

Espen Thorsrud Bragerhaug

Deep Tree Detector

A New Method for Detecting
Individual Trees from ALS Data Using
Projected Features and Deep Learning

Master's thesis in Master of Science in Engineering and ICT
Supervisor: Hongchao Fan

June 2021

Espen Thorsrud Bragerhaug

Deep Tree Detector

A New Method for Detecting
Individual Trees from ALS Data Using
Projected Features and Deep Learning

Master's thesis in Master of Science in Engineering and ICT
Supervisor: Hongchao Fan
June 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Civil and Environmental Engineering

Abstract

Having an up-to-date tree inventory in a forest or a city is valuable for forest monitoring and ecological research. Manually detecting individual trees for these inventories is time-consuming and costly work. In this study methods for automatically detecting trees from ALS data are researched. A new method for detecting individual trees is created by combining deep learning models from the realm of visual intelligence with the rich point clouds gathered through remote sensing. The proposed method is based on projecting point clouds onto a rasterized XY-plane, creating images from the point clouds. The spatial information is preserved by projecting valuable features from the point cloud onto the image. The values are then stored on the image-channels. A synthetic dataset is created using extensive data augmentation, to account for the lack of labelled data available. A Mask-RCNN model is trained on the synthetic images, and is tested on real data from a scanned area in Trondheim. The trained model reaches a precision and recall of 82% and 82% respectively, on the test area. The model was also tested on a synthetic test set, containing sampled trees not seen during training, arbitrarily positioned on a grid. The model reaches a precision and recall of 97,2% and 96,1% respectively, on the synthetic test set. The results show that this method is more than able to compete with traditional non-deep-learning methods for detecting individual trees.

Sammendrag

Å ha oppdaterte databaser med individuelle trær er nyttig både i skogforvaltning og økologisk forskning. Manuell deteksjon og digital innsamling av enkelttrær er både tidkrevende og dyrt. I denne studien undersøkes metoder for å automatisk detektere trær gjennom luftbåren laserskanning. I masteroppgaven blir også en ny metode for å oppdage trær fra disse fjernmålingene foreslått. Metoden kombinerer metoder fra visuell intelligens, som baserer seg på todimensjonell data, med detaljerte tredimensjonale datasett, fanget med luftbåren lidar. Metoden går ut på å projisere punktskyene på et raster i XY-planet for å danne todimensjonale bilder. Den romlige informasjonen blir bevart gjennom å projisere viktige egenskaper fra punktskyen, og lagre disse verdiene i båndene i bildet. Båndene i bildet som vanligvis blir brukt for å holde rødfarge, grønnfarge og blåfarge blir erstattet med henholdsvis punktetthet, høydedifferanse og høydegradient. For å løse kravet om store mengder annotert data i dyp læring, har et syntetisk datasett blitt skapt gjennom omfattende dataaugmentering. En Mask R-CNN modell har blitt trent på det syntetiske datasettet og blitt testet på reell data fra et skannet område i Trondheim. Modellen ble også testet på et syntetisk testset som ble lagd på samme måtet som treningssettet, men med andre trær. Den ferdige modellen når en presisjonsverdi på 82% og en tilbakekallingsverdi på 82% på det reelle testsettet. På det syntetiske testsettet når modellen en presisjon og tilbakekalling på henholdsvis 97,2% og 96,1%. Resultatene viser at den foreslåtte metoden er i stand til å oppdage flere trær, med høyere presisjon enn tradisjonelle metoder, og legger seg i sjiktet blant andre metoder basert på dyp-læring.

Preface

The study presented represents my master thesis, completing my civil engineering degree at the study program *Engineering and ICT - Geomatics* at Norwegian University of Science and Technology (NTNU) in Trondheim. The subject of the thesis is formed by my interest in spatial data, and the fascination I have for computer vision and deep learning. This project has allowed me to work with cutting-edge technology that I have found very inspiring. All work has been led expertly by my supervisor Hongchao Fan.

The project is an interdisciplinary study, combining methods from Computer Vision, Machine Learning and Remote Sensing. The scope of the project is broad, but gets very specific at times, meaning that not all concepts utilized in the project are explained in depth. It is highly advised that the reader has some basic understanding of machine learning, computer vision and lidar data. Some external sources are suggested throughout the text, should the reader feel the need to read more about the topic.

I want to thank my supervisor Hongchao Fan for presenting the idea of combining image-based deep learning with lidar data. I also want to thank Facebook AI Research (FAIR) for developing, maintaining and contributing to the open-source community a state-of-the-art deep learning framework that is both powerful and user-friendly.

I want to thank my classmates for all the insightful discussions and my family and friends for motivating me throughout the master thesis. Finally, I would like to thank you, Astrid, for your continuous love and support, and for showing me through example that things are never as tough as they may seem.

Contents

Abstract	i
Sammendrag	ii
Preface	iii
Contents	iv
Figures	vi
Tables	vii
1 Introduction	1
1.1 Motivation	1
1.2 State-of-the-art Methods	2
1.3 Objective and Proposed Method	4
1.4 Outline	5
2 Background Knowledge	7
2.1 Convolutional Neural Networks	7
2.1.1 Object detection	9
2.1.2 Segmentation	10
2.2 CNN on Point Clouds	11
2.3 Overfitting	11
2.4 Evaluation Metrics	12
2.4.1 Accuracy, Precision and Recall	12
2.4.2 Intersection over union (IoU)	14
2.4.3 Mean average precision (mAP)	14
3 Deep Tree Detector	16
3.1 Overview	17
3.2 Synthetic training examples	17
3.2.1 Get a random sample	18

3.2.2	Normalize each tree	19
3.2.3	Spread trees onto a spatial grid	19
3.2.4	Map Features to 2D	19
3.2.5	Project annotations to 2D	19
3.3	Transforming Point Clouds to Images	20
3.3.1	Selecting image resolution	21
3.4	Top Down Slicing	23
3.5	Training a tree detection model.	23
4	Experimental Results	25
4.1	Available data	26
4.2	Labelling the data	26
4.3	Performance	28
4.3.1	Performance of various resolutions	31
4.3.2	Performance with various amounts of synthetic images	32
4.3.3	Performance with various amounts of manually labelled trees	34
5	Conclusion and Future Work	38
	Bibliography	41
A	Source Code For Generating Synthetic Images	45

Figures

1.1	Canopy Height Model Illustration	3
1.2	Projecting Point Cloud to Image Illustration	5
2.1	CNN Illustration	8
2.2	Object Detection Output	9
2.3	Semantic Segmentation vs Instance Segmentation	10
2.4	Overfitting Illustration	12
2.5	2x2 Confusion Matrix	13
3.1	Synthetic Sample Pipeline	16
3.2	Manual Extraction of Individual Trees	18
3.3	Dense Groupings of Trees	18
3.4	Projection Resolutions Differences	22
3.5	Synthetic Examples	24
4.1	Scanned Area Used for Experiments	25
4.2	Manual Extraction of Individual Trees	26
4.3	Scanned Test Area	28
4.4	Performance on Various Resolutions	31
4.5	Amount of Images Performance	33
4.6	Bounding Box Accuracy on Various Amounts of Trees	34
4.7	Qualitative Results Test Area	36
4.7	Qualitative Results 20% Holdout	37

Tables

4.1	Manually labelled trees properties	27
4.2	Training parameters	29
4.3	Final performance on the various datasets.	30

Chapter 1

Introduction

In this chapter the general problem studied, and the motivations behind it will be introduced. The problem will be explained, and the goals for the project will be presented. State-of-the-art methods will be elaborated as a reference for the rest of the thesis.

1.1 Motivation

Maintaining up-to-date tree inventories in both cities and forests is important for several reasons. Tree inventories are used to track issues with pest and urban attacks [1]. The tree inventories make it possible to look at the ecosystem services that trees provide in a city [2]. The convenience of having these up to date inventories has increased during recent years as they have become more useful. The increased interest has led to a surge in the development of new methods and techniques for acquiring them. The main objective is to effectively and accurately collect the tree data automatically. Tree inventories are a valuable source of data for researchers studying the environmental, social and economic services provided by trees. Fields of studies includes storm-damaged trees and identification of species and dimensions most affected, risk assessments, identification of species composition and diversity, modelling of local climate, impacts on air pollution, urban heat island effects and storm water runoff, assessment of the economic benefit of urban trees, and monetary evaluation of individual trees [3]. These rules also generally applies to rural forests, and maintaining a good digital tree inventory over a forest can give much insight in how the ecological system behaves, and how to best preserve and utilize the resources.

The tree inventories typically include properties like species, height, diameter at breast height (DBH), soil quality, age, and defects. Today these inventories are most often manually collected, tree by tree. This is done by human inspectors, often supported by equipment like terrestrial laser scanners and sonars. The work is costly and time-consuming. Finding ways to automate these tasks is widely researched, as it could save many hours of manual work, as well as widely acquiring valuable data that can be used for research and development. A natural step in automatic detection of trees is to look in the realm of deep learning.

Deep learning and computer vision have been in an emerging state for the last decade. The groundbreaking depth of the models and the ability to utilize graphical processing units (GPUs) has accelerated the performance of deep learning models [4]. Utilizing these new methods on three-dimensional geographical data is still in its infancy, and is therefore an interesting topic to look further into.

1.2 State-of-the-art Methods

Detecting individual trees from ALS point clouds has been a heavily researched topic. Several methods have been developed to solve the task. Generally they can be divided into three types: Canopy Height Model (CHM)-based methods, clustering-based methods and deep learning based methods.

CHM-based models use the CHM as an input. The CHM is generated using a digital surface model (DSM) and a digital terrain model (DTM). The CHM is the DTM subtracted from the DSM. The CHM is typically smoothed using a height-based filter. The filter reduces noise and simplifies the model. The assumption of the method is that local maximum points on the smoothed CHM are individual treetops. The local maximums are used as seed points in a region growing algorithm [5]. The resulting regions are individual trees. CHM-based methods typically work very good at trees that stands alone, but works poorly on dense groups of trees. A comparison of implementations of CHM-methods can be found in this article from Finland [6]. The top-scoring implementations achieve close to 100% accuracy on standalone trees, 65% accuracy on a group of similar trees, and 15% accuracy on trees next to bigger trees.

Clustering models use the point cloud raw, meaning that they are not voxelized as a preprocessing step. Typically the first step is to divide the point cloud into several

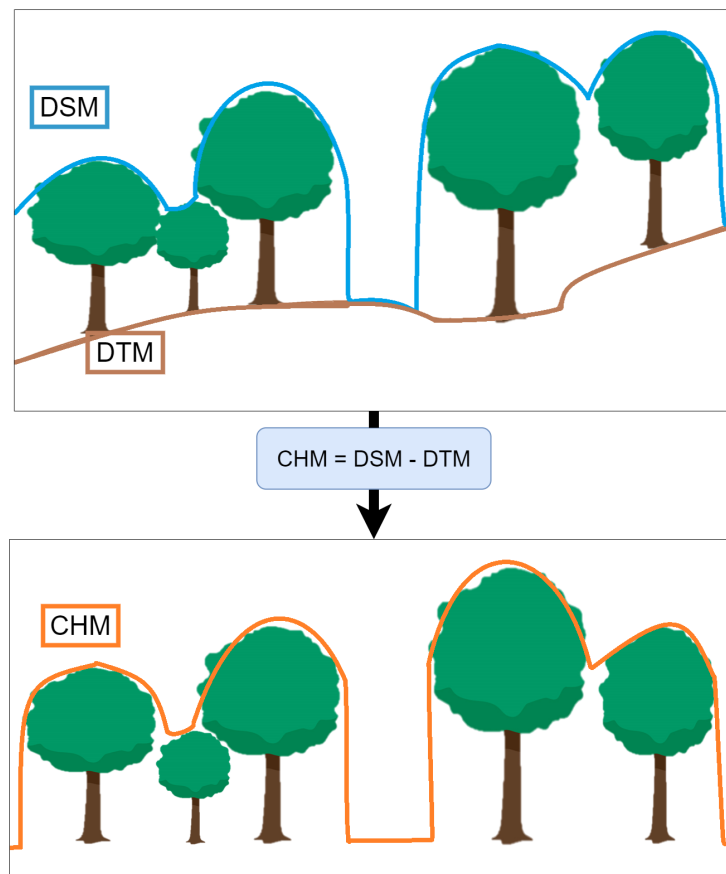


Figure 1.1: Canopy Height Model.

height-slices. These height slices are made to simplify the data by discretizing the z-axis. Each slice can be treated as a 2D plane. The point clouds can be partitioned into any number of slices, but are typically partitioned into equally big slices. The points in each slice are then clustered by a clustering algorithm. The clusters can then be used to model a part of a tree, for example using a RANSAC algorithm. Typically an ellipsoid is approximated at each slice. These ellipsoids are then merged in height, given that they are close enough. The modelled shape that does not match any of the subsequent layers are deleted, while the shapes that does match another are merged with that height slice. The end result is that all height slices are merged together, hopefully finding continuous trees along the way. The method is dependent on dense point clouds, with information along the whole tree trunk. This is an issue with ALS data, because typically ALS-data contain less information below the tree-crown. The

method is however quite robust when used in dense forests. An implementation of the method is presented in [7] and they report an accuracy of 55% in a dense forest.

