DEEP CONVOLUTIONAL NEURAL NETWORK BASED SINGLE TREE DETECTION USING VOLUMETRIC MODULE FROM AIRBORNE LIDAR DATA

HYUNGJU LEE

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN EARTH AND SPACE SCIENCE

YORK UNIVERSITY

TORONTO, ONTARIO

November 2022

Abstract

There was an undeniable success of Deep Learning networks for visual data analytics such as object detection and segmentation in recent years, while the adaptation to tree detection has been rare. In this paper, we pursue to achieve individual tree identification, defined as a detection of an individual tree as each object, with deep convolutional neural networks to create and update tree inventories using LiDAR information. The first objective was to provide a suitable dataset that can be used to test such networks and to create a module that attempts to increase the 3D object detection algorithms' detection accuracy. This novel dataset was created by fusing LiDAR data gathered by Teledyne Optech with field data collected by York University. The second was to develop an appropriate accuracy increasing volumetric module. For this module, the learnable weights concept was introduced, which enable to increase detection precision of the object detection algorithm.

Acknowledgments

I must express my deep appreciation to the people who supported and helped me as a graduate student, including many ups and downs. But, the end of the day became a memorable and wonderful experience of my life because of those people who walked with me.

First, I thank my supervisors, Dr. Gunho Sohn and Dr. Connie Ko. As a person who just entered the world of research, I could be left lost without these two wonderful people. They always provided the utmost level of guidance and support. While being outrageously busy, they took time to review and edit. The patience and encouragement they showed while supervising someone with such little knowledge were genuinely admirable. I am truly honoured to have an opportunity to work with them.

Additionally, I would like to thank Dr. Sohn's lab members. I deeply appreciate Dr. Jungwon Kang and Dr. Yeonjeong Jeong, who guided me technically in implementing and testing deep learning models. Speaking of the technical helps, Maryam Jameela, Sunghwan Yoo and Mohammad Moein Sheikholeslami must be mentioned for the support they provided for network design and brainstorming for this project. Also, I am blessed to have Dr. Yujia Zhang, Dr. Amin Alizadeh Naeini, Afnan Ahmad, Ali Hassan, Zhen Liu, Zahra Arjmandi, Muhammad Kamran, Mostafa Ahmadi and Mohammadjavad Ghorbanalivakili.

I wish to appreciate the financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) and York University.

Lastly, I would like to thank my friends, who always brought me up to my feet when I was down, and my family, who showed their unconditional love that I always admired an endless amount.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Definition
2D	2-Dimensional
3D	3-Dimensional
ALS	Airbourne Laser Scanner
AP	Average Precision
BEV	Birds-Eye-View
CHM	Canopy Height Model
CNN	Convolutional Neural Networks
DCNN	Deep Convolutional Neural Network
DBH	Diameter at Breast Height
FN	False Negatives
FP	False Positives
GPS	Global Positioning System
GE	Google Earth
GSV	Google Street View
GT	Ground Truth
ITCD	Individual Tree Crown Detection and delineation
INS	Inertial Navigation System
IoU	Intersection over Union
LiDAR	Light Detection And Ranging
MLS	Mobile Laser Systems

MLP	Multilayer Perceptron
NMS	Non Maximum Suppression
POV	Point of View
RADAR	RAdio Detection And Ranging
RoI	Regions of Interest
SLAM	Simultaneous Localization And Mapping
SOTA	State Of The Art
SGD	Stochastic Gradient Descent
TLS	Terrestrial Laser Scanners
TN	True Negative
TP	True Positive

Chapter 1.

Introduction

Mapping and inspecting infrastructures have never been simple or easy, regardless of whether natural or artificial. The traditional way of man-powered mapping and inspecting is time-consuming, laborious, and dangerous. Moreover, these man-operated results often meet the problems such as blind zones and non-quantifiable objects. Hence it is costly, and the result is not accurate. So naturally, the industry seeks a more precise, cost-effective, quantifiable automated method for these tasks.

Since the mid-20th century, the aerial photographing method has been applied to solve this expansive problem of inventory and analysis [1], [2]. However, even though these novel data sources improved the efficiency of processes, manual interpretation was still needed by human eyes, making it costly and labor-intensive.

In the forestry community, semi-to fully automating the inventorying and analyzing procedure was critical for forest management. Hence, algorithms for detecting and delineating individual tree crowns from aerial photographed imagery became necessary for the forest

inventory system for time efficiency and accuracy. Delineation of individual tree crowns soon became one of the essential parts of precision forestry for estimating its size, diameter at breast height (DBH), height, and many other aspects [3], [4]. In the community, researchers named this process differently e.g., automatic tree recognition [5], stem number estimation [6], single-tree detection [7] and many others. In this paper, these processes of detecting individual trees are referred to as Individual Tree Crown Detection and delineation (ITCD).

As mentioned above, the ITCD was first based on aerial imagery, moved to the satellite collected images, and then recently, the trend became an integration of different data collection sources [8]. The researchers considered the aerial images the essential data source for ITCD because of their high spatial resolution necessary for detecting individual tree crowns. Soon later the more enhanced satellite imagery from QuckBird or IKONOS was used for ITCD because the resolutions were much improved from the previous satellite images [9], [10]. These imagery data sources are often considered passive.

The terms passive and active sensors are mainly used to distinguish different modes. Active sensing means that the system emits energy and receives parts of the energy reflected from the environment [11] It can also describe active control of movement in vision [12]. In this paper, the former definition of active sensing is used.

Passive remotely sensed data sources have a limitation of dimensions. The target objects, trees, were not fully represented in a 2-Dimensional (2D) imagery. To overcome this, alternative active remote sensing devices such as RAdio Detection And Ranging (RADAR) and Light Detection And Ranging (LiDAR) were suggested. Passive sensed database such as satellite or street imagery only supports 2D information. RGB-D sensors are seldomly used to collect the depth information making the database 3D, but the amount of information it collects cannot be

compared with the active sensors such as RADAR and LiDAR, because such sensors collect x, y and z axis values, GPS locations, RGB values, intensity and many more information of each point collected. Hence, the active sensors brought a new dimension of height to ITCD research. Around the 2000s, [13] presented the usage of active sensors for data collection of individual tree inventory systems Among active sensors, the LiDAR sensor gained popularity due to its efficiency in local-scale inventories [14].

Among different sensors using LiDAR such as Terrestrial Laser Scanners (TLS), and Mobile Laser systems (MLS), due to its high spatial resolution, an Airbourne Laser scanner (ALS) is used as a dominant remote sensing device for 3-Dimensional (3D) surveying and ITCD studies [15], [16]. Then, the utilization of these LiDAR data sources from ALS for urban and forest ITCD in a 3D way with point clouds was proposed to the community.[17]. With the ALS approach, the LiDAR system has been heavily used in mapping scenes in both urban and forest areas.

1.1 Problem domain

Recently, the detection of an individual tree became achievable with novel technologies such as 3D surveying systems with LiDAR sensors and cameras paired with Inertial Navigation System (INS) and Global Positioning System (GPS) devices. Also, computer vision, remote sensing and machine learning technologies have rapidly advanced over the last two decades. This achievement often demonstrates the enormous potential of the image processing technologies leading to the success of detecting and classifying tree instances from an imagery and point cloud acquired by various sensors.

