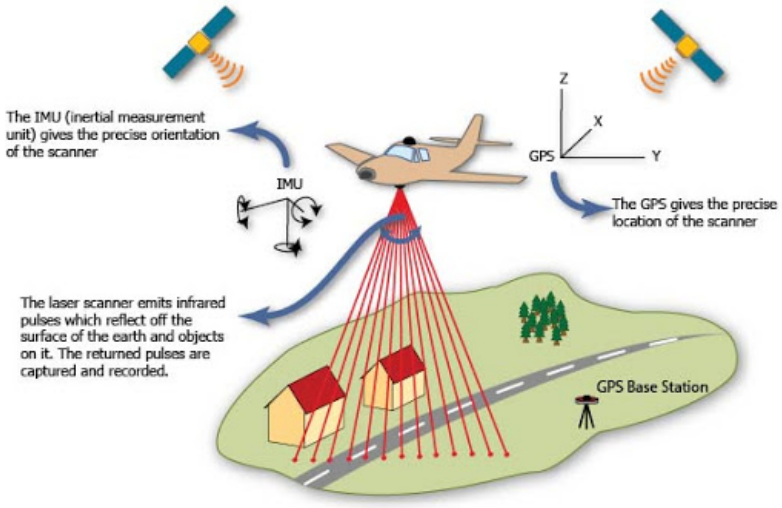



**Figure 2.4:** Airborne laser scanning system: It consists of a laser scanner, a GNSS receiver, an IMU and a camera mounted on an aerial vehicle (here: airplane). The *laser scanner* emits laser pulses to the surface of the earth and objects on it, and measures the time it takes for the signal to return to the scanner to determine the position of the targets in 3D space. The *GNSS* is used to define the location of the scanner, the *IMU* is used to define the orientation of the scanner, and the *camera* is used to capture 2D photos of the scenes during scanning. (Image source: Geospatial Modeling & Visualization [2013])

In photogrammetry, 3D models are produced using 2D photos acquired by cameras mounted on UAVs. It is actually the opposite of photography, which maps the 3D world into 2D images. In fact, for an unknown point that is observed from at least 2 known locations, when the distances and angles of these locations are also known, the location of the point can be determined. Photogrammetry is often used to extract 3D point cloud data out of a set of 2D images.

**Structure from Motion (SfM)** is a low-cost photogrammetric method that can produce point cloud-based 3D models similar to LiDAR, and can be used for high resolution topographic reconstruction (see Figure 2.5), Westoby et al. [2012]. SfM follows the same basic principles as stereoscopic photogrammetry, in a sense that 3D data of a scene or object can be reconstructed using a series of overlapping, offset 2D images. The difference of SfM compared to traditional photogrammetry relies on the fact that the geometry of the scene and the camera positions and orientations are solved automatically, without requiring to manually specify a number of targets with known 3D positions. There are actually solved concurrently using an iterative bundle adjustment procedure that is based on a database of features which are automatically extracted from the multiple overlapping images. This approach is suitable for sets of images that capture the whole 3D structure of the scene from multiple positions and present a high degree of overlap. These images are collected from a moving sensor, which is typically a digital camera mounted on an UAV.

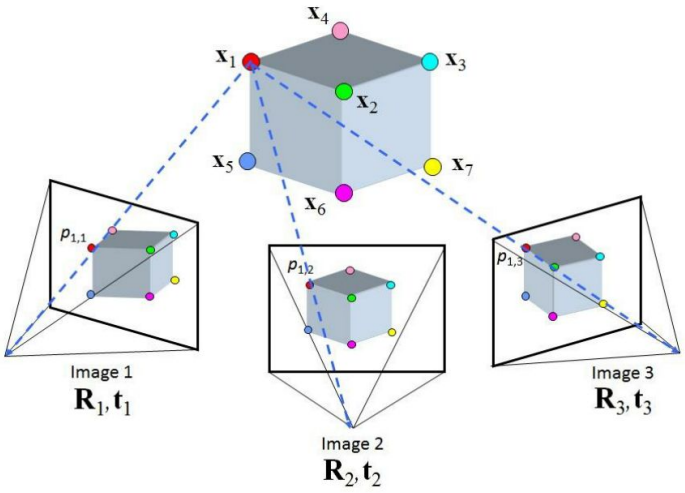



**Figure 2.5:** Structure from motion: The 3D data of a scene or object are reconstructed using a series of overlapping, offset 2D images. The geometry of the scene, and the camera positions and orientations are solved automatically using an iterative bundle adjustment procedure. Here, the structure of the cube is captured from multiple viewpoints (Image 1, Image 2, Image 3). The camera motion (rotation, translation) $\{(R_1, t_1), (R_2, t_2), (R_3, t_3)\}$ and the 3D model of the structure are estimated, by tracking the pixel locations $\{p_{1,1}, p_{1,2}, ...\}$ of specific features $\{x_1, x_2, ...\}$ in the images. (Image source: Yilmaz and Karakus [2013])

As stated, both remote sensing and photogrammetric methods are used to produce georeferenced 3D point clouds of the environment. In this work, the point cloud data used for the representation of railroads derive from 3 different acquisition techniques: terrestrial laser scanning, airborne laser scanning and structure from motion, abbreviated as TLS, UAV-LiDAR and UAV-SfM, respectively. Each of these techniques has some advantages and disadvantages, considering the resolution of point cloud data they can produce, the size of the area that is able to be mapped, the time it takes to do so, and the quality of mapping different objects of the environment. A comparison among these 3 techniques, taking into account the aforementioned criteria, is presented in Table 2.2. Here, point density refers to surface density, as explained in Subsection 3.1.2.

**Table 2.2:** Comparison among different acquisition techniques used in this work: TLS, UAV-LiDAR, UAV-SfM.

| Criteria | TLS | UAV-LiDAR | UAV-SfM |
|---|---|---|---|
| **Resolution** | Higher than UAV-LiDAR and lower than UAV-SfM (mean point density: $4,259$ $pts/m^2$). | The lowest (mean point density: $2,363$ $pts/m^2$). | The highest (mean point density: $44,750$ $pts/m^2$). |
| **Area** | Larger than UAV-SfM and smaller than UAV-LiDAR (maximum range: 270 $m$). | The largest (maximum range: 550 $m$). | The smallest (maximum range: up to 100 $m$). |
| **Time** | The fastest (measurement rate: up to 1 million $pts/sec$). | Faster than UAV-SfM and more time-consuming than TLS (measurement rate: up to $350,000$ $pts/sec$). | The most time-consuming (includes data collection and post-processing). |
| **Quality** | Relatively uniform sampling. | Relatively uniform sampling. | Different reconstruction quality for different objects. |

## 2.3 MACHINE LEARNING

Machine Learning (ML) is a subset of Artificial Intelligence (AI) where computers are programmed to learn from data, Russell [2018]. In the beginning of "intelligent" applications, several systems used handcoded rules to process data, Müller et al. [2016]. These manually crafted decision rules are feasible for some particular applications, especially when humans have a good insight of the process to model. However, when using handcoded rules to make decisions, the logic needed to make a decision is determined for a specific task, while human experts should have a deep understanding of the way a decision should be taken to design those rules. And that is where machine learning comes in. In a problem that many long lists of rules are necessary, ML methods can simplify the implementation and improve the performance, Russell [2018]. Also, ML is very handy for complex problems which do not have a solution using a traditional approach and for non-stable environments that require adaptation to new data.

There are several different types of ML, depending on whether:

- They have been trained by humans or not.
- They can learn incrementally.
- They build a model by comparing new data to already existing data, or detecting new patterns in the data.

Focusing on the first type, ML algorithms can be divided into 4 categories, according to the type of human supervision during training:

- Supervised learning.
- Unsupervised learning.
- Semi-supervised learning.
- Reinforcement learning.

In **supervised learning** the data fed into the algorithm are labeled. This means that the data are tagged with one or multiple labels. The most significant supervised algorithms are: *K-nearest neighbors*, *Linear regression*, *Neural networks*, *Support vector machines*, *Logistic regression*, *Decision trees and Random forests*. Supervised learning is the type of learning used for classification tasks. Classification is related to categorization. In point cloud classification tasks there are two types of classification: object classification and point-wise classification. In object classification, or simply referred as classification, a class label is assigned to each cluster of points that represents an object class. On the other hand, in point-wise classification, more often referred to as semantic segmentation, a class label representing an object type is assigned to each individual point of the point cloud.

On the flip side, in **unsupervised learning** the data are unlabeled. The most important unsupervised algorithms are: *Clustering*, *Association rule learning*, *Visualization and dimensionality reduction (such as Principal Component Analysis, PCA)*. **Semi-supervised learning** falls between supervised and unsupervised learning, as it combines a small amount of labeled data with a large amount of unlabeled data. **Reinforcement Learning** is the last type of ML technique in this category, where a software agent observes the environment and takes actions according to a policy, aiming to maximize the notion of cumulative reward. This type of learning can be found in many robotics applications.

Machine learning systems require data of good quality and sufficient quantity to work efficiently. When working with training data that contain lots of errors and outliers, the system will not be able to work properly and detect patters effectively. The algorithm will be able to learn only if the training data contain enough, meaningful features. Features are information about the data used to distinguish them with each other. Therefore, the most significant part is to develop and select the most useful features for the training data.

The trained model should have the ability to *generalize*, i.e. adapt to new, previously unseen data. To make sure that the model is working well, new cases should be tried out and the performance of the model should be evaluated based on those data. If the model is inadequate, its performance using the new data will be poor. For this purpose, the data should be divided into 2 sets, one for training and one for testing, such that the first one can be used to train the model and the second one to evaluate the performance of the model. Usually, $60\% - 80\%$ of the data is used for training and $40\% - 20\%$ for testing. In some cases, the data are split into 3 sets, where the third one is used for validation and tuning model's hyperparameters during training.

In ML, overgeneralization is called *"overfitting"*. This means that the model performs well when working with the training data, but fails to generalize on any other data, such that it is overfitted to the training set. This occurs when the model is too complex for the amount of data provided, or there is a large invariance between training and test data (e.g. imbalance or very different feature values between training and test set). In order to tackle this problem, the amount of training data should be increased, including also more diverse data in case of invariance, or a more simple model with less parameters should be selected. On the other hand, *"underfitting"* is the counterpart of overfitting. Underfitting occurs when the model is too simple to capture relationships between data features and a target variable, or, in other words, it is not able to learn. An underfitted model results in problematic outcomes on unseen data and often performs poorly even on training data. This problem can be encountered by choosing a more complex model.

## 2.4 RANDOM FOREST

Random Forest is a type of ML algorithm, belonging to the category of supervised learning, Hartshorn [2016]. It is typically used for classification and regression problems. In classification tasks, it is used to categorize data based on their features.

After training the model to classify the given data, the model should be able to predict classification results of other unseen data. Random Forest is a simple and handy ML algorithm that can be used for a wide range of applications and perform fairly good at all of them. However, it might not be as efficient as a specialized algorithm for a specific task.

Random Forest is actually a collection of *decision trees* generated using a random subset of data. The term "Random" comes from the randomness in selecting the subset of data, while the term "Forest" refers to the bunch of decision trees constructed during the training process. A decision tree is the process to decide which class the data belong to. Practically, it is a flow chart of questions the algorithm goes through to partition data. The algorithm starts at the top and continues downwards, with each tree in series, answering sequential questions. At each point, a decision should be made which way to choose between 2 options. Eventually, the algorithm reaches the bottom and comes to a conclusion regarding the outcome. In case the decision tree is good, unknown data can be used and follow the route to be classified correctly. But if it is not good enough, the data can take the wrong path and be assigned to the wrong category.

A Random Forest consists of multiple decision trees, each one generated slightly differently (see Figure 2.6). In order to generate the decision trees, a training set is necessary. The training set should contain some useful features of the data that are able to provide a clear distinction between the different classes, and the labels of the data that indicate which data belong to which class. When passing the features and labels of the data into the algorithm, it will evaluate the features and select those that will generate the best decision trees. Finally, Random Forest performs *majority voting* among all its trees to predict the output class for each input data, that is the class that the greatest part of trees have voted for. The data used for training and prediction should have the same number and type of features.

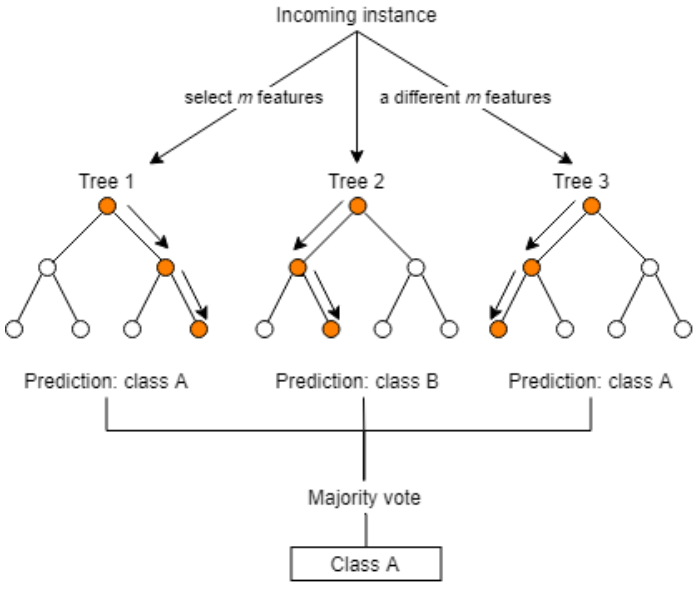



**Figure 2.6:** Random Forest: Decision trees generated to predict the final class label of the incoming instance. Here, 3 decision trees are constructed by selecting different features, and class A is finally assigned to the incoming instance after majority voting. (Image source: Thomas Wood [2020])

Random Forest is a powerful model on many different problems. It is fast, easy to interpret, performs well on large datasets, and is able to handle both categorical and numerical data. However, decision trees are prone to overfitting. To prevent overfitting, the depth of the decision trees can be controlled. Apart from the maximum depth, there are also other parameters that can be tuned, such as the number of

decision trees that will be constructed in the forest and the minimum number of samples required to split an internal node. Random Forest has already been applied for classification problems in several various fields. Nonetheless, although this method has been employed in a few studies related to railroads, it has not been implemented yet on point cloud data from railways.

## 2.5 DEEP LEARNING

Deep Learning (DL) is a subfield of ML for learning representations from data by learning successive *layers* of increasingly meaningful representations, Chollet [2017]. The term "deep" in deep learning refers to the multiple layers of representation, and the number of those layers in the model represents the depth of the model. The basic difference between deep learning and other machine learning approaches is that the latter mainly focus on 1 or 2 layers of representations of the data, i.e. shallow learning.

In deep learning, the layered representations are generally learned through models called **Neural Networks (NNs)**, where the layers are placed on top of each other (see Figure 2.7). Hence, deep learning is just like machine learning in mapping inputs to targets by observing many examples of inputs and targets, but using a deep sequence of simple data transformations (layers), where transformations are learned by exposure to examples. The transformation applied by each layer is parameterized by its *weights* (the parameters of the layer). In that case, learning means finding the optimal values for the weights of all the layers in the network to map inputs to targets as correctly as possible.

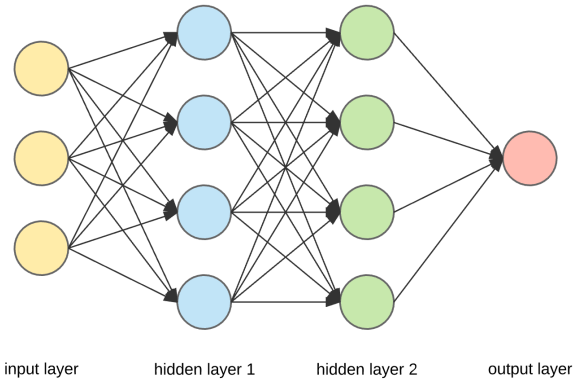



**Figure 2.7:** A fully connected neural network with 2 hidden layers. (Image source: Francesco Lelli [2019])

However, finding the best values of all the parameters at the same time is not simple. The *loss function* (also called objective function) is used to compute a distance score, using the predictions from the network and the true targets, that evaluates the performance of the network on this specific example. This score is then used as feedback to adjust the values of the weights so that the loss score will be decreased. This adjustment is done using the *optimizer* by implementing the *backpropagation* algorithm, which computes the gradient of the loss function with respect to the weights. Initially, random values are given to the network, leading to outputs far away from the targets, as well as high loss score. However, with every instance the network processes during training, the weights are updated closer to the correct direction and the loss score reduces. After repeating this procedure several times, the resulting set of weight values is the one that minimizes the loss function. Thus, the network is trained and the outputs are as close as possible to the targets.

**Convolutional Neural Networks (CNNs)** are similar to ordinary NNs, while using convolutions and pooling to reduce the number of parameters, allowing the

network to be deeper and with much less parameters, Aghdam and Heravi [2017]. *Convolution* is the operation where 2 functions are multiplied to produce a third function (see Figure 2.8). In image classification, 2D convolution is applied by using an input image and a kernel, that is a 2D matrix of weights that works as a filter. The kernel slides over the pixels of the input image, by performing element-wise multiplication with the part of the input it is on, and then summing up the products into a single output pixel. This process is repeated until all the locations in the image are covered, converting the spatial information from the 2D image to a 2D matrix of features, the feature map.

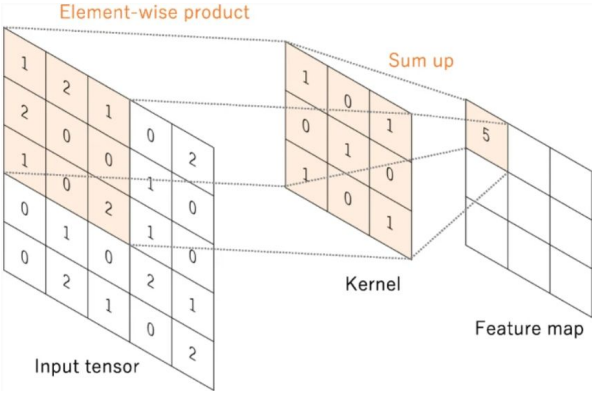



**Figure 2.8:** Convolution: The kernel slides over the input image, by performing element-wise multiplication and summing up the products to produce the feature map. Here, the convolution is applied using a $3 \times 3$ kernel. (Image source: Yamashita et al. [2018])

*Pooling*, also referred to as downsampling, is mainly used to reduce the dimensionality of feature maps. The features of the feature map generated by a convolution layer are summarized by the pooling layer. In that way, subsequent operations are then performed on the summarized features instead of the precisely positioned features created by the convolution layer, making the model more robust to variations in the position of the features in the input image. Pooling involves selecting a pooling operation to be applied to feature maps. The two most common pooling operations are: *max-pooling* and *average-pooling* (see Figure 2.9). In max-pooling, the maximum value for each patch of the feature map is selected and for average-pooling the average value is calculated.

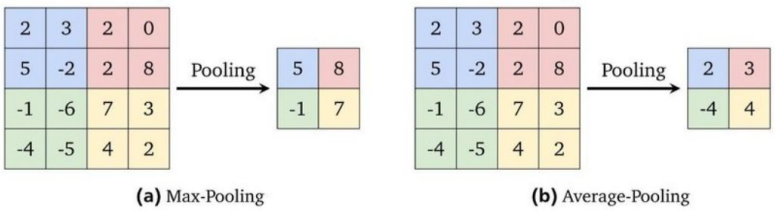



**Figure 2.9:** (a) Max-pooling and (b) Average-pooling: The kernel calculates the (a) maximum and (b) average value for each patch of the feature map. Here, a $2 \times 2$ kernel is used for both operations. (Image source: Guissous [2019])

Nonetheless, 3D point clouds differ from 2D images, since they are fundamentally irregular, unstructured and unordered. In order to apply CNNs to 3D point cloud data similarly to images for classification tasks, there are methods called *projection-based* methods, which first project the unstructured point cloud into a regular representation, and then apply 2D or 3D convolution to classify the data, Guo et al. [2020]. In projection-based methods, there are multi-view-based methods, that project an unstructured point cloud into 2D images, and volumetric-based methods that convert a point cloud into a 3D volumetric representation (3D grid). Other projection-based

methods, specifically for semantic segmentation tasks, are methods using spherical, permutohedral lattice or hybrid representation of point clouds. Lately, *point-based* methods have gained ground on DL applications on point clouds, designed to directly manipulate raw point clouds, without first passing them to an intermediate regular representation using voxelization or projection. Point-based networks include pointwise Multi-Layer Perceptron (MLP) networks, convolution-based networks, graph-based networks and other networks. PointNet, Qi et al. [2017], is a pioneer method in deep learning on raw point clouds, that belongs to the MLP methods.

**PointNet** is a unified architecture that directly inputs point clouds and outputs either class labels for the whole input (shape classification) or per point labels for each input point (shape segmentation). In the simplest form, each point is represented by just the $XYZ$ coordinates, while additional dimensions can be added, including local and global features. Key to this approach is the use of max-pooling as a single *symmetric function*. The use of a symmetric function aims to maintain permutation invariance, since its value is the same regardless the order of the arguments. The network learns a set of optimization functions that choose informative points, while encoding the reason for their selection. The final fully connected layers of the network use these optimal values to predict per point labels for shape segmentation or aggregate them into a single *global descriptor* that encodes the entire input 3D object for shape classification. In the semantic segmentation task, PointNet achieved 78.62% OA on Stanford large-scale 3D Indoor Spaces (S3DIS), an indoor benchmark dataset with 13 classes.

The majority of the top-of-the-line DL classification methods achieve state-of-the-art Overall Accuracy (OA) of more than 90% on ModelNet40, Guo et al. [2020]. ModelNet40 is a Computer-Aided Design (CAD) model dataset, which consists of 40 man-made object categories for shape classification and recognition. In the respective semantic segmentation methods, the OA mostly drops below 90% on S3DIS. However, the applicability of the methods depends on the specific task. For indoor environments, shape classification is preferred over semantic segmentation, since there are objects that are mainly discrete, such as chairs, tables and bookcases in a room. On the flip side, for outdoor environments, semantic segmentation is usually preferred. This is because the objects that can be found in an outdoor environment are mainly continuous, rather than discrete, and extend over a wide area. For instance, rails, sleepers and overhead wires are some of such objects on a railroad.

## 2.6 DGCNN

Dynamic Graph Convolutional Neural Network (DGCNN), Wang et al. [2019b], is a graph-based network, inspired by PointNet and convolution operations. Unlike PointNet that works on individual points, DGCNN exploits local geometric structures, as it applies convolution operations on the edges of neighboring pairs of points and constructs a local neighborhood graph. PointNet operates on each point independently using a series of MLPs to acquire features in a higher dimension. It then applies a symmetric function to accumulate features from all points into an 1-Dimensional (1D) global feature vector, maintaining permutation invariance. In the segmentation model, the global descriptor is concatenated with the local descriptors, that is the features generated from each MLP, to compute the classification scores for $p$ classes. Nonetheless, PointNet processes points individually, without taking into account local neighborhood information.

There are several extensions of PointNet that improve the performance of the original model by considering neighborhoods of points, instead of acting on each point independently, allowing the network to exploit local features. Mainly, in order to maintain permutation invariance, these methods treat points independently at local scale, while neglecting the geometric relationships among points. *Edge Convolution (EdgeConv)* is an operation that captures local geometric structure and maintains

permutation invariance (see Figure 2.10). It generates edge features which describe the relationships between a point and its neighborhood, rather than generating point features directly from their embeddings. EdgeConv is permutation invariant, since it is made to be invariant to the ordering of neighbors. Since EdgeConv constructs a local graph (*k-Nearest Neighbors, k-NN, graph*) and learns the embeddings for the edges, it is capable of grouping points in Euclidean space, as well as in semantic space. EdgeConv can be integrated into existing deep learning models to enhance their performance. It has already been integrated into the basic version of PointNet, while excluding feature transformation, and the deep learning method produced is called DGCNN.

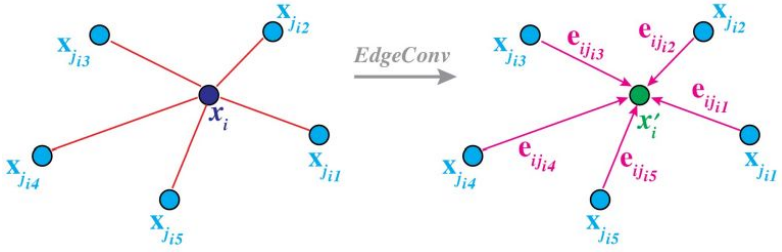



**Figure 2.10:** EdgeConv: The output is calculated by aggregating all the edge features associated with the edges coming from each connected vertex, using max-pooling symmetric function. Here, the k-NN graph is constructed using $x_i$ vetrex and 5 neighbors, and $x'_i$ is the resulting output. (Image source: Wang et al. [2019b])

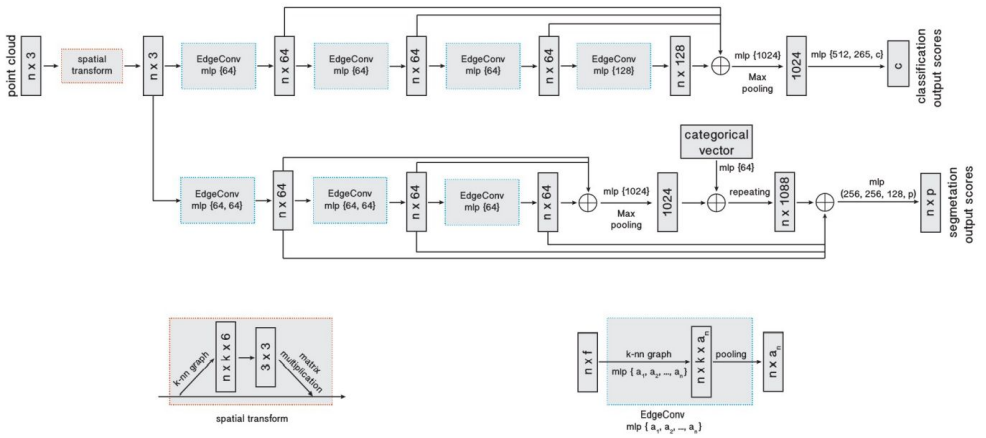



**Figure 2.11:** DGCNN: The model architectures for classification (top branch) and segmentation (bottom branch).
The *classification model* inputs $n$ number of points, calculates $k$ edge features for each point at each EdgeConv layer, and aggregates those features for each point to compute the corresponding EdgeConv responses. The features output from the last EdgeConv layer are then aggregated into an 1D global descriptor. The global descriptor is used to calculate the classification scores for $c$ classes.
The *segmentation model* extends the classification model by concatenating ($\bigoplus$) the global descriptor and the local descriptors (the EdgeConv outputs) for each point to compute the classification scores per point for $p$ semantic labels.
The *point cloud transform block* applies an estimated $3 \times 3$ matrix in order to align the input point set to a canonical space. To estimate the matrix, a tensor is used that concatenates the coordinates of each point and the coordinate differences between the point and its $k$ nearest neighbors.
The *EdgeConv block* inputs an $n \times f$ tensor, computes the edge features for each point using an MLP with the number of layer neurons being $\{a_1, a_2, ..., a_n\}$, and creates an $n \times a_n$ tensor using max-pooling among neighboring edge features. (Image source: Wang et al. [2019b])

In contrast to graph CNNs that are fixed, the graph in DGCNN is dynamically updated after each layer of the network. Within the network, the set of $k$-nearest neighbors of a point is computed from the sequence of embeddings and changes from layer to layer. Proximity in feature space is different than proximity in the input, which causes non-local diffusion of information throughout the point cloud. The architecture of DGCNN is presented in Figure 2.11. The present research makes use of the semantic segmentation model for point-by-point classification. Using this model, DGCNN achieved 84.1% OA on the indoor benchmark S3DIS.

Although there are no studies yet applying DGCNN on railways, other researchers have already extended its semantic segmentation application from indoor to outdoor scenes, using aerial point cloud data from urban environments. In this context, Widyaningrum et al. [2020] applied DGCNN for pointwise semantic segmentation of an airborne point cloud from the Surabaya city of Indonesia into 4 land cover categories of bare-land, trees, buildings and roads. In that study, 3 different tailor-made feature combinations were employed, that is RGB, original LiDAR features (intensity, return number and number of returns), and 2 spectral colors and intensity (red, green and intensity), and achieved OA of 81.05%, 76.13% and 79.81%, respectively. Also, Bai [2020] implemented DGCNN to semantically segment the aerial point cloud dataset of the Current Elevation File Netherlands (AHN), and specifically of AHN3, into 4 classes of ground, building, water and others. Moreover, the impact of the effective range was investigated by adjusting the 2 parameters of block size and neighborhood size, and concluded that enlarging the block size improves the OA but cannot ensure better per class segmentation results. Finally, using a block size of 50 m and 20 neighbors $k$, the best OA of 93.28% was achieved, while the accuracy was further improved by combining segmentation results from smaller scale (block size of 30 m) with those from larger scale (block size of 50 m) and using 20 neighbors.