Deep learning based models used to detect individual trees are still in its infancy. Due to the increase in available 3D information deep learning based models are expected to be the most important domain for developing new breakthroughs in ecological observations and modelling in the future [8]. The methods are divided into voxel-based and point-based models. For detecting individual trees typically voxel-based methods are used. A deep learning method was implemented by Sydney University [9]. The first step in the method is to voxelize the points. The points are then encoded using birds-eye view projection (BEV). The BEV-projection results in a three channel image with preserved 3D information. The BEV-projection focused on encoding the vertical density, maximum height, and average return. Since the method is voxel-based, the vertical density is found by summing all occupied voxels in the z-axis for each XY-grid location, and dividing this sum on the total number of voxels in the z-axis. The max height is found as the highest occupied cell in a XY-location. The average return is found by calculating the mean of all return values in the z-axis for each XY-grid location. The resulting projected image is then fed to a pretrained Faster R-CNN model [10], to predict individual trees. This step outputs a bounding box around all regions the model thinks contains an individual tree. The deep learning based models has achieved very good results. The method achieved an accuracy of 96%, a precision of 100%, and a recall of 92.8%. The test area consisted of a fairly dense forest area. The data was collected using a Riegl VUX-1 LiDAR equipped on a helicopter that flew approximately 60-90metres from the ground. This resulted in a very dense point cloud, with approximately 300-700 points per m^2 . This is about 50 times as dense as the data available for the experiments in this master thesis. The results shows that the deep learning models are able to achieve state-of-the-art results.

1.3 Objective and Proposed Method

The objective of this master thesis is to look into how deep learning object detection models can be utilized on point clouds to detect individual trees. Previous work has shown that both using deep learning directly on the point cloud, as well as projecting the point cloud down to an image has been viable methods for detecting the trees. The

objective will be to generate a new method for detecting individual trees by combining computer vision and point clouds in a new way. The proposed method will consist of several steps. First the point cloud will be rasterized. Each point in the cloud will be projected down to the XY-plane, and fitted into the appropriate cell in the raster. For each cell in the raster, the points will be aggregated to generate key values from their properties related to the z-value. In turn each cell will compute its **density**, **height difference** and **height gradient**. These values will be stored as channels in the raster, resulting in a traditional 3-channel image. These images will then be fed into a trained object detection model, which will predict individual trees in the image in form of bounding boxes and segmentation masks. The bounding boxes can then be used to find out which points in the point cloud related to which tree.

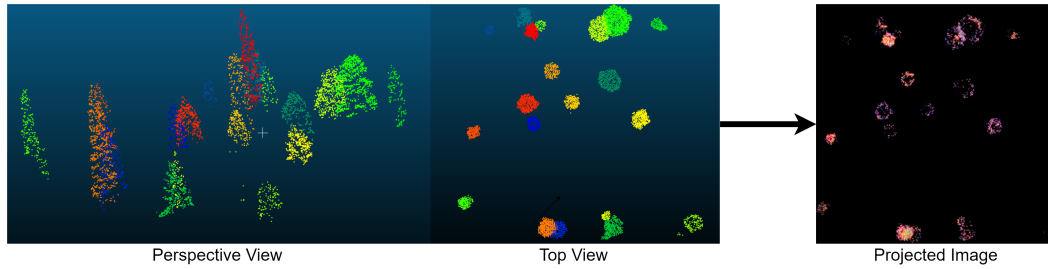


Figure 1.2: Projecting the point cloud to a 3-channel image

If the results of the object detection model are able to come near the results achieved on known datasets like COCO [11] and CityScapes [12], this method based on deep learning will be able to compete with traditional state-of-the-art methods used directly on the point cloud data.

1.4 Outline

In the following chapters a method for automatically detecting individual trees from point clouds acquired through airborne laser scans will be presented.

In chapter 2 the background theory for the method will be explained. Concepts of deep learning, specifically convolutional neural networks (CNNs) will be elaborated. Object detection and segmentation, which is probably the most important tasks for this project, will be presented. Common difficulties and shortcomings when training a CNN on 3D-data will be elaborated. How to surpass these difficulties will also be

explained. Finally, metrics used for evaluating a CNN model will be explained. Specifically precision, recall, mean average precision and intersection over union will be explained, and how they are calculated will be explained.

In chapter 3 the proposed method "Deep Tree Detector" will be explained in detail. Every step will be thoroughly explained, both in method and in implementation. The approach to preserve three-dimensional features when projecting a point cloud onto a 2D-plane will be explained, and an algorithm for doing this will be presented. A pipeline for generating training example will be proposed. The training examples consists of sampled point cloud trees. In addition it will be talked about how an object detection model is trained and what open-source resources exists.

In chapter 4 the experiments and results will be presented. The performance of the method will be presented and discussed. During the project several experiments has been carried out to understand what factors influences the final method. Specifically experiments highlighting the effects of generating several images from the same data source is looked into. The resolution of the images generated is experimented with. Finally it was investigated how the amount of manually labelled trees that were affecting the performance of the model.

In chapter 5 the master thesis will be concluded by summarizing the method and the thesis. In addition, future work and outlook will be proposed, so that new research can build upon and enhance the presented method in this thesis.

Chapter 2

Background Knowledge

This chapter will present the theoretical background that will form the basis of the following experiments and analyses. The reader will get a basic understanding on supervised learning and CNNs. How CNNs are utilized on 3D point clouds will be presented. The reader will also get an introduction on the computer vision tasks *object detection* and *instance segmentation*. Finally the reader will get an introduction on metrics used for evaluating a deep learning model. The reader is expected to have a basic conceptual understanding of machine learning.

2.1 Convolutional Neural Networks

CNNs are one of many deep learning models that has received a lot of attention in the last decade (2011-). The breakthrough for CNNs happened when AlexNet [4] achieved groundbreaking performance in 2011 on the ImageNet challenge [13]. Utilization of GPUs made it possible to process bigger networks. AlexNet applied more hidden layers to the neural net, resulting in a deeper model that was able to classify images more accurately. The workflow of a CNN is illustrated in the figure 2.1.

CNNs are based on shared weights and convolution kernels that slide along the input features. The input features are typically an image. As in all neural nets, the output is forwarded through the net, increasing feature dimension when moving deeper down the network. This enables interpreting of the spatial properties of the input at many levels. CNNs rely on all inputs having the same size, and are ordered in the same way. That makes it very suitable for images. All images in the dataset must have the

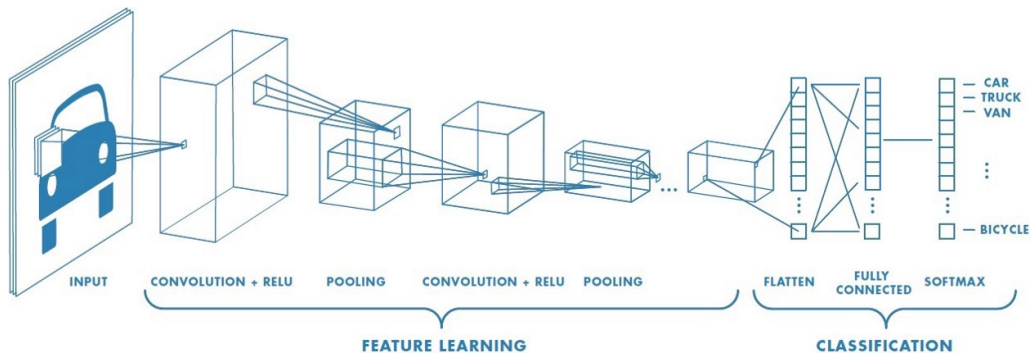


Figure 2.1: An illustration of how CNN models work. Image from Towards Data-science[14]

same height and width for the CNN to work. Traditionally CNNs were only used for image classification but has later been extended to solve other computer vision tasks like object detection, image segmentation and instance segmentation.

CNNs are different from traditional neural networks in that they have more specific connections between the layers. Typically in a neural net each neuron in a layer is connected to every neuron in the next layer, a fully connected layer. In CNN these fully connected layers are changed by having shared weights between several neurons. The effect will be the same as dragging a sliding window over the image, combining the regions into more complex features [15]. This will drastically decrease the number of parameters needed, and will also give more focus on local features in the image. When moving deeper into a CNN the features will become more and more combined, and the features will become of higher and higher levels. This makes each layer important in its own way, and typically the layers can be combined in a lot of ways to utilize the generated information. Neural networks like ResNet [16] has connections from not only one layer to the next, but some connections goes directly several layers down. These kinds of complex structures are widely researched, and history has shown that there is a lot of possibilities regarding the architecture of a CNN.

To further understand how a CNN works this paper made by Saad Albawi and Tareq Abed Mohammed [15] is highly recommended.

2.1.1 Object detection

When doing machine learning on images, the objective is to extract some kind of information from the image. Originally, CNN models were used to classify an image. Later these architectures have been extended to fulfill several other classification and regression tasks, one of them being detecting objects inside the image. In contrast to classifying an image, the object detection has two problems. One regression problem, of detecting the position of an object in an image, and one classification problem, predicting the class of that object. The task is referred to as object detection, and is extensively researched.

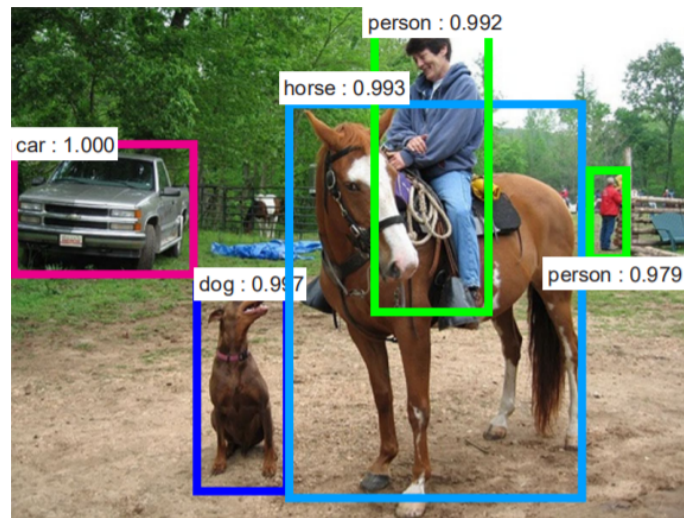


Figure 2.2: An object detection model detects the objects in the image. The outputs are typically a bounding box of the object, a class of the object and a prediction score, telling how confident the model is of its prediction. This image is generated by a Faster Region Based CNN (Faster R-CNN) model [10].

Several different architectures has been created for detecting objects, and the most successful ones includes YOLO[17] Single-Shot-Detector[18] and Faster R-CNN[10]. The architectures are divided into the categories, single-stage and two-stage detectors. The architectures consists of several CNN-modules. The Faster-RCNN architecture is a two-stage architecture. The first stage is a region proposal network. This stage generates regions of interest in the image using a CNN module. The output from this stage is several regions of interest. The second stage consists of classifying these proposed regions by running them through a seperate CNN module to extract high-dimensional features from them. This is done by a CNN module called a feature extractor. Finally

the features are in parallel used to find the accurate position of the region, and the class of the object, The YOLO architecture however, treats the whole problem as one regression task. YOLO is a single-stage object detection model. By doing this it can do both stages in one module, reducing the complexity of the model. The benefits of this is that it becomes very fast, reportedly achieving up to real-time processing speed.

2.1.2 Segmentation

The natural step after object detection is to not only create a bounding box around each object, but accurately outline the object. This is done by classifying each pixel, and reasoning which class that pixel belongs to. **Semantic segmentation** is a task where each pixel is mapped to a class. This can be done by using a fully convolutional network (FCN), such as U-Net [19]. A FCN performs convolution operations all the way, no fully connected layers at the end like traditional CNNs. This is because the output is an image where each pixel has been assigned to a class.