Over the last decade, we have observed considerable successes in Deep Convolutional Neural Networks (DCNN), especially in visual data analytics, which vastly outperforms shallow or non-learnable vision methods [18]. This DCNN architecture makes the problem of learning representation easier and more robust. However, DCNNs require a much larger scale of data to learn a few million parameters for generating accurate probabilities. In this context, many researchers have made extensive efforts to provide various datasets obtained with LiDAR sensors supporting the advancement of DCNN research. These benchmarks include: KITTI [19], H3D [20], Waymo Open [21], and nuScenes [22] for the 3D object detection task and Semantic3D [23], Paris-Lille-3D [24], Semantic KITTI [25], and Toronto3D [26] for the 3D segmentation task. Most of the existing benchmarks have been acquired by MLS for 3D object detection, segmentation, depth estimation and Simultaneous Localization And Mapping (SLAM) purposes. The MLS benchmarks provide a wide horizontal and vertical view suitable for autonomous driving applications.

Airborne Lidar Sensor (ALS) is preferred for environmental applications such as forest inventory management due to greater coverage areas with a Birds-Eye-View (BEV). Compared to MLS, finding a publicly available ALS benchmark or existing tree benchmark built based on ALS that focuses on forest trees is not easy. There is a lack of ALS-based tree benchmarks in an urban setting. Forest trees develop differently than urban trees. Numerous urban trees are usually planted and situated as part of urban planning concepts, whereas forest trees grow naturally. Thus, there is a demand for accessing an urban tree benchmark processed by ALS. Not only the datasets but the algorithms also are heavily optimized towards such objects related to the autonomous driving application and its datasets.

Hence, there are the following voids of 3D object detection study: lack of 3D LiDAR dataset collected by ALS with tree-related annotations; lack of testing results with the existing detection algorithms with such dataset; lack of algorithms optimized for individual tree detections.

1.2 Research objectives

The biased trend of 3D object detection algorithms makes them fundamentally incompatible with objects like trees with many different characteristics than cars, trucks, and other traffic objects. Such objects have tendency of regularized sizing, shape and not superimposable. The simplest example would be a passenger in a moving vehicle, the passenger would not be detected as an individual human object but included as a part of a vehicle [19].

Therefore, this research project's general goal is to develop a novel DCNN for detecting individual tree objects or improve the performance of the state-of-the-art (SOTA) 3D object detection network for individual trees. To achieve such goal, the focus is narrowed down and explicit to two main objectives. First is to generate the compatible dataset with analyzing its in-depth testing results with existing detection algorithms. The second is to implement methods that can increase the accuracy with possible further suggestions for future works.

Figure 1: Examples of masking errors in instance segmentation with Mask R-CNN. Most common errors include the cases as shown above. The similar issues often occur in the 3D domain because of the nature of the individual tree detection. In the 3D domain, the case of (a) and (e) are predominant issue, while the ALS collected point clouds often do not contain the RGB values, which is the effective information to separate occluded tree, to differentiate colors of each tree [27], [28].

This project's research aims to create and implement a deep learning system for detecting and classifying solitary trees in urban/semi-urban areas using Lidar data [29]. displays mistake examples from the instance segmentation. The examples in Figure 1 were generated using the Mask R-CNN network [27], a well-known and widely used system. Even though these examples pertain to the segmentation of 2D instances, the fault kinds also apply to the 3D environment. Similar issues occur, such as the frequent union of two or more trees or the growth of a little tree beneath the canopies of more giant trees. Consequently, in numerous research or datasets, trees are frequently categorized as vegetation or not included at all [19], [30]–[32]. Consequently, I

believe that a specific network may detect individual trees from a scene in which they are obscured instead of grouping them together. It will also function more successfully with other infrastructure objects like poles, pylons, pipes, and railroad tracks. Also, the detection algorithm can be elaborated with more various applications such as smart city, specifically for parking management or safety and security camera managements that must solve tree overlapping problems.

1.3 Research contributions

This research outcome mainly contributes to the investigation of 3D object detection of individual tree objects and the generation of the related dataset in the following collaborations and publications:

- Ko, C., Y. Jeong, H. Lee and G. Sohn. 2022. YUTO Tree-5000 Benchmark. 43rd Canadian Symposium on Remote Sensing, 10th International Conference on Agro-Geoinformatics, July 11 – 14, hybrid, Quebec City, Quebec, Canada.
- York University Teledyne Optech (YUTO) Tree-5000: A Large-scale Airborne LiDAR dataset includes 5717 trees as each individual object with its 3D bounding box information alongside with other specifications of each, such as DBH, latitude, status and more.
- Conducted an extensive investigation and comparative studies of the SOTA networks, including single-stage and two-stage DCNNs.
- Proposed a novel loss function, called Volumetric loss function. This method was designed to enable a regression algorithm to expand and shrink the set anchor boxes more effectively and efficiently by adding features of the learnable weight for improving the Average Precision (AP) performance up to 3% in both Birds-Eye-View (BEV) and 3D.

The outcome of this research project was published in International Conference on Pattern Recognition (ICPR) 2022 and Canadian Symposium on Remote Sensing (CSRS) 2022 [33]. YUTO Tree 5000 will be available to the public.

1.4 Thesis outline

This Chapter introduces the research project with the motivation to generate an appropriate dataset for 3D object detection, while individual trees as its designated object, test the dataset with state-of-the-art algorithms and design a better method to increase the accuracy. The outline of this thesis for the rest of the document is organized as follows:

- Chapter 2 introduces the existing trees or forestry datasets and the detection algorithms. It focuses on the need and trends of the recent studies conducted for the individual tree detections and regarding other elements such as datasets and detection algorithms, while listing comprehensive literature reviews of the subject.
- Chapter 3 highlights the newly built dataset for the individual tree detection algorithms. The list of existing 3D LiDAR datasets is overlayed and discusses why they were unsuitable for individual tree detection studies. Hence, there has been a great need for the proper dataset, and we built a dataset for such studies called YUTO Tree-5000. The chapter will explain how and why this specific dataset was created extensively and compare it with the existing datasets.
- Chapter 4 studies the utilization of the YUTO Tree-5000 dataset with various detection algorithms. There are countless deep learning-based detection algorithms with drastically different approaches. The chapter will present the results from each detection algorithm. It confers the configurations and modified frameworks of the algorithms and reasoning based on the ablation studies and suggests methods that can improve the results.
- Chapter 5 features a novel module design of a detection algorithm using an adaptive volumetric approach explicitly built for individual tree detection algorithms. The chapter defines why a proper utilization of anchor boxes is critical to specific detection networks

and why utilizing a well-designed anchor box size is the optimal way for the given structures of the networks. It provides the implementation of the proposed module with experiments and results of the selected 3D detection algorithm.

- Finally, Chapter 6 addresses the conclusion of this research study. It features the main contributions of this study while highlighting the limitations of the implemented module and recommends possible future extensions.

Chapter 2.

Related works

This section will discuss existing tree/forestry datasets, the SOTA DCNNs for detecting tree/forestry, and 3D objects.

2.1 Existing tree and forestry datasets

In recent years, the use of Convolutional Neural Networks (CNN) for object detection in both the 2D and 3D domains has achieved considerable success. CNN's object detection approaches surpassed other shallow or non-learnable vision algorithms in several respects, including processing speed and accuracy. Furthermore, CNN demonstrated its advantage over previous methods by estimating probabilities by linearly mixing features evaluated by densely layered convolutional layers.

In this part, several articles pertaining to benchmark projects utilizing LiDAR datasets in the forestry sector will be discussed. The European Spatial Data Research Organization (EuroSDR) and the International Society of Photogrammetry and Remote Sensing (ISPRS)

Commission II launched the "Tree Extraction" (2005–2008) [30], [31] as the first initiative in this field. This initiative involved twelve participants from the United States, Canada, Norway, Sweden, Finland, Germany, Austria, Switzerland, Italy, Poland, and Taiwan. This project's objective was to create a dataset by evaluating the quality, accuracy, and practicability of using ALS to extract the following information about individual trees: Tree position (x and y coordinates of the trunk), tree crown delineation, tree height, and crown base height were selected as the criteria (or volume of the tree crown).