## 2.7 CLASSIFICATION OF RAILWAY POINT CLOUDS

The classification of point cloud data of railway environments has only gained attention the past years. Some of the studies attempted to classify entire railway point clouds into several railway components. In this context, Luo et al. [2014] presented a context-based recognition method of multiple railway objects using MLS data. The objective of that research was to apply the Conditional Random Field (CRF) classifier to automatically classify railway objects from MLS data into terrain, railway bed, railway tracks, vegetation, platform, noise barriers, electronic posts, attachments on electronic post and power lines. The results of 2 classification methods of Gaussian Mixture Model - Expectation Maximization (GMM-EM) and CRF, with and without contextual information, were presented for comparison purposes, and the study concluded that the CRF with contextual feature performed the best with 94.43% OA. However, the contextual features used were generated at 4 different levels of point level, line level, volumetric space level and vertical profile level, and the context classification developed in that work is sensitive to the quality of point cloud data. Later, Sánchez-Rodríguez et al. [2018] proposed a methodology for the detection and decomposition of railway tunnels, using 3 different MLS datasets. The data were classified into 6 categories of ground, rails, lining, catenary wires, cantilevers and other elements. The methodology comprised the preprocessing of each point cloud, dividing it into ground and non-ground points, and detecting the elements present in these clouds, using several algorithms including Random Sample Consensus (RANSAC). The point-by-point classification of the tunnel elements returned F1-score values between 71% and 99%.

Other researchers mainly focused on identifying specific objects on railways using point clouds. In this sense, Arastounia [2012] implemented Template Matching (TM) and Region Growing (RG) method to classify 2 3D LiDAR point cloud datasets of railway environments into 3 classes of rails, contact wires and catenary wires. Both TM

and RG were applied on the terrestrial dataset, and since the results from RG were better than those from TM, only RG was applied on the airborne dataset. Overall, RG was proved to be faster and more efficient than TM, but no evaluation was implemented, since there were no reference data. Complementing that study, Arastounia and Oude Elberink [2016] improved the previous work of clustering urban railroad point clouds into rail tracks, contact cables, and catenary cables by employing TM again. The MLS dataset used included only geometrical information, and no intensity or RGB data, and the algorithm applied was based on no assumptions or prior knowledge. In the context of that work, they first applied a coarse classification of the data into points containing rail tracks, those containing contact cables and those containing catenary cables, based on their height. After that, points belonging to rail tracks were identified and points belonging to overhead cables were classified. Finally, the method achieved 97.7% average accuracy and 97.3% average precision. Analytically, the rail tracks were classified with 96.5% accuracy and 98.4% precision, and the cables with 98.4% average accuracy and 96.8% average precision. Following his previous works, Arastounia [2015] developed automated methods for recognizing rail tracks, contact cables, catenary cables, return current cables, masts, and cantilevers from 3D LiDAR data. The recognition of key components of the railroad infrastructure was implemented based on their physical shape, geometrical properties, and topological relationships. The methods achieved 100% accuracy and precision at object level, as well as 96.4% average accuracy and 97.1% average precision at point level.

With regard to the supporting structure and overhead wires of railroad infrastructures, some authors developed different methodologies on classifying railway point clouds into different components of the supporting structures or power cables. In this relation, Sánchez-Rodríguez et al. [2019] suggested a methodology to automatically inspect railway tunnels' power lines using LiDAR point clouds from a mobile mapping system, containing classification of points, distances measurement and standards' compliance. The data were classified into contact wires, suspension wires, and remaining points. The validation of the proposed methodology was carried out under 3 different tunnel point clouds and the methodology achieved F1-scores of more than 93% for the classification of both contact wires and suspension wires in all 3 datasets. Moreover, Corongiu et al. [2020] aimed to classify railway assets, using mobile mapping point clouds from 2 areas of the Italian railway system. A set of candidates were first extracted and preprocessed, and then a 3D modified Fisher vector-deep learning neural net was used in order to classify the candidates into portals, cantilevers and other. The results showed that 100% of portal points, 86.2% of cantilever points and 97.5% of other points were classified correctly.

The novelty of this research relies on the use of 2 machine learning methods of Random Forest and DGCNN to classify the whole railway environment at one go. Additionally, the methods are validated under 3 case studies. While the majority of works done employ point cloud data generated mainly from MLS techniques, or TLS and ALS techniques, the present research incorporates also data produced by photogrammetry.

## 2.8 SUMMARY

Chapter 2 comprised the theoretical background related to this research. Analytically, the chapter included information regarding monitoring railways, point clouds and acquisition techniques, machine learning, Random Forest, deep learning, DGCNN, and classification of railway point clouds. In railway monitoring, the main railway components of rails, sleepers, track bed, overhead wires, masts, trees and other, required to be identified, were analyzed. In point clouds and acquisition techniques, the definition and properties of point clouds were given. Also, the acquisition techniques of remote sensing and photogrammetry, commonly used to collect point cloud data, were described, and the techniques of TLS, UAV-LiDAR and UAV-SfM, used in this work,

were compared to each other. In machine learning, the basic principals of ML were analyzed, and the different types of ML along with some common ML algorithms were presented. In Random Forest, a brief explanation of the structure and way this ML algorithm works was given. In deep learning, the basic principals of this subfield of ML were mentioned, including NNs, CNNs, and deep learning methods used for point cloud data. In DGCNN, the Dynamic Graph Convolutional Neural Network was explained and compared to PointNet that was based on, as well as, the model architectures for classification and segmentation were presented, along with semantic segmentation applications of DGCNN implemented so far. Finally, in classification of railway point clouds, studies related to classification of point clouds from railways were summarized and presented shortly. The next chapter, Chapter 3, presents the data and tools used in this work. In data, the datasets employed, along with the properties of the datasets and their defects and missing data are analyzed, while the tools contain the software and hardware used in the present study.

# 3 DATA AND TOOLS

Chapter 3 includes a brief analysis of the data and tools used in this study, in Section 3.1 and Section 3.2, respectively. Finally, Section 3.3 comprises a summary of this chapter, along with a short introduction to the following chapter.

## 3.1 DATA

Section 3.1 is split into the description of the datasets in Subsection 3.1.1, their properties in Subsection 3.1.2, and their defects and missing data in Subsection 3.1.3.

### 3.1.1 Datasets

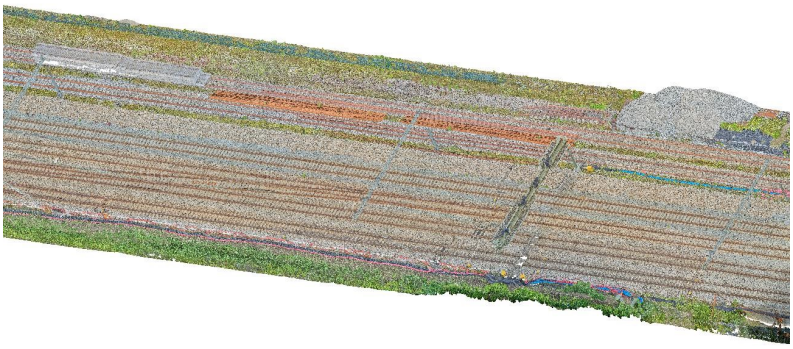
There are 3 types of 3-Dimensional (3D) point cloud data employed in this work (see Figure 3.1). The data come from 2 different areas in the Netherlands, i.e. Groningen and Stroe, and 3 different sources, i.e. Structure from Motion (SfM) photogrammetry, Airborne Laser Scanning (ALS) and Terrestrial Laser Scanning (TLS). The 3 datasets are called Unmanned Aerial Vehicle (UAV)-SfM, UAV-Light Detection And Ranging (LiDAR) and TLS. The collection and processing of the data described below has already been conducted by other projects within CGI and provided to be used for the current research.

**Dataset 1. UAV-SfM: 3D point cloud produced by structure from motion photogrammetric method using photos collected from a drone.**
The area of interest is a region of around 250 m long and 40 m wide, in the area of Groningen. A DJI MATRICE 600 Pro drone and a Phase One iXM-100 camera with a resolution of $11664 \times 8750$ pixels were used to obtain a set of 1169 images. All the images along with 4 Ground Control Points (GCPs) were processed in the photogrammetry software Pix4D, that makes use of the SfM method, to produce a 3D point cloud of $628,167,660$ points in total. The data produced are georeferenced and represented by the $XYZ$ Rijksdriehoeks (RD) coordinates of the points along with their Red Green Blue (RGB) values. RD coordinates consist of a projected Cartesian coordinate system used at national level for the European Netherlands.

**Dataset 2. UAV-LiDAR: 3D point cloud collected from a RiCOPTER LiDAR drone.**
A RIEGL RiCOPTER VUX-SYS airborne laser scanning system was used to collect 3D point cloud data in the area of Stroe. The system consists of a laser scanner (RIEGL), a Global Navigation Satellite System (GNSS) receiver (ANTCOM), an Inertial Measurement Unit (IMU) (Applanix) fixed to the laser scanner, and two camera sensors (Sony Alpha 6000). During the data acquisition, a total of 3 flights was executed at a flying height of around 40 meters relative to the rail tracks. For this project, the scans from the first 2 flights were provided, with 9 scans per flight and a total of $170,078,607$ points from the first flight and $182,108,012$ points from the second flight. The point clouds include a railway of around 460 m long, together with an extensive area of the surrounding environment. They are georeferenced, and contain the $XYZ$ RD coordinates of the points, along with the RGB values, as well as other attributes, such as intensity, number of returns etc.

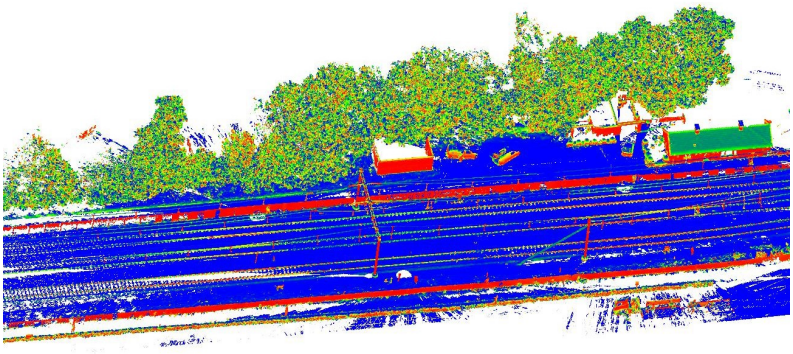
**Dataset 3. TLS: 3D point cloud collected from a terrestrial laser scanner.**
The point cloud was collected using a Leica ScanStation P30 laser scanner. It consists

of 39 scans, which were merged, so the final point cloud consists of 414, 601, 670 points in total. It covers an area of 340 m along track and between 40 and 50 m perpendicular to the track, in the area of Stroe. The point cloud is georeferenced, and contains the $XYZ$ RD coordinates of the points.



**(a)** Dataset 1 - UAV-SfM Groningen

**(b)** Dataset 2 - UAV-LiDAR Stroe



**(c)** Dataset 3 - TLS Stroe

**Figure 3.1:** Data: (a) Dataset 1 - UAV-SfM Groningen, (b) Dataset 2 - UAV-LiDAR Stroe, (c) Dataset 3 - TLS Stroe. Here, only a part of each dataset is illustrated.

### 3.1.2 Properties

The main properties of the datasets are summarized in Table 3.1. The properties are related to the techniques used to collect the datasets, the areas of interest, the attributes of the point clouds, the total number of points and total length of the railways in the datasets, as discussed in Subsection 3.1.1. Additionally, the mean

surface density, mean volume density, and mean point spacing of the point clouds are included in the properties.

**Table 3.1:** Datasets and properties.

| Properties | Dataset 1 | Dataset 2 | Dataset 3 |
|---|---|---|---|
| **Acquisition technique** | UAV-SfM | UAV-LiDAR | TLS |
| **Area** | Groningen | Stroe | Stroe |
| **Attributes** | $X, Y, Z, R, G, B$ | $X, Y, Z, R, G, B$ | $X, Y, Z$ |
| **No of points** | $628,167,660$ | $352,186,619$ | $414,601,670$ |
| **Railway length** $[m]$ | 250 | 460 | 340 |
| **Mean surface density** $[pts/m^2]$ | $44,750$ | $2,363$ | $4,259$ |
| **Mean volume density** $[pts/m^3]$ | $1,678,121$ | $88,624$ | $159,698$ |
| **Mean point spacing** $[m]$ | 0.005 | 0.021 | 0.015 |

The most prominent property of a point cloud is its *point density*. Point density is an indicator of the resolution of the data, so higher density means more information and, therefore, higher resolution, whereas lower density means less information, i.e. lower resolution. Similar to the resolution of a photograph, the point density of a point cloud defines the amount of measurements per area. Typically, "point density" refers to *surface density*, while *volume density* is a similar metric of point density that considers volume areas rather than surface areas. Surface density is calculated for each point in the point cloud as the number of neighbors $N$ within a neighborhood radius $R$ divided by the neighborhood surface (Equation 3.1), and the volume density as the number of neighbors $N$ within a neighborhood radius $R$ divided by the neighborhood volume (Equation 3.2).

$$\text{Surface density} = \frac{N}{\pi R^2} \tag{3.1}$$

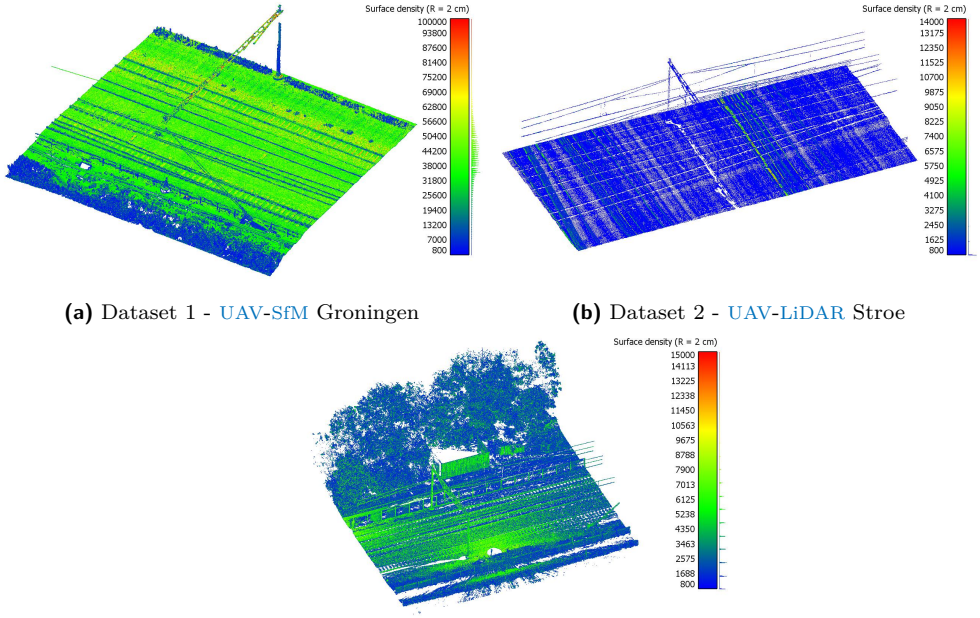
$$\text{Volume density} = \frac{N}{\frac{4}{3}\pi R^3} \tag{3.2}$$

The mean surface density of a point cloud is the mean of the surface densities of all the points $p$, and the mean volume density is the mean of the volume densities of all the points $p$. Since we are referring to point clouds and not to images, there will be empty space between measurements. Hence, another metric used to define the quality of a point cloud is the distance from point to point, the so-called *point spacing*. Point spacing is calculated for each point as the square root of the inverse point density (surface density) (Equation 3.3), and the mean point spacing is the square root of the inverse mean surface density.

$$\text{Point spacing} = \sqrt{\frac{1}{\text{Surface density}}} \tag{3.3}$$

In this work, a radius $R = 2$ cm is used to calculate the mean surface density, mean volume density and mean point spacing of each of the 3 datasets, as an indicator of the quality of the different data. Figure 3.2 illustrates the point density (i.e. surface density) for a part of each dataset. Taking into account the aforementioned metrics,

it is clear that the first dataset has around 10 times higher resolution than the third dataset (10 times more points per surface area/ volume area) and almost 20 times higher resolution than the second dataset, while the third dataset has about 2 times higher resolution than the second dataset. The different resolution of the data is an important factor that can affect the quality of classification results.




**(a)** Dataset 1 - UAV-SfM Groningen

**(b)** Dataset 2 - UAV-LiDAR Stroe

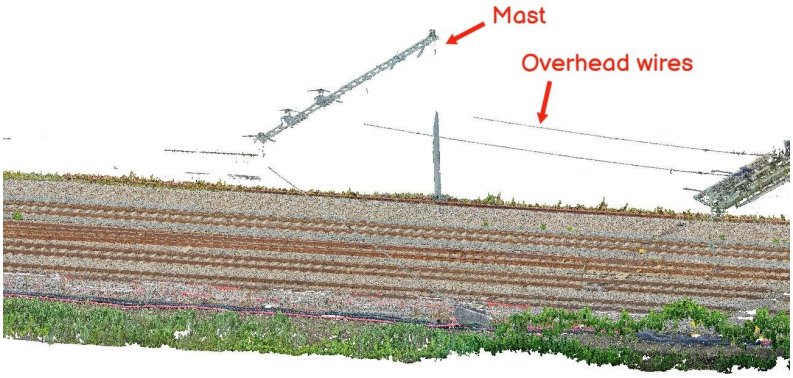
**(c)** Dataset 3 - TLS Stroe

**Figure 3.2:** Point density (surface density) using $R = 2$ cm: (a) Dataset 1 - UAV-SfM Groningen, (b) Dataset 2 - UAV-LiDAR Stroe, (c) Dataset 3 - TLS Stroe. Here, only a part of each dataset is illustrated.

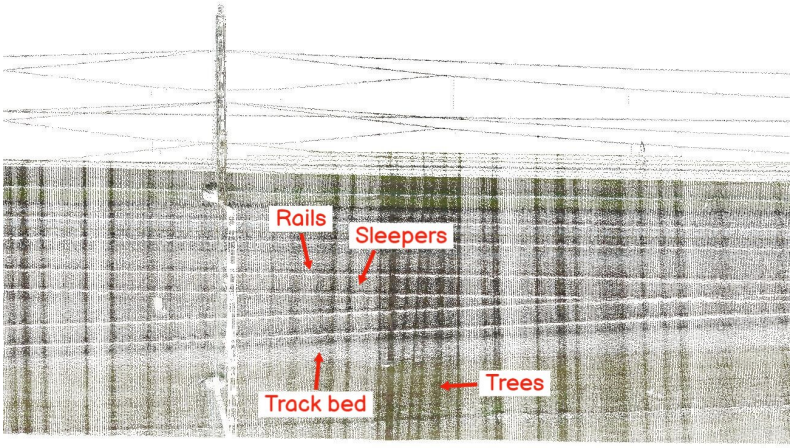
### 3.1.3 Defects and missing data

In each of the 3 datasets, some of the components are representative of the original railway components, whereas other present some defects and missing data. Particularly, in the first dataset, the overhead wires are not well reconstructed, while the masts miss some parts of their structure. The second dataset shows the most faults out of all the datasets used in this work, since the rails go zigzag and sometimes present mismatches, the sleepers are not visible, and the track bed and trees contain stripes of missing data. In the third dataset, some sleepers and some parts of the track bed include gaps of data, while the trees are much bigger than the ones in the other 2 datasets.

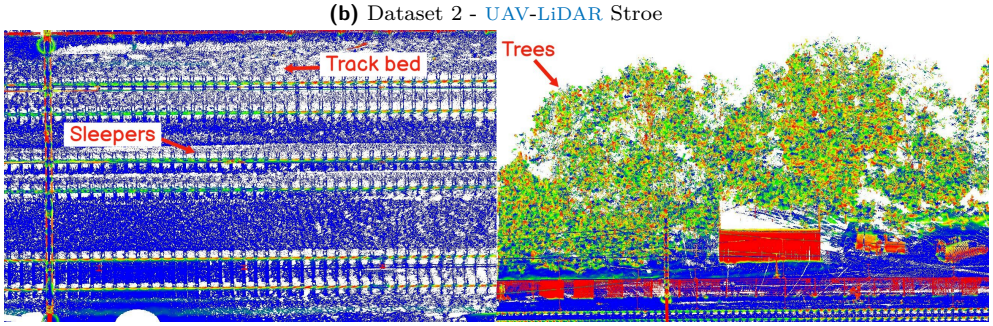
The different quality of the components in the different datasets is another significant factor that can influence the quality of classification results. Table 3.2 summarizes, for each component and each dataset, whether the components are well represented, as well as the defects and missing data of those which present issues, and Figure 3.3 displays some instances of the components with faults and missing data for each dataset.




**(a)** Dataset 1 - UAV-SfM Groningen




**(b)** Dataset 2 - UAV-LiDAR Stroe




**(c)** Dataset 3 - TLS Stroe

**Figure 3.3:** Components with defects and missing data: (a) Dataset 1 - UAV-SfM Groningen, (b) Dataset 2 - UAV-LiDAR Stroe, (c) Dataset 3 - TLS Stroe. Here, only some instances are given.

**Table 3.2:** Assessment of the quality of each component in each dataset. If the component is well represented, ✓is used. Else, the defect or missing data is reported.

| Class | Dataset 1 | Dataset 2 | Dataset 3 |
|---|---|---|---|
| **Rails** | ✓ | Present mismatches and go zigzag. | ✓ |
| **Sleepers** | ✓ | Not visible. | Include gaps of data. |
| **Track bed** | ✓ | Contains stripes of missing data. | Includes gaps of data. |
| **Masts** | Not fully reconstructed. | ✓ | ✓ |
| **Overhead wires** | Not well reconstructed. | ✓ | ✓ |
| **Trees** | ✓ | Contain stripes of missing data. | Much bigger. |
| **Other** | ✓ | ✓ | ✓ |

## 3.2 TOOLS

Section 3.2 is divided into Subsection 3.2.1, which contains the software used to accomplish this study, and Subsection 3.2.2 containing the hardware used.

### 3.2.1 Software

The entire project is implemented in `Python` programming language, using `Python 3.7.4` in the local computer and `Python 3.6.8` in the High Performance Computing (HPC) cluster. The libraries used during the project are `laspy` for reading the data provided in .las format, `numpy`, `pandas` and `scipy` for numerical, dataframe and spatial operations, respectively, and `os`, `sys`, `glob`, `shutil` and `natsort` for managing the files paths along with the files. Also, laserchicken is used for computing the echo ratio features, `scikit-learn` and `pickle` for classification processes, `argparse` for command-line interfaces, `tqdm` for following the progress of loops, and `matplotlib` and `seaborn` for visualizing the results.

Dynamic Graph Convolutional Neural Network (DGCNN) has 3 implementations for semantic segmentation tasks, 1 in `Tensorflow` provided by the authors of DGCNN, and 2 in `PyTorch` by other users. The present research makes use of the `PyTorch` implementation by An Tao, that achieves significant better results on the indoor benchmark Stanford large-scale 3D Indoor Spaces (S3DIS) than the authors' `Tensorflow` implementation (Yue Wang et al.), as reported by the authors of DGCNN, Wang et al. [2019b]. Specifically, `PyTorch 1.7` is used for the DGCNN implementation.

Furthermore, the open source 3D point cloud and mesh processing software, `Cloud Compare`, is used for labeling, subsampling and cropping the point clouds, calculating the geometric features, and visualizing the results in more detail.

### 3.2.2 Hardware

All the processes are first tested locally with a small amount of data on an MSI Prestige P65 Creator 8RD laptop using the Spyder Integrated Development Environment (IDE). Afterwards, they are fully implemented on the remote HPC cluster of TU Delft. Dur-

ing the training process on the cluster, 2 Central Processing Units (CPUs) are used for the Random Forest implementation given that more memory is required, and 2 Graphics Processing Units (GPUs) and Compute Unified Device Architecture (CUDA) for GPU acceleration for the DGCNN implementation. The specifications of the machines, including the CPU, GPU, memory and Operating System (OS), are as follows.

**MSI Prestige P65 Creator 8RD**

- **CPU:** Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- **GPU:** NVIDIA GeForce GTX 1050 Ti
- **Memory:** 16GB
- **OS:** Windows 10 Home 64-bits

**HPC cluster**

An HPC cluster is a cluster of servers, where each server is commonly referred to as a node. The scheduler prioritizes the submitted to the cluster jobs, and assigns them to compute nodes accordingly.

- **CPU:** Multiple CPUs on all nodes.
- **GPU:** Some nodes also have GPUs which support the CUDA platform. Three different types of GPUs are available:
  - NVIDIA Tesla P100
  - NVIDIA GeForce GTX 1080 Ti
  - NVIDIA GeForce RTX 2080 Ti
- **Memory:** Large main memories on all nodes.
- **OS:** Linux CentOS 7 64-bits

**Compute nodes**

| Node | Features | CPUs | Cores | Memory |
|---|---|---|---|---|
| ess-[1-4] | ht, avx, avx2 | 32 | 16 | 128 Gb |
| gauss, markov, neumann | ht, avx | 32 | 16 | 192 Gb |
| grs-[1-4] | ht, avx, avx2 | 32 | 16 | 256 Gb |
| watson | ht, avx | 32 | 16 | 256 Gb |
| maxwell, parzen, viterbi | ht, avx, avx2 | 56 | 28 | 256 Gb |
| hopper | ht, avx, avx2 | 64 | 32 | 512 Gb |
| 100plus | ht, avx, avx2 | 64 | 32 | 768 Gb |
| insy-node[01-06] | ht, avx, avx2 | 64 | 32 | 256 Gb |

**Figure 3.4:** HPC cluster nodes. (Image source: TU Delft)

In order to connect remotely from the local computer to the HPC cluster, run commands and submit jobs, the open source program `PuTTY` is used. Secure Shell (SSH) is a network protocol that gives users a secure way to access a computer over an unsecured network, while an SSH client program is used for remotely logging in and performing operations on computers running the Linux operating system. `PuTTY` is one of the graphical SSH client programs that can be used on Windows. Apart from `PuTTY`, the free software `FileZilla` is used for transferring files from the local computer to the HPC cluster and vice versa. It uses Secure Copy Protocol (SCP) which transfers files between computers using the SSH File Transfer Protocol (SFTP), a file protocol for transferring large files over the web.

## 3.3 SUMMARY

Chapter 3 contained the data and tools employed in this work. In data, the 3 different point cloud datasets of UAV-SfM Groningen, UAV-LiDAR Stroe and TLS Stroe were presented and visualized, including the collection procedure and final product of each dataset. Moreover, the properties of the datasets were analyzed, containing the acquisition techniques used to collect the data, the areas of interest, the attributes and

total number of points of the points clouds, and the length of the railways mapped, while focusing mainly on the mean surface density, mean volume density and mean point spacing of the point clouds. Additionally, the defects and missing data of the different railway components in each dataset were mentioned and visualized. In tools, the software used to accomplish this project was described, containing the `Python` libraries, DGCNN implementation and software programs employed. Furthermore, the hardware utilized in this work was explained, including the specifications of the local computer and HPC cluster utilized, together with the programs used to remotely connect from the local computer to the HPC cluster, execute jobs and transfer files between the machines. The following chapter, Chapter 4, shows the methodology followed to achieve this work. In particular, it contains the preprocessing of the data, the methods and evaluation metrics used, and the combination of methods implemented. Preprocessing comprises the classes, training-test splitting, labeling, subsampling, and other datasets. In methods, the Random Forest and DGCNN implementations are presented, while the evaluation metrics contain the classification metrics and predicted probabilities computed to assess the methods.

# 4 METHODOLOGY

Chapter 4 describes the methodology followed in this study to perform point-by-point classification of colored and uncolored 3D point clouds of railroads, using the 2 different Machine Learning (ML) methods of Random Forest and Dynamic Graph Convolutional Neural Network (DGCNN). Figure 4.1 shows the flowchart of the methodology, including the input data used, the different processes implemented and the output data resulted from the processing. The main goal of this work is to perform point-wise classification of 3 different point cloud datasets of railways using an ensemble approach, that is Random Forest, and a Deep Learning (DL) approach, that is DGCNN. The first phase includes the preprocessing of the data. Initially, the classes representing the dominant components on a railway are chosen to be detected. Then, the first dataset (Unmanned Aerial Vehicle (UAV)-Structure from Motion (SfM) Groningen) is used for training and testing the 2 different methods, and 2 different areas within this dataset are used 1 as training set and 1 as test set. The training and test sets are then labeled manually and subsampled to save computational time and power.