Semantic segmentation does however not entirely fit the task of detecting individual trees, because when each pixel has been assigned to either a tree or not, there is still no way to determine what pixels belong to each specific tree. This is where **instance segmentation** becomes the appropriate task. Instance segmentation segments each instance present in the image, which makes it possible to detect individual trees and accurately detect the outlines of it. This has been achieved using single-stage architectures such as U-Net and E-Net, by adding focal loss [20, 21]. It has also been implemented in two-stage detector such as the Faster R-CNN architectures. The leading model in instance segmentation has been the Mask RCNN [22]. It utilizes many complex architectural tricks such as feature pyramid networks and region of interest aligning (ROI-aligning) to accurately and precisely pinpointing each segment included in an instance of a class.

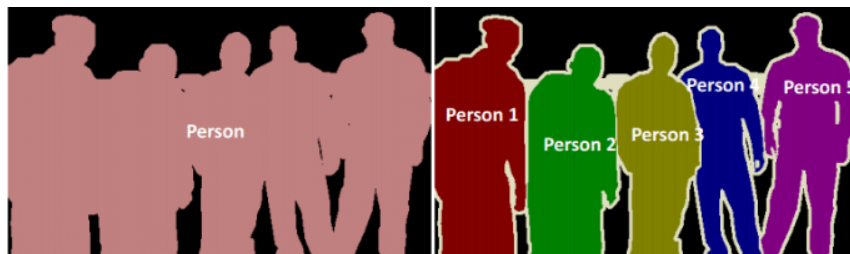


Figure 2.3: The difference between semantic- and instance segmentation [23]

2.2 CNN on Point Clouds

Using deep learning on point clouds is challenging for number of reasons. Deep learning frameworks for computer vision typically relies on the fixed data structure of images, which the point clouds does not have. The main issues with point clouds are: [24]

- **Irregularity:** Data points are not sampled evenly over an entire region.
- **Unstructured:** Each point is scanned independently with no fixed distance to its neighbours. The point cloud is not placed on a regular grid.
- **Unorderedness:** Data points do not hold any topological information in terms of how they are ordered in the data structure.

Several methods have been developed to enable deep learning for point clouds. Methods can be put into two main categories. The first category consists of methods that uses the point cloud raw, like PointNet [25], DGCNN [26] and SpiderCNN [27]. The second category structures the data into grids, like VoxNet [28], ShapePFCN [29] and SplatNet [30]. The methods that structurizes the data will be able to use CNNs, since the data will have a fixed, given number of voxels, pixels or cells. The raw data methods will not be able to use CNNs, since the conditions will not be met.

2.3 Overfitting

CNNs is a data-driven approach. The method relies on thousands of images to be able to understand the underlying structure that defines a class of objects. When training a CNN, the model is iterating through a dataset several times, each time predicting what classes the image contains. The CNN is then presented with the *ground truth*, the real classes the image contains. The error is computed through a loss function. The error is then used to update the network. So for each iteration the network is tuned to learn the classes of the training images. The problem is this; Is the CNN learning the underlying general structure of that class, or is it just memorizing the input data? This problem is called **overfitting**.

Overfitting is when the model fails to learn a generalized structure that defines a class, but instead memorizes the distinct entities of that class. When training a CNN model this can be continuously monitored by using a test set. The test set is only used

to evaluate the model during training, and the network should *not* update its weights when seeing this data. If only the training dataset is used to evaluate the model, there is no way to tell whether the model is getting better, or if its just memorizing the training examples. By using a separate test set the generalization can be tested.

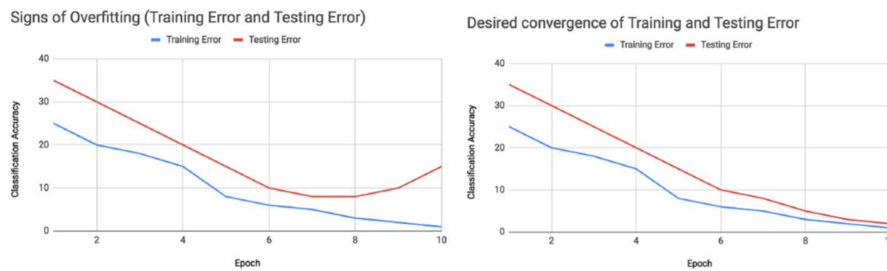


Figure 2.4: Overfitting can be visualized by plotting the training dataset evaluation against a test dataset evaluation. See that on the left figure the training error is gradually decreasing, but this does not generalize to the testing error. On the right side both converge to zero, and thus the model is generalizing well. The figure is from "A survey on Image Data Augmentation for Deep Learning" [31].

There are several solutions to overfitting, which are designed to punish the model for fitting the data. Data augmentation is a technique that tackles the root of the problem of overfitting, *the dataset*. Data augmentation assumes that the lack of generalization is the lack of crucial examples in the training dataset. Data augmentation handles this by changing the existing examples in the dataset. Examples of augmentations are scaling, rotation and including random noise.

2.4 Evaluation Metrics

Metrics are measures of quantitative assessment which can be used for evaluating the performance of a model. For a predictive model the most important performance metrics are the accuracy, precision and recall.

2.4.1 Accuracy, Precision and Recall

Accuracy, precision and recall are measures that tells how well the predictive qualities of a model are. They are calculated with the 4 possible prediction values: True positive

(TP), true negative (TN), false positive (FP) and false negative (FN). These values makes up the confusion matrix.

		Predicted Value	
Actual Value		TP	FN
		FP	TN

Figure 2.5: 2x2 Confusion Matrix

The *accuracy* of a model is the most intuitive metric. It is simply the ratio of correct predictions to the total amount of predictions. Accuracy makes most sense on datasets where the amount of false positives and false negatives are equal. In the experiments presented here there is simply one class to detect; tree. Therefore, the predictive values will only be true and false positives. In other words, accuracy is not a good metric for this use.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.1)$$

The *Precision* is the models ability to predict the correct value, and is defined by how many of the predicted values that are actually correct. The *recall* on the other hand is the models ability to detect all ground truths. In terms of object detection recall is the ability to find all objects present in an image, precision is the ability to *only* predict only the correct objects. Precision and recall can be formulated mathematically with the following equations:

$$Recall = \frac{TP}{TP + FN} \quad (2.2)$$

$$Precision = \frac{TP}{TP + FP} \quad (2.3)$$

Typically, the goal is to maximize both precision and recall. Having high recall is not very good if the precision is low and vice versa. Therefore a third metric, called F1-score is typically used. F1-score combines the precision and recall into one metric,

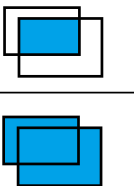
and is good to use when the goal of a model is to have balance between both precision and recall. F1-score is calculated by the following formula:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (2.4)$$

Object detection is about predicting bounding boxes around instances in the image. An important implementation detail when testing the model is how accurate the bounding box has to be. The ground truth bounding box is typically manually generated, and will rarely be perfectly fitted to the object. Therefore, a threshold has to be set on how accurately the predicted bounding box must match the ground truth bounding box to be considered a correct prediction.

2.4.2 Intersection over union (IoU)

Intersection over union is an evaluation metric that measures how accurately a predicted bounding box matches the ground truth bounding box. Mathematically the IoU is defined with the following formula.

$$IoU = \frac{AreaOfOverlap}{AreaOfUnion}$$


The IoU is often used as a threshold for defining if a prediction is valid or not. Depending on how accurate the model is required to be, the IoU-threshold can be adjusted. Typically an IoU-value > 0.5 is considered a valid prediction. The IoU is not the most interesting metric in itself, but is used to compute more complex metrics. One of those metrics is the mean average precision.

2.4.3 Mean average precision (mAP)

Mean average precision is used for evaluation in almost all object detection benchmarks. The metric gives a good overall score of the performance of the model. In

general it is used on systems that detect multiple classes. The mAP is found by computing the average precision on each class in the prediction model, and taking the mean of all the average precisions. In this project there is only one object class, trees, and the mean in mAP goes away. The average precision (AP) is the area under the precision-recall curve, and can be specified mathematically with the formula below.

$$AP = \int_0^1 Precision(Recall) \quad (2.5)$$

The reason for calculating precision as a function of recall is that the model outputs a confidence level from 0 to 100%, and to evaluate the robustness of the model all confidence levels have to be considered to know how well it performs. In practice, the confidence levels of the predictions are used to find the different levels of recall. The highest precision at each level of recall can then be identified. The creators of the COCO-dataset[11] states that currently a 101-point interpolation method is used for calculating the mAP, meaning that the maximum precision are found for recall-values in the range: {0, 0.1, ..., 0.99, 1.0}.

By using this method, average precision can be found at a certain IoU-threshold. In the COCO-challenges mAP and AP is used interchangeably, and the main performance metric is just called AP. In practice AP means the mean AP over the IoU-thresholds .5:.05:.95, meaning the mean AP of IoU thresholds in the range: {0.5, 0.55, ..., 0.9, 0.95}. Other popular metrics are the AP50 and AP75, respectively meaning the average precision at IoU-threshold .50 and .75. Mean average precision is a difficult metric to understand, because it holds so much information. However it is a very strong at determining the performance of a predictive model, and is widely used.

Chapter 3

Deep Tree Detector

In this chapter, a method for combining point clouds with CNN architectures is proposed. Firstly, a collection of individual trees are manually labelled at point level. These trees are augmented and used several times to create synthetic training examples. By projecting the point clouds to 2D space and rasterize the points into an image, the data becomes available for training a CNN. The features contained in the point clouds are preserved by exchanging the RGB-channels in the image with features from the point cloud. Namely these features are height, height gradient, and point density. This enables the use of complex state-of-the-art models in the image based CNN domain on what initially is a point cloud.

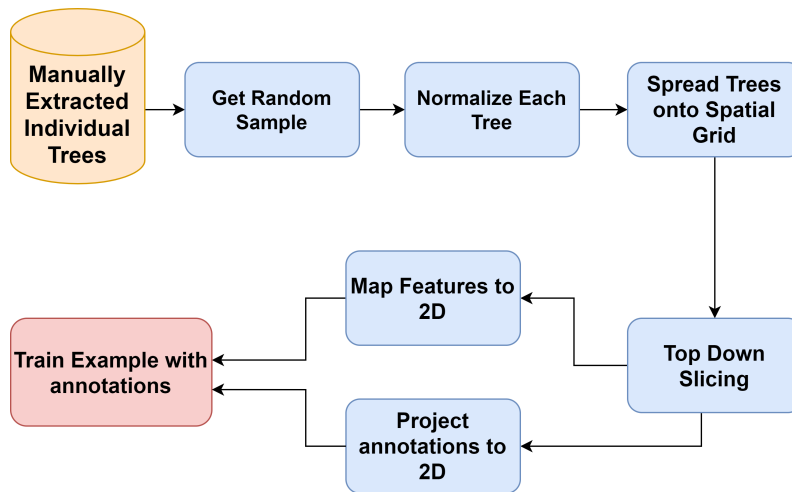


Figure 3.1: The pipeline for creating a synthetic sample.

3.1 Overview

The method for creating a deep tree detector consists of several steps.

1. **Create a collection of individual trees.** The trees are labelled at point level. The collection should be split into a train set and a test set.
2. **Create a synthetic dataset** by randomly sampling from the tree collection. The trees are randomly sampled to generate new examples. The process is defined by the following steps.
 - a. **Get a random sample** of n trees. The sample should vary in size, to make the model robust for detecting various amount of trees in each image.
 - b. **Normalize each tree** by dividing all points by the maximum point, leaving all points in the range $[0-1]$.
 - c. **Spread trees onto a spatial grid.** This should be done randomly to create unique examples. In the experiments it was also allowed to put trees on the same spot, creating real life scenarios of intertwining trees.
 - d. **Map Features to 2D** by running the grid through the Map2D algorithm (Map2D). Simultaneously **project annotations to 2D**. This is done by creating the polygon that surrounds all points for each tree, and export this polygon into a JavaScript Object Notation (JSON) File.
3. **Train an object detection model** using the synthetic train set. The object detection model can be any model that inputs images and outputs prediction boxes and/or instance segmentation masks.

3.2 Synthetic training examples

Labelling instances is essential for training a model. In this project, trees had to be manually extracted from the scanned area, and put into separate files. Extracting one individual tree approximately took 2-5 minutes per tree, depending on how densely the tree was scanned, and if there were any occlusions. Clustered trees typically took longer time to extract. Extracting thousands of individual trees would take a lot of time. To avoid using too much resources on manual extraction, data augmentation was used. By randomly sampling extracted individual trees it is possible to generate unlimited synthetic examples from just a few extracted trees. A synthetic dataset was

formed this way.