Then, between 2011 and 2014, the NEWFOR (New Technologies for a better mountain FORest timber mobilization) initiative benchmarked eight ALS-based single-tree detection algorithms [34]. The primary objective of the NEWFOR project was to create a sustainable and adaptable management system for mountain forest resources. For this project, eight locations were examined. The ALS LiDAR data was collected between 2005 and 2013, with point densities ranging from 5 to 121 points (pts) per square meter. Four sensors were utilized for this project: Riegl LMS-Q560, Riegl LMS-Q680i, Optech ALTM 3100, and Leica ALS 70. The range of its research area was between 0.3 hectare and 1.2 ha. The study found a link between autonomous tree detection algorithms and forest inventory information. Finally, University of Eastern Finland researchers [32] conducted a benchmarking study to compare the performance of tree-detecting algorithms. Five ALS datasets were collected between 2006 and 2008, with point densities ranging from 1.5 to 30 pts / m2. The study sites ranged in size from 0.05 to 0.64 hectares. This study utilized the Optech ALTM 3100, TopoSys Harrier 56, Toposys Falcon II, and TopEye MKII ALS scanners. In this study, six tree detection techniques were evaluated. The project concluded that forest architecture, tree density, and clustering affected the algorithmic performance.

EuroSDR and the Finnish Geospatial Research Institute initiated the international TLS benchmarking project in 2014. This project's objective was to examine and contrast the strengths and shortcomings of the TLS domain, which is vastly distinct from the ALS domain, for extracting tree properties [35]. With 24 test plots, 18 methods were evaluated. The DeepForest [36] benchmark dataset for tree detection algorithms was released in 2020. This RGB, LiDAR, and hyperspectral dataset was used to detect trees in 2D image space using BEV Point of View (POV). The point density of the LiDAR dataset was roughly 5 points per square meter. For network training, 2D bounding boxes and RGB photos were utilized. A benchmark evaluation was conducted using the Deepforest Python module created by its authors [36].

Recent research by Schmohl et al. utilized 10,864 manually labelled ground truth trees to test a network for individual tree recognition. This study provided cylindrical shapes for tree annotation and prediction and found that the circular (2D) Mean Average Precision (mAP) was 0.5-0.1% greater [37]. The point density was recorded for the first return as 16.6 points per square meter. In addition, the authors decided to omit the height and z coordinates from their loss functions; hence, most of the prediction and assessment were conducted in 2D. In addition, the integration of segmentation and detection findings was accomplished by extensive postprocessing outside of the DCNN. Therefore, training and testing of the dataset could not be completed end-to-end if the dataset were made available to the community. Unfortunately, the authors did not indicate whether this dataset would be available. Their conclusion was that "the DCNNs trained on this ground truth would not necessarily learn to identify particular trees or tree points accurately but would rather mimic the training data" [37].

2.2 DCNN for 3D object detection

There were countless attempts to build a tree detection system, as was explained in the introduction. Before the significant utilization attempts with DCNN, mathematical algorithms were the most popular and efficient way of executing ITCD. Among various methods, the watershed algorithm [38] is considered the most effective and efficient for image segmentation for tree crown delineation because of its high edge delineation accuracy and efficiency [39]. For example, Huang et al. [40] proposed a Marker-Controlled Watershed Segmentation Algorithm with the bias field image as a substitute for the original RGB image in ITCD utilizing the smooth varying feature in the bias field and achieved a great result. Their work addresses the critical problem of occlusion, which is one of the biggest challenges in tree detection and is inspired by the occlusion-aware R-CNN proposed by Zhang et al. [41]. Lumniz et al. [28] used street-level imagery and Mask R-CNN for tree detection, and tree location was estimated from depth. These related works are based on 2D object detection. In contrast, our work is focused on using ALS and 3D object detection to understand the objects' real-world sizes and positions. In this section, it will be focused that DCNN as an object detecting method, but not other methods such as watershed algorithms, etc. The general knowledge upon DCNN for object detection and categorized structures of it will be discussed in the following.

2.2.1 3D object detection

Object detection is one of the tasks that are related to information inferencing from the given domain. The domain can be 2D imagery, 3D points clouds which be from various of sensors such as Sound Navigation and Ranging (SONAR), RAdio Detection And Ranging (RADAR) or as in this paper, LiDAR. Unfortunately, there is not yet a predominantly accepted definition of object detection. However, it is loosely defined as the task of finding the location all the instances of a selected object from the given source data. The localization be defined as the

center of the object image, as a bounding box containing the object, or other shapes such as cylinders [37].

The center of the object is almost always defined according to the given domain, in this case, in 3D. It is respect to the descriptions such as its position, scale, etc. Hence to illustrate this in more mathematical manner, let S be a scene from the given dataset while O(S) be the set of N_S^* object descriptions.

Equation 1. Object description in the given scenes

$$O(S) = \left\{(Y_1^*, Z_1^*), \dots, (Y_i^*, Z_i^*), \dots, \left(Y_{N_i^*}^*, Z_{N_i^*}^*\right)\right\}$$

As in Equation 1, $Y_i^* \in \Upsilon$ denotes the category of i[th] object. Υ denotes for the set of included categories. In this paper, there is only one selected category, which is tree. Hence, this denotation may be seemed irrelevant. However, for the future works, the specification of each tree in the dataset would be used for classification of each tree. This specification information can be considered as categories. Therefore, it should be still considered, nevertheless. $Z_{N_i^*}^*$ denotes the location, scale, and geometry. The position of the center of the object would be $(x_{center}, y_{center}, z_{center}) \in R^3$. The notation for the bounding of this object would be $(x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}) \in R^6$.

In this paper, the 3D object detection with bounding box labeling is utilized, hence, these notations would be valid to describe. However, in other cases of different domains and encompassing the object, different ways of denotation would be required. With these denotations established, object detection can be defined as the follow.

Equation 2. Object detection

$$\text{Detection}(S,\lambda) = \left\{(Y_1, Z_1), \ldots, (Y_i, Z_i), \ldots, \left(Y_{N_i(\lambda)}, Z_{N_i(\lambda)}\right)\right\}$$

The operation point λ allows to manage the errors of false alarms and missed detections. There are widely accepted terms to describe these detection results which are: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). Naturally, false alarms of detections are referred to as FP and missed detections referred to as FN.

2.2.2 One-stage vs two-stage

There are two main streams of 3D object detection can be divided into one-stage approaches and two-stage approaches. A DCNN with a single-stage strategy [42]–[44] transforms the 3D point cloud into more compact representations, such as voxels or BEV pictures. It then employs CNN directly to create predictions, in this instance bounding boxes, in a fully convolutional fashion as can be seen in Figure 2. Now, this uncomplicated approach makes the network straightforward and cost-effective. However, its cost is borne by the resulting pocketbook. As indicated, the operation would utilize the gradually downscaled feature map. It eventually loses spatial resolution, rendering it incapable of analyzing point cloud data structure information. The one-stage detector for processing the sparse point cloud is less precise than its counterpart.