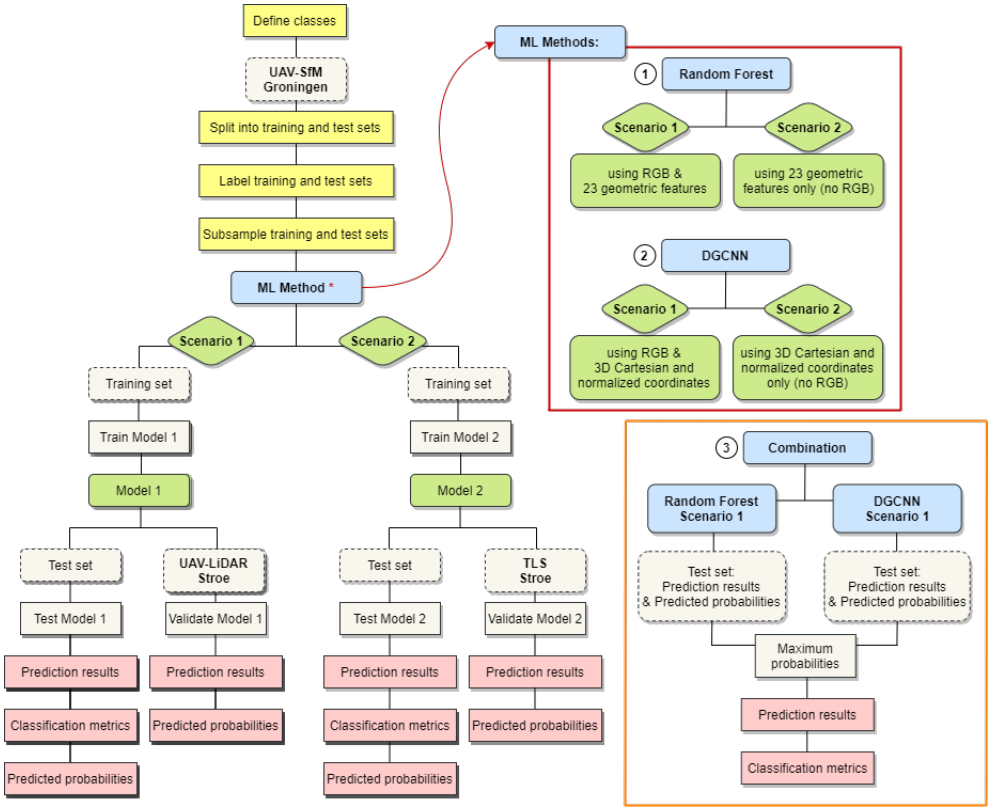



**Figure 4.1:** Flowchart of methodology: Input data are shown in beige - dashed border boxes, preprocessing in yellow, methods in blue, scenarios in green, intermediate processes in beige - solid border boxes, and results in pink. The red box shows the 2 ML methods and 2 scenarios implemented, and the orange box the combination of Random Forest and DGCNN for scenario 1.

In the next phase, firstly Random Forest and then DGCNN are applied to perform the classification. In both methods, the training set is used, once including Red Green Blue (RGB) values and once excluding them, to train the model. In the first

case, the test set is then used with RGB to test the performance of the trained model and evaluate it based on the computed classification metrics. Afterwards, the trained model is also used to predict the second dataset (UAV-Light Detection And Ranging (LiDAR) Stroe), which is unlabeled and contains RGB values. Its performance on that different dataset is evaluated visually. In the second case, the test set is used without RGB to test the performance of the trained model and its performance is evaluated again based on the classification metrics. After that, the trained model is used to predict also the third dataset (Terrestrial Laser Scanning (TLS) Stroe), which is unlabeled but does not contain RGB values, and evaluate its performance on that dataset visually. The predicted class probabilities are also calculated for all predicted results, in order to examine the uncertainty of the models regarding the output classes.

Particularly, in the first scenario of Random Forest the RGB values are used together with 23 geometric features, while in the second scenario only the geometric features are considered. On the other hand, in the first scenario of DGCNN, the 3D Cartesian coordinates of the points are used along with the color values and the normalized 3D coordinates of the points, whereas in the second scenario only the Cartesian and normalized coordinates of the points are taken into account. In the final phase, the prediction results from Random Forest and DGCNN are combined to further improve the final results. The combination is applied only to scenario 1, as a first try. There, the predicted label for each point is picked either from Random Forest or from DGCNN, depending on which method predicted the label with higher probability, and the respective classification metrics are computed.

Chapter 4 is organized as follows: Section 4.1 describes the preprocessing of the data, Section 4.2 presents the methods implemented, Section 4.3 refers to the evaluation metrics used to evaluate the performance and uncertainty of the trained models, Section 4.4 explains the combination of methods, and Section 4.5 contains a summary of this chapter, together with a short introduction to the upcoming chapter.

## 4.1 PREPROCESSING

Section 4.1 is divided into the following subsections: Subsection 4.1.1 presents the classes selected to represent the dominant components on railways, Subsection 4.1.2 shows the training-test splitting procedure, Subsection 4.1.3 refers to the labeling procedure, Subsection 4.1.4 analyzes the subsampling procedure, and Subsection 4.1.5 presents the other datasets employed.

### 4.1.1 Classes

The key components in a railway environment that would be interesting to detect are

**Table 4.1:** Classes and colors used to represent them.

| Class | Color |
|---|---|
| Rails | Blue |
| Sleepers | Red |
| Track bed | Gray |
| Masts | Purple |
| Overhead wires | Yellow |
| Trees | Green |
| Other | Black |

the rails, sleepers, track bed, masts and overhead wires, as well as surrounding trees and other nearby objects, as discussed in Section 2.1. Table 4.1 shows the 7 classes selected for the classification of railroads, together with the colors used to represent each class in the classification tasks.

### 4.1.2 Training-test splitting

In order to split the data into training and test sets, there are different methods used, such as *random*, *spatial*, *temporal* splitting. In random splitting, the training and test sets are generated randomly, by assigning a percentage of random samples to the training set and the remaining samples to the test set, totally disregarding the distribution or proportions of the classes in the sets. Typically, in this method, $60\% - 80\%$ of the samples are used for training and $40\% - 20\%$ for testing.

While random splitting is indeed the best approach for many ML problems, it is not always the right solution. This depends on the type and nature of data. For spatial data, random splitting results in training and test points very close to each other. This leads to leaking information from the training to the test set, meaning that the 2 sets share information, and evaluating the performance of the model based on the test set will not provide reliable results.

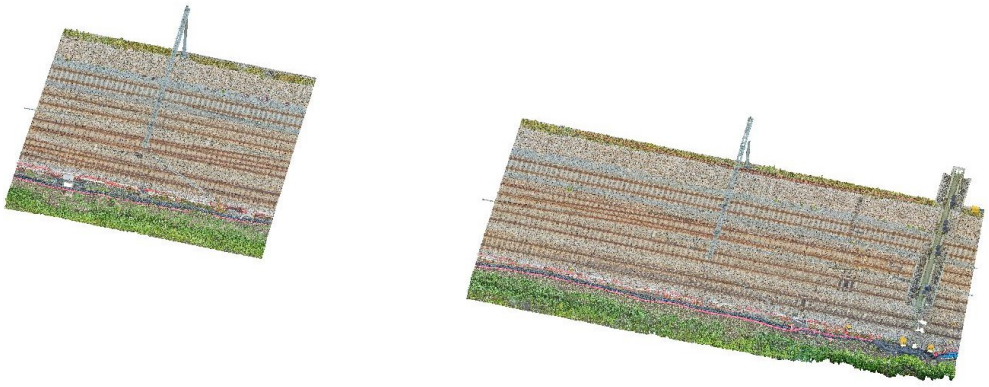


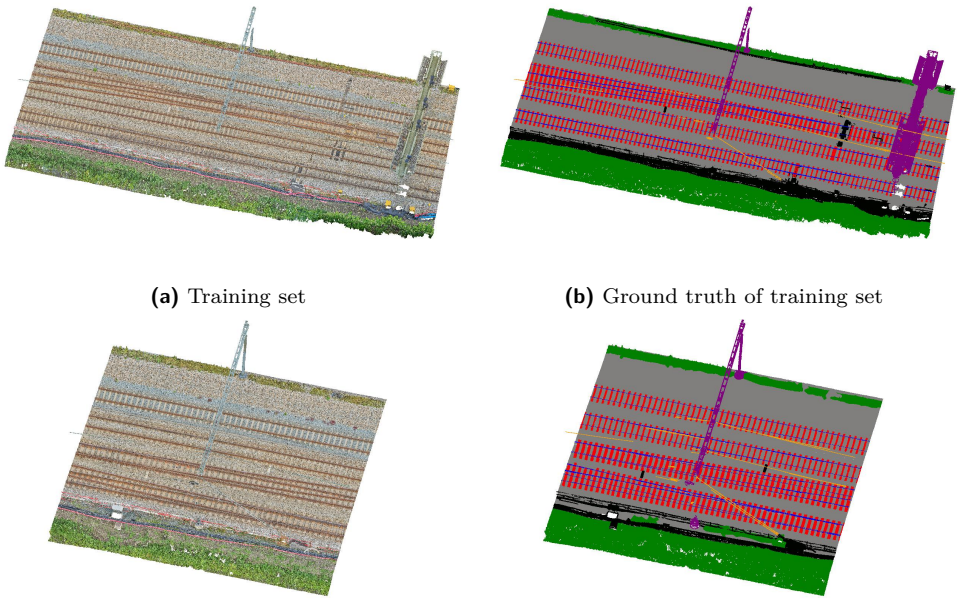
**Figure 4.2:** Dataset 1 - UAV-SfM Groningen: Training set (right) and test set (left).

Another way to split the data is spatial splitting. In this approach, the training and test sets are split spatially, so each set represents a different area of the dataset. As its name suggests, this method is preferred when working with spatial data. In this research, the training and test sets derive from 2 different areas within the first dataset, which is the dataset with the highest resolution (see Figure 4.2). The training set is around 60 m long and consists of $90,922,704$ points and the test set around 30 m long and consists of $47,248,170$ points. That is around 66% of the total points are used for training and 34% for testing, thus maintaining a meaningful ratio between training and test set. Between the 2 sets, there is a distance of about 25 m, that can guarantee a more reliable evaluation of the performance of the models.

### 4.1.3 Labeling

As usually, the training and test sets should be labeled in order to train the models and evaluate their performance quantitatively. Although there are some approaches developed for automatic or semi-automatic labeling of training data, the training and test sets here are annotated manually, since 2-Dimensional (2D) representation of the scenes is not available. 2D representations would help to detect and label the required objects, and automate this procedure. Figure 4.3 shows the training set together with the labeled training set (ground truth of training set), and the test set together with the labeled test set (ground truth of test set). Note that the ground truth of both

sets cannot be absolutely correct since they are generated manually, and the factor of human error is introduced.



**(a)** Training set

**(b)** Ground truth of training set

**(c)** Test set

**(d)** Ground truth of test set

**Figure 4.3:** Dataset 1 - UAV-SfM Groningen: (a) Training set, (b) Ground truth of training set, (c) Test set, (d) Ground truth of test set.

### 4.1.4 Subsampling

There are 3 methods of subsampling point cloud data, *random*, *spatial* and *octree*. In random mode, the specified number of points is picked in a random manner. In spatial mode, the points selected are the ones with the specified minimum distance between 2 points, while in octree mode, the level of subdivision of the octree is specified and the nearest point to each octree cell center is kept. Since the first dataset is characterized by very high resolution, the training and test sets are subsampled. Spatial subsampling is applied, as the resulting local density is constant. Both sets are subsampled with a minimum distance of 2 cm between 2 points, aiming to save computational time and power in the phase of processing. After subampling, the training set from $90,922,704$ points now contains $17,434,745$ points, whereas the test set from $47,248,170$ points now contains $9,371,921$ points. The total number of points and the number of points of each class of the training and test sets, before and after subsampling, are presented analytically in Table 4.2.

Also, the barplots in Figure 4.4 and Figure 4.5 show a comparison of the number of points of each class of the training and test set, respectively, before and after subsampling. It is clear that there is a great imbalance among the different classes, with the track bed containing the largest amount of points, and the overhead wires the smallest amount of points. Moreover, the ratio among the different classes is similar in the training and test sets, as well as before and after subsampling. Subsampling is applied only to the first dataset, while the second and third datasets are retained in their original resolution. After subsampling, the first dataset has around 1.5 times higher resolution than the third dataset and almost 3 times higher resolution than the second dataset. Table 4.3 summarizes the data and their properties, before and after subsampling.

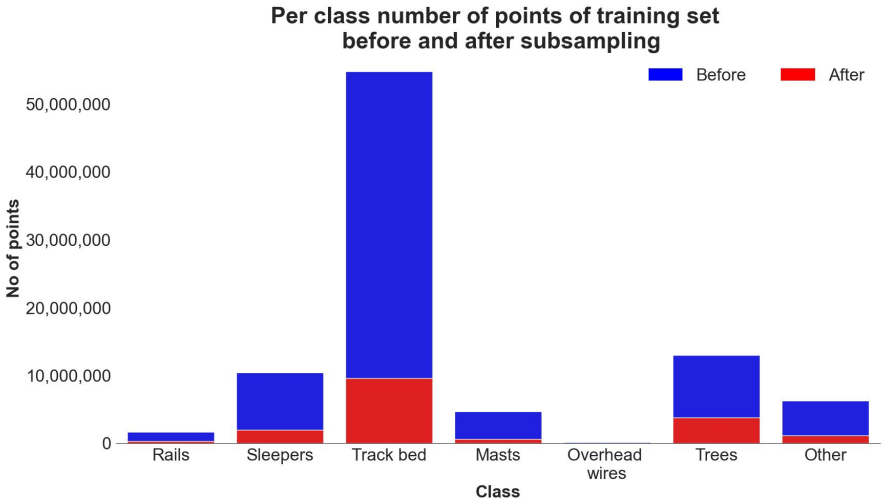



**Figure 4.4:** Comparison of per class number of points of training set, before (in blue) and after (in red) subsampling.

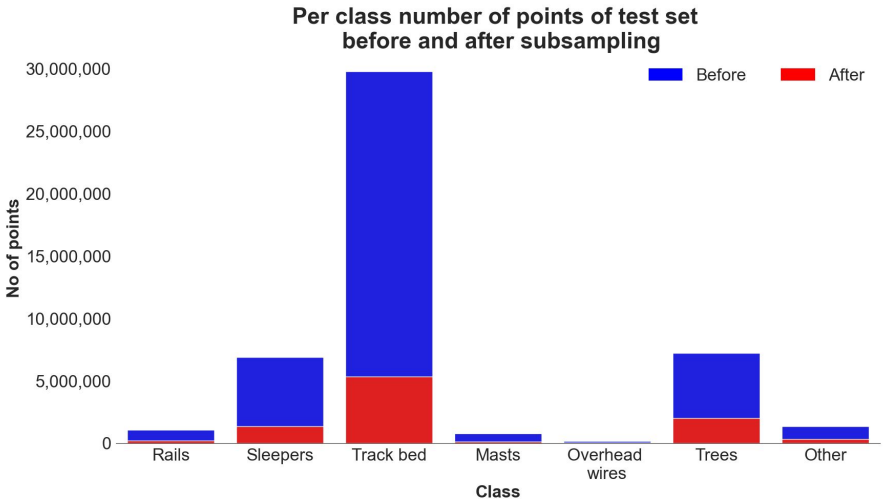



**Figure 4.5:** Comparison of per class number of points of test set, before (in blue) and after (in red) subsampling.

**Table 4.2:** Total and per class number of points of training and test set, before and after subsampling.

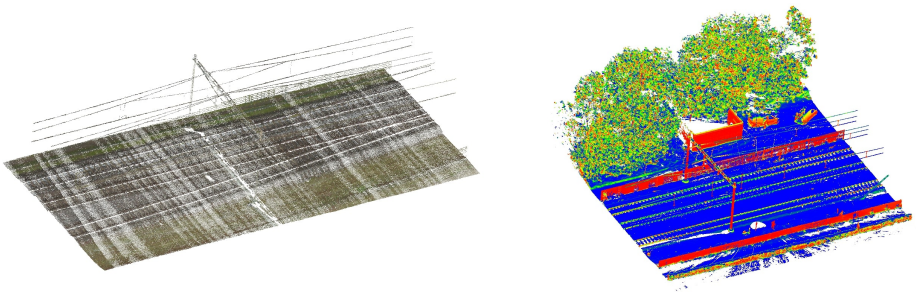
| | No of points | | | |
|---|---|---|---|---|
| | Training set | | Test set | |
| **Class** | **Before** | **After** | **Before** | **After** |
| **Rails** | 1,635,030 | 294,823 | 1,062,337 | 194,966 |
| **Sleepers** | 10,409,688 | 1,992,252 | 6,897,730 | 1,343,083 |
| **Track bed** | 54,745,675 | 9,597,236 | 29,744,047 | 5,328,493 |
| **Masts** | 4,695,603 | 629,487 | 801,066 | 131,426 |
| **Overheard wires** | 156,060 | 25,060 | 164,209 | 27,685 |
| **Trees** | 12,998,017 | 3,776,034 | 7,206,675 | 2,006,955 |
| **Other** | 6,282,631 | 1,119,853 | 1,372,106 | 339,313 |
| **Total** | 90,922,704 | 17,434,745 | 47,248,170 | 9,371,921 |

**Table 4.3:** Datasets and properties, before and after subsampling.

| **Properties** | **Dataset 1 Before** | **Dataset 1 After** | **Dataset 2** | **Dataset 3** |
|---|---|---|---|---|
| **Acquisition technique** | UAV-SfM | UAV-SfM | UAV-LiDAR | TLS |
| **Area** | Groningen | Groningen | Stroe | Stroe |
| **Attributes** | $X, Y, Z, R, G, B$ | $X, Y, Z, R, G, B$ | $X, Y, Z, R, G, B$ | $X, Y, Z$ |
| **Mean surface density** $[pts/m^2]$ | 44,750 | 6,668 | 2,363 | 4,259 |
| **Mean volume density** $[pts/m^3]$ | 1,678,121 | 250,045 | 88,624 | 159,698 |
| **Mean point spacing** $[m]$ | 0.005 | 0.012 | 0.021 | 0.015 |

### 4.1.5 Other datasets

The second and third datasets are used for visual validation of the trained models in the first and second scenario, respectively, for both methods. To this end, 1 segment of about 40 m long is chosen from each dataset. The segment from the second dataset consists of 692,310 points in total, and the segment from the third dataset consists of 13,439,478 points. Especially for the UAV-LiDAR Stroe dataset, it is not only cropped along track to reduce the amount of data, but also the surrounding environment is left out, so as to center on the railway. The 2 segments can be seen in Figure 4.6.



**(a)** Dataset 2 - UAV-LiDAR Stroe **(b)** Dataset 3 - TLS Stroe

**Figure 4.6:** Segments: (a) Dataset 2 - UAV-LiDAR Stroe, (b) Dataset 3 - TLS Stroe.

## 4.2 METHODS

Section 4.2 is divided into Subsection 4.2.1 which includes the Random Forest implementation, and Subsection 4.2.2 that describes the DGCNN implementation.

### 4.2.1 Random Forest

The Random Forest classifier is trained using the training set, which contains the labels of the points along with some features that can provide distinction between the different classes. In order to test the performance of the trained model, the test set is used, that also includes the labels of the points together with the same features as in the training set. Furthermore, for the prediction of the unlabeled segments only the features of the points are used. After trying out different features in different combinations, the features eventually used for scenario 1 and 2 of Random Forest are presented in Table 4.4. In scenario 1, 26 features are used in total, while in scenario 2, 23 features are used in total. In both scenarios the same features are used, except for the 3 extra features in the first scenario being the color values.

**Table 4.4:** Features used for scenario 1 and 2 of Random Forest.

| **Features** | |
|---|---|
| **Scenario 1** | **Scenario 2** |
| R, G, B | |
| Linearity, Planarity, Sphericity, Omnivariance, Anisotropy, Eigenentropy, Sum of eigenvalues, Surface variation, Verticality, Roughness, Mean curvature | Linearity, Planarity, Sphericity, Omnivariance, Anisotropy, Eigenentropy, Sum of eigenvalues, Surface variation, Verticality, Roughness, Mean curvature |
| *(using neighborhood radius of 2 cm)* | *(using neighborhood radius of 2 cm)* |
| Linearity, Planarity, Sphericity, Omnivariance, Anisotropy, Eigenentropy, Sum of eigenvalues, Surface variation, Verticality, Roughness, Mean curvature | Linearity, Planarity, Sphericity, Omnivariance, Anisotropy, Eigenentropy, Sum of eigenvalues, Surface variation, Verticality, Roughness, Mean curvature |
| *(using neighborhood radius of 20 cm)* | *(using neighborhood radius of 20 cm)* |
| Echo ratio | Echo ratio |
| *(using neighborhood radius of 2 cm)* | *(using neighborhood radius of 2 cm)* |
| Total=26 | Total=23 |

Therefore, for scenario 1, the inputs for each point are 1 value for red, 1 for green, 1 for blue, and 1 for each of the 23 geometric features. Hence, in case we have 1000 points, the input is an array of 1000 rows (1000 points) and 26 columns (26 features), so $1000 \times 26$. For scenario 2, in turn, the inputs for each point are 1 value for each of the 23 geometric features, and for 1000 points the input array is of size $1000 \times 23$. For the training and test sets, the input array contains plus 1 column for the labels (with values from 0 to 6 since we have 7 classes), in order to train and evaluate the models. For the 2 unlabeled segments from the other 2 datasets, the label column is obviously not included. When labels are not included, a set of points can still be predicted using a trained model, but the results can only be verified visually since the ground truth is not available for comparison.

The geometric features of *linearity*, *planarity*, *sphericity*, *omnivariance*, *anisotropy*, *eigenentropy*, *sum of eigenvalues*, *surface variation* (or *change of curvature*) and *verticality* are computed using Principal Component Analysis (PCA). PCA is an unsupervised technique for reducing the dimensionality of multidimensional data, while increasing interpretability and minimizing information loss, Jolliffe and Cadima [2016].

It does so by transforming a dataset of interrelated variables to a new uncorrelated set of variables, Principal Components (PCs), where the first few PCs retain most of the variation of the entire dataset. Hence, the first PC is a linear combination of all the original variables that has the largest variation, the second PC is also a linear combination of all the variables that has the second largest variation, and so on. The eigenvectors of the covariance matrix are actually the directions of the axes with the most variance that we call principal components, and the eigenvalues are the coefficients attached to the eigenvectors that represent the amount of variance carried in each PC.

The eigenvalues $\lambda_{1,2,3}$ resulting from PCA and sorted as $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and the corresponding eigenvectors $\vec{e}_{1,2,3}$ are then used to calculate local features called geometric features, including dimensionality and other measures. Different neighborhood sizes consider different number of neighboring points and result in different feature values. All the types of geometric features used in Random Forest can be seen in Table 4.5 along with a short description of their meaning and the definition used to compute them. Indicatively, the geometric features computed for the training set have been

**Table 4.5:** Geometric features, description and definition of them.

| Geometric feature | Description | Definition |
|---|---|---|
| Linearity | Shows whether a set of points can be modeled by a 3D line. | $L_\lambda = \frac{\lambda_1 - \lambda_2}{\lambda_1}$ |
| Planarity | Describes the smoothness of a surface. | $P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1}$ |
| Sphericity | Shows how closely the shape of an object resembles that of a perfect sphere. | $S_\lambda = \frac{\lambda_3}{\lambda_1}$ |
| Omnivariance | Shows how a neighborhood of points spreads inhomogeneously across a 3D volume. | $O_\lambda = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3}$ |
| Anisotropy | Discriminates between orientated and non-orientated objects. | $A_\lambda = \frac{\lambda_1 - \lambda_3}{\lambda_1}$ |
| Eigenentropy | A measure of the order or disorder of 3D points within the covariance ellipsoid. | $E_\lambda = \sum_{i=1}^{3} \lambda_i \ln(\lambda_i)$ |
| Sum of eigenvalues | Describes the total variation which is the sum of the squared distances of the points of a neighborhood from their centroid. | $\Sigma_\lambda = \lambda_1 + \lambda_2 + \lambda_3$ |
| Surface variation | Approximates the change of curvature in the neighborhood. | $C_\lambda = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$ |
| Verticality | Shows how vertical an object is. | $V_\lambda = 1 - \lvert[001] \cdot \vec{e_3}\rvert$ |
| Roughness | Describes the surface texture. | The distance between each point and the best fitting plane (least squares plane) computed on its nearest neighbors. |
| Mean curvature | Approximates the mean curvature in the neighborhood. | Estimated by best fitting a quadric around each point. |
| Echo ratio | A measure of local transparency and roughness. | $ER = \frac{n_{3D}}{n_{2D}} \cdot 100$ |

visualized and given in Appendix A.

To train the Random Forest classifier, there are several parameters that can be defined, such as number of estimators, criterion, maximum depth etc. The parameters used in this work for both scenarios are set to default (More information can be found in the documentation: sklearn.ensemble.RandomForestClassifier).

### 4.2.2 DGCNN

In the indoor benchmark Stanford large-scale 3D Indoor Spaces (S3DIS), the data are structured in 6 areas, where each area contains a different number of rooms. Usually, the first 5 areas are used for training the model, and the last one for testing the performance of the model. Similarly to S3DIS, the data in this work are structured in 4 areas, 3 consisting of the training set and 1 of the test set. The other datasets are structured in 1 area each. Each area contains 10 subareas, corresponding to the rooms in the S3DIS dataset. Each subarea is converted to a `.npy` file, where the 3-Dimensional (3D) coordinates of the points are shifted by subtracting the minimum 3D coordinates in the subarea.

As stated in Subsection 3.2.1, the `PyTorch` implementation by An Tao is used, including also the additions and modifications by Qian Bai to use directly the `.npy` files of the subareas to load points block by block, instead of using the `HDF5` files (`.h5` files) as usually. Thus, the size of the blocks can be used directly as input parameter to the implementation. Moreover, in this implementation, all points of each block are used in testing, instead of randomly sampling each block with a specific number of points, to get a full picture of the results. This implementation is further extended by incorporating the preparation and prediction of unlabeled point clouds. Since in the `.npy` files used the 3D coordinates of the points in each subarea are shifted, after exporting each subarea with the predicted labels, the $XYZ$ coordinates of the points are converted back to the original ones. Intermediate additions are also applied to automate the entire implementation. Considering that this implementation is built to work for point clouds with 3D coordinates and color values (scenario 1), another implementation is made for point clouds that do not contain RGB values (scenario 2).

In scenario 1, 9 features are used in total, the 3D Cartesian coordinates ($XYZ$), RGB values, and normalized 3D coordinates ($X_nY_nZ_n$) of the points. The normalized coordinates are computed by subtracting the central coordinates of each subarea. The use of normalized coordinates can help the learning of the network to be more robust. In the second scenario, 6 features are used in total, which are the same as in the first scenario, except for the color values. In detail, for scenario 1, the inputs for each point are 1 value for $X$, 1 for $Y$, 1 for $Z$, 1 for red, 1 for green, 1 for blue, 1 for $X_n$, 1 for $Y_n$, and 1 for $Z_n$. Hence, for 1000 points the input array is of size $1000 \times 9$. Similarly, for scenario 2 the inputs for each point are 1 value for $X$, 1 for $Y$, 1 for $Z$, 1 for $X_n$, 1 for $Y_n$, and 1 for $Z_n$, and for 1000 points the input array is of size $1000 \times 6$. Alike to Random Forest, the input array of the training and test sets additionally includes 1 column for the labels, while that of the other 2 unlabeled segments does not. Table 4.6 summarizes the features used for scenario 1 and 2 of DGCNN.

**Table 4.6:** Features used for scenario 1 and 2 of DGCNN.

| Features | |
|---|---|
| **Scenario 1** | **Scenario 2** |
| $X$, $Y$, $Z$ | $X$, $Y$, $Z$ |
| R, G, B | $X_n$, $Y_n$, $Z_n$ |
| $X_n$, $Y_n$, $Z_n$ | |
| Total=9 | Total=6 |

DGCNN is based on PointNet, where the point cloud is split into 3D blocks with a specified block size before used as input to the neural network. In the training process, $n$ number of points with $f$ dimensions are randomly sampled for each single block and fed into the neural network. Thus, the network processes each block individually, and this process is repeated until all blocks making up the point cloud are selected. In the present research, 4096 points are randomly sampled from each block and used to train the network. The block size is set to 1 m, since in the attempt of increasing the block size, the Overall Accuracy (OA) of the trained model is reduced. Such a small block size is recommended for point clouds with high resolution, while point clouds of large areas and low resolution can be processed in larger blocks.

The receptive field is the region in the input space that the features of the Convolutional Neural Network (CNN) are computed. It is affected by the block size, but also by the size of the neighborhood (number of neighbors) used to construct the k-Nearest Neighbors (k-NN) graphs in Edge Convolution (EdgeConv). In the current study, the number $k$ of nearest neighbors is set to 20 for all EdgeConv layers, as in the original implementation of DGCNN. Unlike to the original implementation that discards blocks of less than 100 points, and the start of the next block contains the same points as the end of the previous block, in this implementation blocks containing less than 20 points are discarded, and each block contains exclusively different points, starting after the end of the previous block. This aims to keep as many points of the original points as possible and avoid duplicates. Obviously, the output points are less than the original points, since the blocks ought to include at least as many points as those used to construct the k-NN graphs (here: at least 20 points), and blocks of less than those points cannot be processed.

During training, the batch size is set to 10, meaning that 10 blocks are processed at the same time. Therefore, the network is fed by an $b \times n \times f$ array, where $b$ is the number of batches, that is an $10 \times 4096 \times 9$ array for scenario 1, and an $10 \times 4096 \times 6$ array for scenario 2. To test the performance of the trained model and predict unlabeled point clouds, a batch size of 1 is used. A different batch size is used in testing and prediction since all points from each block are used, and each block has a different number of points, but tensors of different size cannot be stacked together.

**Table 4.7:** Parameters used for both scenario 1 and 2 of DGCNN.

| Parameters | |
|---|---|
| Block size [m] | 1 |
| Batch size (training) | 10 |
| Batch size (testing/prediction) | 1 |
| Number of epochs | 50 |
| Number of neighbors | 20 |
| Number of points | 4096 |
| Optimizer | Adam |
| Learning rate | 0.001 |
| Decay rate | 0.5 |
| Momentum | 0.9 |

In order to optimize the network, the Adam optimizer is used. The initial learning rate that controls the adjustment of the weights of the network is set to 0.001, and the decay rate which determines the rate at which the learning rate decays is set to 0.5. The momentum, that accumulates past observations to achieve faster convergence, is set to 0.9. The values of the above parameters are as suggested by DGCNN. The

network is trained for 50 epochs (50 times), and the resulting model of DGCNN is the best among the ones occurred from 50 epochs of training. The parameters used both for scenario 1 and 2 of DGCNN are summarized in Table 4.7.