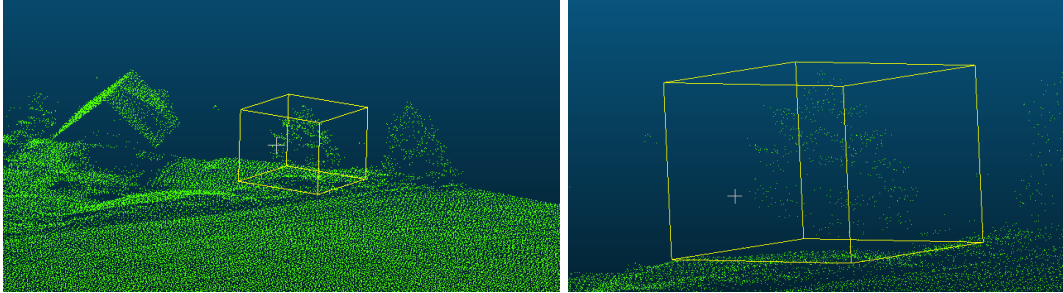


Figure 3.2: Extracting individual trees is time consuming work.

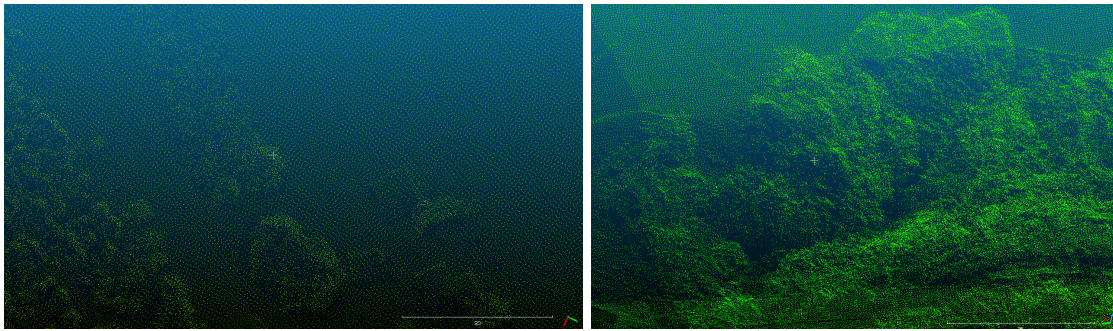


Figure 3.3: Extracting individual trees is not always easy in dense forests like in these images.

The pipeline for creating random samples from a dataset of manually extracted trees is shown in Figure 3.1.

3.2.1 Get a random sample

A random sample is taken from the collection of manually extracted individual trees. To create variation in each sample, the amount of trees in the sample is a random integer within the interval $[1, 24]$. A specific tree in the collection will have the same orientation each time it is loaded from the collection. To prevent this the points are rotated a random number of degrees. This will create more diverse samples and increase the usefulness of having several images containing the same trees.

3.2.2 Normalize each tree

Each tree in the random sample is normalized by dividing each z-value by the maximum z-value in the points. The x- and y-values are divided by the largest of the maximum x- or y-value. The reason for dividing both the x- and y-axis on the same value is to prevent the ratio from getting skewed. Skewing the ratio could be done as a data augmentation step, but to preserve the authenticity of the data this is not implemented at this point.

3.2.3 Spread trees onto a spatial grid

In this step each tree gets a place on a quadratic, spatial grid. The trees are scattered randomly on the grid, with real numbers. This will cause some dense, and some scattered groupings of trees. The size of the grid varies, to increase the variation between scattered and dense forests. The trees are placed truly random, without any check of how close together they get. In theory two trees can be placed exactly on top of each other. This adds to the dataset variation, and should be useful since this could also be the case in the real world, where two trees grow around each other. Rules can be added to create different examples. In example, the trees could be placed with integers, and that position could be removed for future trees, removing the chance of having two trees atop each other. Since the trees are normalized between 0 and 1, each tree would then be completely isolated on its position.

3.2.4 Map Features to 2D

In this step the spatial grid from the previous step is split into a given resolution. The points and the grid is then inputted into the Map2D algorithm (Map2D). The Map2D algorithm is specified later, but preserves the 3D information onto 3 channels. Resulting in an image on the XY-plane with various 3D qualities preserved in the RGB-channels.

3.2.5 Project annotations to 2D

In this step the convex hull of each tree on the grid is projected onto a raster of the same resolution as the resolution set in Map Features to 2D. The convex hull is then

used as the segmentation mask of that tree on the image. In addition the max and min XY point is used as bounding box coordinates, for the object detection task. The annotations are then stored on COCO format [11], and saved to a JSON file.

3.3 Transforming Point Clouds to Images

Transforming Point Clouds to an image is a big task. It takes away a dimension, removing very much spatial information. The challenge is to find a way to preserve the most crucial information in the transformation. Map2D is an algorithm that uses rasterization to project the point cloud onto an image in the XY-plane. To preserve the properties implied with z axis, the RGB-channels in the image are utilized. The algorithm starts by defining an empty image with a given resolution, and an empty raster with the same resolution. This raster is defined in the XY-plane. The projection puts each point from the point cloud into a cell in the generated raster. For detecting individual trees the point density at each cell is useful. Typically the points are densest where the vegetation is dense, which in turn will give information regarding trees. The point density is therefore an important feature for detecting trees. The point density is preserved by counting all points that falls within each pixel. The density value of each pixel is stored in the first band of the image. The second preserved feature is the height. The height is obviously lost when projecting a point cloud down to the XY-plane. The height is important for detecting trees, but even more important is the local height, meaning the difference between the highest and lowest point in a cell. This height information is preserved by storing the difference between the top and bottom point in each cell. Finally the height gradient is considered. This feature will preserve the topological differences between a cell and its neighbours, and will be able to locate the steep height differences that appears from a tree top down to the ground. The height gradient is found by calculating the gradient value for all 8-connected pixels. The algorithm for creating these images is shown in the following figure.

Algorithm 1: Map2D

Result: Image I **Input:** Pointcloud \mathbf{P} containing n points;Image resolution r ; $I = \text{Zeros}(r, r, 3)$; $C = \text{Cell}(r, r)$;**for** $p \in P$ **do** $x = \lceil p_x \times r \rceil$ $y = \lceil p_y \times r \rceil$; $I(x, y, 1)++$; $C(x, y).APPEND(p_i)$;**end****for** $u \leftarrow 1$ **to** r **do** **for** $v \leftarrow 1$ **to** r **do** $cell = C[u, v]$; $I(u, v, 2) = |max(cell_z) - min(cell_z)|$; **end****end****for** $u \leftarrow 1$ **to** r **do** **for** $v \leftarrow 1$ **to** r **do** $I(u, v, 3) = \sum_{i=-1}^1 \sum_{j=-1}^1 |I(u+i, v+j, 2) - I(u, v, 2)|$; **end****end****return** I

3.3.1 Selecting image resolution

When selecting the image resolution it is important to achieve high enough detail to detect the individual trees. Higher resolution is however not necessarily better. Higher resolution images takes more time to train, as there is more data to process. In addition the Map2D algorithm focuses on grouping points into cells. The image resolution will decide the size of these cells. If the cells are too small there will not be grouped enough

points in each cell. This will decrease the quality of the preserved attributes height, density, and height gradient. For example, when calculating the height gradient the 8-neighborhood is evaluated. If the resolution is too high the 8-neighborhood might not be populated by any points, and thus leaving the height gradient channel empty and useless. The difference between high and low resolutions can easily be seen in the following figure.

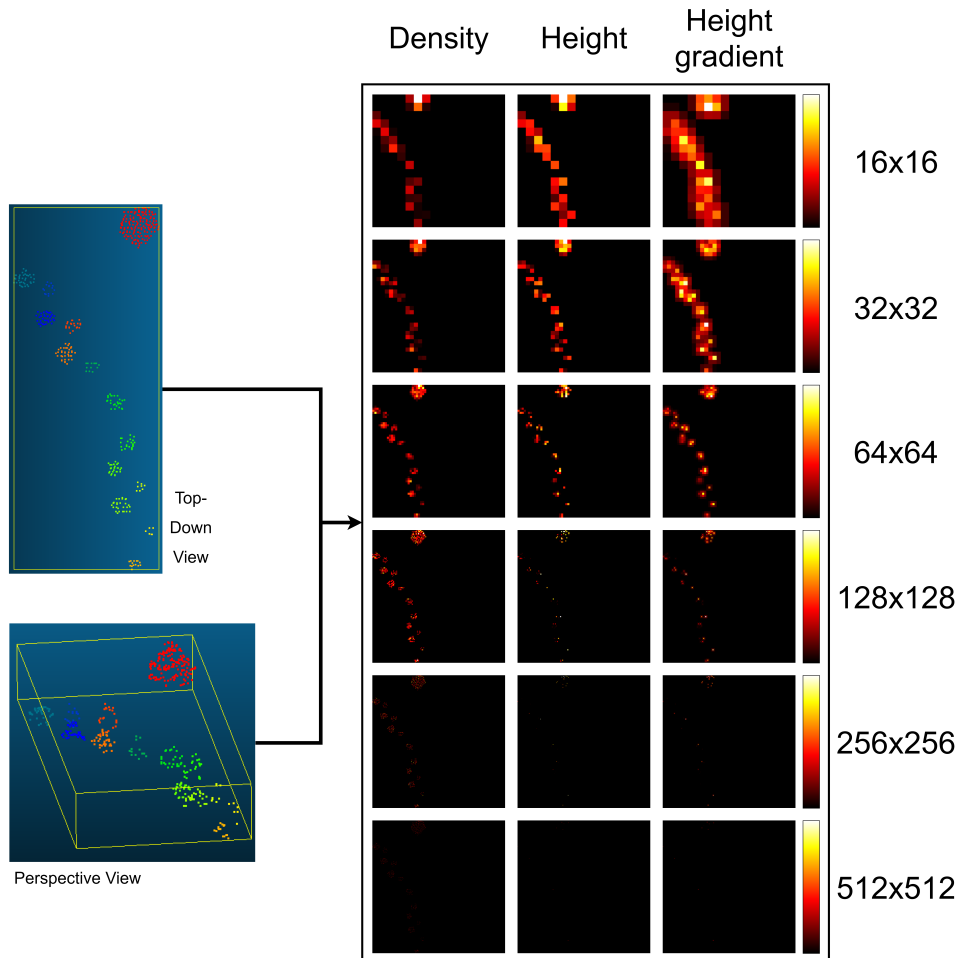


Figure 3.4: The trees will be represented differently on different resolutions.

From the figure 3.4 it is easy to see that too small resolution will not distinguish the smaller trees. On the other hand, the biggest resolution will at many cases have empty cells, and at most only populated by one point. This gives very little variety in the image values. It becomes a trade-off between having meaningful feature channels,

and preserving the detailed spatial positions of the XY points. The resolutions focused on in the experiments were 64x64 128x128, 256x256 and 512x512. The results of the various resolutions are presented in the results-section.

3.4 Top Down Slicing

Top down slicing is one of the modules in the pipeline. It is inspired by the height-slicing methods. Top down slicing is used to separate the point cloud into several height layers. The normalized point cloud P is partitioned into k subsets based on the z-value of each point. Each partition is equally distributed, and the most wanted partitioning would be a three-way split at 0.33, 0.67 and 1.0 respectively. This split would be mostly beneficial as it would split the trees into three distinct parts, the crown, the stem and the scattered shrub.

After splitting the points, each subset could be used separately in the Map2D algorithm, Leaving more information related to the height in the images. An object detection model could be adjusted to take these three individual images as an input and jointly predicting bounding boxes or segmentation masks, since the images overlap perfectly. Alternatively the images could be concatenated, leaving us with one 9-channel image.

Sadly, there was no time to do this in this master thesis. Several frameworks and finished models was tested, but none of the models could be adjusted to map three images to one target, or input a 9-channel image. Therefore, the top-down slicing module was discarded, and the points were not partitioned into height-layers. This lead to some untapped potential in the final model, but the general concepts still remains valid. Source code for doing the slicing is added in Appendix A.

3.5 Training a tree detection model.

The synthetic images generated with the Map2D algorithm are used to train an object detection model. The model used for the experiments is the Mask R-CNN [22]. The Mask R-CNN model extends Faster R-CNN [10] by adding a branch for predicting segmentation masks on each region of interest. The result is that it not only detects a bounding box for each tree, but a segmentation mask for each tree as well. The Mask

R-CNN model surpassed all previous models on the COCO instance segmentation task [11] in 2017. Although there are several models that has exceeded the performance in later years, Mask R-CNN is still one of the best publicly available models that can be used. And since implementing a object detection model from scratch is very time consuming, even with great frameworks like PyTorch [32] and TensorFlow [33], choosing the Mask R-CNN model was an easy choice.