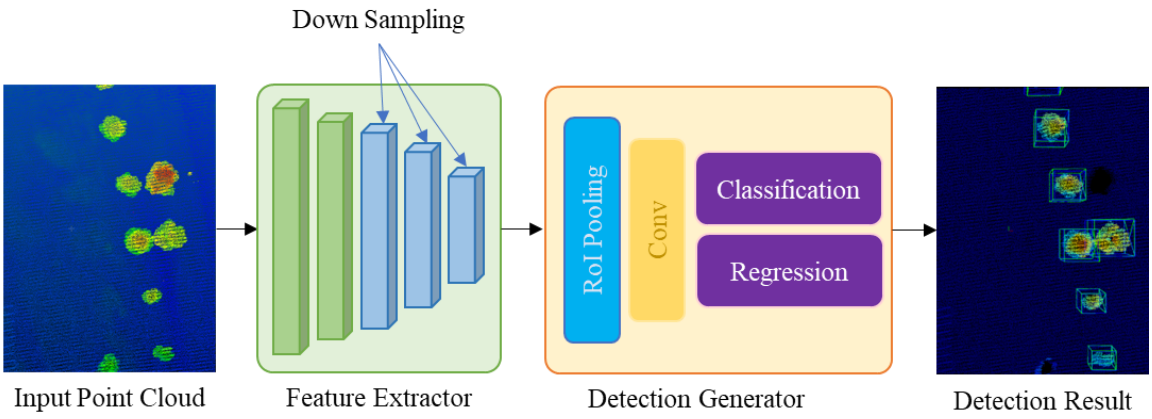


Figure 2. One-stage approach algorithms structure design.

As its name suggests, the two-stage technique of making predictions involves more steps than the one-stage approach [45]–[50], which are: region proposal and classification and localization. This region proposal step presents the classifier with class-agnostic boxes that locates the ground truth. This means that it only detects the foreground objects. Again, whilst the term foreground is not industry-defined due to the novelty of the area of this study, it is often defined as a set that contains all specific desired classes in a scene [51]. For this procedure, different approach can be applied to achieve these proposals. For example selective search methods are used in R-CNN and Fast R-CNN, or Region Proposal Network (RPN) method was used in Faster R-CNN [52]–[54]. By separating this detection procedure into two, the second stage would operate on the crops made from the pooled results of the first stage of selective search (Fast R-CNN) or RPN (Faster R-CNN). The reasons why to make the algorithm with two steps is because of the limitation of the receptive field. In the case of Faster R-CNN, the input is transformed into a feature map with limited spatial resolution through a CNN. For each feature map, the RPN head estimates if the features correspond to an object and the head regress the detection box, and this box regression process is based on the final feature map. Since RPN created a rough estimate of the bounding box, the second stage of Faster RCNN can utilize all

features contained in the predicted bounding box and can correct the estimate. However, inevitably, there would be many inference steps per scene. Therefore, performance is not as fast as one-stage detection algorithm.

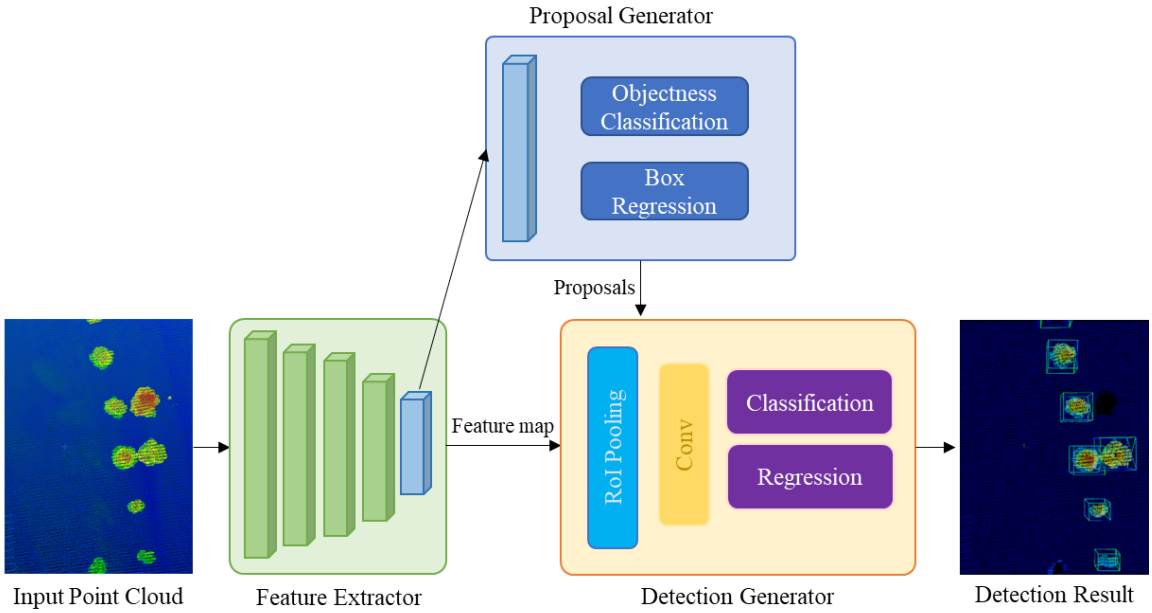


Figure 3. Two-stage approach algorithms structure design.

2.2.3 Anchor-based vs anchor-free

Faster R-CNN [54] first presented the concept of use an anchor for 3D object detection. Since then, numerous object detectors have improved their networks' precision by employing this anchoring concept. Nevertheless, the regression head which performs localization that is fed with the feature map, can only process one known item at a time. Therefore, there may be interference with the regression if more than one object is present because every object affects the regression head. This issue was resolved with the use of regression head per item. However, the simultaneous presence of many objects of the same class remains problematic.

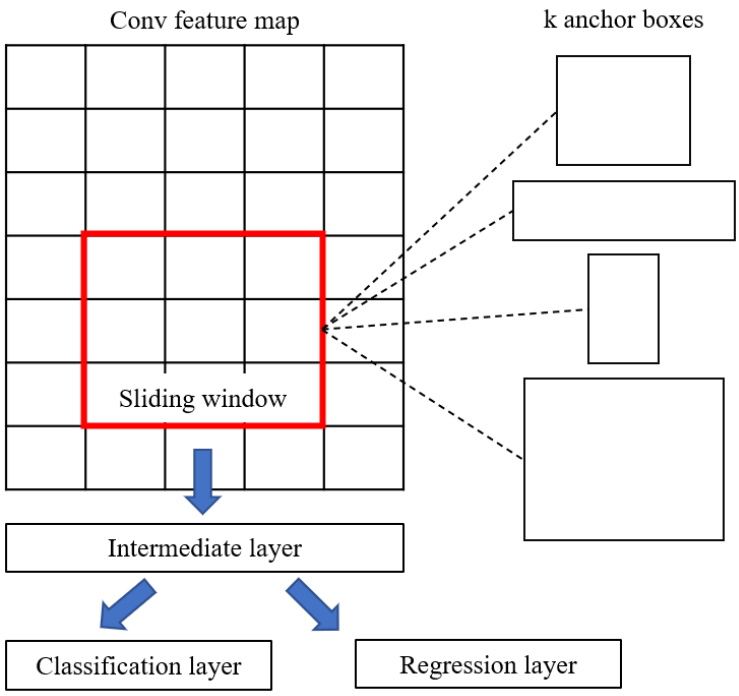


Figure 4. Anchor function example. k anchors are set for each spatial location of the final feature maps. Classification score for each class including background is predicted for each anchor and regression coordinates are predicted for anchors that has greater IoU than the predefined threshold. First introduced in [55].

To circumvent interference, the network constrains each regression head to a specific portion of the scene. However, it does not completely eliminate the interference between the items that are extremely close together. Therefore, it does not function when two objects of the same class overlap. The anchor box is consequently introduced to confine each regression head. When the ground truth of an item has a high Intersection over Union (IoU) value with a particular anchor, the regression head uses this anchor to predict the final bounding box. Then, each regression head would be spatially independent and interfere with the others to a minimal degree.