## 4.3 EVALUATION METRICS

Section 4.3 is organized in the following subsections: Subsection 4.3.1 with the classification metrics used to evaluate the performance of the trained models, and Subsection 4.3.2 with the predicted probabilities computed to investigate the uncertainty of the trained models.

### 4.3.1 Classification metrics

There are several classification metrics used to assess the performance of a ML model in classification tasks. In order to evaluate the performance of the trained Random Forest and DGCNN models, the classification metrics used are the OA of the model, the precision, recall and F1-score of each class, and an elaborate confusion matrix. *Accuracy* is the number of correct (True Positive (TP) and True Negative (TN)) predictions over the total number of input samples (TP and TN and False Positive (FP) and False Negative (FN)) (Equation 4.1). *Precision* is the number of correct positive predictions over the number of positive predictions (Equation 4.2), while *recall* (or *sensitivity* or *true positive rate*) is the number of correct positive predictions over the number of all relevant samples (all samples that should have been predicted as positive) (Equation 4.3).

*F1-score* (or *F-score* or *F-measure*) is the harmonic mean of precision and recall (Equation 4.4). Typically, it is preferred over other evaluation metrics as it considers both precision and recall to compute the score, being a trade-off between those 2 metrics. It actually tells how precise the classifier is (the number of instances it classifies correctly), and how robust it is (the number of instances it misses). Accuracy, precision, recall and F1-score range between 0 and 1, where 1 is the best value. They can also be expressed in terms of percentages. Moreover, the *confusion matrix* provides a detailed breakdown of the number of correct (true) and incorrect (false) classifications for each class. In its simplest form, a confusion matrix looks like in Figure 4.7.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.4}$$

Additionally, the impurity-based feature importances are computed for the trained Random Forest classifiers. The feature importances are not used to evaluate the performance of a classifier, but indicate the contribution of each feature to the training

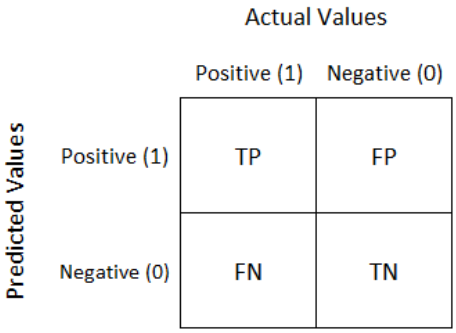



**Figure 4.7:** Confusion matrix for binary classification (positive and negative). The correctly (TP and TN) and wrongly (FP and FN) classified points. (Image source: Sarang Narkhede [2018])

process. The higher the feature importance, the more important the feature, while all importances together sum up to 1. The importances, also known as *Gini* importances, are calculated as the normalized total reduction of the utilized criterion. Considering that there is a large imbalance among the different classes, the *average Per class Accuracy (avPA)* is also computed for DGCNN. Moreover, the *mean Intersection over Union (mIoU)* is calculated, which is a widely used evaluation metric in semantic segmentation applications. Practically, Intersection over Union (IoU) is the area of overlap between ground truth and segmentation results divided by the area of union of them (see Equation 4.5 and Figure 4.8). The mIoU is then the mean IoU of all the classes.

$$IoU = \frac{TP}{TP + FP + FN} \tag{4.5}$$

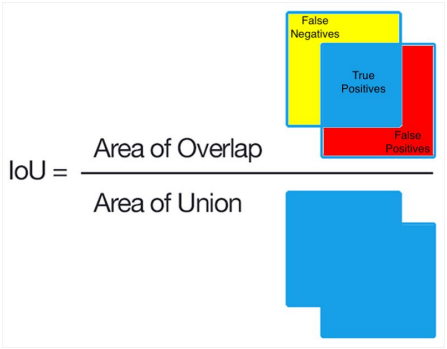



**Figure 4.8:** IoU: The area of overlap between ground truth and segmentation results (top blue part) over the area of union of them (bottom blue part / top yellow part and top blue part and top red part). (Image source: Nigel M. Parsad [2018])

### 4.3.2 Predicted probabilities

Random Forest performs majority voting among all its decision trees to predict the output class of any input point. Hence, the predicted class of each point is the one with the highest mean probability. The predicted class probabilities for each point, are computed as the fraction of the total number of trees that vote for each specific class. Let us assume a Random Forest that consists of 100 decision trees, 3 classes A, B and C, and 1 point that should be attributed to 1 of these 3 classes. In case that 50 out of 100 trees have voted for class A, 30 for class B and 20 for class C, the

predicted class for that point will be A. The predicted class probabilities will be 0.5, 0.3 and 0.2, which are actually the probability of this point belonging to class A, B and C, respectively. In this work, the predicted class probabilities are computed, and the highest probabilities of the points are used to investigate the probabilities of the points in each class. This can provide an indication regarding the uncertainty of the classifier for each output class.

In DGCNN, after testing or prediction, apart from the predicted labels of the points, the network outputs also the segmentation scores of all classes for each point, the so-called *logits*. In ML, logits are the unnormalized (raw) predictions of a model that vary in the range [-inf, inf]. But because interpreting those raw values is not easy, they are ordinarily passed to a normalization function. In multi-class classification problems, *softmax* is usually used to convert the final scores to class probabilities ([0, 1]). Softmax is actually used as a last activation function of a neural network in order to normalize the output scores of the network to a probability distribution over predicted classes. Prior to applying softmax, some scores can be negative or larger than 1 and may not add up to 1, but after softmax each score will be in the interval [0, 1] and the scores will sum up to 1, so they can be interpreted as probabilities.

For each point, the softmax function takes as input a vector $l$ (logits) of $p$ values, where $p$ is the total number of classes, and outputs the class probabilities $P$. The probability $P_i$ of the $i^{th}$ class is calculated by dividing the exponential of the $l_i$ value by the sum of exponentials of the $l$ values of all the classes (Equation 4.6). After applying the softmax function to the output scores, similar to Random Forest, the highest probability for each point is computed, and the probabilities of all the points assigned to each class are analyzed, in order to examine the uncertainty of the classifier for each class.

$$P_i = \sigma(l)_i = \frac{e^{l_i}}{\sum_{j=1}^{p} e^{l_j}} \tag{4.6}$$

## 4.4 COMBINATION OF METHODS

The results from Random Forest and DGCNN are eventually combined to better exploit the positive outcomes of each method and further improve the final results. For each point of a predicted set there is a predicted label from Random Forest and one from DGCNN, accompanied by the probability with which the label is assigned by Random Forest and the one by DGCNN. After the combination, each point of the set receives the label of the method predicted with the highest probability. For instance, if point 1 is assigned to rails with a probability of 70% by Random Forest and to masts with 80% probability by DGCNN, eventually point 1 is assigned to masts. Since DGCNN results in less points than the originals, given that blocks of less than 20 points are discarded, the aforementioned comparison only applies to common output points, while points missing from DGCNN simply receive the labels assigned by Random Forest. Finally, the classification metrics of OA, per class precision, recall and F1-score, and confusion matrix are computed to be compared with the ones resulted from each method individually. The combination is implemented here only on the predicted test set from scenario 1, as a first experiment, but could be applied in the same way on any predicted set.

## 4.5 SUMMARY

Chapter 4 comprised the methodology followed in this research to perform pointwise classification of point cloud data from railways. Specifically, it contained a general overview of the methodology used, and described the preprocessing of the data, the methods applied in this work, as well as the evaluation metrics employed, and the combination of methods implemented. In preprocessing, the classes and colors used to represent the main components on railways were presented, while the spatial training-test splitting method used to split the first dataset into training and test sets was analyzed, and the sets produced were presented and illustrated. Also, the labeling procedure of manual labeling implemented to annotate the training and test sets was presented, and the labeled sets produced were visualized. Furthermore, the spatial subsampling method used to subsample the training and test sets was described, the total and per class number of points before and after subsampling were compared, and the new properties of the first dataset after subsampling were analyzed. Additionally, the segments used from the second and third dataset to visually validate the trained models were presented and visualized. In methods, the Random Forest implementation was described, including the features and parameters used to train the Random Forest classifiers. Moreover, the DGCNN implementation was explained, containing the preparation of the data and the additions and modifications applied to the implementation used, as well as the features and parameters used to train the DGCNN models. In evaluation metrics, the classification metrics and predicted probabilities used to evaluate the performance and investigate the uncertainty of the trained models of the 2 methods were explained. Eventually, in combination of methods, the procedure followed to combine the prediction results from scenario 1 of Random Forest and DGCNN was described. The next chapter, Chapter 5, shows the results produced in this research. Particularly, it contains the classification results obtained from Random Forest scenario 1 and 2, as well as those from DGCNN scenario 1 and 2, and the ones from the combination of both methods.

# 5 RESULTS

Chapter 5 includes the results from Random Forest, Dynamic Graph Convolutional Neural Network (DGCNN), and the combination of both methods in Section 5.1, Section 5.2 and Section 5.3, respectively.

## 5.1 RANDOM FOREST

Section 5.1 is split into Subsection 5.1.1 and Subsection 5.1.2, which contain the results from scenario 1 and 2 of Random Forest, respectively.

### 5.1.1 Scenario 1: using RGB and 23 geometric features

In the first scenario, Random Forest employs Red Green Blue (RGB) values and 23 geometric features. It is trained and evaluated using the labeled training and test sets from the Unmanned Aerial Vehicle (UAV)-Structure from Motion (SfM) Groningen dataset, and is validated using an unlabeled segment from the UAV-Light Detection And Ranging (LiDAR) Stroe dataset, as discussed in Chapter 4. In this scenario, training Random Forest lasts 6 hours in total, and the memory used during training is 125.58 GB.

#### UAV-SfM Groningen test set

Figure 5.1 shows the predicted test set. Some misclassification results are indicated in cyan boxes, with box A showing the confusion of parts of rails and sleepers with track bed, and boxes B some misclassification results of masts. Particularly, some straight components of the top part of the mast are misclassified as rails, while the base of the pole as other and trees. The boxes C show some misclassification results of overhead wires, which are mainly confused with other objects or rails. Also, Table 5.1 summarizes the results, including the Overall Accuracy (OA) of the model, the precision, recall and F1-score computed for each class in percentages, and the total number of points used for each class. All the confusion matrices calculated in this work are included in Section B.1 of Appendix B, and used to interpret misclassification among the different classes. The results show that the model achieves 88.65% OA. The track bed is the most well-classified class and the overhead wires the least well-classified class, according to F1-score measure, which correspond to the classes with the largest and smallest amount of points, respectively. The same classes have the highest and lowest recall values, respectively, while rails show the highest precision and masts the lowest one. The rails, sleepers and trees are mostly misclassified as track bed, the track bed, masts and other as trees, and the overhead wires as other. Hence, for most of the classes, the largest amount of wrongly classified points are misclassified as either track bed or trees. However, for all classes, the greatest amount of points are correctly identified.

Additionally, the impurity-based feature importances computed for the Random Forest classifiers are included in Section B.2 of Appendix B. Importances is a useful tool that can help evaluating the importance of the features during training, omit features that do not contribute significantly to the trained model and add other more important features. Here, the geometric features computed using a neighborhood

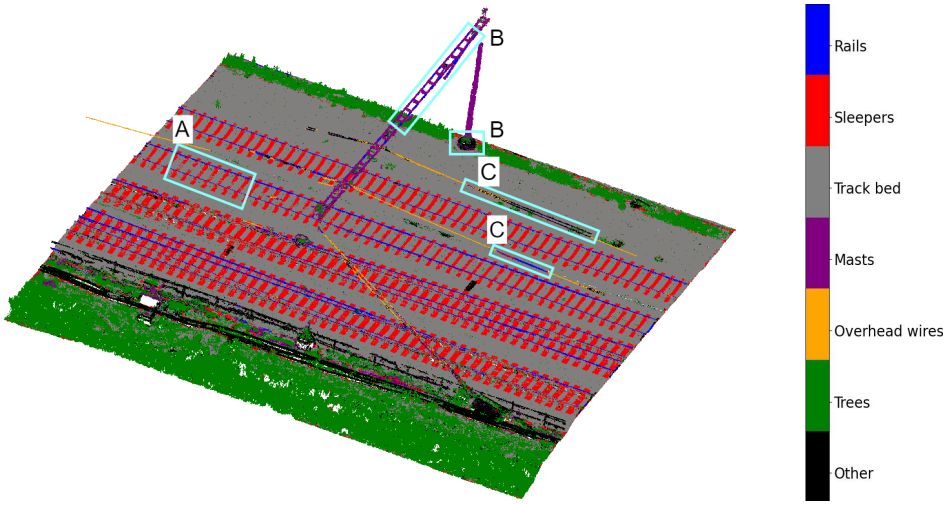



**Figure 5.1:** {Random Forest - Scenario 1} Predicted test set. Some negative results are indicated in cyan boxes. Some wrongly classified parts of rails and sleepers are shown in box A, of masts in boxes B, and of overhead wires in boxes C.

**Table 5.1:** {Random Forest - Scenario 1} OA and per class precision, recall, F1-score and number of points.

| Class | Precision [%] | Recall [%] | F1-score [%] | No of points |
|---|---|---|---|---|
| **Rails** | 91.14 | 80.84 | 85.68 | 194,966 |
| **Sleepers** | 90.20 | 85.72 | 87.90 | 1,343,083 |
| **Track bed** | 91.06 | 94.14 | 92.57 | 5,328,493 |
| **Masts** | 63.97 | 60.99 | 62.44 | 131,426 |
| **Overhead wires** | 85.05 | 46.83 | 60.40 | 27,685 |
| **Trees** | 86.35 | 83.81 | 85.06 | 2,006,955 |
| **Other** | 64.02 | 61.45 | 62.71 | 339,313 |
| **OA [%]: 88.65** | | | | |

radius of 20 cm contribute more, in the majority, than the ones computed using a neighborhood radius of 2 cm, meaning that features developed with a larger neighborhood size are more informative for the model. Moreover, color values are included in the most important features used to build this model. Therefore, omitting the RGB values in the next scenario can affect negatively the classification results.

The histograms of the probabilities with which the points of each output class are assigned to that class are shown in Figure 5.2. Each histogram is accompanied by a density plot, that is a smoothed, continuous version of the histogram. In all output classes, the majority of points are predicted with a probability between 90% and 100%, expect for the masts where most of the points are predicted with a probability between 40% and 50%. This fact implies the low uncertainty of the model in predicting almost all the output classes. In rails, sleepers, track bed and trees, fewer and fewer points are predicted with less and less probability. On the flip side, in masts, overhead wires and other, the amount of points varies among the different probability intervals, indicating that the model presents higher uncertainty when predicting those classes. Besides, these are the 3 classes with the lowest F-1 scores.

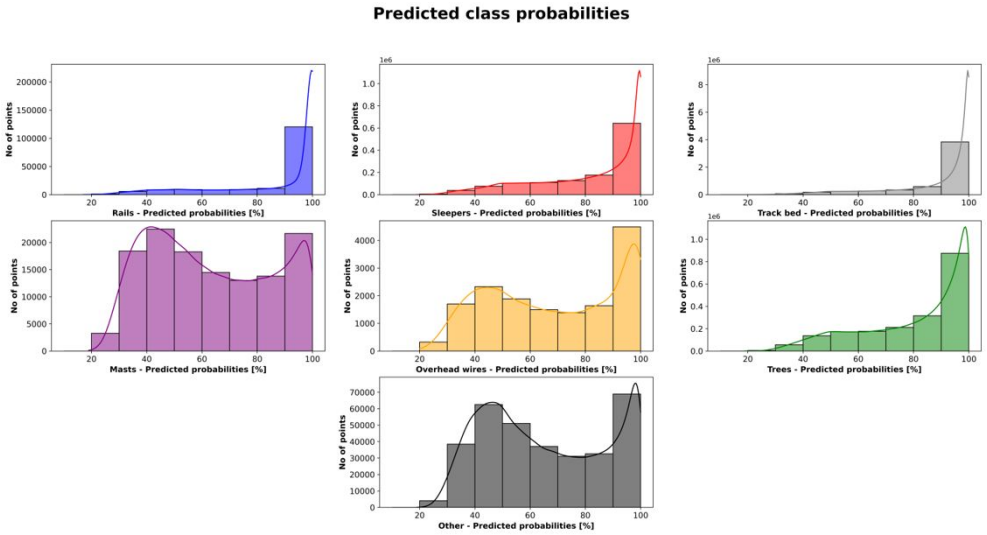



**Figure 5.2:** {Random Forest - Scenario 1} Histograms and density plots of predicted probabilities of output classes of test set.

### UAV-LiDAR Stroe segment

Figure 5.3 illustrates the predicted UAV-LiDAR Stroe segment, with some correctly and wrongly classified parts indicated in pink and cyan boxes, respectively. As can be observed, the track bed is classified mostly as track bed but mixed with trees and sleepers (cyan box A), while the trees are classified mostly as trees but mixed with track bed and especially sleepers (cyan box B). As expected, the sleepers cannot be identified (cyan box C), since they are not even visible to the human eye. Although the top overhead wires are mainly identified (pink box A), the bottom wires that are straight are classified as rails (cyan box D), and the rails as trees (cyan box E). This confusion can be justified by the fact that the model has been trained on the UAV-SfM Groningen dataset, where the wires are not well-reconstructed and not representative of normal wires on railways. Regarding the mast, only the poles are identified (pink

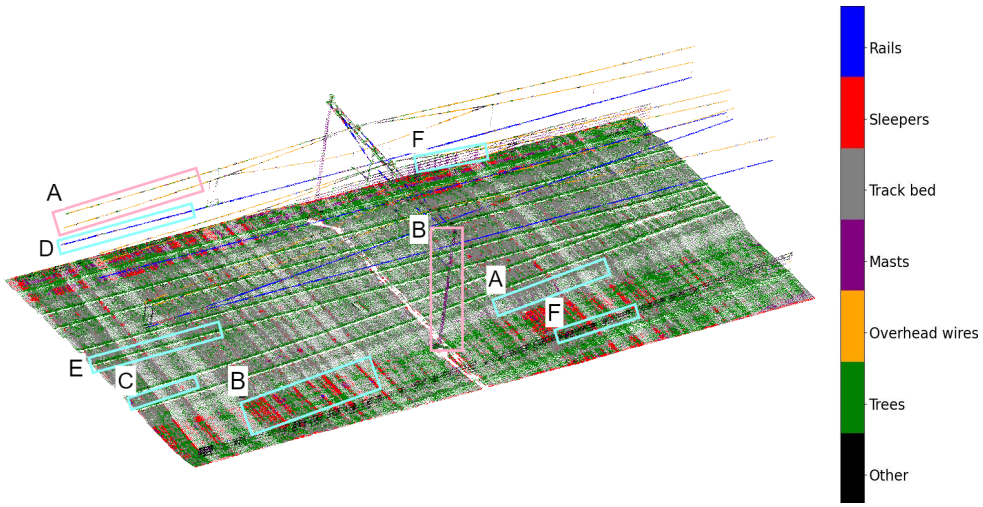



**Figure 5.3:** {Random Forest - Scenario 1} Predicted UAV-LiDAR Stroe segment. Some positive and negative results are indicated in pink and cyan boxes, respectively. Some correctly classified parts of overhead wires are shown in pink box A, and of masts in pink box B. Some wrongly classified parts of track bed are shown in cyan box A, of trees in cyan box B, of sleepers in cyan box C, of overhead wires in cyan box D, of rails in cyan box E, and of other in cyan boxes F.

box B), while the fence on the one side of the railroad is mainly classified as other but on the other side as masts (cyan boxes F).

The histograms along with the density plots of the predicted probabilities for each output class of the UAV-LiDAR Stroe segment are shown in Figure 5.4. The confusion in the predicted classes is also imprinted on the probabilities. The largest amount of points in all classes are predicted with a probability either between 30% and 40% (rails, sleepers, masts, other) or between 40% and 50% (track bed, overhead wires, trees). The predicted probabilities of the remaining points are mainly centered around these intervals. Thus, the low probability values indicate the high uncertainty of the model in predicting more or less all the output classes.

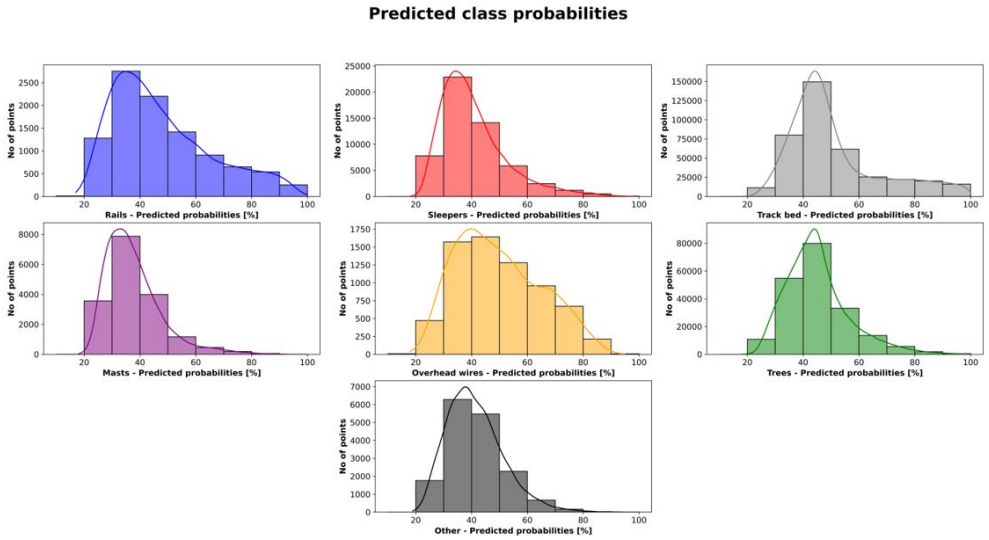



**Figure 5.4:** {Random Forest - Scenario 1} Histograms and density plots of predicted probabilities of output classes of UAV-LiDAR Stroe segment.

### 5.1.2 Scenario 2: using 23 geometric features only (no RGB)

In the second scenario of Random Forest, only 23 geometric features are employed. The model is trained and evaluated again using the labeled training and test sets from the UAV-SfM Groningen dataset, but the utilized time and memory are 1 hour and 18 GB less than in the previous scenario. Moreover, the trained model is validated here using an unlabeled segment from the Terrestrial Laser Scanning (TLS) Stroe dataset that does not contain color values.

#### UAV-SfM Groningen test set

The classification results for the test set can be seen in Figure 5.5 and Table 5.2. As can be observed in Figure 5.5, there is a similar pattern in the prediction results compared to the previous scenario, with the most obvious difference that some parts of the rails are misclassified as trees, as indicated in box D. In this case, the model achieves an OA of 83.39%, that is around 5% less than when taking into account also the color values in the first scenario of Random Forest. Similar to scenario 1, track bed is the most well-classified class according to F1-scores, which is the class containing most of the points. However, unlike to scenario 1, the least well-classified class is masts, which is the second class with the fewest points. The same classes show the largest and smallest, respectively, precision and recall values. Also, the classes are mainly misclassified just like in the first scenario. In total, when ignoring color values both the OA of the model and the quality of the per class classification results drop significantly.

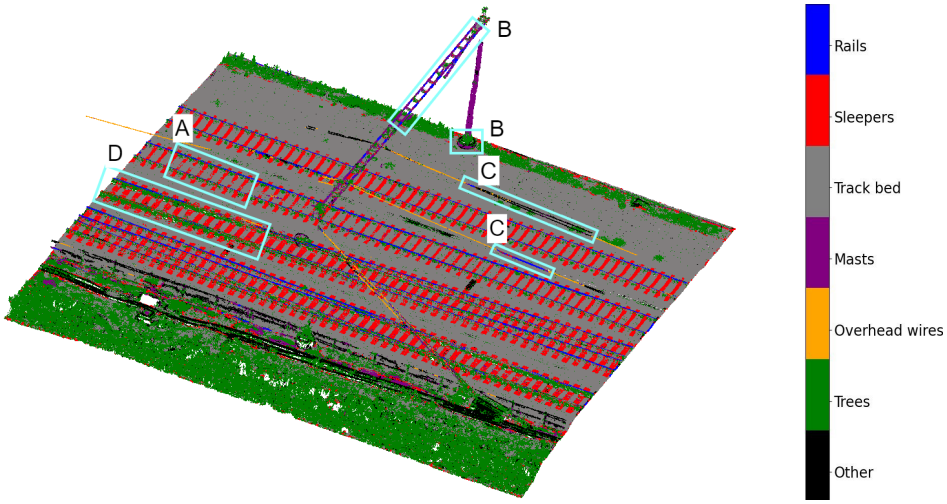



**Figure 5.5:** {Random Forest - Scenario 2} Predicted test set. Some negative results are indicated in cyan boxes. Some wrongly classified parts of rails and sleepers are shown in box A, of masts in boxes B, of overhead wires in boxes C, and of rails only in box D.

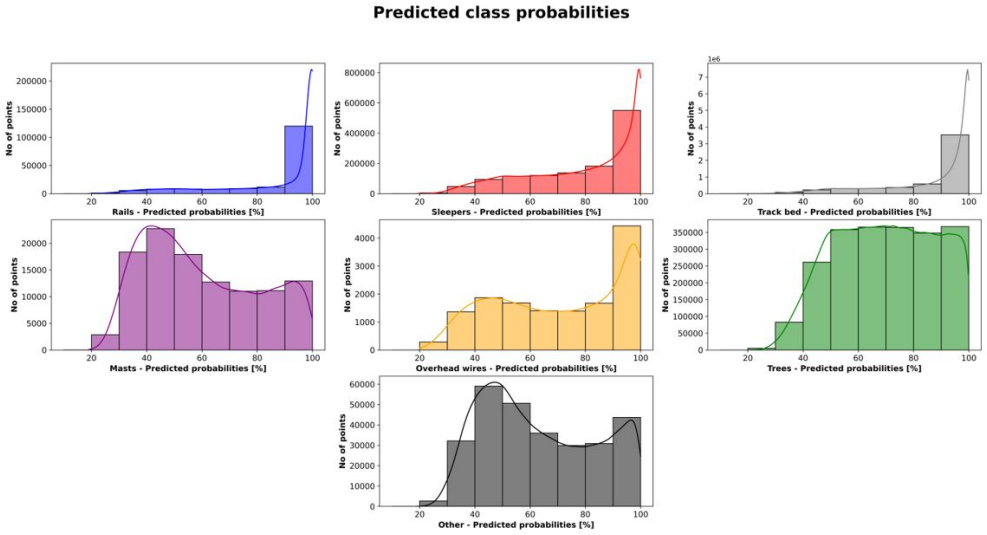



**Figure 5.6:** {Random Forest - Scenario 2} Histograms and density plots of predicted probabilities of output classes of test set.

**Table 5.2:** {Random Forest - Scenario 2} OA and per class precision, recall, F1-score and number of points.

| Class | Precision [%] | Recall [%] | F1-score [%] | No of points |
|---|---|---|---|---|
| **Rails** | 88.60 | 77.65 | 82.76 | 194,966 |
| **Sleepers** | 87.13 | 81.05 | 83.98 | 1,343,083 |
| **Track bed** | 89.13 | 90.16 | 89.64 | 5,328,493 |
| **Masts** | 47.67 | 39.76 | 43.36 | 131,426 |
| **Overhead wires** | 80.89 | 41.20 | 54.59 | 27,685 |
| **Trees** | 71.94 | 77.19 | 74.47 | 2,006,955 |
| **Other** | 55.61 | 46.68 | 50.76 | 339,313 |
| **OA [%]: 83.39** | | | | |

The predicted probabilities in scenario 2 (see Figure 5.6) show that, apart from the masts and other where the largest amount of points are predicted with a probability between 40% and 50%, in the remaining classes most of the points are predicted with a probability between 90% and 100%, showing low model uncertainty overall. Particularly for masts, trees and other, there is a large spread of points over the different probability intervals, that indicates higher uncertainty of the model in predicting those classes.

### TLS Stroe segment

In this scenario, the TLS Stroe segment is predicted (see Figure 5.7). Similar to the predicted UAV-LiDAR Stroe segment in scenario 1 of Random Forest, the top overhead wires are mainly identified as wires (pink box A), but the bottom straight wires are classified as rails (cyan box A), and the rails as trees (cyan box B). Another similarity is that only the poles of the mast are identified (pink box B), while the track bed is classified mostly as track bed but mixed with trees and sleepers (cyan box C). The trees are mainly classified as trees, but mixed especially with sleepers, other and masts (cyan box D), while the sleepers are partly identified (pink box C). Other objects, such as fences, are classified also as other, but mainly as masts (cyan boxes E). Overall, there is a similar classification pattern between the UAV-LiDAR Stroe segment predicted in scenario 1 and the TLS Stroe segment in scenario 2.