The detection model is made available through Detectron2 [34]. Detectron2 is an open-source project made available by Facebook AI Research (FAIR). The project supports several different computer vision tasks, including object detection and instance segmentation. The project includes a model zoo that contains several pre-trained model architectures. These models will therefore quickly adapt to new tasks and data because the parameters have been tuned beforehand. Detectron2 works best when the data has been annotated in COCO format. This includes having two directories, one with images, and one with annotations in JSON format. The annotation files includes all bounding boxes and segmentation masks for training and evaluation. Detectron2 handles all the actual training as long as the data is in the correct format and all images are readable. A configuration can be specified for the training model. This configuration can be used to create a specific training environment for the model. This includes setting the learning rate, the loss functions, batch-sizes, and more.

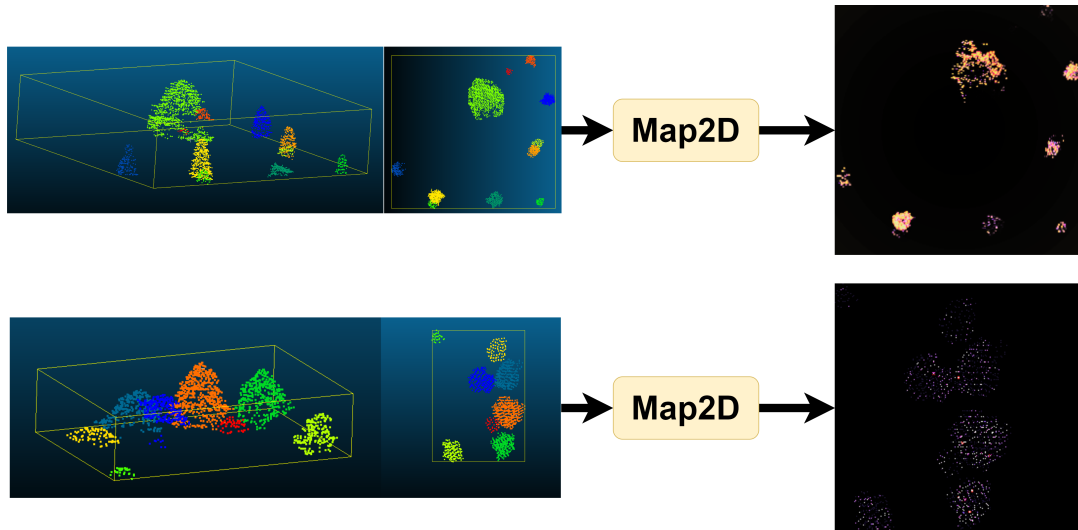


Figure 3.5: Synthetic Examples

Chapter 4

Experimental Results

In this chapter the experiments and results will be presented. The results will also be discussed. Firstly the available data will be introduced. Then the reader will get insights in the methods to process these data to generate results. Finally it will be presented how the various methods and choices affected the results.

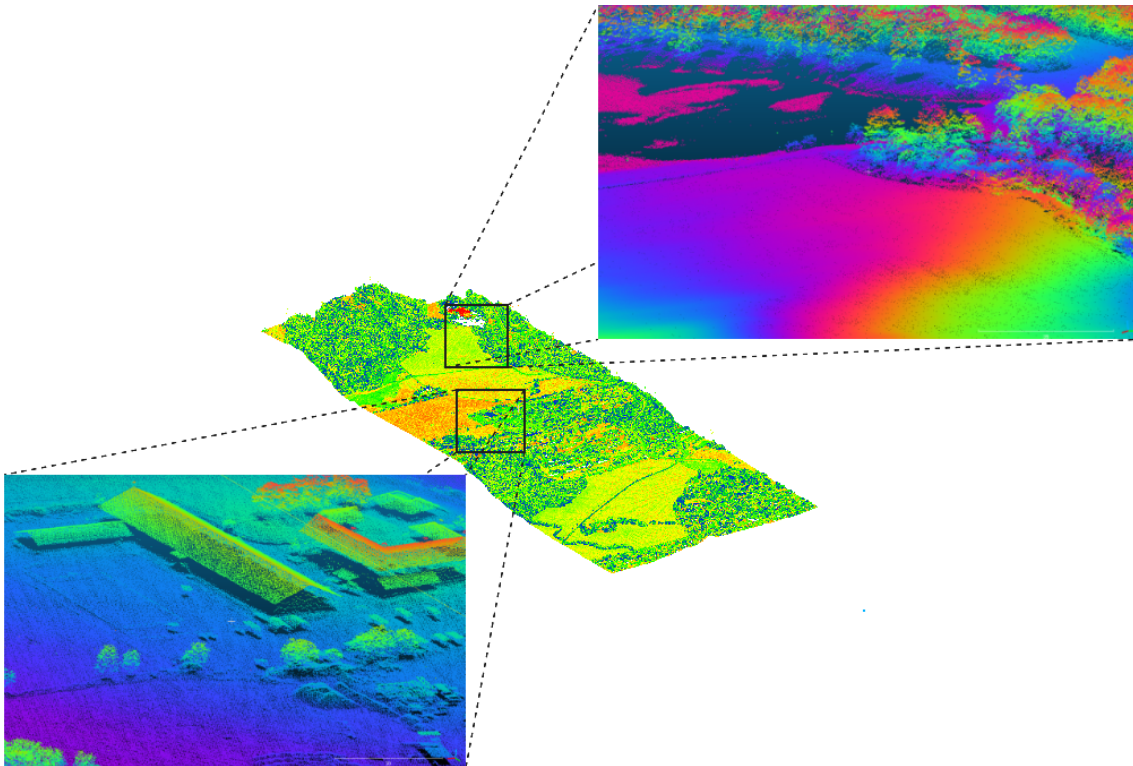


Figure 4.1: Scanned Area Used for Experiments

4.1 Available data

For the experiments in this master thesis a scanned area in Trondheim was made available. The data was scanned using airborne lidar. It covers an area east in Trondheim (Ranheim). The original point cloud is shown in Figure 4.1.

The point cloud consists of 24,126,903 points over an 800x2400 metres area, which means an average point density of about 10 points/ m^2 . The points presented are not classified in any way, but does contain some scalar fields such as number of returns, intensity and scan angle rank. These attributes are helpful for visualization, but only the positional data are used for detecting individual trees. To train a model on this data it is first necessary to include some targets for the training, and this is done by labelling the data.

4.2 Labelling the data

In order to use the data for supervised learning, the first step is to label data. In this experiment, 200 trees were manually labelled into individual trees. The process of labelling was done by clipping trees into point cloud segments, so that each individual tree became an individual point cloud file. For labelling the trees, the open-source software CloudCompare[35] was used. The software enables a simple view and a basic toolkit for manipulating a point cloud. All individual trees from the point cloud were gathered using this software.

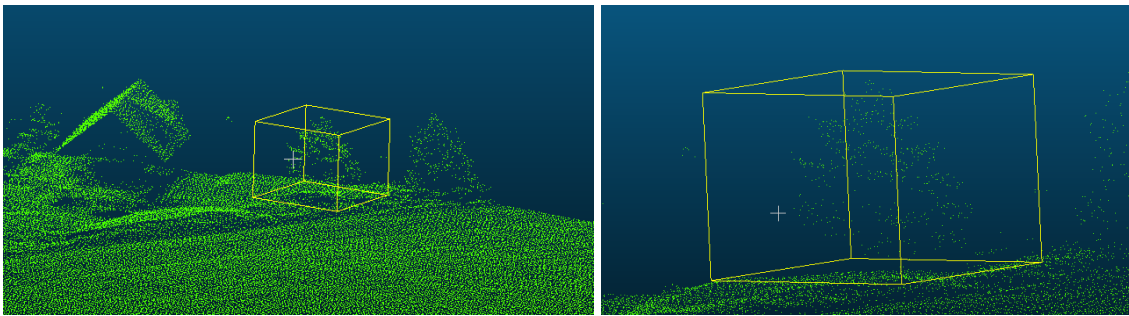


Figure 4.2: Example on how the trees were manually extracted using CloudCompare [35].

There are some obvious challenges with manually labelling the data. For trees that are not surrounded by anything else it is straightforward to include all points

belonging to that tree. However when working on densely grouped trees it is harder to see which points belong to which trees. The trees can be very close together, and even have a shared crown, which makes it hard to distinguish each tree. One thing that helped this process was to use Google Maps Streetview. By looking at the trees from different angles it was easier to tell what was individual trees, especially in cases where the trees branched out or merged together. An even better method would be looking at the trees physically on site and simultaneously labelling the data points. The more obscure examples were not chosen for labelling, making sure that the examples that were labelled, were also correct. This led to an oversampling of standalone trees, which is a problem because the model should also learn how to detect trees that are close to other trees. The assumption is that the synthetic sampling will solve this, since the method will put trees close together at random. In the following sections this assumption will be tested.

MANUALLY LABELLED TREES

Property	Value
Average amount of points	472 Points
Average diameter	8,41 Metres
Average height	12,61 Metres

Table 4.1: Manually labelled trees properties

The trees were labelled randomly along all of the point cloud. The diversity of the trees was preserved by taking both big and small trees, in steep and flat terrain. In addition both "cultivated trees" close to houses and in backyards, as well as trees in the middle of a forest, was segmented. A suitable test set was found in the middle of the original point cloud at a small farm. In this area all trees within was segmented, and sorted into a separate collection, which would only be used for testing, and not training.

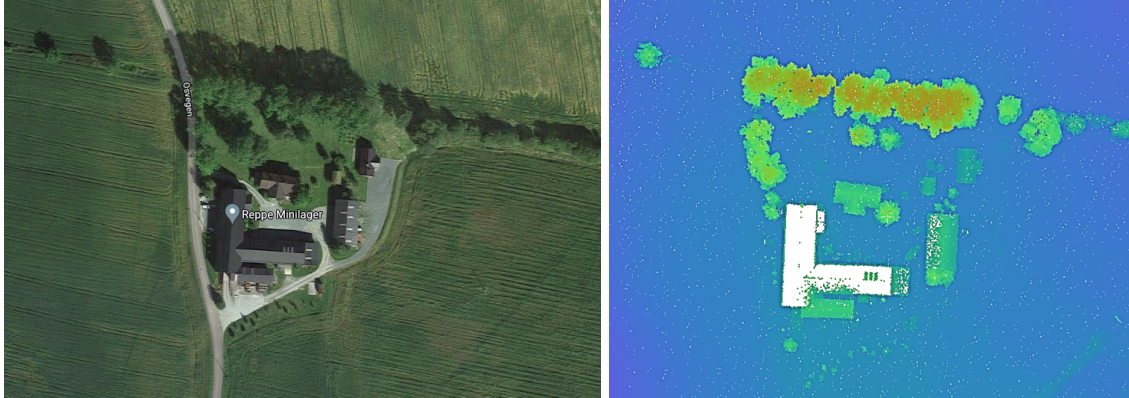


Figure 4.3: Scanned area used for experiments. Google maps satellite image to the left, available point cloud to the right.

One main source of error is the manual labelling of the test area. Since the area needed to be fully labelled, this included many difficult trees. Since the available data lacks density, it was not always easy to see what points belonged to what tree, and at some points this might have led to mislabelling of points. To care for this error source it was looked at images from google streetview and satellite images to accord for how many trees there actually was in tight groups of trees. Thus, the number of trees should be correct, however, the specific points may at several cases be wrongly labelled, and it might also have happened that some trees have been merged by error. To get a quantitative measure for this, a new validation set was introduced. The new validation set consists of 20% of the labelled trees outside of the test area. When training this data set, 20 % of the trees were held out, and used as validation set instead. Since these trees generally consists of trees standing alone it can with high certainty be said that it has been correctly labelled. In addition, the trees in this set generally contain denser points, because they are not occluded by other trees. Therefore these results can give us an indication on how well the model will perform on denser datasets, and whether the test set can be trusted or not.

4.3 Performance

The model was trained for 20k iterations, using 50k images that were randomly generated by sampling from 200 individual trees. The model chosen for the training was the deepest Mask R-CNN model available in the model zoo of Detectron2. This is a Mask

R-CNN model with a ResNext 101 backbone. The model was tested on three different datasets. The parameters of the trained model is shown in the following table.