Consequently, these anchors function as references and spatial restrictions that restrict the geographic extent of the regression head. Then, regions outside of this anchor would have

minimal or no effect on the regression procedure. Since the domain is moved from 2D to 3D, anchors include the z-axis values. Anchors used in a 3D domain do not always take boxes' shapes. For example, the proposal generation module of STD [49] generates proposals from point-based spherical anchors instead of anchor boxes. The authors claim that the process reduces the number of anchors.

Occasionally, networks do not use anchor generation to gain efficiency, as anchor approaches invariably incur large memory and computational costs for improved recall performance. In this regard, a DCNN for 3D object detection without using the anchors is called an anchor-free approach. Instead, a foreground segmentation network was used for PointRCNN [50]. Point RCNN was used as the anchor-free network for this study. It uses 3D bounding boxes to build a ground-truth segmentation mask. In the first stage, foreground points are segmented, and a small number of bounding box proposals are generated simultaneously from the segmented points. This solution avoids requiring a considerable number of 3D anchor boxes throughout the entire 3D space, as prior systems did, reducing the necessary amount of computing. And all networks other than Point RCNN embraced the anchor-based approach.

The PIXOR [43] technique assigns all pixels inside ground truth bounding boxes as positive samples in the BEV feature map, despite the fact that it has not been validated with our in-house dataset in this research. Other range-view-based approaches designate as positive samples pixels on range-view maps that are contained within 3D ground truth boxes. Layers Rangedet [56], LaserNet [57], and Range-Conditioned Dilation RCD [58]. Other anchor-free detection networks define the detection problem as a key point detection problem, as illustrated by the following networks: AFDet [59]; RSN [60]; CenterPoint++ [61]. In these instances, the

concept of centeredness is employed to obtain the relevant properties; thus, anchors can be deprecated.

2.2.4 Points representation

For 3D object recognition using DCNN, numerous approaches of encoding point clouds have been developed. Frameworks with voxel-based and point-based representations are two of the most prevalent forms. However, how to represent LiDAR-acquired 3D data, which consists primarily of point clouds, is up to the user. Most existing 3D object detection networks mainly analyze point clouds as voxel grids, point clouds, graphs, or 2D images, or a combination of these.

CNNs are predominantly applied to data with regular structures, such as the 2D pixel array, in the past. To apply CNNs to unordered 3D point cloud data, the data must be partitioned/processed into regular grids of a specific size that describe the data distribution in 3D space. Typically, the size of the grid is proportional to the data's resolution. By identifying the occupied voxels as visible, occluded, or self-occluded, voxel-based representation can encode 3D shape and viewpoint information. In addition, 3D convolution and pooling techniques can be applied directly to voxel grids.

Since numerous 3D object detectors were constructed using the voxel-based framework, its limitations were discovered. First, not all voxel representations are usable since they contain both occupied and unoccupied scanning environment regions. In light of this inefficient data representation, the significant need for computer storage is unnecessary. Second, it is difficult to establish the size of the grid, which impacts the scale of input data and may disrupt the spatial link between points. Thirdly, computational and memory demands increase quadratically with

resolution. Existing voxel-based models are therefore kept at low 3D resolutions, and the most common grid size is 303.

Point cloud data, as opposed to volumetric 3D data representation, can preserve 3D geospatial information and internal local structure. In addition, the local receptive fields confine the voxel-based models that scan the environment with fixed strides. For point clouds, however, the input data and the measure determine the range.

The primary focus of point cloud-based deep learning models is the resolution of permutation problems. Although they treat points independently at local scales in order to preserve permutation invariance. This independence disregards the geometric relationships between points and their neighbors, exposing a fundamental constraint that leads to the absence of local features.

In general, voxel-based algorithms rasterize unprocessed points into voxel-grids. Then, 2D and 3D CNN are utilized to produce 3D suggestions, such as VoxelNet [44] and SECOND [62]. In general, point-based algorithms accept points as input. Then, set abstraction is used to obtain the point characteristics for box prediction, such as Pointnet [63].

2.3 Summary

In this chapter, the related works were presented. Firstly, the existing tree and forestry datasets were introduced. As it was navigated in chapter 2.1, it was evident that there is no suitable enough dataset for this specific study. Therefore, it was proved that building a custom dataset, which suite the objectives of this study, is necessary. Secondly, 3D object detection methods, especially DCNN related, were explored. The term of object detection was defined to be used in the further research and types of DCNN related object detection structures were

presented. It was essential to review such materials not only because it gave the ideas of which types of models should be tested with the custom dataset we would build, and also what structure would be optimal to be utilized for individual single tree detection.

Chapter 3.

Generating 3D Single-tree Detection Dataset: YUTO Tree-5000

Providing accurate and efficient scene perceptions of environments that suits the purpose of the application are crucial, regardless of types of applications, including mapping of area, autonomous driving, or 3D model reconstruction. Especially, DCNNs require a much larger scale of data to learn a few million parameters for generating accurate probabilities. Hence inevitably, there has been a great need for the proper datasets.

Numerous researchers have exerted tremendous effort in this setting to provide diverse datasets collected with LiDAR sensors to promote DCNN research. Among these benchmarks is the data set, which was covered in Chapter 2.1. In addition, the heightened interest in autonomous driving applications necessitated an extensive data set. Therefore, many of the existing standards for 3D object detection, segmentation, depth estimation, and SLAM have been acquired via MLS.

The MLS benchmarks offer an expansive horizontal and vertical field of view suited for autonomous driving applications. For environmental applications like forest inventory monitoring, ALS is favored over BEV because to BEV's larger coverage areas. Compared to MLS, it is challenging to locate a publicly accessible ALS benchmark, or an existing tree benchmark based on ALS that focuses on forest trees. In an urban area, ALS-based tree benchmarks are lacking. Urban trees grow differently than forest trees. Consequently, there is an intense desire for access to an urban tree benchmark processed by ALS for this study and the discipline as a whole.

3.1 Motivation

To bridge the gap between the demand for and the availability of ALS-based tree benchmarks, this project generated a large-scale tree benchmark dataset for the community to advance DCNNs for detecting single-trees. The dataset was collected in an urban setting. In woodlots, it contains both planted trees (which often have little or no overlap) and naturally growing trees (which typically have large overlap). This article describes how we created our dataset utilizing ALS collected during the leaf-on season, fieldwork, and publicly available images for annotation such as Google Street View (GSV) and Google Earth (GE). This new dataset has the following elements:

1. Segmented ALS dataset with four manually validated classes (Figure 5a);
2. Field-collected tree data with specific information about each tree;
3. Manually adjusted GPS treetop locations (Figure 5b);
4. Semi-automatically generated boundaries of individual tree crowns (Figure 5c);
5. Generated 3D bounding boxes representing individual trees (Figure 5d).

The main contributions of this work are:

- Generate a new urban tree benchmark for 3D single-tree detection using ALS;
- Develop a unique workflow of the semi-automatic fusing tree inventory database with ALS data; and
- Conduct a comparative analysis of the SOTA DCNNs for 3D object detections utilizing our new tree benchmark.