The respective predicted probabilities in Figure 5.8 intimate that the majority of points in rails and overhead wires are predicted with a probability in the interval [90%, 100%], and in trees in the interval [80%, 90%]. In practise, the top wires and trees are indeed relatively well-classified. Regarding the rails, they are predicted with very high probabilities, although they are actually completely confused with bottom wires. This is plausible considering that the shape of bottom wires is pretty similar to that of rails, and the model is not able to learn the difference between the 2 objects during training, since the wires in the training set are not well-reconstructed. In other words, the similar shape of the 2 objects makes the algorithm be very sure when predicting rails, although, in reality, they are not. On the other hand, the largest amount of points in sleepers are predicted with probabilities between 30% and 40%, and in the rest of the classes (track bed, masts, other) with probabilities between 40% and 50%. Therefore, just like in the predicted segment of the first scenario, the model shows high uncertainty in total when predicting a different dataset than the one it was trained on.

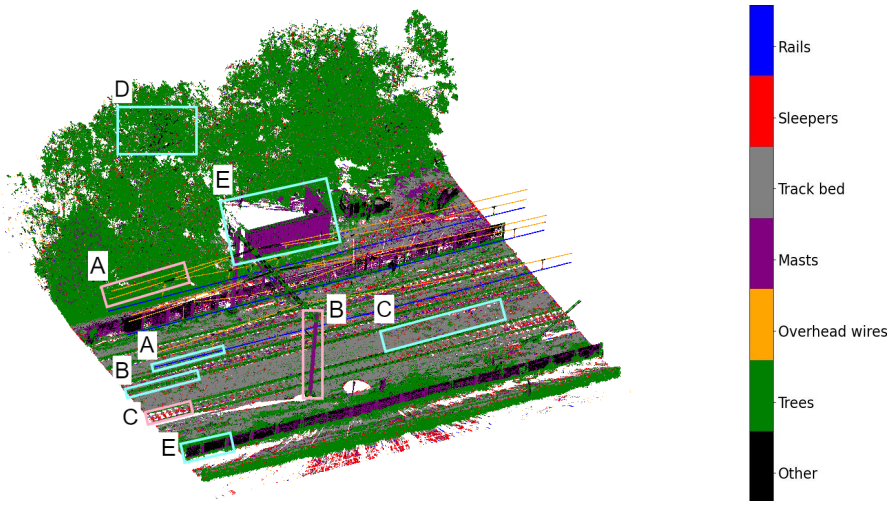



**Figure 5.7:** {Random Forest - Scenario 2} Predicted TLS Stroe segment. Some positive and negative results are indicated in pink and cyan boxes, respectively. Some correctly classified parts of overhead wires are shown in pink box A, of masts in pink box B, and of sleepers in pink box C. Some wrongly classified parts of overhead wires are shown in cyan box A, of rails in cyan box B, of track bed in cyan box C, of trees in cyan box D, and of other in cyan boxes E.

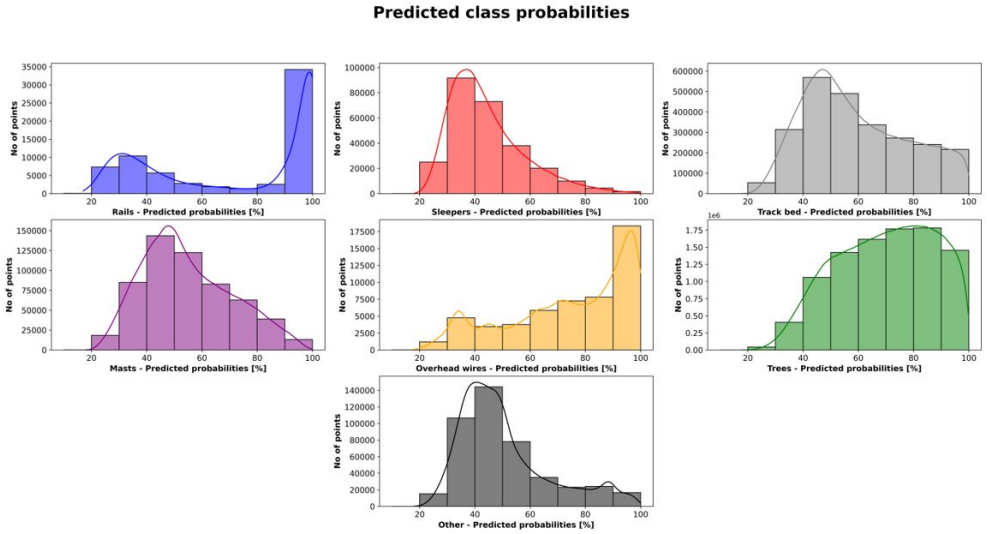



**Figure 5.8:** {Random Forest - Scenario 2} Histograms and density plots of predicted probabilities of output classes of TLS Stroe segment.

## 5.2 DGCNN

Section 5.2 is divided into Subsection 5.2.1 and Subsection 5.2.2, including the results from scenario 1 and 2 of DGCNN, respectively.

### 5.2.1 Scenario 1: using RGB and 3D Cartesian and normalized coordinates

Unlike Random Forest that makes use of user-developed geometric features, in DGCNN only some LiDAR features are utilized. Specifically, in the first scenario of DGCNN, the $XYZ$ Cartesian coordinates of the points are used along with their RGB values and their $X_nY_nZ_n$ normalized coordinates. Similar to Random Forest scenario 1, the model is trained and tested using the labeled training and test sets from the UAV-SfM Groningen dataset, and is validated using an unlabeled segment from the UAV-LiDAR Stroe dataset. However, DGCNN uses much less memory than Random Forest, but takes more time for training. In particular, the utilized time and memory are 6 hours and 5.84 GB, respectively.

#### UAV-SfM Groningen test set

The prediction of the test set is illustrated in Figure 5.9, where the most significant misclassification of rails and sleepers with track bed is pointed out in box A, and of track bed with sleepers in box B. It is worth mentioning that the mast and the overhead wires seem fully identified in this case. The results in Table 5.3 show that the model achieves 0.5% higher OA compared to scenario 1 of Random Forest, i.e. 89.19%. The model also reaches 84.77% avPA and 74.57% mIoU. While the trees get the highest recall and F1-score values, and the other the lowest precision and F1-score values, the highest precision is met in masts and the lowest recall in sleepers. Analytically, most of the points in rails and sleepers are misclassified as track bed, in masts and trees as other, in track bed as sleepers, in overhead wires as masts, and in other as trees. Nonetheless, as in the 2 scenarios of Random Forest, most of the points in all output classes are classified correctly. Although DGCNN reaches slightly higher OA than Random Forest in the corresponding scenario, Random Forest manages to classify better important components such as rails, sleepers and track bed, according to F1-scores. Moreover, Random Forest outputs the same amount of points taking as input, but the total number of output points from DGCNN are 9,371,681. That is

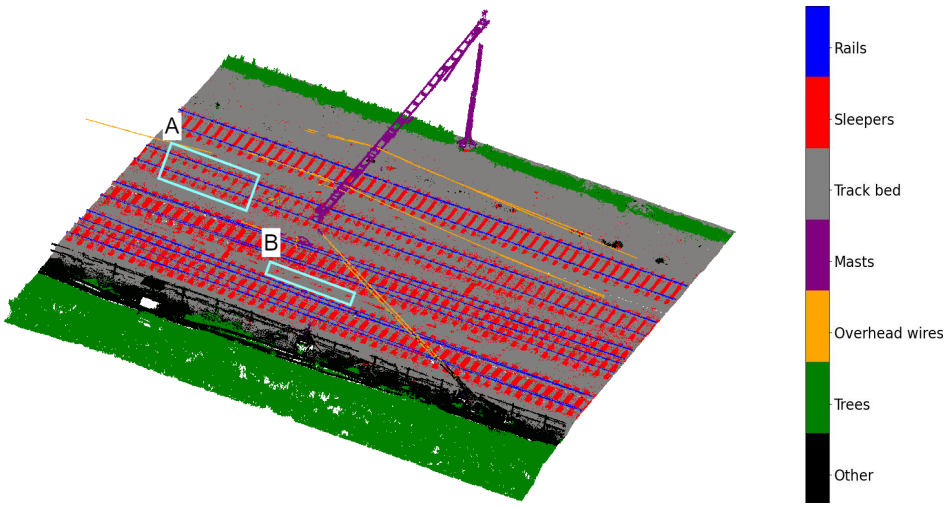



**Figure 5.9:** {DGCNN - Scenario 1} Predicted test set. Some negative results are indicated in cyan boxes. Some wrongly classified parts of rails and sleepers are shown in box A, and of track bed in box B.

240 less points compared to the original ones, since blocks containing less points than the selected neighborhood size (that is less than 20 points) are discarded.

**Table 5.3:** {DGCNN - Scenario 1} OA, avPA and mIoU, and per class precision, recall, F1-score and number of points.

| Class | Precision [%] | Recall [%] | F1-score [%] | No of points |
|---|---|---|---|---|
| **Rails** | 80.73 | 85.98 | 83.27 | 194,966 |
| **Sleepers** | 81.41 | 71.91 | 76.37 | 1,343,080 |
| **Track bed** | 91.49 | 92.57 | 92.03 | 5,328,327 |
| **Masts** | 97.38 | 82.33 | 89.22 | 131,408 |
| **Overhead wires** | 95.95 | 85.32 | 90.32 | 27,685 |
| **Trees** | 95.73 | 93.90 | 94.81 | 2,006,902 |
| **Other** | 57.25 | 81.41 | 67.23 | 339,313 |
| **OA [%]: 89.19** | **avPA [%]: 84.77** | | **mIoU [%]: 74.57** | |

Regarding the predicted probabilities (see Figure 5.10), in all output classes, most of the points are classified with a probability in the range [70%, 80%], except for the overhead wires and other which are predicted with a probability in the range [80%, 90%] and [60%, 70%], respectively. Hence, the model predicts most of the classes with relatively low uncertainty, but still higher than in Random Forest. Furthermore, all classes, but overhead wires, show a similar pattern. The amount of points increases as the probability interval increases, until reaching the interval with the greatest amount of points, whereas, after that, the amount of points decreases dramatically.

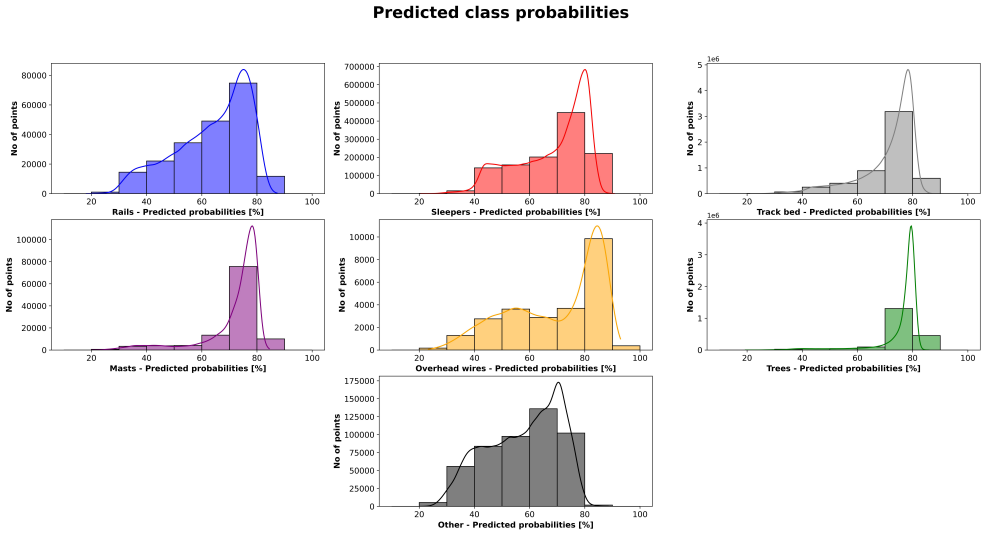



**Figure 5.10:** {DGCNN - Scenario 1} Histograms and density plots of predicted probabilities of output classes of test set.

### UAV-LiDAR Stroe segment

Overall, this model fails to predict the UAV-LiDAR Stroe segment correctly, as can be seen in Figure 5.11. Specifically, the mast is partly classified as mast (pink box A) and partly as trees (cyan box A), and the overhead wires are mostly predicted as masts (cyan box B), whereas the remaining environment is mainly classified as trees and sleepers. The low resolution of this dataset compared to the training set, that is 3 times lower than that of the training set after subsampling the latter, as well as the many defects and missing data it contains, create difficulties in its classification.

Also, the low predicted probabilities seen in Figure 5.12 verify the high uncertainty of the model in predicting all the classes of this dataset. Most of the points in sleepers, track bed and other are predicted with a probability between 30% and 40%, in masts and trees between 40% and 50%, and in overhead wires between 20% and 30%. It is worth mentioning, that no points are assigned to rails, while overhead wires consist of very few points.

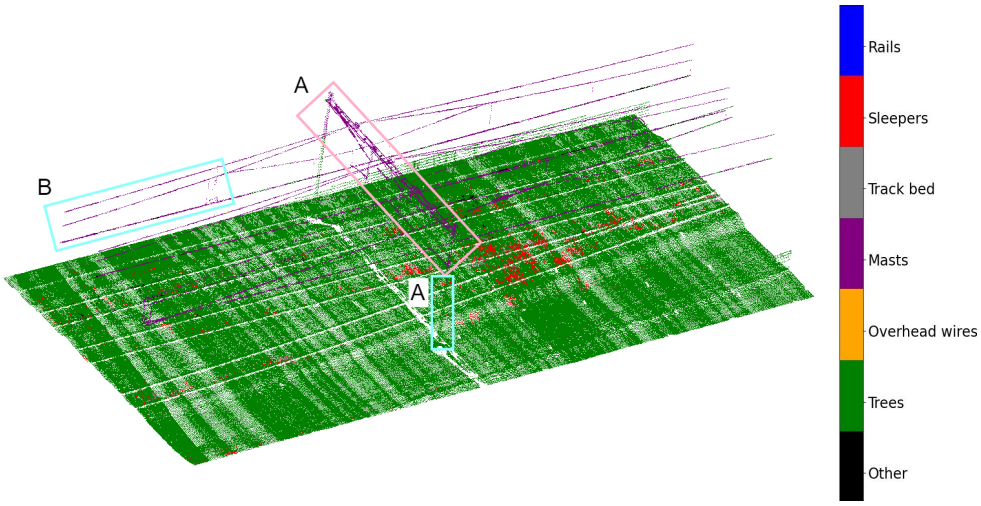



**Figure 5.11:** {DGCNN - Scenario 1} Predicted UAV-LiDAR Stroe segment. Some positive and negative results are indicated in pink and cyan boxes, respectively. Some correctly classified parts of masts are shown in pink box A. Some wrongly classified parts of masts are shown in cyan box A, and of overhead wires in cyan box B.

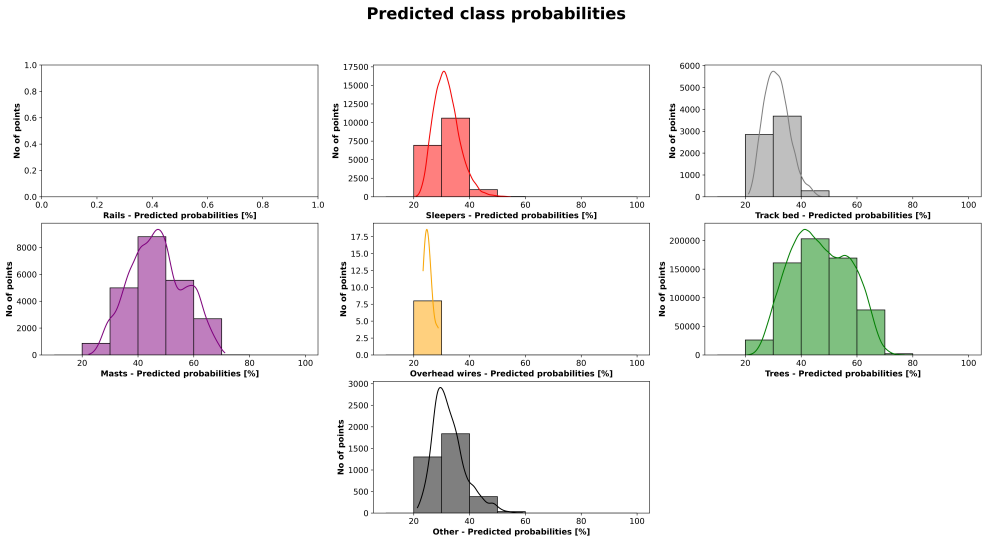



**Figure 5.12:** {DGCNN - Scenario 1} Histograms and density plots of predicted probabilities of output classes of UAV-LiDAR Stroe segment.

### 5.2.2 Scenario 2: using 3D Cartesian and normalized coordinates only (no RGB)

In scenario 2 of DGCNN, only the 3D Cartesian and normalized coordinates of the points are employed, while ignoring the color values. Corresponding to the second scenario of Random Forest, the model is trained and evaluated using the labeled

training and test sets from the UAV-SfM Groningen dataset, and is validated using an unlabeled segment from the TLS Stroe dataset. Also, training DGCNN takes around half an hour and 0.5 GB less when omitting the color values in this scenario.

### UAV-SfM Groningen test set

The classification results for the test set from this scenario are presented in Figure 5.13, where there is still confusion of rails and sleepers with track bed (box A), but not as much between track bed and sleepers as in the previous scenario. Moreover, the mast still looks fully identified, but in this case there are parts of overhead wires misclassified as masts and other objects (boxes B). In total, the model achieves 88.20% OA, 75.77% avPA and 65.73% mIoU, that is about 1%, 9% and 9%, respectively, less than when considering also the color values in the first scenario of DGCNN (see Table 5.4). Hence, color helps to not only slightly improve the OA, but also produce significantly more balanced results among the different classes. Also, the OA in this scenario of DGCNN is around 5% more than in Random Forest scenario 2. Although the difference in OA between DGCNN and Random Forest is much bigger when using also the RGB values (4% more in scenario 2 than in 1), Random Forest is still able to provide better classification results of rails and sleepers, according to F1-score values.

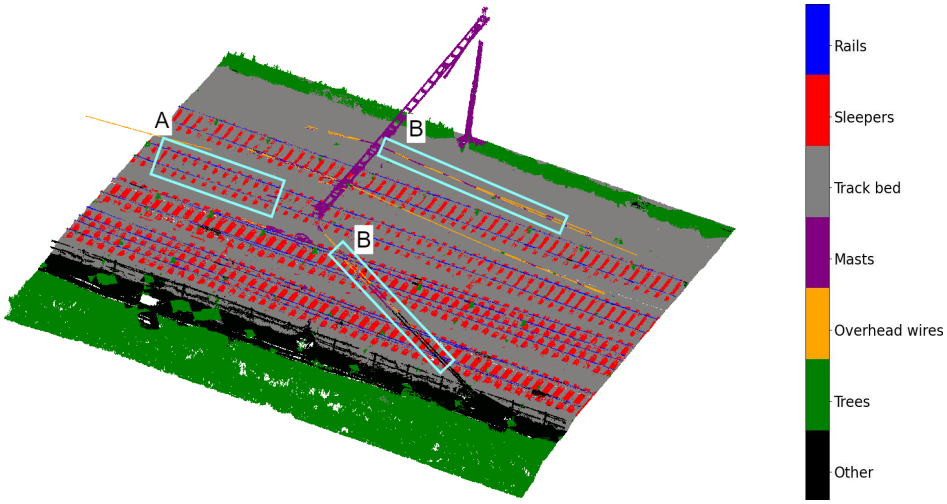



**Figure 5.13:** {DGCNN - Scenario 2} Predicted test set. Some negative results are indicated in cyan boxes. Some wrongly classified parts of rails and sleepers are shown in box A, and of overhead wires in boxes B.

**Table 5.4:** {DGCNN - Scenario 2} OA, avPA and mIoU, and per class precision, recall, F1-score and number of points.

| Class | Precision [%] | Recall [%] | F1-score [%] | No of points |
|---|---|---|---|---|
| **Rails** | 82.55 | 56.71 | 67.23 | 194,966 |
| **Sleepers** | 89.95 | 65.16 | 75.57 | 1,343,080 |
| **Track bed** | 89.49 | 93.96 | 91.67 | 5,328,327 |
| **Masts** | 89.15 | 87.79 | 88.46 | 131,408 |
| **Overhead wires** | 99.64 | 47.48 | 64.31 | 27,685 |
| **Trees** | 92.59 | 92.19 | 92.39 | 2,006,902 |
| **Other** | 55.79 | 87.11 | 68.02 | 339,313 |
| **OA [%]: 88.20** | **avPA [%]: 75.77** | | **mIoU [%]: 65.73** | |

In this case, the overhead wires have the highest precision value and the lowest recall and F1-score values, while the track bed, trees and other have the highest recall, highest F1-score and lowest precision, respectively. The classes are mostly misclassified as in the first scenario of DGCNN, except for the track bed which is mainly confused with other objects. Also, the total number of output points is the same as in scenario 1 of DGCNN, so less than the original points. Considering the probabilities (see Figure 5.14), the model presents low uncertainty in predicting most of the classes, but rails and overhead wires. Moreover, just like in scenario 1, DGCNN shows more uncertainty than Random Forest. In particular, the greatest amount of points in rails and overhead wires are predicted with a probability in the interval [40%, 50%], in sleepers in the interval [60%, 70%], in masts, trees and other in the interval [70%, 80%], and in track bed in the interval [80%, 90%]. The rest of the points are mainly spread around these probability intervals.

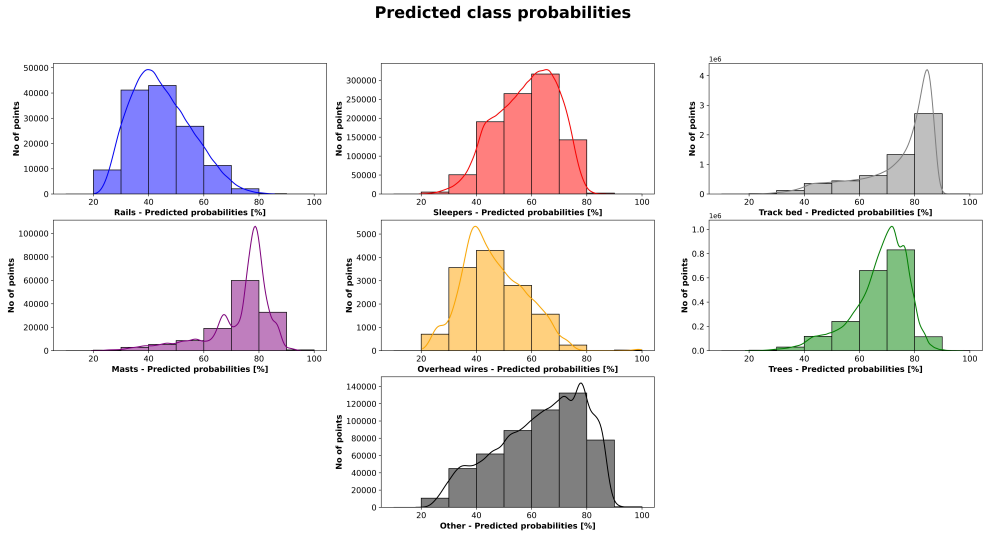



**Figure 5.14:** {DGCNN - Scenario 2} Histograms and density plots of predicted probabilities of output classes of test set.

### TLS Stroe segment

In this method, the classification results for the predicted TLS Stroe segment (see Figure 5.15) show that the majority of points are assigned to either masts, trees or track bed, in contrast to the results from Random Forest scenario 2 that there was more variation among the output categories. More specifically, the mast and overhead wires are classified as masts (pink boxes A and cyan box A), the rails, sleepers and track bed mostly as track bed and partly as trees (cyan box B and pink box B), and the trees and other mainly as trees and masts (pink boxes C, cyan box C and cyan box D). In other words, the output class masts contains the mast but also the wires and some parts of the trees, the output class track bed contains the track bed but also the rails and sleepers, and the output class trees contains the trees but also the other objects, while missing a great part of the trees that is assigned to masts.

What is already clear from Figure 5.15, it can also be observed in the probabilities in Figure 5.16, where a great amount of points are assigned to track bed, masts and trees, and very few points to the remaining classes. The largest amount of points in rails and sleepers are predicted with a probability between 20% and 30%, and in other between 30% and 40%. The high uncertainty of the model in predicting those classes is verified by the fact that these classes are not even visible in Figure 5.15. On the other hand, the majority of points in masts and trees are classified with a probability between 60% and 70%, while in track bed with a probability between 80% and 90%. As expected from the classification results, the model presents relatively low uncertainty in predicting the aforementioned classes, since the greatest part of these

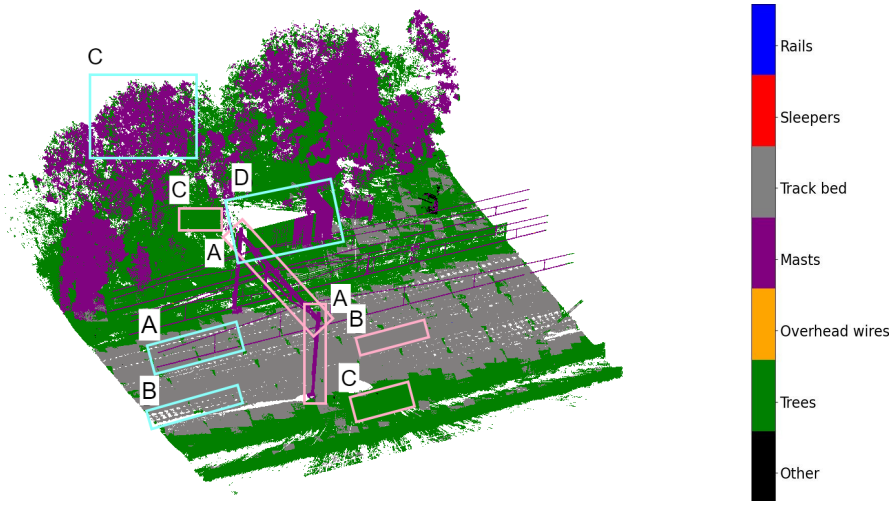



**Figure 5.15:** {DGCNN - Scenario 2} Predicted TLS Stroe segment. Some positive and negative results are indicated in pink and cyan boxes, respectively. Some correctly classified parts of masts are shown in pink boxes A, of track bed in pink box B, and of trees in pink boxes C. Some wrongly classified parts of overhead wires are shown in cyan box A, of rails and sleepers in cyan box B, of trees in cyan box C, and of other in cyan box D.

classes is classified correctly. Paradoxically, the largest amount of points in overhead wires are predicted with a probability in the range [90%, 100%]. The low uncertainty of the model in predicting the wires is different from what would be expected, given that, in practise, the entire wire system is predicted as masts, but the total number of points assigned to wires is anyway small.

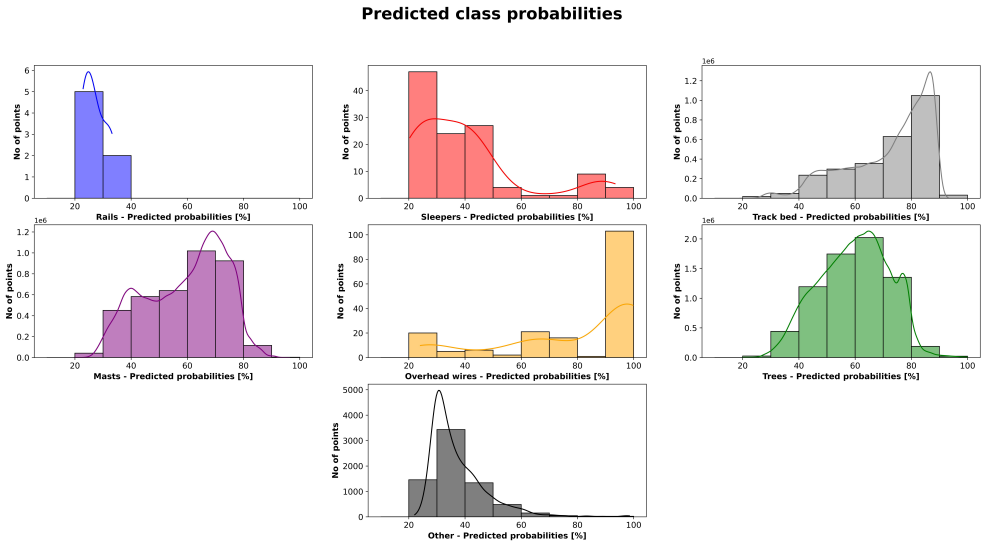



**Figure 5.16:** {DGCNN - Scenario 2} Histograms and density plots of predicted probabilities of output classes of TLS Stroe segment.

## 5.3 COMBINATION OF RANDOM FOREST AND DGCNN

In the combination of Random Forest and DGCNN, the prediction results for the test set from Random Forest scenario 1 are combined with those from DGCNN scenario 1. The label finally assigned to each point is the one predicted by the method with the highest probability, whereas for points missing from DGCNN the labels assigned

by Random Forest are used.

### UAV-SfM Groningen test set

The predicted test set from the combination of the 2 methods for scenario 1 as illustrated in Figure 5.17 shows an overall improvement of the results. Actually, 39% of the total points are predicted by Random Forest and 61% by DGCNN. Considering that Random Forest scenario 1 provides better results for the rails, sleepers and track bed, and DGCNN scenario 1 for the rest of the classes, the combination of both methods results mainly in a combination of the positive results from Random Forest with those from DGCNN. As can be seen, the rails, sleepers and track bed present small misclassifications, as indicated in boxes A and D. On the other hand, the mast and overhead wires are almost fully identified, with just a few negative results as well (boxes B and C).