TRAINING PARAMETERS

Backbone	ResNext 101 32x8d
Pretrained on dataset	ImageNet
Images Per Batch	4
Learning Rate	.0005
Number of iterations	15k

Table 4.2: Training parameters

Test Area contains the images containing all the trees from the chosen test area as shown in 4.3. The trees here are used "as is", which means that they have not been altered in any way, they are standing exactly where they were from the original scanned point cloud. It should be noted that there is only 61 individual trees in this dataset, and since the images are not augmented, each tree only appears one time. The area is split into four images.

Synthetic Test Area also consists of the trees from the chosen test area. However, these trees have been sampled from and been augmented several times to generate synthetic images. 500 images were generated for testing, each image containing a random sample of 1 to 24 trees, resulting in about 5000 trees in total.

20% Holdout was the final dataset used for testing. Unlike the other two datasets, this dataset was made by randomly sampling from trees outside of the test area, that was generally only used for training. When testing this dataset, the model was retrained using only 80% of the trees, leaving 40 individual trees for generating testing images. In total 500 images were generated by randomly sampling from these 40 trees, leaving us with about 5000 trees in total.

The performance of the model on each dataset can be seen in the table below. The precision and recall metrics are calculated for a IoU-threshold of 0.5. Each score displayed in the graph is the best score achieved during training. A model was trained with similar parameters, except that it was trained using the ResNet50 backbone. This is a more shallow network, with only 50 layers, compared to ResNext101 which has 101 layers. ResNext 101 also is a newer generation, with more residual connections

than before [36]. The architectures of the different backbones can be viewed in the paper of ResNet [16] and the paper of ResNext [36].

PERFORMANCE

Dataset	Precision	Recall	ResNext101 AP	ResNet50 AP
Test Area	0.820	0.820	0.503	0.479
Synthetic Test Area	0.809	0.805	0.599	0.545
20% Holdout	0.972	0.961	0.812	

Table 4.3: Final performance on the various datasets.

The results shows that for the real test-data the precision is 0.820 and the recall is 0.820. These trees represents a combination of simple, and hard examples. Especially it is worth noting that the trees might not be correctly labeled, as there is no way to accurately, manually label trees when at times the point clouds are very sparse, and the trees are densely grouped. Considering that the test site is quite a complex area, containing many trees merged together, these results are definitely able to compete with the traditional CHM-based- and Height-Slicing models, which does not get over 65% accuracy in the state-of-the-art implementations. These results do not, however, compete with the best deep learning models that exists. There are many disadvantages in these experiments. One of the major being the dataset. The data used for the results in the deep learning model [9] is 50 times more dense then the data used in this project. It is expected that the model would perform better with denser data, since it would increase the amount of detail captured in the feature projection.

The model was also tested on the synthetic testset. The accuracy was similiar to the test trees, diverging with only 1.1% and 1.5% percent for the precision and the recall, respectively. This indicates that the random sampling works well, and that it preserves the difficulty often found in real examples.

To account for the difficulty of accurately labelling the trees a test was also carried out on the validation set, containing 20% of the training trees. These trees were taken out before training, and were only used for testing. The fact that this dataset gets an

precision and recall of 0.972 and 0.966 respectively gives a clear indication that the model is very good at predicting individual trees, given that the point cloud for each tree is dense, and complete.

By looking at some results it is more easy to determine what exactly is the upside and downside of the method. In the end of this chapter some of the results have been visualized. The visualizations includes three samples from the 20% Holdout set, and three samples from the real test area. The predictions has been displayed in the projected images, and the predictions are not at this point converted back to the point clouds, since this gives most intuitive view of the predictions. The images are displayed in figure 4.7.

4.3.1 Performance of various resolutions

Choosing the resolution of the images was a discussed topic. An experiment was carried out to test the effects of the various resolutions. 50k images was generated for the quadratic resolutions 64, 128, 256 and 512. The model was then in turn trained on the various resolutions for 5000 iterations. Results were generated during training for both the *Test Set* and the *Synthetic Test Set*.

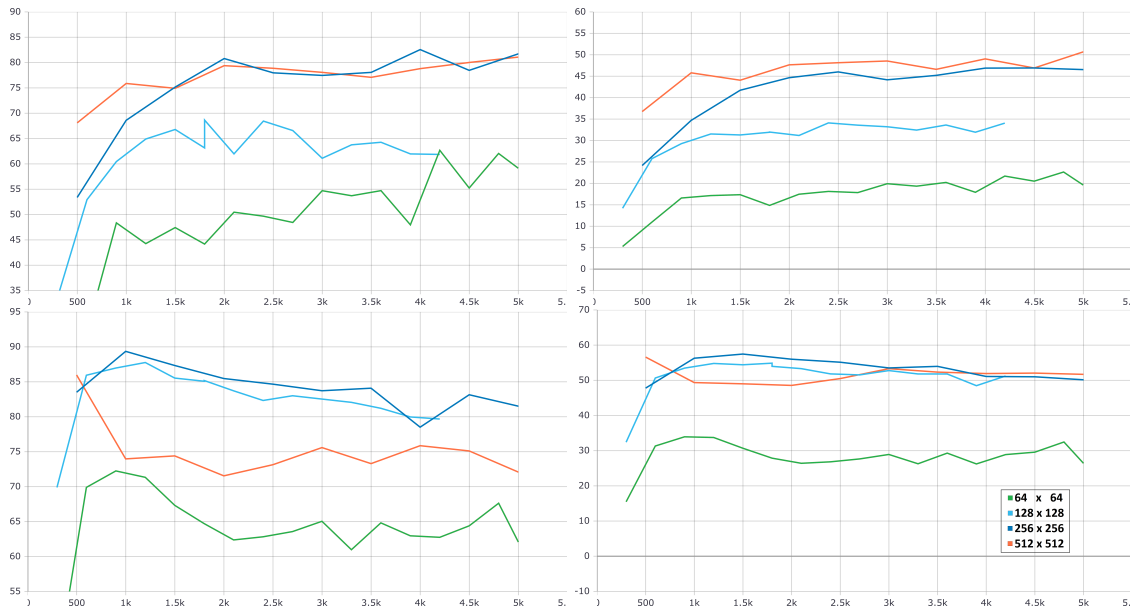


Figure 4.4: Performance during training for various resolutions. Upper left: AP50 on the real test area. Upper right: AP on the real test area. Lower left: AP50 on the synthetic test area. Lower right: AP on the synthetic test area.

The tendency is overall that with higher resolution, comes higher performance. This is an expected result, as you get more details and information with higher resolutions. It should be noted that overall 256x256 and 512x512 performed quite similar. When the trees get rasterized at too high resolutions, each cell will rarely get more than 0 or 1 points in them, reducing the variety and quality of the calculated features, leaving only the detailed positional information for the images to learn from. However it seems that this does not affect the performance of the model, probably because it gets more detailed positional data in return.

When comparing the resolutions 256 and 512 it is possible to see that 256 performed better on the AP50 metric, while 512 performed best on the AP metric. This indicates that the 256 resolution is better at predicting the trees, but less accurately pinpoints the position of the bounding box, while the 512 gives more accurate bounding boxes but predicts fewer trees. This makes sense, as the 512x512 resolution images has more spatial detail due to its cell size, and can more precisely pinpoint the position.

It is also important to note that higher resolutions has several disadvantages. The images use more storage space, they take longer time to generate, and they increase the training time. It was decided that 256x256 seemed to be the ideal trade-off between performance and the disadvantages mentioned above.

4.3.2 Performance with various amounts of synthetic images

An assumption taken when creating this method was that synthetic data helped the model generalize, and increasing the amount of synthetic data would increase the performance of the model. An experiment was carried out to determine how the amount of synthetic data affected the final model. For this experiment, the training set was partitioned into seven subsets. The subsets contained 10, 50, 100, 500, 1000, 5000, and 10k images, respectively. Each subset is built upon the previous, meaning that the 50 image set contains all images from the 10 image set, the 100 image set contains all images from the 50 images set and so on. This is done to ensure that no smaller set has any "better" images. The model was trained over again for each subset, and the final model for each training set was tested on the Test Area dataset. Each model was used at various IoU thresholds, and the F1-score was calculated for each threshold. The results can be seen in the following graph.

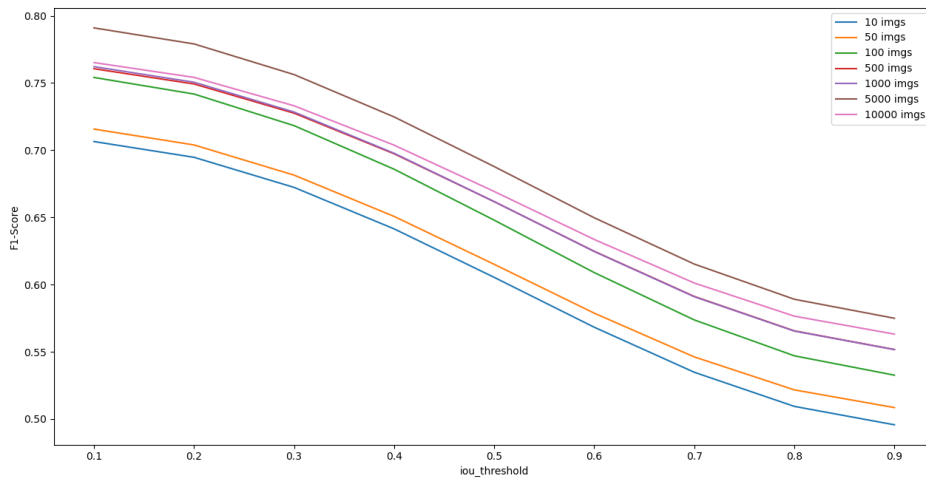


Figure 4.5: F1 score for various IoU-thresholds. The graph clearly indicates a correlation between overall higher F1-score at higher amounts of data.

From the chart it can be seen that the results gradually become better until the peak on 5000 images. After 5000 images the performance surprisingly decreases. This can be an indication that 5000 is enough data to maximize the potential of the data augmentation. Note that all of the 5000 images contain 4 to 20 trees. The amount of trees to sample from is only 200, meaning that each tree is used approximately 250 times in various orientations, positions and sizes. It might make sense that the potential is used up.

There is no explanation to why 5000 images should be better than 10000 images, since information is just included when having more images, making it possible for the model to better learn the underlying features. Since the model is trained for a similar amount of iterations there is also no reason for 10k images causing more overfitting than 5000 images. The reason for it may be coincidental, either being because of too little test data, or an "unlucky" run of the model. The model is learning stochastically, meaning that each run will differ from each other, even when the variables and parameters are the same.

4.3.3 Performance with various amounts of manually labelled trees

The tree detection method in this paper is based on using synthetic data for training a model. Synthetic examples are created by augmenting manually labelled trees. It is assumed that the same trees can be used several times to generate a synthetic dataset, however it is not known how the amount of labelled trees affects the final results. In this experiment the effect of each individual tree is investigated. In total 200 trees were manually labelled. The 200 trees were in this experiment sub-sampled to generate various datasets with various amounts of manually labelled trees. Since the previous experiments showed that 5k images were enough to utilize the potential of a dataset of 200 trees, that amount of generated examples was used for this experiment as well. The experiment was tested with arbitrarily chosen trees for the amounts less than 200. The amounts tested was: 5, 10, 20, 50, 100 and 200. Each subset was built upon the previous, meaning that the 10 tree subset also included the same trees as in the 5 tree subset, and the 50 tree subset included the same trees as the 20 tree subset and so on. For each amount of individually labelled trees a new batch of 5000 images was created, sampled from the trees in the respective subset. It was then trained for 5k iterations on the Mask R-CNN model. The final model for each subset was used for testing on the Test Area dataset. The results are shown in the following graph.

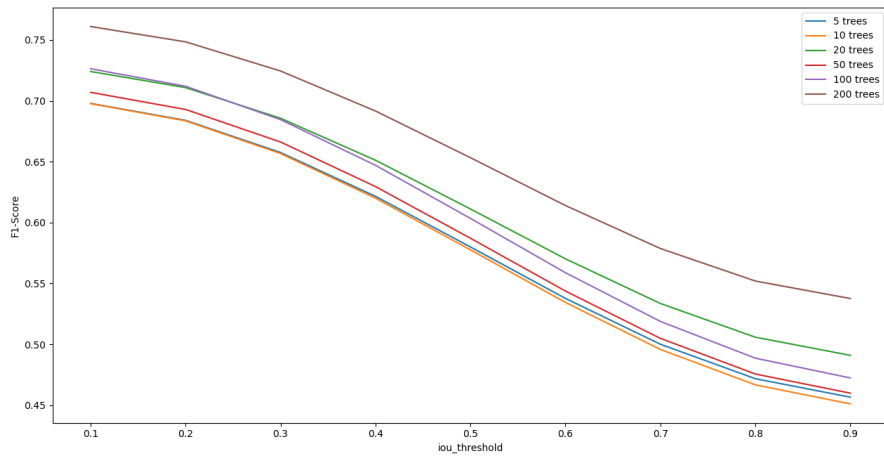


Figure 4.6: F1-score over IoU-thresholds. Results on bounding boxes.