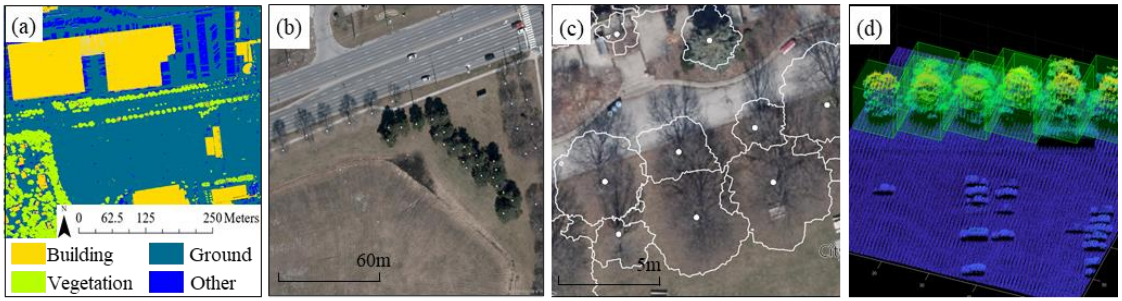


Figure 5. YUTO Tree-5000 Benchmark items. (a) an example of the classified (segmented) point cloud after human inspection, (b) white dots are the examples of tree location (c) white lines are the examples of derived tree boundaries, (d) examples of the 3D tree bounding box.

The summarization of the previously discussed existing tree-related datasets is in table 1. Large-scale LiDAR benchmark datasets do not have tree-related data, whereas existing tree benchmark datasets are primarily conducted in forest environments. Also, many of these datasets are not publicly available. Amongst these studies, point density varies from 2 to 28 pts/m^2. Unlike the work from the studies of Weinstein et al. [36], bounding boxes of YUTO Tree-5000 are 3D instead of 2D, allowing 3D detectors to be seamlessly applied. Finally, unlike the works of [37], this dataset was tested with eight SOTA 3D object detectors without postprocessing, based on only (x, y, z) as input and releasing our dataset to the public, benefiting the community. This dataset addresses tree detection in an urban context by recognizing the distinction between

urban and forest trees. Also, the point density is superior to that of other publicly accessible benchmark datasets. This benchmark dataset addresses the bottleneck of data availability for trees, which is that the tree data was not specifically for the individual trees as individual objects but mostly categorized as semantically classified areas.

Citation	Point density	Availability	Urban domain	Sensor platform	DL tested
[30], [31]	2	No	No	ALS	No
[34]	28	Yes	No	ALS	No
[32]	12	No	No	ALS	No
[35]	n/a	No	No	TLS	No
[36]	5	Yes	No	ALS	Yes
[37]	17	No	Yes	ALS	Yes
YUTO Tree-5000, 2022	**119**	Yes	**Yes**	**ALS**	Yes

Table 1. Summary of the existing tree benchmark studies. Dataset comparison between YUTO Tree-5000 and other related datasets. Point density shown in pts/m^2, availability denotes that its available for public access. Urban domain column cites that if the dataset includes scenes with roads, buildings, and other artificial structures. Deep Leaning (DL) tested column notes that if the dataset has been tested with any deep learning based object detection algorithm.

3.2 Data acquisition

YUTO Tree-5000 benchmark dataset has been derived from two primary sources: ALS data and field-collected data.

3.2.1 LiDAR Data Acquisition

For surveying to capture designated area for generating point cloud dataset, multiple options of LiDAR sensors are available. From Table 1, [30], [31] utilized Optech ALTM 2033 as their main sensor. For the cases of [32], [34], the datasets were knitted from multiple different area. Therefore, the sensors that they operated at each location were different. For [34], Riegl LMS-Q560, Riegl LMS-Q680i, Optech ALTM 3100 and Leica ALS 70 were utilized. For [32],

ALTM 3100, TopoSys Harrier 56, Toposys Falcon II and TopEye MKII were selected for their survey. [35]–[37] did not specify the choice of their sensors that they operated to gather their information.

In this paper, with the collaboration with Teledyne Optech, there were multiple sets of data collected with different types of sensors available for this study. Among all the datasets collected, Galaxy sensor based airborne LiDAR was selected. The Galaxy sensor could offer 2 MHz effective pulse repetition frequency provides on-the-ground point density and up to 8 returns per pulse provide increased vertical resolution. Both features were critical for this study to collect the point cloud data with stable point density level throughout the entire area of choice.

The acquisition of this data took place on 23rd September 2018 with ALS sensors Galaxy-PRIME, owned by Teledyne Optech, in Toronto, Ontario. The average flying height for the mission was 1829 m above ground level. York University Keele campus area (same extent as field data) covers approximately 4.3 km^2 spanning 1.8 km in the north-south direction and 2.4 km in the east-west direction. The average point density was calculated by dividing the total number of points recorded by the area of flight coverage. Due to overlap in some areas, the overall average point density is approximately 119 pts / m^2. The collected areas are shown in Figure 6 with further descriptions.

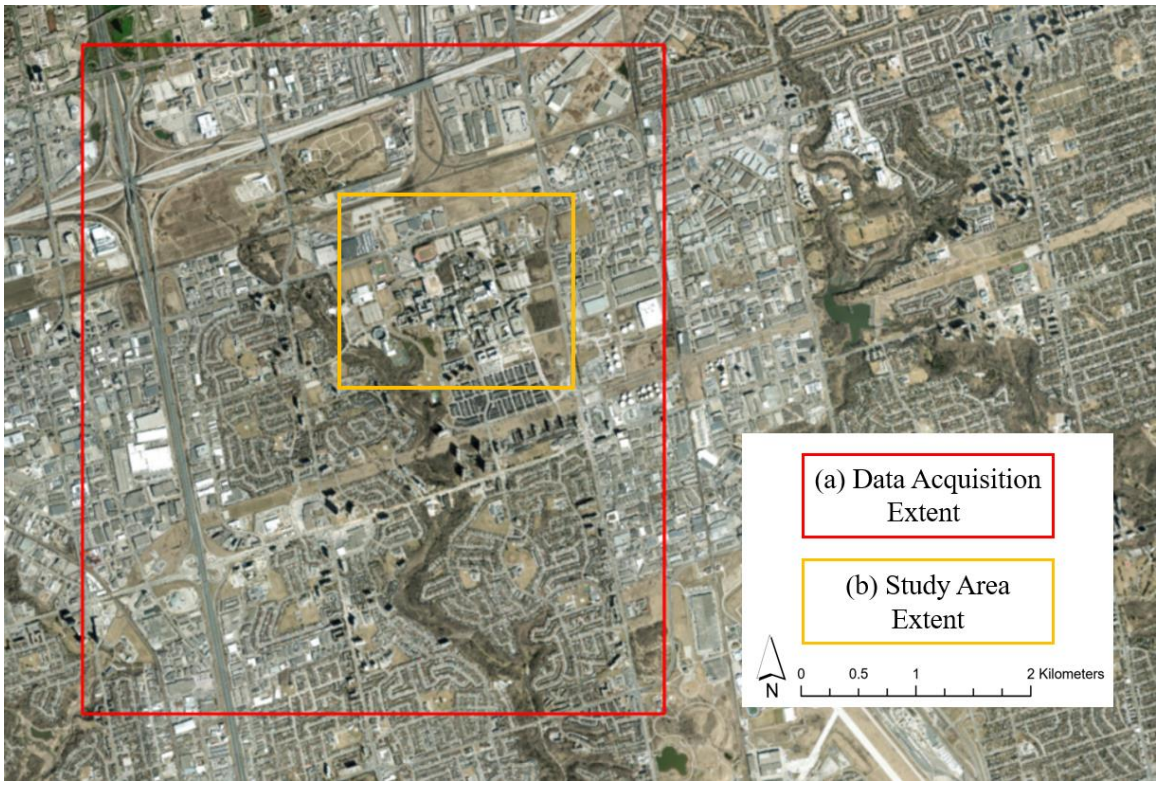


Figure 6. Lidar data acquisition map. (a) red lined box on the map is the area that was scanned with ALS devices, (b) yellow lined box is the area which that York university possessed the tree data, which resulted to become the area that this study ultimately focused on.