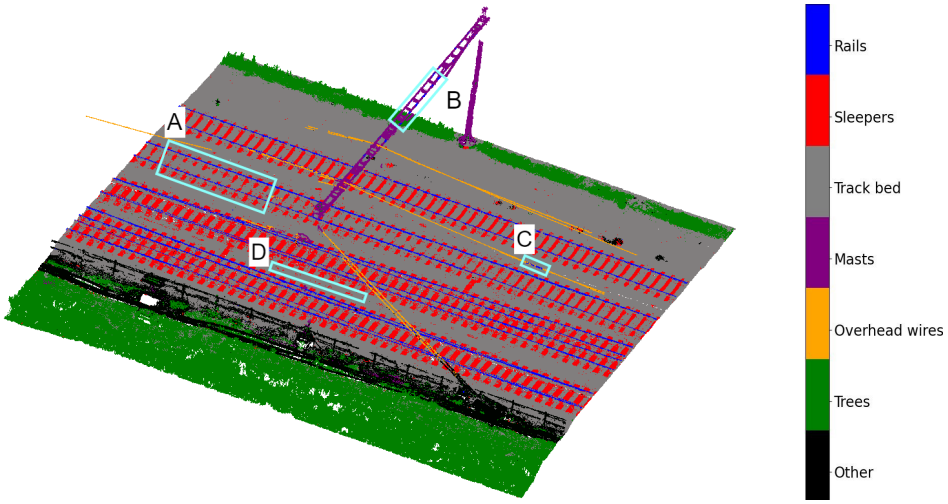



**Figure 5.17:** {Combination of Random Forest and DGCNN - Scenario 1} Predicted test set. Some negative results are indicated in cyan boxes. Some wrongly classified parts of rails and sleepers are shown in box A, of masts in box B, of overhead wires in box C, and of track bed in box D.

**Table 5.5:** {Combination of Random Forest and DGCNN - Scenario 1} OA and per class precision, recall, F1-score and number of points.

| Class | Precision [%] | Recall [%] | F1-score [%] | No of points |
|---|---|---|---|---|
| **Rails** | 84.35 | 91.37 | 87.72 | 194,966 |
| **Sleepers** | 89.32 | 76.70 | 82.53 | 1,343,083 |
| **Track bed** | 91.66 | 94.88 | 93.24 | 5,328,493 |
| **Masts** | 93.56 | 79.35 | 85.87 | 131,426 |
| **Overhead wires** | 96.05 | 83.96 | 89.60 | 27,685 |
| **Trees** | 94.86 | 91.27 | 93.03 | 2,006,955 |
| **Other** | 62.36 | 78.11 | 69.35 | 339,313 |
| **OA [%]: 90.57** | | | | |

The combination improves the OA by around 2% compared to Random Forest scenario 1 and around 1.5% compared to DGCNN scenario 1, with an OA value of 90.57% (see Table 5.5). The per class precision, recall and F1-score values are higher than the corresponding ones either of both methods or at least of one of the two.

In more detail, the track bed has the highest F1-score and recall values, while the other objects the lowest F1-score and precision values. Also, the overhead wires show the highest precision, and the sleepers the lowest recall. In this case, the majority of points in rails, sleepers and trees are misclassified as track bed, in track bed as sleepers, in masts as other, in overhead wires as masts, and in other as trees.

# 6 DISCUSSION

In this research, 2 Machine Learning (ML) methods are implemented, aiming to compare the performance of an ensemble method, that is Random Forest, with that of a Deep Learning (DL) method, that is Dynamic Graph Convolutional Neural Network (DGCNN), in pointwise classification of different 3-Dimensional (3D) point cloud data of railway environments into 7 classes of rails, sleepers, track bed, masts, overhead wires, trees and other. To this end, 2 different scenarios are developed for each method, with scenario 1 considering the color values of point clouds and scenario 2 ignoring them. In particular, in the first scenario of Random Forest 26 features are used in total, including Red Green Blue (RGB) values and 23 geometric features, while in the second scenario of Random Forest only the 23 geometric features are used. On the other hand, in scenario 1 of DGCNN 9 features are used including the 3D Cartesian coordinates of the points, RGB values and 3D normalized coordinates, while in scenario 2 of DGCNN only 6 features are used containing only the Cartesian and normalized coordinates of the points.

The 3 different datasets employed in this work consist of an Unmanned Aerial Vehicle (UAV)-Structure from Motion (SfM) dataset from Groningen with RGB values, an UAV-Light Detection And Ranging (LiDAR) dataset from Stroe with color values as well, and a Terrestrial Laser Scanning (TLS) dataset from Stroe, with only the 3D Cartesian coordinates of the points. For both methods and both scenarios, a training and test set from the UAV-SfM Groningen dataset are used to train the model and evaluate its performance according to several evaluation metrics. Afterwards, the first model of both methods is validated visually on the UAV-LiDAR Stroe dataset, and the second model on the TLS Stroe dataset. Finally, the results for the test set from Random Forest and DGCNN scenario 1 are combined, as a first attempt to further improve the final results. To this end, the predicted label for each point in the final prediction results of the test set is the one resulted from the method that predicted it with higher probability, while for points missing from DGCNN the label from Random Forest is assigned. For a step-by-step description of the methodology refer to Figure 4.1. Chapter 6 is split into 2 sections, containing a comparison among the different methods and scenarios in Section 6.1, and discussing the applicability and limitations of the methods in Section 6.2.

## 6.1 COMPARISON OF METHODS AND SCENARIOS

In this section all the results presented in Chapter 5 are summarized together, in order to compare the performance of the different methods and scenarios applied in this work. The time and memory used to train the different models in Random Forest and DGCNN scenario 1 and 2 can be seen in Table 6.1. Although typically neural networks require high computational power, in this case, Random Forest utilizes much more memory than DGCNN. This is because a large amount of features are employed for Random Forest, making the training procedure much more demanding. However, DGCNN is still more time-consuming than Random Forest, considering also that 2 Graphics Processing Units (GPUs) and Compute Unified Device Architecture (CUDA) are used for DGCNN, which speed up training, compared to the 2 Central Processing Units (CPUs) used for Random Forest due to the high memory required. Obviously, adding extra features like color values, increases the computational time and memory

in both methods. Specifically, in the first scenario of Random Forest, training takes 1 more hour compared to the second scenario, and the utilized memory is about 18 GB more. On the other hand, when taking into account also the color values in scenario 1 of DGCNN, the computational time and utilized memory increase by around half an hour and 0.5 GB, respectively. Inferentially, the computational time difference among the different methods and scenarios, as well as the utilized memory difference between the 2 scenarios of each method are trivial, but the utilized memory difference between the 2 methods is remarkable.

**Table 6.1:** Comparison of computational time and utilized memory for training of different methods and scenarios.

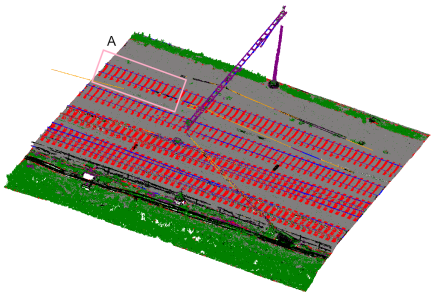
| | Random Forest | Random Forest | DGCNN | DGCNN |
|---|---|---|---|---|
| **Efficiency** | **Scenario 1** | **Scenario 2** | **Scenario 1** | **Scenario 2** |
| **Computational time [hrs]** | 6 | 5 | 7 | 6.5 |
| **Utilized Memory [GB]** | 125.58 | 107.55 | 5.65 | 5.17 |

### UAV-SfM Groningen test set

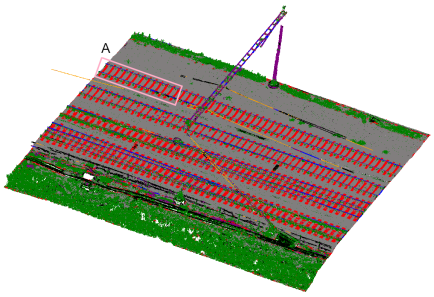
The prediction results of the test set from the different methods and scenarios are illustrated together in Figure 6.1. For both methods, in scenario 2 there is more confusion among the classes than in scenario 1. Moreover, Random Forest scenario 1 provides better classification results for rails, sleepers and track bed (Figure 6.1a - box A) than DGCNN scenario 1, while Random Forest scenario 2 gives better results for rails and sleepers (Figure 6.1b - box A) than DGCNN scenario 2. Therefore, DGCNN seems to be more able to classify correctly more components than Random Forest. It is noteworthy that, DGCNN can fully identify the mast (Figure 6.1c and Figure 6.1d - boxes A), whereas DGCNN scenario 1 gives also a full picture of the overhead wires (Figure 6.1c - box B).

Moreover, Figure 6.2, shows the barplots of Overall Accuracy (OA) and per class precision, recall and F1-score of the different methods and scenarios (for the definition of the classification metrics refer to Subsection 4.3.1). In both methods, the OA increases when taking into account the color values, with 5.26% higher accuracy in scenario 1 than 2 for Random Forest and 0.99% more accuracy in scenario 1 than 2 for DGCNN. Also, DGCNN scenario 1 achieves 0.54% higher OA than Random Forest scenario 1, and DGCNN scenario 2 achieves 4.81% higher OA than Random Forest scenario 2. Overall, no method and scenario reaches the best precision, recall and F1-score values for all classes. The best precision for rails, sleepers and other results from the first scenario of Random Forest, for track bed, masts and trees from the first scenario of DGCNN, and for overhead wires from the second scenario of DGCNN. Also, the highest recall values for sleepers and track bed are derived from Random Forest scenario 1, for rails, overhead wires and trees from DGCNN scenario 1, and for masts and other from DGCNN scenario 2. Regarding the F1-score metric, scenario 1 of Random Forest results in the highest values for rails, sleepers and track bed, scenario 1 of DGCNN for masts, overhead wires and trees, and scenario 2 of DGCNN for other.

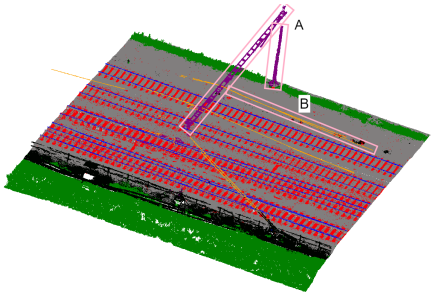
Additionally, Table 6.2 summarizes, for all methods and scenarios, the predicted class probability intervals in which the majority of points in the output classes are predicted. Random Forest shows much less uncertainty than DGCNN in predicting the majority of points for most of the classes. Random Forest scenario 1 and 2 present the same probability intervals for all classes, except for other objects where most of the points are predicted with much lower probability in scenario 2 than 1. On the other hand, DGCNN scenario 1 presents lower uncertainty than DGCNN scenario 2 in general, but not for all classes.



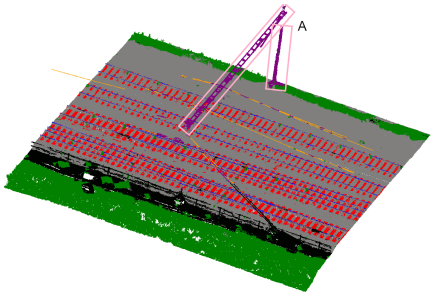

**(a)** Random Forest - Scenario 1: using RGB and 23 geometric features




**(b)** Random Forest - Scenario 2: using 23 geometric features only (no RGB)

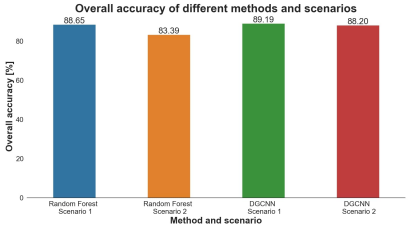



**(c)** DGCNN - Scenario 1: using RGB and 3D Cartesian and normalized coordinates

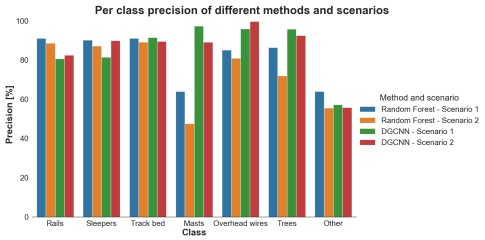



**(d)** DGCNN - Scenario 2: using 3D Cartesian and normalized coordinates only (no RGB)

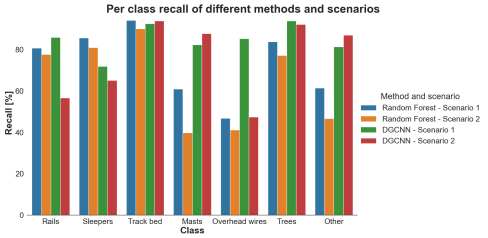
**Figure 6.1:** Comparison of predicted test set: (a) Random Forest - Scenario 1, (b) Random Forest - Scenario 2, (c) DGCNN - Scenario 1, (d) DGCNN - Scenario 2. Rails are shown in blue, sleepers in red, track bed in gray, masts in purple, overhead wires in yellow, trees in green, and other in black. Some positive results are indicated in pink boxes. Some correctly classified parts: (a) of rails, sleepers and track bed are shown in box A, (b) of rails and sleepers in box A, (c) of masts in boxes A, and of overhead wires in box B, (d) of masts in boxes A.



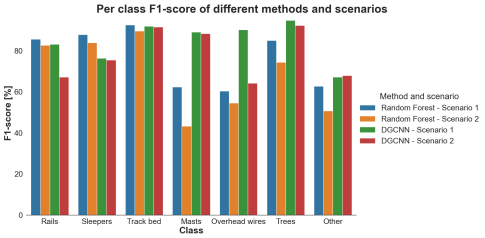

**(a)** OA of different methods and scenarios.




**(b)** Per class precision of different methods and scenarios.




**(c)** Per class recall of different methods and scenarios.




**(d)** Per class F1-score of different methods and scenarios.

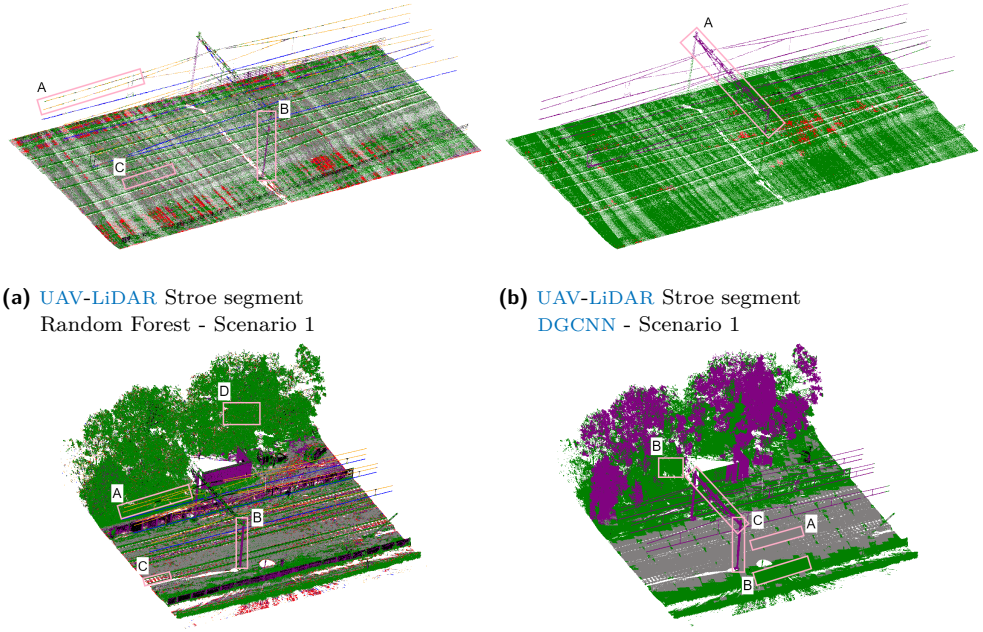
**Figure 6.2:** Comparison of: (a) OA, (b) per class precision, (c) per class recall, (d) per class F1-score of different methods and scenarios.

**Table 6.2:** Comparison of per class predicted probability intervals (for the majority of points) of test set of different methods and scenarios.

| | Predicted class probabilities [%] | | | |
|---|---|---|---|---|
| | Random Forest | Random Forest | DGCNN | DGCNN |
| Class | Scenario 1 | Scenario 2 | Scenario 1 | Scenario 2 |
| **Rails** | 90 – 100 | 90 – 100 | 70 – 80 | 40 – 50 |
| **Sleepers** | 90 – 100 | 90 – 100 | 70 – 80 | 60 – 70 |
| **Track bed** | 90 – 100 | 90 – 100 | 70 – 80 | 80 – 90 |
| **Masts** | 40 – 50 | 40 – 50 | 70 – 80 | 70 – 80 |
| **Overhead wires** | 90 – 100 | 90 – 100 | 80 – 90 | 40 – 50 |
| **Trees** | 90 – 100 | 90 – 100 | 70 – 80 | 70 – 80 |
| **Other** | 90 – 100 | 40 – 50 | 60 – 70 | 70 – 80 |

### UAV-LiDAR Stroe segment & TLS Stroe segment

The UAV-LiDAR Stroe segment as predicted in the first scenario of Random Forest and DGCNN can be seen in Figure 6.3a and Figure 6.3b. In general, the low resolution of this dataset creates difficulties in classification for both methods. Apart from




**(a)** UAV-LiDAR Stroe segment Random Forest - Scenario 1

**(b)** UAV-LiDAR Stroe segment DGCNN - Scenario 1

**(c)** TLS Stroe segment Random Forest - Scenario 2

**(d)** TLS Stroe segment DGCNN - Scenario 2

**Figure 6.3:** Comparison of predicted: (a) UAV-LiDAR Stroe segment by Random Forest - Scenario 1, (b) UAV-LiDAR Stroe segment by DGCNN - Scenario 1, (c) TLS Stroe segment by Random Forest - Scenario 2, (d) TLS Stroe segment by DGCNN - Scenario 2. Rails are shown in blue, sleepers in red, track bed in gray, masts in purple, overhead wires in yellow, trees in green, and other in black. Some positive results are indicated in pink boxes. Some correctly classified parts: (a) of overhead wires are shown in box A, of masts in box B, and of track bed in box C, (b) of masts in box A, (c) of overhead wires in box A, of masts in box B, of sleepers in box C, and of trees in box D, (d) of track bed in box A, of trees in boxes B, and of masts in boxes C.

the fact that this dataset has 3 times lower resolution than the UAV-SfM dataset used for training the models, it also contains defective components and missing data. However, Random Forest is able to predict the top overhead wires and the poles of the mast correctly, while the track bed is partly identified (Figure 6.3a - boxes A, B and C). On the other hand, DGCNN is able to predict the top part of the mast correctly (Figure 6.3b - box A), but the whole ground part is mostly classified as trees. The confusion in the output classes from both methods depicted in Figure 6.3a and Figure 6.3b is also imprinted on the predicted class probabilities (see Table 6.3). Both methods present high uncertainty (less than 50% probability) in predicting all the output classes, with some trivial differences between the 2 methods.

**Table 6.3:** Comparison of per class predicted probability intervals (for the majority of points) of UAV-LiDAR Stroe segment of different methods and scenarios.

| | **Predicted class probabilities [%]** | |
|---|---|---|
| | **Random Forest** | **DGCNN** |
| **Class** | **Scenario 1** | **Scenario 1** |
| **Rails** | 30 – 40 | - |
| **Sleepers** | 30 – 40 | 30 – 40 |
| **Track bed** | 40 – 50 | 30 – 40 |
| **Masts** | 30 – 40 | 40 – 50 |
| **Overhead wires** | 40 – 50 | 20 – 30 |
| **Trees** | 40 – 50 | 40 – 50 |
| **Other** | 30 – 40 | 30 – 40 |

The TLS Stroe segment as predicted in the second scenario of Random Forest and DGCNN is illustrated in Figure 6.3c and Figure 6.3d. In total, Random Forest is able to predict more classes of this dataset (Figure 6.3c - boxes A - D), but with more confusion among the different classes. On the flip side, DGCNN shows a more clear classification picture of the data, but assigns the points mainly to 3 classes, that is track bed, trees and masts (Figure 6.3d - boxes A - C). Moreover, Random Forest is much more capable of predicting the big trees in this dataset that are not met in the training set, while DGCNN misclassifies large parts of this kind of trees, although it is able to classify low vegetation correctly.

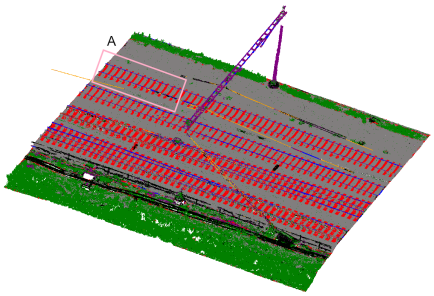
Nonetheless, the classification results are still better than those in the UAV-LiDAR Stroe segment, as expected, since the quality of the TLS Stroe dataset is quite better. In fact, the latter has 2 times higher resolution than the UAV-LiDAR Stroe dataset. That is the reason why the predicted class probabilities in Table 6.4 are somehow higher than the respective in Table 6.3. Generally, Random Forest shows higher probabilities for the majority of points in most of the classes than DGCNN, but both methods present relatively high uncertainty in predicting most of the classes. It is worth mentioning, that it is possible sometimes for the methods to predict an output class with very low uncertainty, although the predicted class is actually completely wrong. For instance, Random Forest predicts rails with very high probabilities, although they practically represent the bottom overhead wires. This stems from the fact that rails and bottom overhead wires are components that look very alike to each other, and, at the same time, the model has been trained on a dataset where the overhead wires are not representative of the original ones on railways.

**Table 6.4:** Comparison of per class predicted probability intervals (for the majority of points) of TLS Stroe segment of different methods and scenarios.

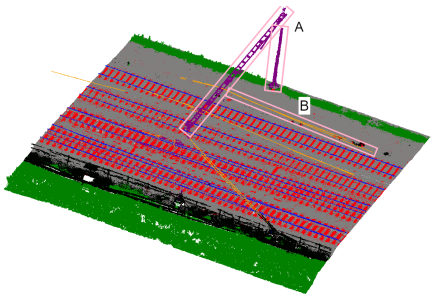
| | Predicted class probabilities [%] | |
|---|---|---|
| | Random Forest | DGCNN |
| Class | Scenario 2 | Scenario 2 |
| **Rails** | 90 – 100 | 20 – 30 |
| **Sleepers** | 30 – 40 | 20 – 30 |
| **Track bed** | 40 – 50 | 80 – 90 |
| **Masts** | 40 – 50 | 60 – 70 |
| **Overhead wires** | 90 – 100 | 90 – 100 |
| **Trees** | 80 – 90 | 60 – 70 |
| **Other** | 40 – 50 | 30 – 40 |

**Combined and independent methods for scenario 1: UAV-SfM Groningen test set**

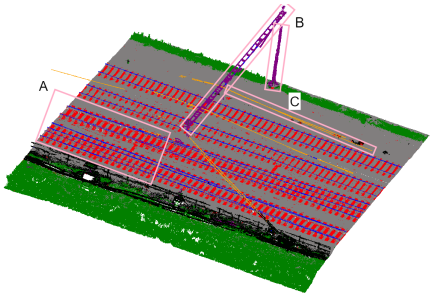
The predicted test set from Random Forest, DGCNN, and the combination of both methods for scenario 1 can be seen all together in Figure 6.4, while Figure 6.5 depicts the barplots of Overall Accuracy (OA) and per class precision, recall and F1-score of




**(a)** Random Forest - Scenario 1



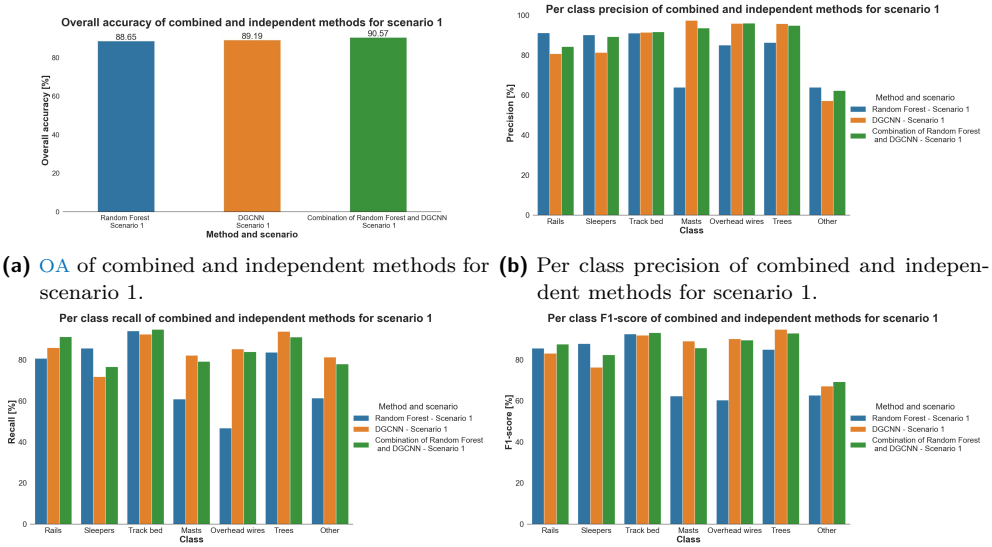

**(b)** DGCNN - Scenario 1




**(c)** Combination of Random Forest and DGCNN - Scenario 1

**Figure 6.4:** Comparison of predicted test set of combined and independent methods for scenario 1: (a) Random Forest - Scenario 1, (b) DGCNN - Scenario 1, (c) Combination of Random Forest and DGCNN - Scenario 1. Rails are shown in blue, sleepers in red, track bed in gray, masts in purple, overhead wires in yellow, trees in green, and other in black. Some positive results are indicated in pink boxes. Some correctly classified parts: (a) of rails, sleepers and track bed are shown in box A, (b) of masts in boxes A, and of overhead wires in box B, (c) of rails, sleepers and track bed in box A, of masts in boxes B, and of overhead wires in box C.

the combined and independent methods. Both the prediction results as well as the classification metrics show that the combination of the 2 methods using the predicted probabilities helps to improve the final results compared to those from the individual methods. The combination makes, mainly, use of the best results from each method. Thus, as can be observed in Figure 6.4c, there is a more clear picture of the rails, sleepers and track bed as indicated in box A, the mast is fully identified (boxes B), while the overhead wires are almost fully correctly classified (box C). Furthermore, the combination results in about 2% higher OA than Random Forest, and about 1.5% than DGCNN. Also, for all classes, it achieves better precision, recall and F1-score values than at least one of the two methods, and even the best ones in some cases.




**(a)** OA of combined and independent methods for scenario 1.

**(b)** Per class precision of combined and independent methods for scenario 1.

**(c)** Per class recall of combined and independent methods for scenario 1.

**(d)** Per class F1-score of combined and independent methods for scenario 1.

**Figure 6.5:** Comparison of: (a) OA, (b) per class precision, (c) per class recall, (d) per class F1-score of combined and independent methods for scenario 1.

## 6.2 APPLICABILITY AND LIMITATIONS OF METHODS

### Comparison with existing methods

Earlier studies applied classification of railway point clouds into several railway components, while identifying some common and some different components compared to this work, as discussed in Section 2.7. Comparing this research to studies where the full railway environment was classified from point cloud data, Luo et al. [2014] applied a Conditional Random Field (CRF) classifier to classify railway point clouds from Mobile Laser Scanning (MLS) data into terrain, railway bed, railway tracks, vegetation, platform, noise barriers, electronic posts, attachments on electronic post and power lines, and achieved 94.43% OA using the CRF classifier with contextual information. However, due to the use of contextual features, the context classification developed in that work is sensitive to the quality of point cloud data. In this work, the best OA of 89.19% is reached by DGCNN scenario 1, which is around 5.2% less than in the aforementioned work. Nonetheless, this method excels in the sense that the user does not have to draw insights from the data to create any special features. Instead, it simply uses the basic features of coordinates and colors of the points that most LiDAR data already contain, and the rest of the features are created automatically within the

network. Also, when Random Forest scenario 1 and DGCNN scenario 1 are combined, the difference with CRF in OA decreases to about 3.9%.

Furthermore, Sánchez-Rodríguez et al. [2018] classified 3 different MLS point clouds of railway tunnels into 6 categories of ground, rails, lining, catenary wires, cantilevers and other elements. Analytically, the data were first preprocessed, and then divided into ground and non-ground points to detect the components found in these clouds, using several different algorithms. In that work, no OA was reported, but the F1-score values of all the classes and for all the 3 datasets were between 71% and 99%. Compared to our work, the best overall F1-scores are also met in the first scenario of DGCNN, where the scores range from 67% to 94%, that is 4% less in the lower limit and 5% less in the upper limit. However, the advantage of the present research is that after the preparation of the data, both Random Forest and DGCNN are able to classify the entire railway environment without taking any intermediate steps, and the trained classifiers can directly be used for the classification of other railway point clouds. Moreover, in the combined method, the F1-score values range between 69% to 93%, so the lower limit is 2% less than in the related work and 2% more than in DGCNN scenario 1, while the upper limit is 6% less and 1% less, respectively.