It can be seen that the model trained with 200 trees is the best performing one. The

model trained with only 20 unique trees also shows a surprisingly good score. This can indicate that it is possible to create good models with a limited amount of manually labelled data when data augmentation is used extensively. It is however quite unusual that 20 trees is getting a better score than 100 trees. The reason for this might be overfitting. The 20 trees might match the test trees best, and thus causing the model to overfit against these types of trees. An important source of error is the amount of trees in the test set. 61 trees is not enough data to conclude with anything, as the sample size is not big enough to get a valid data variance. It is however interesting to see that the model can perform well even when the amount of manually labelled trees is extremely low.

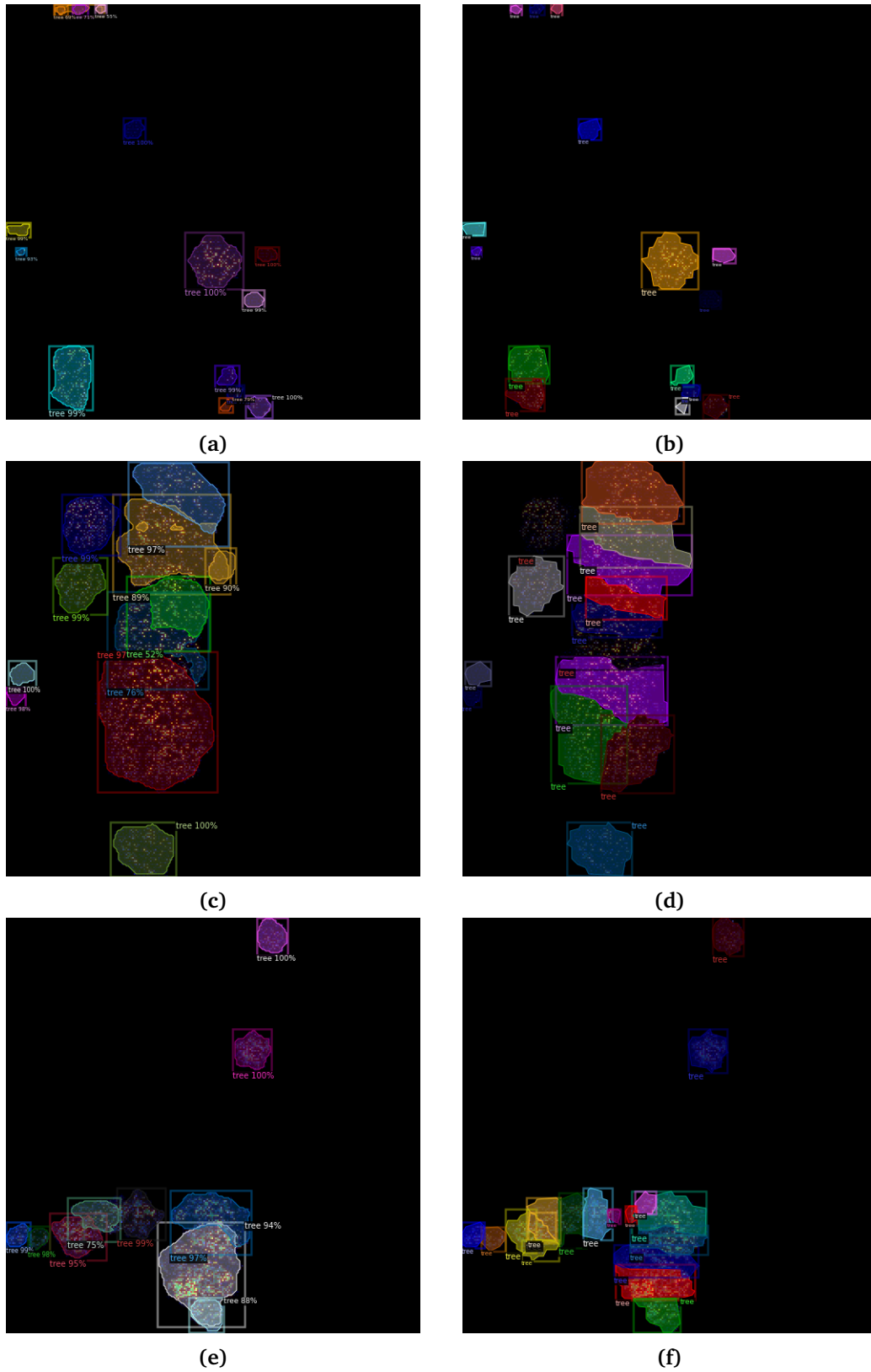
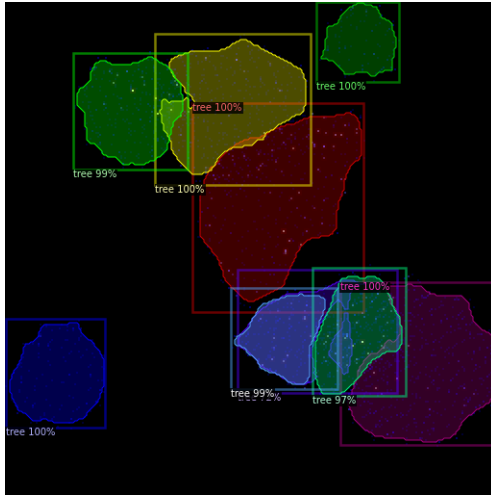
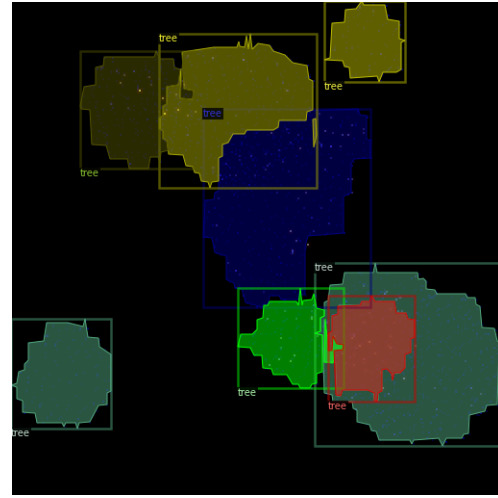


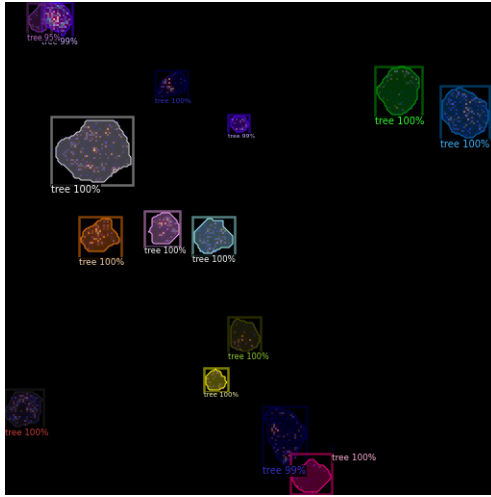
Figure 4.7: Predictions to the left against ground truths on the right. These are taken from the 20% Holdout Set, which has the best segmented trees



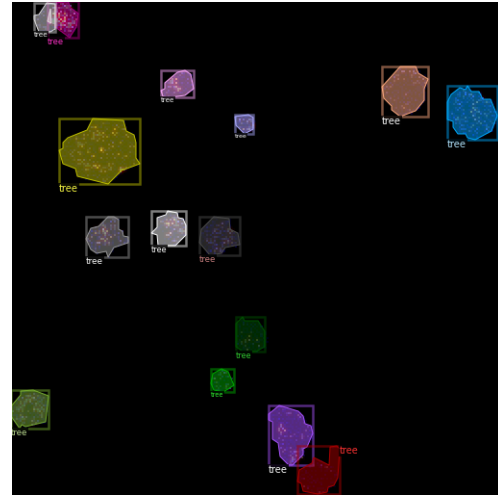
(g)



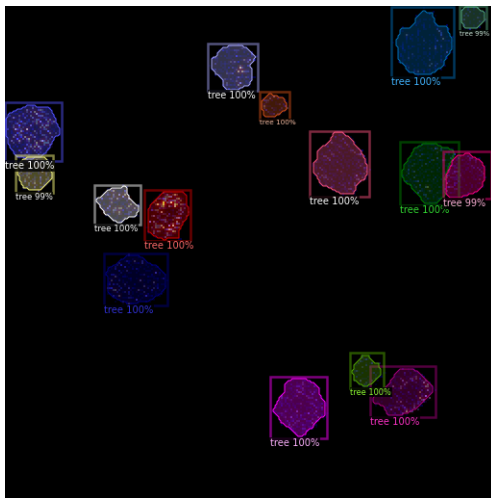
(h)



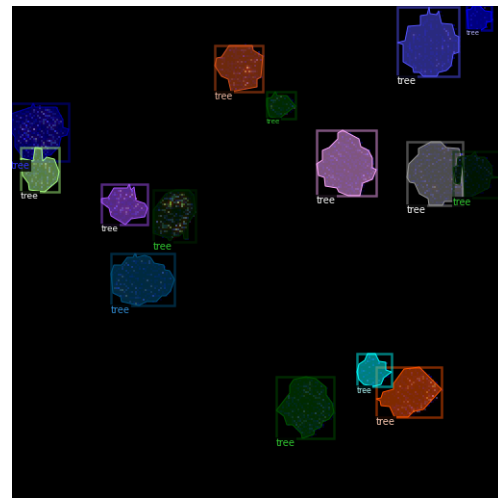
(i)



(j)



(k)



(l)

Figure 4.7: Predictions to the left against ground truths on the right. These are taken from the Test Area Set, which has the some occlusion in the segmented trees

Chapter 5

Conclusion and Future Work

In this chapter the work will be summarized and concluded. Future work will be suggested for anybody who would like to continue the journey of detecting individual trees using feature projection and deep learning.

In this master thesis the main objective was to create a new method for automatically detecting individual trees from point clouds. A feature projection algorithm has been designed and implemented, to create images from the point clouds. A Mask R-CNN model has been trained to detect individual trees in the projection images. A fully automatic method for creating a synthetic dataset has been implemented and used to account for the lack of labelled data. Various setups of the process has been tested, including using different image resolutions, using different amounts of individually labelled trees, and having various amounts of synthetic images in the training dataset.

Results from the experiments confirms that the method is feasible, with a precision of 82.0% and a recall of 82.0% on a real test area, and a precision and recall of 97,2% and 96,1%, respectively, on a synthetic test set. These results shows that the method is able to compete with traditional methods. The method may be able to compete with other deep learning methods as well, however, due to the difference in data quality between the reported results, this can not be concluded.

When testing various amounts of individually labelled trees the results showed that the synthetic sampling worked surprisingly well. It was possible to create decent prediction models with only 20 manually labelled trees in the training dataset. It was also tested how many times the trees could be used in the training set before the model stopped improving. These results showed that each tree could be used approximately

250 times in different sampled images before the results converged to a maximum. This shows clear indication that synthetic sampling is a good method to use in these kinds of detection tasks.

The results does however lack validity, due to the size of the testing dataset. The testing dataset does only contain 61 trees, which means that the models results could change a lot when tested on more trees. In addition, the method has solely focused on detecting individual trees, and has not included any other classes in the training or testing of the model. It can not be reasoned whether the final model is able to distinguish trees from other objects like buildings, streetlights, electrical masts and cars or not.

Nevertheless, the methods shows promising results, and the Deep Tree Detector method has proved to be a method that should be further investigated.

Future Work

There are several things that could be improved in this project. The validity of the results could become a lot stronger if more labelled data was collected. It would also be highly beneficial if the trees were collected at other locations, since that would give the dataset more variety. There are several datasets used in a wide range of different studies. Reaching out to the authors of those studies would be a step in the right direction for acquiring more labelled data, that will both uncover more potential and more challenges.

The density of the point cloud was a problem during this project. Density of the point clouds limits the variety of the projected features. Often there were not enough points to give distinguishable values to the different pixels. If work is to be continued it would be very interesting to look at point clouds with higher density, and compare the models predictive qualities on various densities.