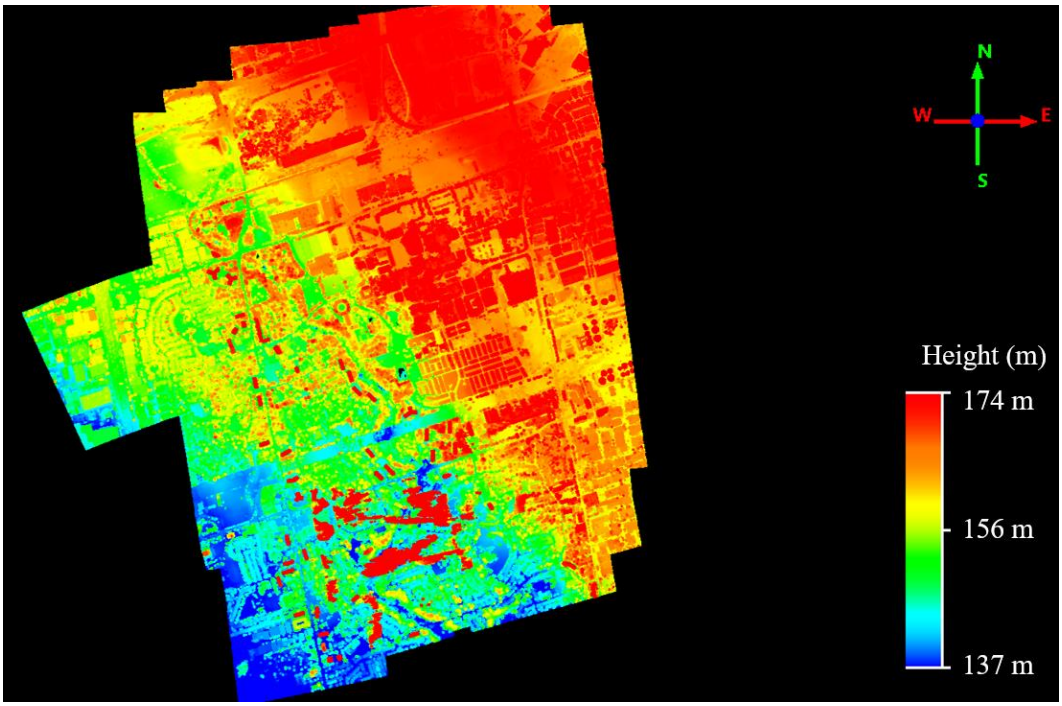


Figure 7. Height heat map of entire ALS collected area. Area of Figure 6a is shown in this figure. This visualization is 1:10 down sampled to be loaded to the machine.

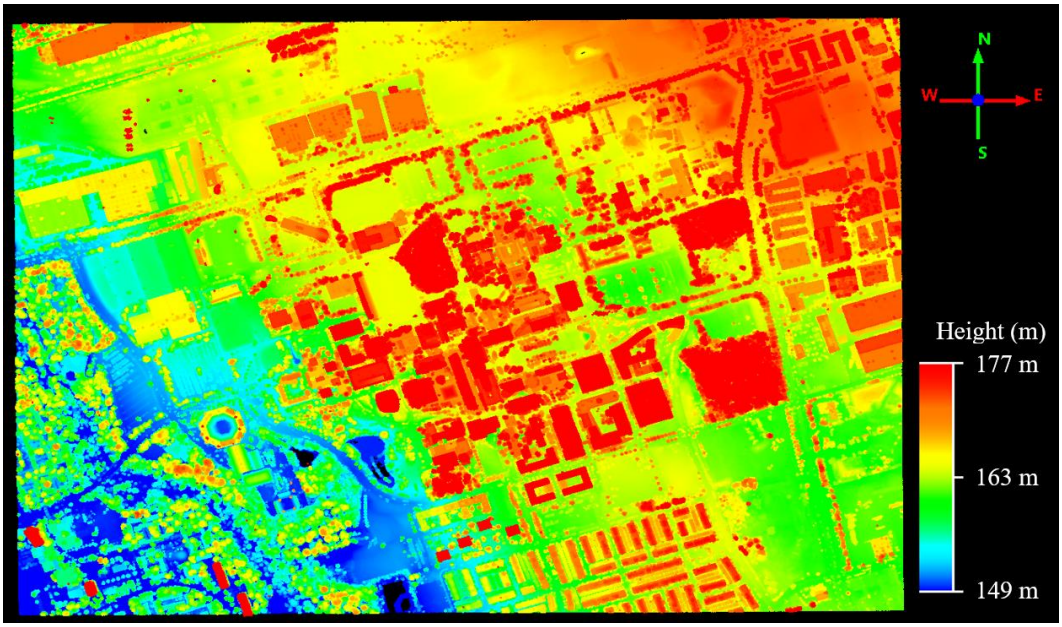


Figure 8. Height heat map of studied area. Area of Figure 6b is shown in this figure.

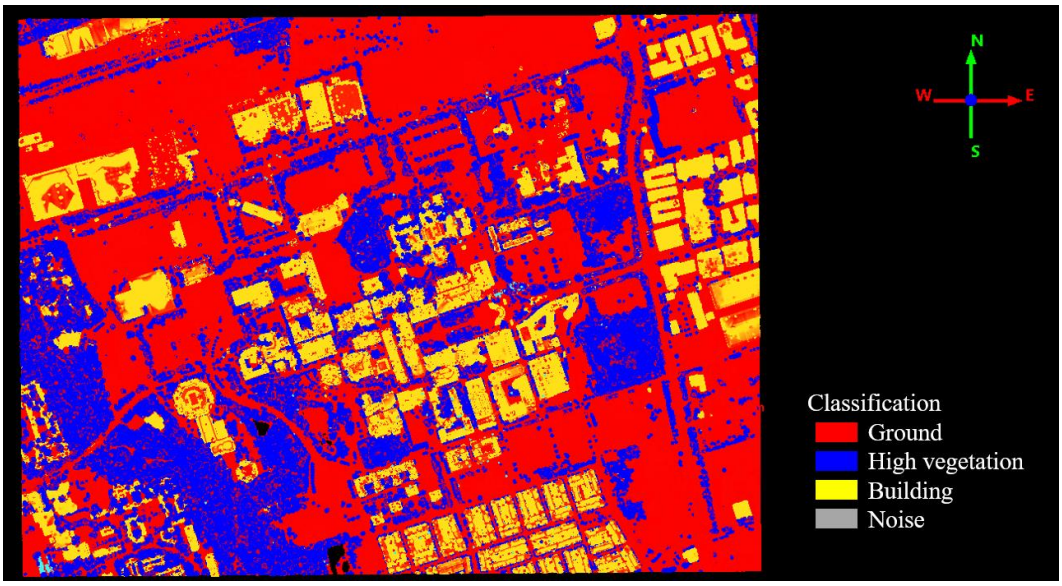


Figure 9. Classification heat map of studied area. Area of Figure 6b is shown in this figure.