**Table 6.5:** Comparison of methods applied with existing methods.

| Method | OA [%] | F1-score [%] |
|---|---|---|
| **Random Forest - Scenario 1** | 88.65 | 60 – 92 |
| **Random Forest - Scenario 2** | 83.39 | 43 – 89 |
| **DGCNN - Scenario 1** | 89.19 | 67 – 94 |
| **DGCNN - Scenario 2** | 88.20 | 64 – 92 |
| **Combination of Random Forest and DGCNN - Scenario 1** | 90.57 | 69 – 93 |
| **CRF with contextual information** (Luo et al. [2014]) | 94.43 | - |
| **Composed method** (Sánchez-Rodríguez et al. [2018]) | - | 71 – 99 |

Table 6.5 provides a comparison between the methods applied in this work and existing methods. In any case, we cannot draw safe conclusion regarding the performance of different methods in the classification of railway point clouds, considering that there are no benchmark datasets for this purpose. Instead, the methods are applied on different datasets with different characteristics, and the categories selected to be identified do not coincide in the different projects.

### Applicability and pros and cons of methods

Overall, both methods used in this study approximate the state-of-the-art performance usually achieved in this field, although they are not absolutely comparable to other methods, as stated before. In addition, the fact that the training data here are created manually should be taken into account, since manual annotation inserts additional errors. The main limitation of both methods is that they are sensitive to the quality of the data used. Both methods perform fairly good at predicting the dataset they are trained on, but fail to generalize on other datasets with different point density and different object quality. Also, the lower the resolution and the more the defects and missing data in a dataset, the worse the prediction results. Hence, the UAV-LiDAR Stroe dataset that has 2 times lower resolution and more defects and missing data than the TLS dataset, gives worse prediction results, despite of the fact that it also contains color values. Therefore, the methods can perform well on point

clouds of similar quality and characteristics, but they are not applicable on every possible dataset.

Clearly, in both methods, the color values help to improve, in general, the overall and per class classification results on the same data. Additionally, for both methods, high resolution data need to be subsampled in order to reduce the computational time and memory during training. In particular, the required resources for Random Forest are already high. This means that retaining the original resolution of the training set would make it impossible to train Random Forest, since we should request such a large memory that we would never get priority on the High Performance Computing (HPC) cluster. The main issue for DGCNN when not applying subsampling relies mostly on the testing procedure, given that all the points are used for testing and the required memory is also not available.

Another big drawback of Random Forest, is that the geometric features developed to train the classifiers are made to fit the training data. For different training data, different geometric features should be incorporated, that are able to provide a clear distinction between the different classes. This is a trial-and-error procedure that can be time-consuming until reaching a sufficient set of feature, while their calculation requires computational time and power. Moreover, the use of a large amount of features increases the computation complexity of the model. Regarding DGCNN, some of the parameters also need to be defined by trial and error to suit the data. Furthermore, DGCNN results in slightly less points than the original ones, since blocks containing less points than the neighborhood size are discarded. The difference depends on the distribution of the data in 3D space, but normally the amount of input and output points does not differ significantly and does not affect the visualization of the results. Another downside of this implementation is that, for large block or neighborhood sizes, the testing and prediction processes, that use the full input points, require large amount of memory not available even in HPC machines.

Generally speaking, although DGCNN performs better than Random Forest, we cannot come to absolute conclusion on which method is actually better, since the geometric features used in Random Forest are developed by the user, and different features can result in different outcomes. Nonetheless, Random Forest is more controllable and comprehensible, since different features can be incorporated to improve the performance of the model. On the contrary, DGCNN uses some basic features and the remaining features are generated within the network, making the main procedure a black box to the user. Obviously, this brings us back to the previous statement that the amount of features required from Random Forest to be efficient, makes the whole procedure more time-consuming and computationally complex. Therefore, although Random Forest is easier to implement than DGCNN, the preparation of the data before feeding the classifier consists of its biggest disadvantage.

Considering the accuracy requirements when classifying objects in a railway infrastructure, according to CGI, the methods should reach state-of-the-art OA of more than 90%. The first scenario of both methods does not meet the requirements, but still approaches them with an OA value of 88.65% for Random Forest and 89.19% for DGCNN. However, both methods are very promising, and the combination of the first scenario of both methods offers the possibility of achieving the required accuracy, with an OA value of 90.57%. Moreover, in this work, the railway point clouds are classified into 7 categories of rails, sleepers, track bed, masts, overhead wires, trees and other. Other potential classes could be to separate the actual masts from the insulators, as well as decompose the overhead wires into their several components, such as catenary wires, contact wires, droppers and other wires. To further extend this work, next steps can be to reconstruct the classified objects from the point clouds, and examine whether the position of the objects deviates by less than 1.5 cm from the original objects to finally meet the requirements of ProRail.

# 7 CONCLUSION AND RECOMMENDATIONS

Chapter 7 is divided into the conclusion related to this work in Section 7.1, and the recommendations for future work in Section 7.2.

## 7.1 CONCLUSION

The objective of this research is to investigate to what extent can a railway infrastructure be mapped in its asset components using 3-Dimensional (3D) point cloud data and Machine Learning (ML). Specifically, 2 ML methods are applied, an ensemble method which is Random Forest, and a Deep Learning (DL) method which is Dynamic Graph Convolutional Neural Network (DGCNN), to classify railway point clouds into 7 categories of rails, sleepers, track bed, masts, overhead wires, trees and other. The methods are validated under 3 case studies, an Unmanned Aerial Vehicle (UAV)-Structure from Motion (SfM) dataset from Groningen with Red Green Blue (RGB) values, an UAV-Light Detection And Ranging (LiDAR) dataset from Stroe with RGB as well, and a Terrestrial Laser Scanning (TLS) dataset from Stroe, with only the 3D Cartesian coordinates of the points. For each method, 2 scenarios are developed, scenario 1 and 2, for classifying colored and uncolored point clouds, respectively. The first scenario of Random Forest employs RGB values and 23 geometric features and the second only the 23 geometric features, whereas scenario 1 of DGCNN uses the 3D Cartesian coordinates, RGB values and 3D normalized coordinates of the points and scenario 2 only the Cartesian and normalized coordinates of the points. In this context, a training and test set from the UAV-SfM Groningen dataset is used to train the 4 models and evaluate their performance. After that, the unlabeled UAV-LiDAR Stroe dataset and TLS Stroe dataset are used to validate visually the performance of the models in the first and second scenario, respectively, of both methods. Finally, a first attempt to combine both methods is applied to further improve the final results. For this purpose, the prediction results for the test set from the first scenario of Random Forest and DGCNN are combined using for each point the predicted label from the method that predicted it with higher probability, and for missing from DGCNN points the predicted label from Random Forest.

In general, both methods show very promising results, which verify the high potential of the combination of ML techniques and 3D point clouds in classifying railway infrastructures in their asset components sufficiently. The research subquestions, as structured in Section 1.3 into 3 main categories of mapping requirements, machine learning approaches and project possibilities, are answered subsequently in the following order: machine learning approaches, project possibilities, mapping requirements.

**Machine learning approaches:**

In order to train and test the models, the training and test sets are chosen from the dataset with the highest resolution out of the 3 datasets used in this research. The remaining 2 datasets have not been labeled to avoid extra manual labor. Also, we aim to keep the typical ratio of $60\% - 80\%$ points for training and $40\% - 20\%$ for testing, while the 2 sets have a distance of around 25 meters to ensure more reliable evaluation of the models. Particularly for Random Forest, the parameters are not optimized, but a set of useful features have been developed, which can distinguish the

different categories. Regarding the DGCNN implementation, most of the parameters are kept as suggested by the authors, but others like block and batch size are tuned to suit the data, while the final models are the optimal ones after 50 epochs of training.

The performance of the models is evaluated based on several classification metrics, including the Overall Accuracy (OA) of the model, the per class precision, recall and F1-score, as well as an elaborate confusion matrix that shows clearly the confusion between the different classes. Additionally, for DGCNN, the mean Intersection over Union (mIoU) and the average Per class Accuracy (avPA) are calculated. The mIoU is a common metric used in segmentation tasks, while the avPA is a useful measure when the classes are imbalanced. Incorporating all these metrics in the study can help getting a better insight of the results, and compare them to other related works. Furthermore, in order to investigate the uncertainty of the models on classifying the different railway components, the predicted class probabilities are computed for each predicted segment, and the highest probabilities for all the points of each output class are summarized.

**Project possibilities:**

The resulting OA of Random Forest scenario 1, Random Forest scenario 2, DGCNN scenario 1, DGCNN scenario 2, and the combination of Random Forest and DGCNN scenario 1 is 88.65%, 83.39%, 89.19%, 88.20% and 90.57%, respectively. The corresponding per class F1-scores range between 60% and 92%, 43% and 89%, 67% and 94%, 64% and 92%, and 69% and 93%. Overall, the methods developed approximate state-of-the-art performance, compared to other related works. However, they are not absolutely comparable, given that other approaches employ different data and classify railway point clouds into different categories. Regarding the predicted probabilities, the uncertainty of the model is more often in line with the results. Although there are cases where the classifiers confuse components of similar shape and structure, which are then assigned to a completely different class, usually the models show lower uncertainty when the prediction results are better and the other way around. The predicted probabilities can then be exploited to further improve the final results, like in the combination applied in this work.

When comparing the 2 different methods implemented in this study, we cannot draw safe conclusion regarding which method performs better in general, since the geometric features used for Random Forest are developed by the author, and different feature combinations can give different outcomes. Nonetheless, Random Forest is easier to implement and more handy and manageable than DGCNN, but the features used as input to the algorithm need to be defined by trial and error, their computation is time-consuming, while they also increase the computation complexity during training. On the flip side, DGCNN uses only some basic features commonly included in LiDAR point clouds, but works like a black box, making it hard for the user to control the outputs. Additionally, some of the parameters of the models should be tuned accordingly to fit the data. Also, the resulting points are less than the original points, since blocks with points less than the neighborhood size are discarded. The difference between original and output points depends on the distribution of the data in 3D space, but it is typically very small and does not affect the visualization of the results. Another drawback of this implementation is that large block or neighborhood sizes can cause out-of-memory error in testing or prediction, where all the input points are used. Moreover, for both methods, high resolution data need to be subsampled to be fed into the algorithms, in order to reduce the required computational time and memory and make their processing feasible, but this depends always on the point density and total number of points of the input point clouds. In total, the big advantage of both methods is that the trained models can be used directly to predict any railway point cloud, after preparing it to the required input format.

However, both methods are not able to generalize when applied on data with very different resolution and object quality. This does not preclude them from being appli-

cable to other data from different sensor systems and different areas, considering that the 3 case studies in this work vary significantly in terms of point density, while all 3 datasets present defects and missing data. Therefore, it is already too optimistic trying to generalize on such data, and it would be of great interest to test those methods also on other datasets with higher overall quality. Furthermore, the contribution of color in point clouds is important, since they improve the OA of the models, as well as the classification results of the individual classes. When ignoring those values, the results are still acceptable, but taking them into account helps in approaching even closer to state-of-the-art accuracy.

**Mapping requirements:**

The accuracy requirements when classifying objects in a railway infrastructure are more than 90% OA, as defined by CGI, to be aligned with the accuracy of top-of-the-line methods in this field. The methods implemented in this work are able to exceed 90% OA when combined. Hence, the individual methods are already very close to this goal, but the combination makes the results even more promising and manages to finally meet the requirements. Moreover, in the present research, the railway point clouds are classified into 7 categories, which are rails, sleepers, track bed, masts, overhead wires, trees and other. The work can be improved by identifying the actual masts and the insulators, as well as the different cables in the overhead wires as separate classes. Finally, to further extend this work and achieve the requirements of ProRail, the classified objects from the point clouds can be reconstructed to examine whether the detected objects deviate by less than 1.5 cm from the original objects.

## 7.2 RECOMMENDATIONS

The interesting results of this research pave the way for future implementations, while there are still different approaches to consider and room for improvement. Some recommendations for future work are listed below:

- Train Random Forest and DGCNN using multiple data from different sensor systems and different areas.
- Test the performance of Random Forest and DGCNN on other railway point clouds with more advanced object quality.
- Export the features generated by DGCNN and use them as input to the Random Forest classifiers.
- Train Random Forest using contextual features.
- Separate first ground and non-ground points, and then apply Random Forest and DGCNN to classify the 2 groups of points individually.
- Test and compare the performance of other ML methods in classification of railway point clouds.
- Add more classes to further decompose railway point clouds into more components.
- Create a benchmark dataset to compare the performance of different methods in classification of point clouds of railway environments.
- Reconstruct the classified objects from the point clouds and examine if the detected objects meet the requirements of ProRail (deviate by less than 1.5 cm from the original objects).

Additionally, some technical hints for future implementations are given below:

- Training data: Apply spatial splitting to split spatial data into training and test sets. Ideally, retain some distance between training and test sets to avoid information leakage from training to test set, and ensure more reliable evaluation of the performance of the model. Also, maintain the ratio $60-80\%$ for training and $40-20\%$ for testing, and make sure that training data are enough to make the model able to learn. If the performance is poor, there is a high possibility that the training data are not sufficient and more data should be incorporated.

- Random Forest: Develop useful features that can provide clear distinction between the different classes. Then, use feature importances to check which features do not contribute significantly to the training process and leave them out. The more features you use the higher the computational complexity during training. Specifically for the geometric features, `CloudCompare` offers a faster and more elegant way to compute and viluazlize them, while the respective implementation in Python increases the computational time significantly (mainly because of the use of `KDTree` to compute the neighbordoods). Before feeding the data into the Random Forest classifier, shuffle them randomly and normalize them using a scaler. Then apply the same scaler to any data used for testing or prediction. In that way, you can make the learning of the algorithm more robust.

- DGCNN: Prepare the data in the same for as the indoor benchmark Stanford large-scale 3D Indoor Spaces (S3DIS), based on which the method is developed. Then, define the block size that best suits your data by trial and error, as it is one of the most crucial factors in the learning of the network. Larger block sizes can be applied for point clouds of larger areas and lower point density.

# BIBLIOGRAPHY


Aghdam, H. H. and Heravi, E. J. (2017). *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer.

Arastounia, M. (2012). *Automatic Classification of LiDAR Point Clouds in a Railway Environment*. Thesis, University of Twente, Faculty of Geo-Information and Earth Observation (ITC).

Arastounia, M. (2015). Automated Recognition of Railroad Infrastructure in Rural Areas from LIDAR Data. *Remote Sensing*, 7(11):14916–14938. doi: 10.3390/rs71114916.

Arastounia, M. and Oude Elberink, S. (2016). Application of Template Matching for Improving Classification of Urban Railroad Point Clouds. *Sensors*, 16(12):2112. doi: 10.3390/s16122112.

Bai, Q. (2020). *Semantic Segmentation of AHN3 Point Clouds with DGCNN*. Additional Thesis, Delft University of Technology, Department of Geoscience and Remote Sensing.

Bello, S. A., Yu, S., Wang, C., Adam, J. M., and Li, J. (2020). Review: Deep Learning on 3D Point Clouds. *Remote Sensing*, 12(11):1729. doi: 10.3390/rs12111729.

Chollet, F. (2017). *Deep Learning with Python*. Manning.

Corongiu, M., Masiero, A., and Tucci, G. (2020). Classification of railway assets in mobile mapping point clouds. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43:219–225. doi: 10.5194/isprs-archives-XLIII-B1-2020-219-2020.

Cozza, A. (2005). *Railways EMC: Assessment of Infrastructure Impact*. PhD Dissertation, Politecnico di Torino, Doctorate in Electronic and Communications Engineering.

Dong, P. and Chen, Q. (2017). *LiDAR Remote Sensing and Applications*. CRC Press.

Guissous, A. E. (2019). Skin Lesion Classification using Deep Neural Network. *arXiv preprint*. arXiv: 1911.07817.

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., and Bennamoun, M. (2020). Deep Learning for 3D Point Clouds: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. doi: 10.1109/TPAMI.2020.3005434.

Hartshorn, S. (2016). *Machine Learning with Random Forests and Decision Trees: A Visual Guide for Beginners*. Kindle Direct Publishing.

Jolliffe, I. T. and Cadima, J. (2016). Principal Component Analysis: A Review and Recent Developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202. doi: 10.1098/rsta.2015.0202.

Luo, C., Jwa, Y., and Sohn, G. (2014). Context-based Multiple Railway Object Recognition from Mobile Laser Scanning Data. In *2014 IEEE Geoscience and Remote Sensing Symposium*, pages 3602–3605. doi: 10.1109/IGARSS.2014.6947262.

Müller, A. C., Guido, S., et al. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, Inc.

Oguchi, T., Yuichi, S. H., and Wasklewicz, T. (2011). *Developments in Earth Surface Processes*. Elsevier.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85. doi: 10.1109/CVPR.2017.16.

Russell, R. (2018). *Machine Learning: Step-by-Step Guide to Implement Machine Learning Algorithms with Python*. CreateSpace.

Sánchez-Rodríguez, A., Riveiro, B., Soilán, M., and González-deSantos, L. (2018). Automated Detection and Decomposition of Railway Tunnels from Mobile Laser Scanning Datasets. *Automation in Construction*, 96:171–179. doi: 10.1016/j.autcon.2018.09.014.

Sánchez-Rodríguez, A., Soilán, M., Cabaleiro, M., and Arias, P. (2019). Automated Inspection of Railway Tunnels' Power Line using LiDAR Point Clouds. *Remote Sensing*, 11(21):2567. doi: 10.3390/rs11212567.

Virtanen, J.-P., Hyyppä, H., Kurkela, M., Vaaja, M., Alho, P., and Hyyppä, J. (2014). Rapid Prototyping—A Tool for Presenting 3-Dimensional Digital Models Produced by Terrestrial Laser Scanning. *ISPRS International Journal of Geo-Information*, 3(3):871–890. doi: 10.3390/ijgi3030871.

Wang, H., Cai, Z., Luo, H., Wang, C., Li, P., Yang, W., Ren, S., and Li, J. (2012). Automatic Road Extraction from Mobile Laser Scanning Data. In *2012 International Conference on Computer Vision in Remote Sensing*, pages 136–139. doi: 10.1109/CVRS.2012.6421248.

Wang, Y., Chen, Q., Zhu, Q., Liu, L., Li, C., and Zheng, D. (2019a). A Survey of Mobile Laser Scanning Applications and Key Techniques over Urban Areas. *Remote Sensing*, 11(13):1540. doi: 10.3390/rs11131540.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019b). Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions On Graphics (TOG)*, 38(5):1–12. doi: 10.1145/3326362.

Westoby, M., Brasington, J., Glasser, N., Hambrey, M., and Reynolds, J. (2012). 'Structure-from-Motion' Photogrammetry: A Low-cost, Effective Tool for Geoscience Applications. *Geomorphology*, 179:300–314. doi: 10.1016/j.geomorph.2012.08.021.

Widyaningrum, E., Fajari, M., Lindenbergh, R., and Hahn, M. (2020). Tailored Features for Semantic Segmentation with a DGCNN using Free Training Samples of a Colored Airborne Point Cloud. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 43(B2):339–346. doi: 10.5194/isprs-archives-XLIII-B2-2020-339-2020.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). Convolutional Neural Networks: An Overview and Application in Radiology. *Insights into Imaging*, 9(4):611–629. doi: 10.1007/s13244-018-0639-9.

Yilmaz, O. and Karakus, F. (2013). Stereo and Kinect Fusion for Continuous 3D Reconstruction and Visual Odometry. In *2013 International Conference on Electronics, Computer and Computation (ICECCO)*, pages 115–118. doi: 10.1109/ICECCO.2013.6718242.

# A GEOMETRIC FEATURES

Appendix A includes the visualization of the geometric features of the training set computed using a neighborhood radius of 2 cm in Section A.1, and a neighborhood radius of 20 cm in Section A.2.