It was not focused on testing various object detection models in this thesis. The Mask R-CNN was simply chosen because it is acclaimed, and it was the strongest performing model available in Detectron2's model zoo. Other object detection models could be more suitable for the task of detecting trees. In addition there exists newer and better models to this day, however, they are not as accessible. Seeking out authors of these models could be done, and a comparison study could be made to see which

model is best suited to the task of detecting individual trees.

The top down slicing module was discussed in chapter 3. It involved slicing z-axis to preserve even more spatial information. In this thesis it was not prioritized to implement this module as it was not compatible with the frameworks used in the experiments. Implementing and utilizing this module in an object detection model should increase the performance of the model. Future work should include this module as it is expected to preserve more spatial data, and thus leading to better performance of the method.

Bibliography

- [1] M. J. Raupp, A. B. Cumming and E. C. Raupp, 'Street tree diversity in eastern north america and its potential for tree loss to exotic borers,' 2006.
- [2] J. Kronenberg, K. Hubacek *et al.*, 'Urban ecosystem services.,' *Landscape and Urban Planning*, vol. 109, no. 1, pp. 1–127, 2013.
- [3] A. B. Nielsen, J. Östberg, T. Delshammar *et al.*, 'Review of urban tree inventory methods used to collect data at single-tree level,' *Arboriculture & Urban Forestry*, vol. 40, no. 2, pp. 96–111, 2014.
- [4] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks,' in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [5] R. Adams and L. Bischof, 'Seeded region growing,' *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 6, pp. 641–647, 1994.
- [6] H. Kaartinen, J. Hyypä, X. Yu, M. Vastaranta, H. Hyypä, A. Kukko, M. Holopainen, C. Heipke, M. Hirschmugl, F. Morsdorf *et al.*, 'An international comparison of individual tree detection and extraction using airborne laser scanning,' *Remote Sensing*, vol. 4, no. 4, pp. 950–974, 2012.
- [7] M. Dalponte, F. Reyes, K. Kandare and D. Gianelle, 'Delineation of individual tree crowns from als and hyperspectral data: A comparison among four methods,' *European Journal of Remote Sensing*, vol. 48, no. 1, pp. 365–382, 2015.
- [8] Q. Guo, Y. Su, T. Hu, H. Guan, S. Jin, J. Zhang, X. Zhao, K. Xu, D. Wei, M. Kelly *et al.*, 'Lidar boosts 3d ecological observations and modelings: A review and perspective,' *IEEE Geoscience and Remote Sensing Magazine*, no. 99, pp. 0–0, 2020.

- [9] L. Windrim and M. Bryson, 'Detection, segmentation, and model fitting of individual tree stems from airborne laser scanning of forests using deep learning,' *Remote Sensing*, vol. 12, no. 9, p. 1469, 2020.
- [10] S. Ren, K. He, R. Girshick and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV].
- [11] Coco, <https://cocodataset.org/>. Coco, Visited 28.09.2020.
- [12] Cityscapes, <https://www.cityscapes-dataset.com/>. Cityscapes, Visited 13.02.2021.
- [13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, 'Imagenet: A large-scale hierarchical image database,' in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [14] S. Saha, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Towards Datascience, Visited 15.04.2021.
- [15] S. Albawi, T. A. Mohammed and S. Al-Zawi, 'Understanding of a convolutional neural network,' in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.
- [16] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].
- [17] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV].
- [18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, 'Ssd: Single shot multibox detector,' *Lecture Notes in Computer Science*, pp. 21–37, 2016, ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2.
- [19] O. Ronneberger, P. Fischer and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV].
- [20] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, 'Focal loss for dense object detection,' in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

- [21] B. De Brabandere, D. Neven and L. Van Gool, ‘Semantic instance segmentation for autonomous driving,’ in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 478–480. DOI: 10.1109/CVPRW.2017.66.
- [22] K. He, G. Gkioxari, P. Dollár and R. Girshick, *Mask r-cnn*, 2018. arXiv: 1703.06870 [cs.CV].
- [23] V. Varatharasan, H.-S. Shin, A. Tsourdos and N. Colosimo, ‘Improving learning effectiveness for object detection and classification in cluttered backgrounds,’ Feb. 2020.
- [24] S. A. Bello, S. Yu, C. Wang, J. M. Adam and J. Li, ‘Review: Deep learning on 3d point clouds,’ *Remote Sensing*, vol. 12, no. 11, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12111729. [Online]. Available: <https://www.mdpi.com/2072-4292/12/11/1729>.
- [25] C. R. Qi, H. Su, K. Mo and L. J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, 2017. arXiv: 1612.00593 [cs.CV].
- [26] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein and J. M. Solomon, *Dynamic graph cnn for learning on point clouds*, 2019. arXiv: 1801.07829 [cs.CV].
- [27] Y. Xu, T. Fan, M. Xu, L. Zeng and Y. Qiao, *Spidercnn: Deep learning on point sets with parameterized convolutional filters*, 2018. arXiv: 1803.11527 [cs.CV].
- [28] D. Maturana and S. Scherer, ‘Voxnet: A 3d convolutional neural network for real-time object recognition,’ in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928. DOI: 10.1109/IROS.2015.7353481.
- [29] E. Kalogerakis, M. Averkiou, S. Maji and S. Chaudhuri, *3d shape segmentation with projective convolutional networks*, 2017. arXiv: 1612.02808 [cs.CV].
- [30] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang and J. Kautz, *Splatnet: Sparse lattice networks for point cloud processing*, 2018. arXiv: 1802.08275 [cs.CV].
- [31] C. Shorten and T. M. Khoshgoftaar, ‘A survey on image data augmentation for deep learning,’ *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.
- [32] PyTorch, <https://pytorch.org/>. PyTorch, Visited 21.10.2020.

- [33] TensorFlow, <https://www.tensorflow.org/>. TensorFlow, Visited 21.10.2020.
- [34] Facebook, <https://github.com/facebookresearch/detectron2>. Facebook, Visited 16.11.2020.
- [35] CloudCompare, <https://www.danielgm.net/cc/>. CloudCompare, Visited 03.05.2021.
- [36] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, *Aggregated residual transformations for deep neural networks*, 2017. arXiv: 1611.05431 [cs.CV].

Appendix A

Source Code For Generating Synthetic Images

```
import pandas as pd

def get_random_sample(
    num_trees: int = 32, width: int = 8,
    height: int = 8, resolution: int = 64,
    tree_glob: str = "train_trees/*.las", image_id = 0):

    random_trees = get_n_random_trees(
        num_trees, tree_glob=tree_glob,
    )
    random_trees = preprocess_trees(random_trees)

    scattered_trees_on_grid = scatter_trees_on_grid(
        random_trees, [width, height],
    )
    max_xy = max([
        scattered_trees_on_grid.points["x"].max() - scattered_trees_on_grid.points["x"].min(),
        scattered_trees_on_grid.points["y"].max() - scattered_trees_on_grid.points["y"].min()
    ])
    scattered_trees_on_grid.points = normalize(scattered_trees_on_grid.points, max_xy)
    sequences = top_down_slicing(scattered_trees_on_grid.points, 3)
    image = Map2D(sequences[-1], r = resolution)

    label_image, label = create_label_image(sequences[-1], r = resolution, image_id=image_id)
    return image, label_image, label
```

```

import numpy as np

def Mapping_M(slice: pd.DataFrame, r: int = 256):
    n = slice.shape[0]

    x_list = list(slice['x'])
    y_list = list(slice['y'])

    I = np.zeros((r,r,3))
    C = {} #C = (r, r)
    for i in range(r):
        for j in range(r):
            C[i,j] = []

    for j in (range(n)):
        x = math.floor(x_list[j] * r) if math.floor(x_list[j] * r) < r else r-1
        y = math.floor(y_list[j] * r) if math.floor(y_list[j] * r) < r else r-1
        I[x,y,0] = I[x,y,0] + 1 #Appending to the density
        C[x,y].append(slice.iloc[j]['z'])

    for h in (range(r)):
        for t in range(r):
            z = C[h,t]
            val = abs(max(z) - min(z)) if z else 0
            I[h,t,1] = val

    for h in (range(r)):
        for t in range(r):
            grad_h = 0
            for i in [-1,0,1]:
                for j in [-1,0,1]:
                    if h+i >= r or h+i < 0 or t+j >= r or t+j < 0:
                        continue
                    grad_h += abs(I[h+i,t+j,1] - I[h,t,1])
            I[h,t,2] = grad_h

    scale_factors = 255 / np.amax(I,axis=(0,1))
    for x in range(I.shape[0]):
        for y in range(I.shape[1]):
            for z in range(I.shape[2]):
                I[x,y,z] = scale_factors[z]*I[x,y,z]

    return I

```

```

import Pyntcloud

def get_n_random_trees(num_trees, tree_glob = "train_trees/*.las"):
    """
    Returns n random trees from the given directory
    the format of the returned trees are loaded pyntcloud objects
    """
    files = shuffle(glob(tree_glob))
    files = files[:num_trees] if len(files) >= num_trees else files

    trees = []
    for file in files:
        tree = PyntCloud.from_file(file)
        tree.points["x"], tree.points["y"] = rotate(
            tree.points["x"], tree.points["y"],
            math.radians(randint(1,360))
        )
        trees.append(tree)
    return trees

```

```

def preprocess_trees(random_trees: list):
    """
    For each tree, put the lowest point in origo
    For all trees, find maximum value, divide all trees on this maxima.
    For all trees, scatter
    """
    cols = ['x', 'y', 'z']
    for tree in random_trees:
        tree.points = put_in_origo(tree.points, cols)
    col_max_values = get_maximum_xyz(random_trees, cols)
    random_trees = normalize_trees_on_max_xyz(random_trees, cols, col_max_values)

    return random_trees

```

```

import pandas as pd
def normalize(pc: pd.DataFrame, max_val: float):
    cols = ['x', 'y']
    pc[cols] = pc[cols].apply(lambda x: (x - x.min()), axis=0)
    pc[cols] = pc[cols].apply(lambda x: (x/max_val), axis=0)

    pc[["z"]] = pc[["z"]].apply(lambda x: (x - x.min()), axis=0)
    pc[["z"]] = pc[["z"]].apply(lambda x: (x/x.max()), axis=0)

    return pc

```

```

import pandas as pd
from random import uniform

def scatter_trees_on_grid(random_trees: list, width_height: list, allow_clustering: bool):
    """
        Scatters the trees randomly on a grid
        Returns a PyntCloud with all the trees scattered and merged into one
    """
    axes = ["x", "y"]
    label_id = 1

    if allow_clustering:
        for tree in random_trees:
            for axis, max in zip(axes, width_height):
                tree.points[axis] = tree.points[axis].add(round(uniform(0, max-1), 3))
                tree.points["label_id"] = label_id
            label_id += 1
    else:
        raise Exception("Invalid..")

    merged_trees = random_trees[0]
    for tree in random_trees[1:]:
        merged_trees.points = pd.concat([merged_trees.points, tree.points])

    return merged_trees

```

```

import pandas as pd
import math

def top_down_slicing(points: pd.DataFrame, k = 3):
    """
        Adds a column 'slice' to the dataframe. the k slices are equal in length
        given k=3:
            slice 1 is the scattered shrub
            slice 2 is the rod-shaped trunk
            slice 3 is the spherical canopy
        returns a list of the sequences
    """
    points['slice'] = points.apply(lambda row:
        math.ceil(row["z"] * k) if math.ceil(row["z"] * k) > 0
        else 1, axis=1)
    slices = []
    for s in range(1, k + 1):
        slices.append(points[points['slice'] <= s])

    return slices

```

```
from random import randint
import json
import cv2

if __name__ == "__main__":

    num_samples = 500
    resolution = 256
    train_val_ratio = 1
    for i in range(num_samples):
        width_height = randint(3,8)
        num_trees = randint(1,width_height*3)
        example, target, labels = get_random_sample(
            num_trees, tree_glob="train_trees/train/*.las",
            image_id=i, resolution=resolution,
            width=width_height, height = width_height,
        )

        cv2.imwrite("data/train/images/"+str(i)+".png", example)
        with open("data/train/labels/"+str(i)+".json", "w") as f:
            f.write(json.dumps(labels, indent = 4))

    if i % train_val_ratio == 0:
        example, target, labels = get_random_sample(
            num_trees=num_trees, tree_glob="train_trees/val/*.las",
            image_id=int(i/train_val_ratio), resolution=resolution,
            width=width_height, height = width_height
        )
        cv2.imwrite("data/val/images/"+str(int(i/train_val_ratio))+".png", example)
        with open("data/val/labels/"+str(int(i/train_val_ratio))+".json", "w") as f:
            f.write(json.dumps(labels, indent = 4))
```