## A.1 USING A NEIGHBORHOOD RADIUS OF 2 CM

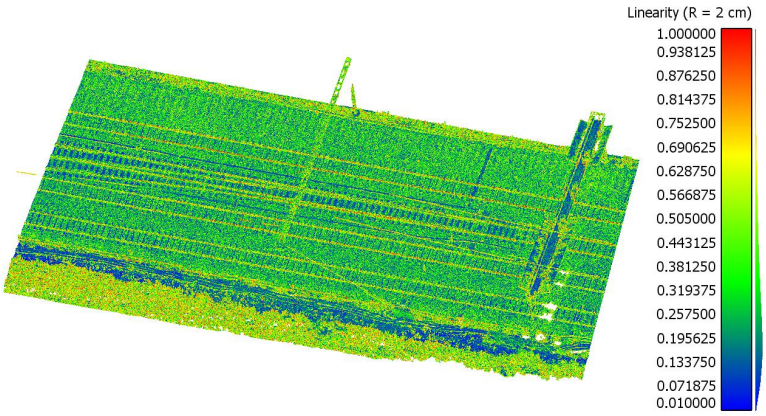



**Figure A.1:** Linearity ($R = 2$ cm).

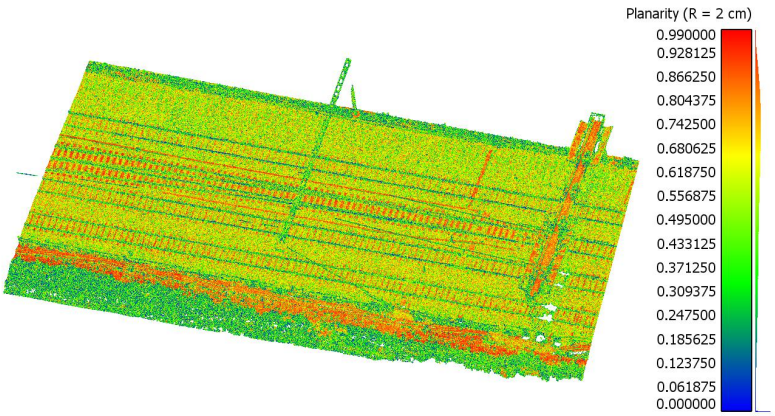



**Figure A.2:** Planarity ($R = 2$ cm).

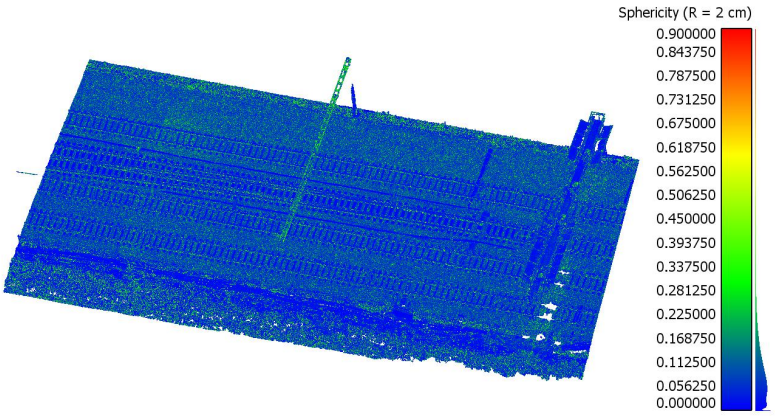



**Figure A.3:** Sphericity ($R = 2$ cm).

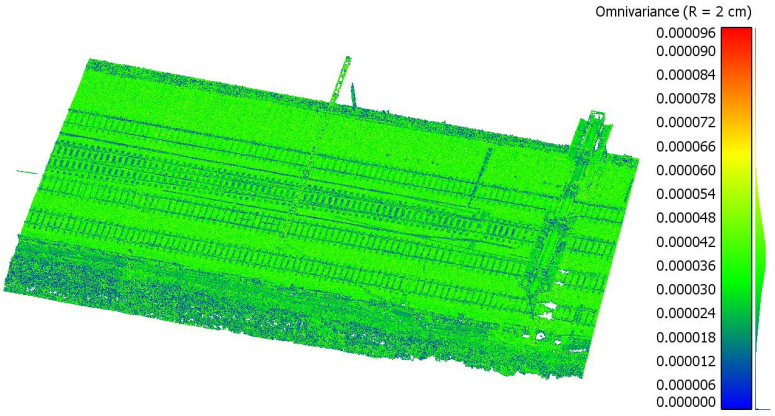



**Figure A.4:** Omnivariance ($R = 2$ cm).

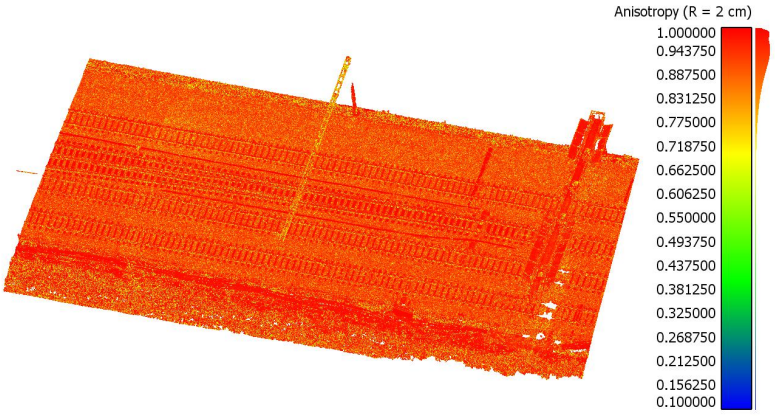



**Figure A.5:** Anisotropy ($R = 2$ cm).

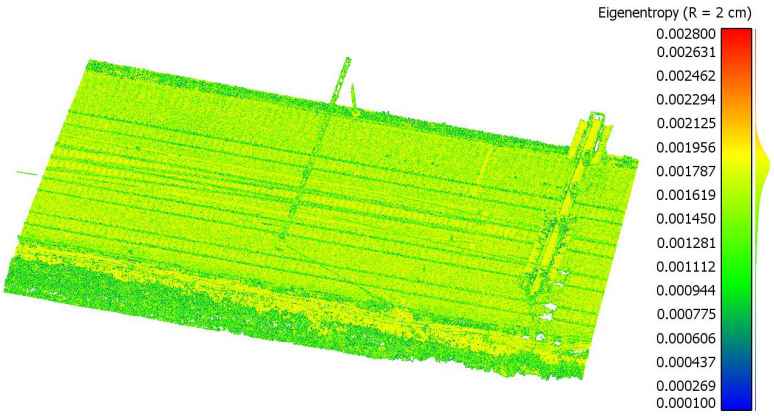



**Figure A.6:** Eigenentropy ($R = 2$ cm).

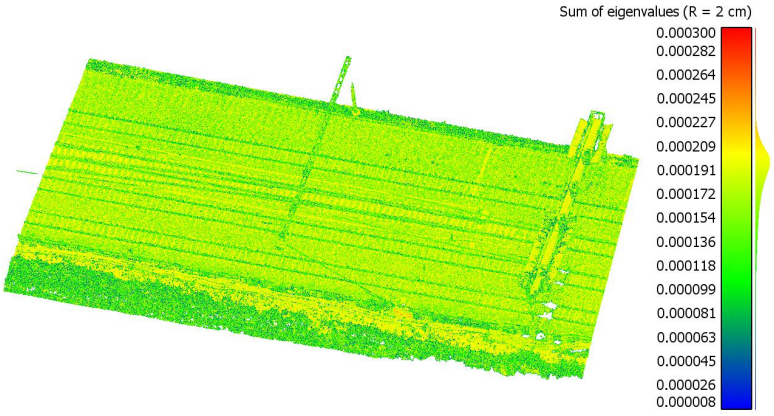



**Figure A.7:** Sum of eigenvalues ($R = 2$ cm).

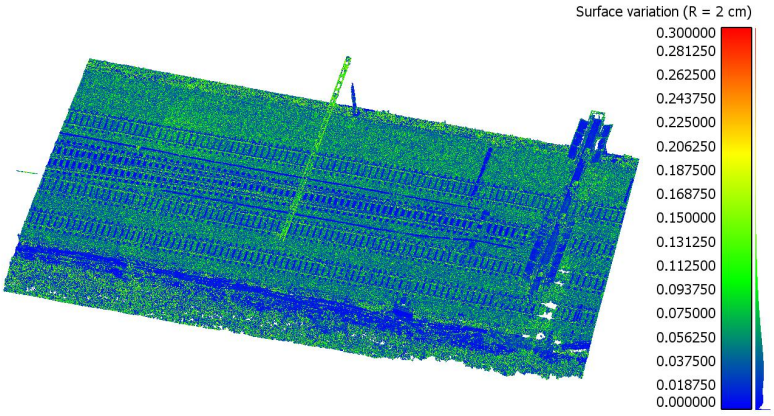



**Figure A.8:** Surface variation ($R = 2$ cm).

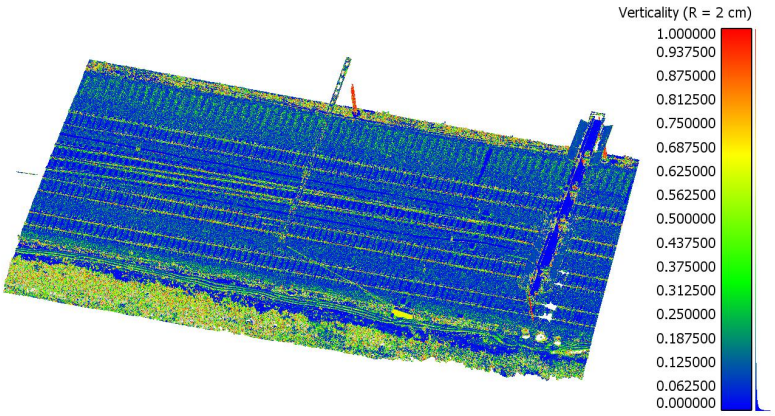



**Figure A.9:** Verticality ($R = 2$ cm).

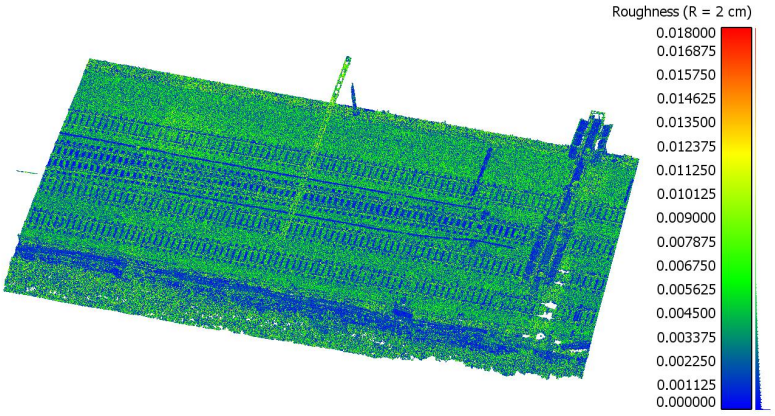



**Figure A.10:** Roughness ($R = 2$ cm).

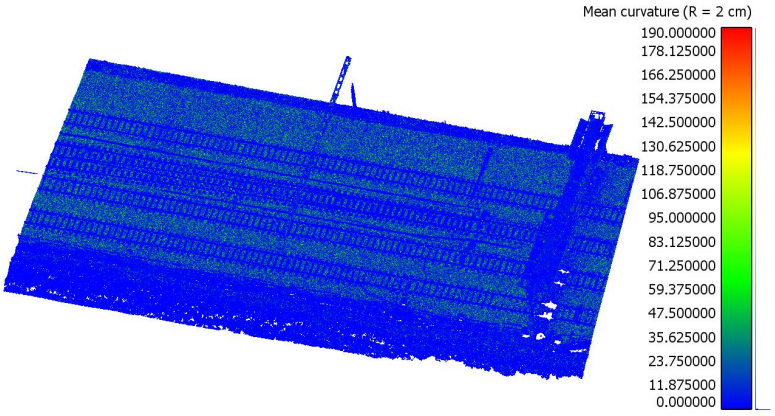



**Figure A.11:** Mean curvature ($R = 2$ cm).

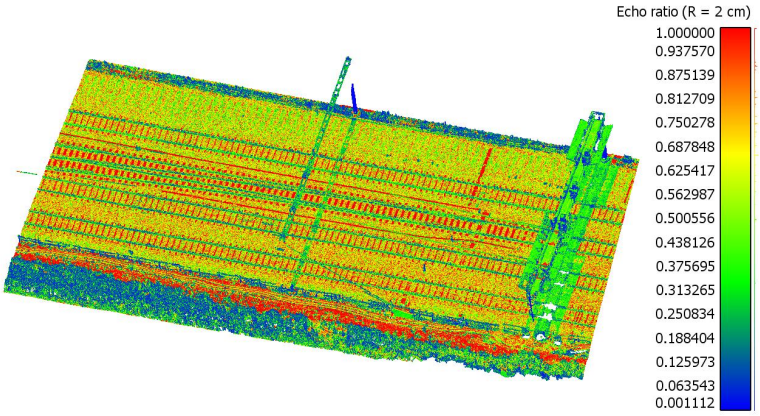



**Figure A.12:** Echo ratio ($R = 2$ cm).

## A.2 USING A NEIGHBORHOOD RADIUS OF 20 CM

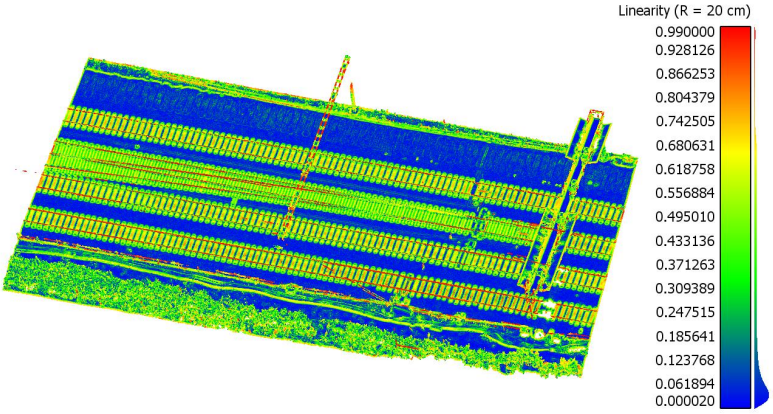



**Figure A.13:** Linearity ($R = 20$ cm).

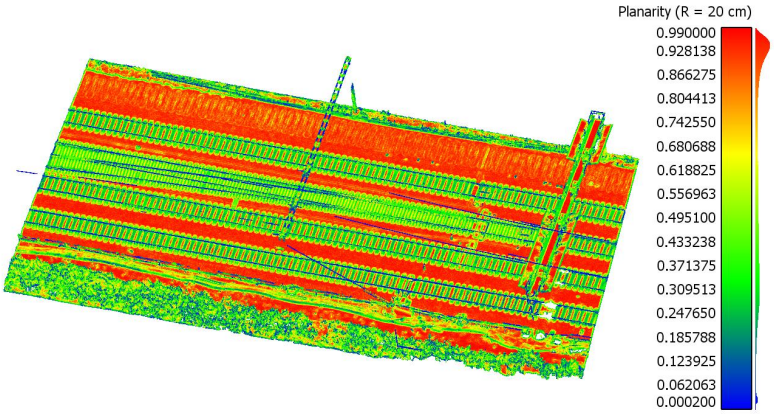



**Figure A.14:** Planarity ($R = 20$ cm).

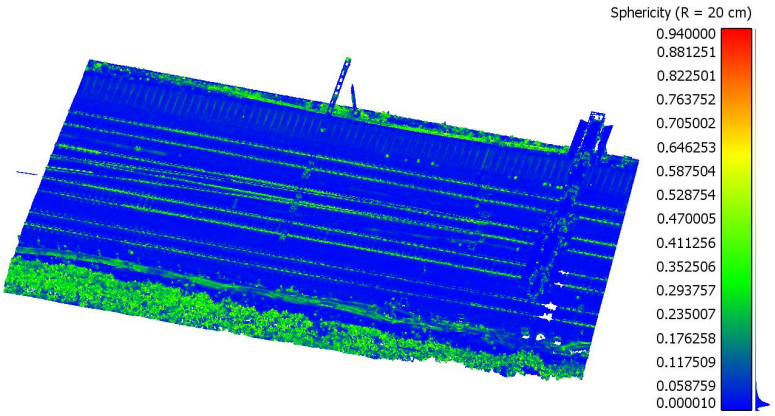



**Figure A.15:** Sphericity ($R = 20$ cm).

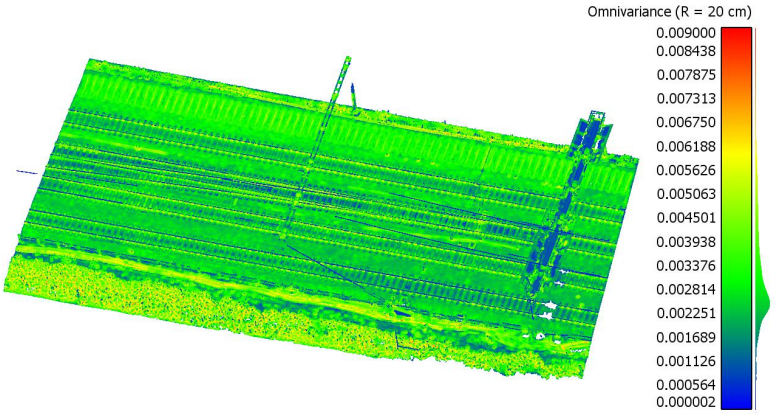



**Figure A.16:** Omnivariance ($R = 20$ cm).

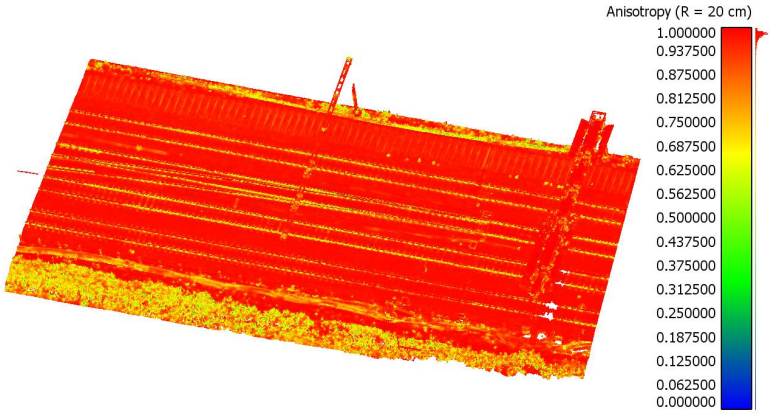



**Figure A.17:** Anisotropy ($R = 20$ cm).

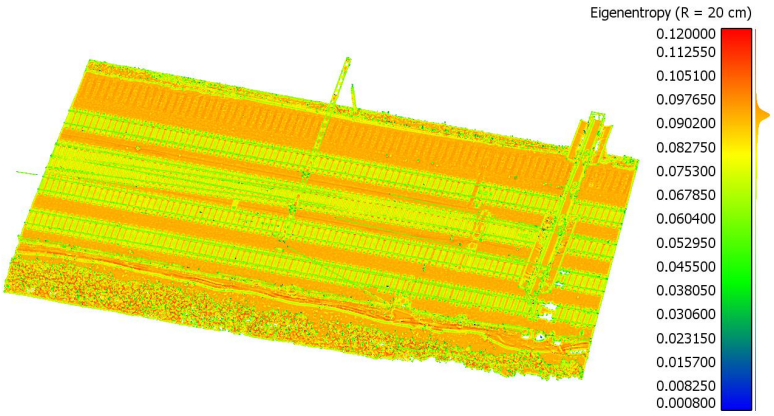



**Figure A.18:** Eigenentropy ($R = 20$ cm).

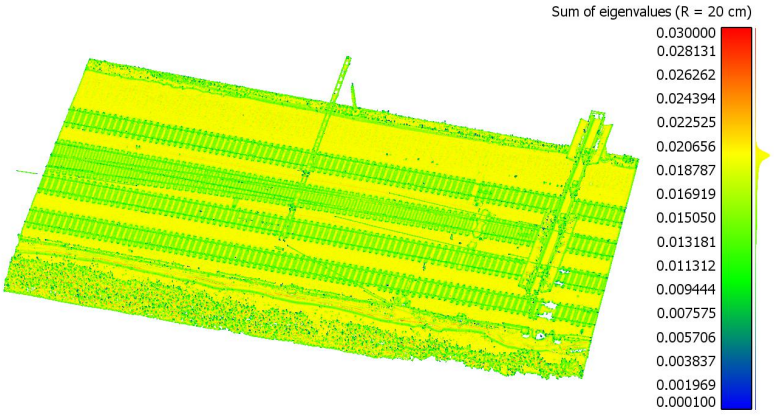



**Figure A.19:** Sum of eigenvalues ($R = 20$ cm).

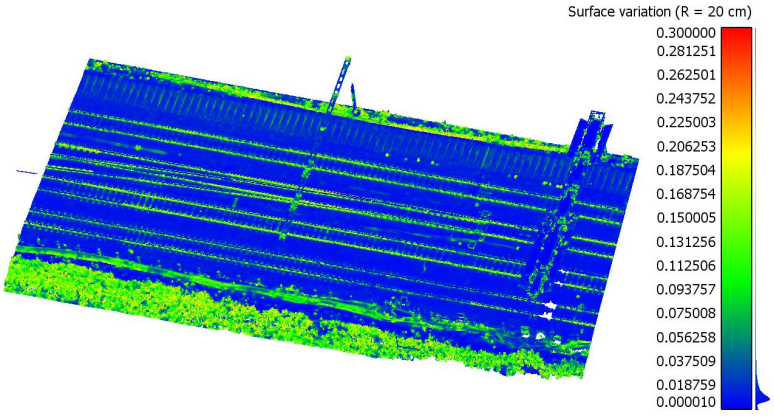



**Figure A.20:** Surface variation ($R = 20$ cm).

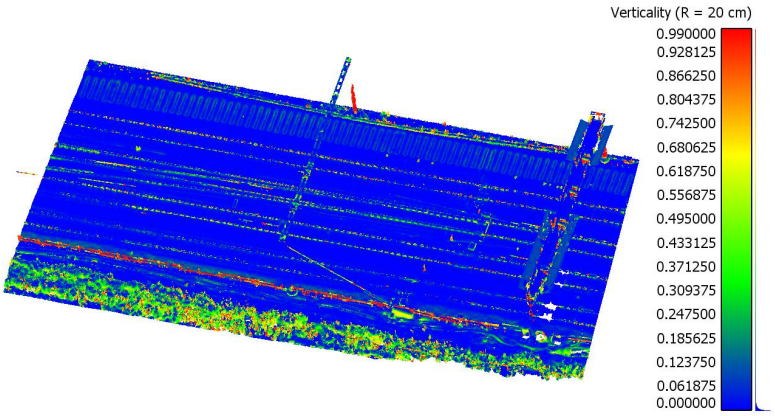



**Figure A.21:** Verticality ($R = 20$ cm).

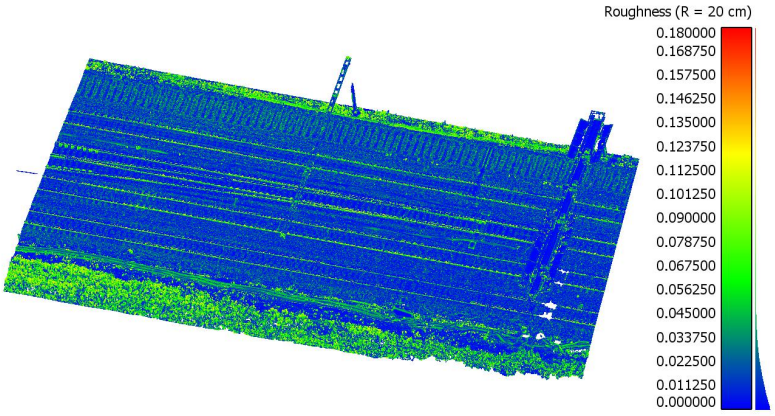



**Figure A.22:** Roughness ($R = 20$ cm).

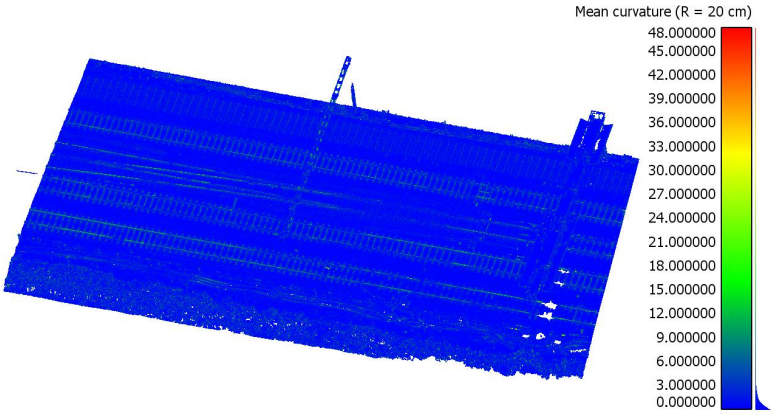



**Figure A.23:** Mean curvature ($R = 20$ cm).

# B ADDITIONAL RESULTS

Appendix B contains the confusion matrices in Section B.1 and feature importances in Section B.2.

## B.1 CONFUSION MATRICES

In each confusion matrix, the columns represent the actual classes and the rows the predicted classes. In the main matrix, each cell shows the number and percentage of points of each actual class that correspond to each predicted class, where green and red cells represent the correctly and wrongly classified points, respectively. The right column shows the number of points for each predicted class in white, the precision percentages in green, and the remaining percentages in red, the bottom row shows the number of points for each actual class in white, the recall percentages in green, and the remaining percentages in red, and the bottom right cell shows the total number of points in white, the OA percentage in green and the remaining percentage in red.

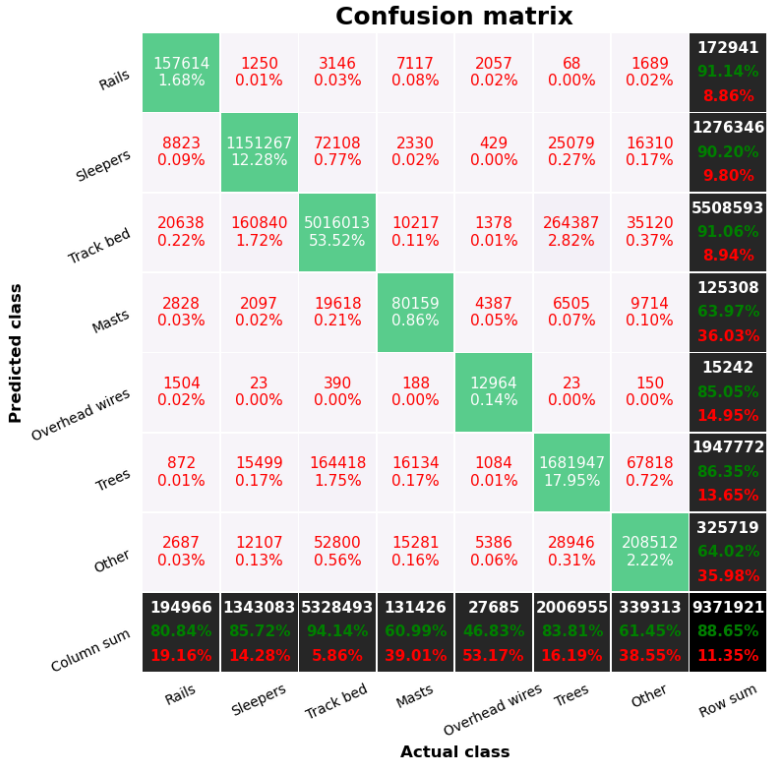



**Figure B.1:** {Random Forest - Scenario 1} Confusion matrix.

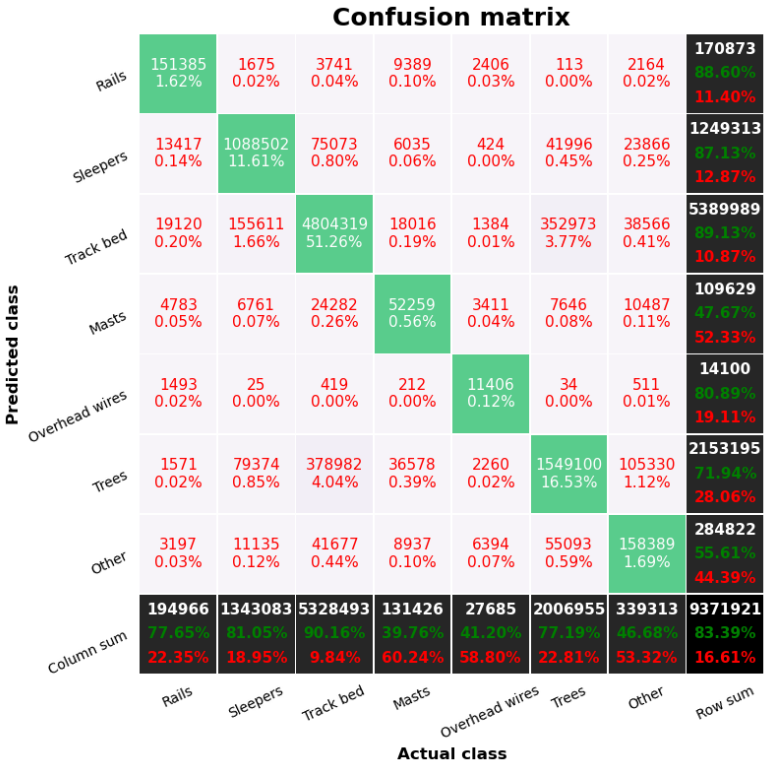



**Figure B.2:** {Random Forest - Scenario 2} Confusion matrix.

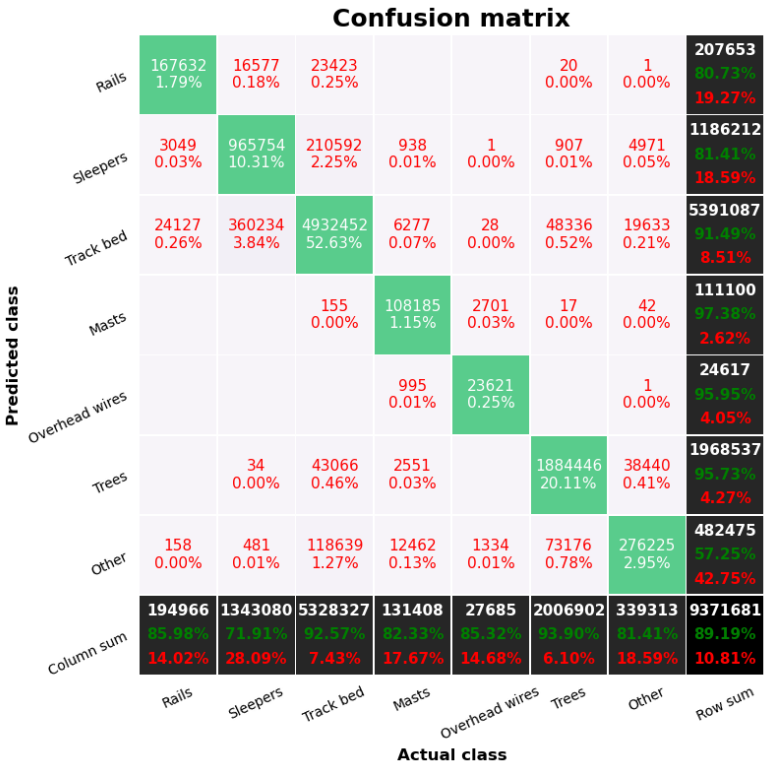



**Figure B.3:** {DGCNN - Scenario 1} Confusion matrix.

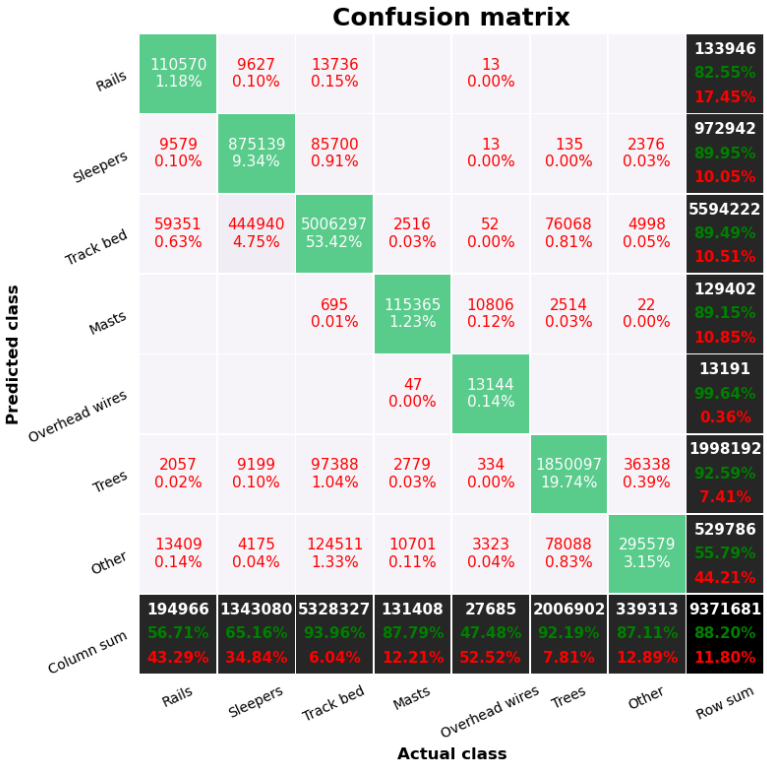



**Figure B.4:** {DGCNN - Scenario 2} Confusion matrix.

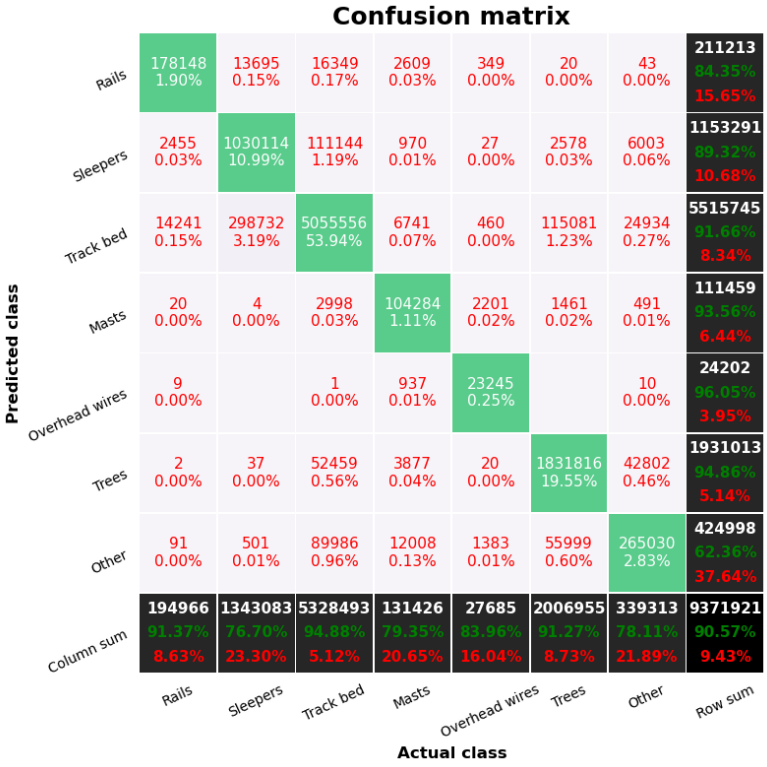



**Figure B.5:** {Combination of Random Forest and DGCNN - Scenario 1} Confusion matrix.

## B.2 FEATURE IMPORTANCES

Feature importances show the importance of RGB in green, geometric features with 2 cm neighborhood radius in blue, and geometric features with 20 cm neighborhood radius in red.

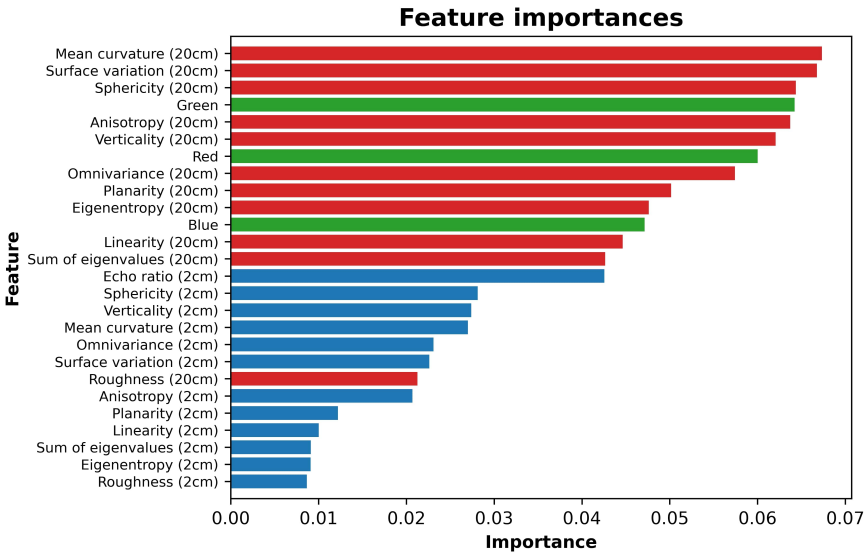



**Figure B.6:** {Random Forest - Scenario 1} Feature importances.

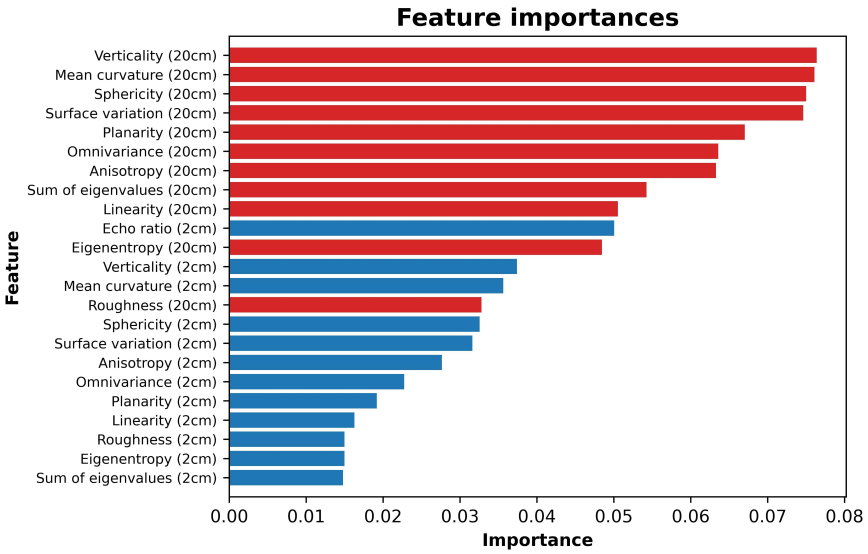



**Figure B.7:** {Random Forest - Scenario 2} Feature importances.

## COLOPHON

This document was typeset using LaTeX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.


TUDelft
CGI