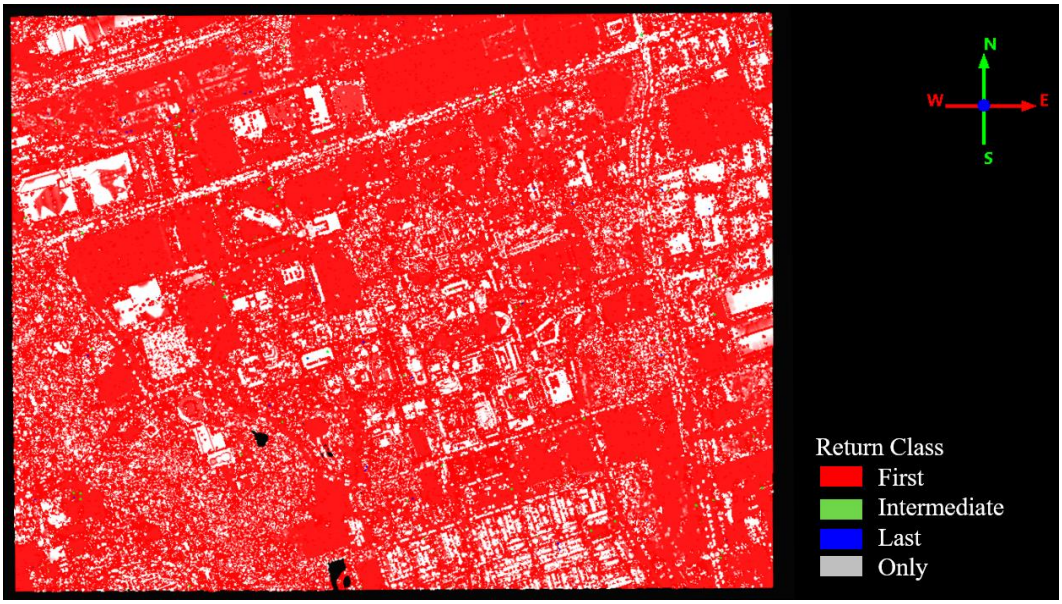


Figure 10. Return class heat map of studied area. Area of Figure 6b is shown in this figure.

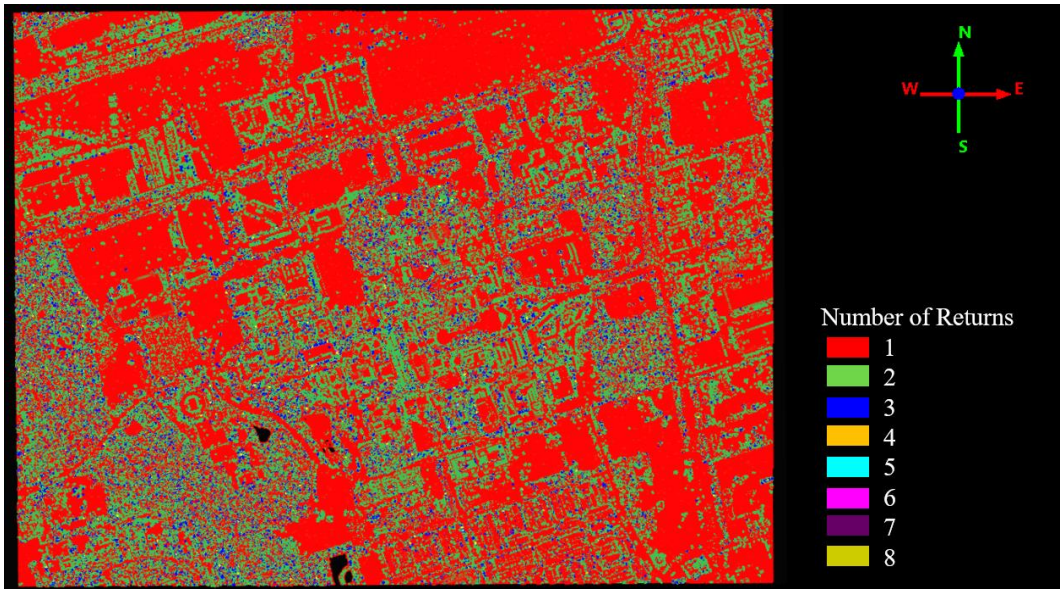


Figure 11. Number of returns heat map of studied area. Area of Figure 6b is shown in this figure.

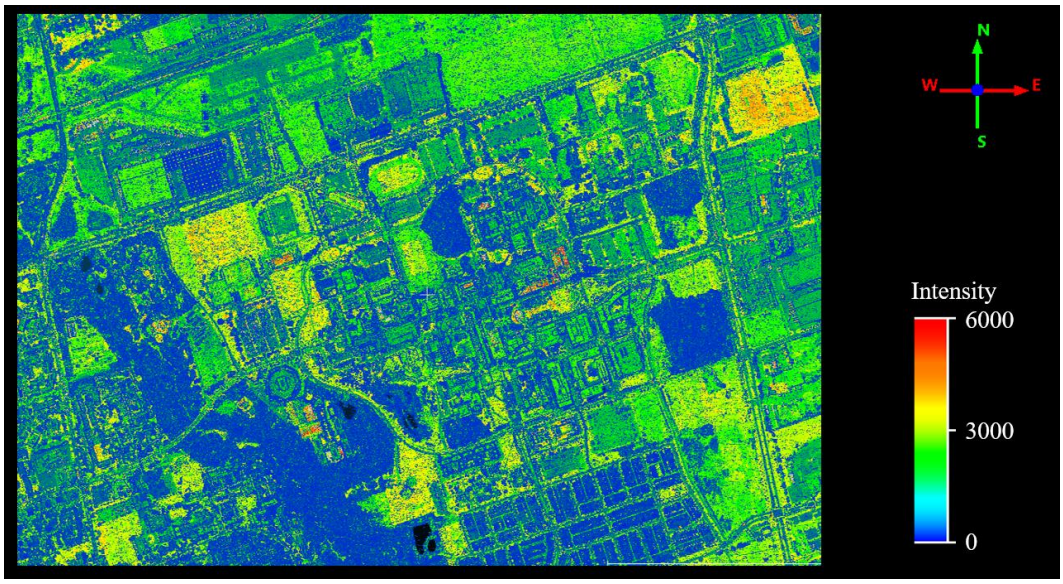


Figure 12. Intensity heat map of studied area. Area of Figure 6b is shown in this figure.

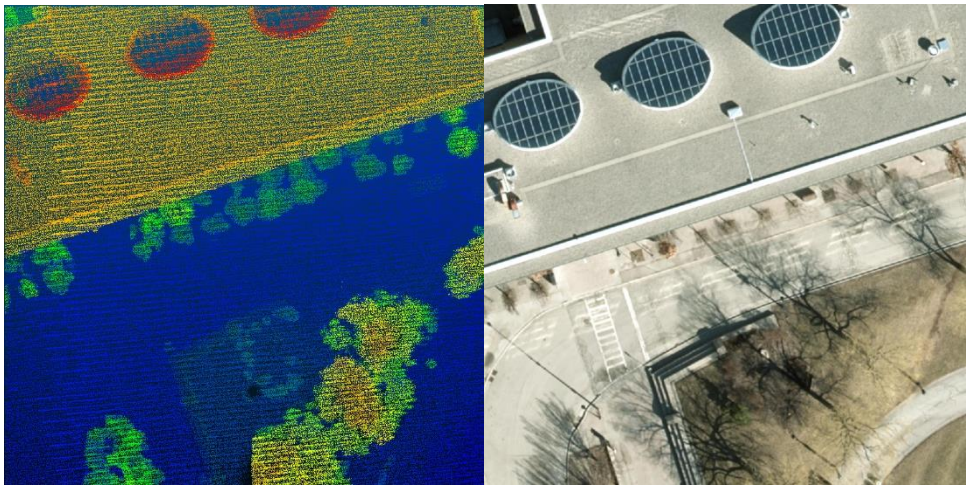


Figure 13. Lidar point cloud vs google satellite view. The resolution of the LiDAR tile is sufficient to distinguish the objects in the satellite imagery.

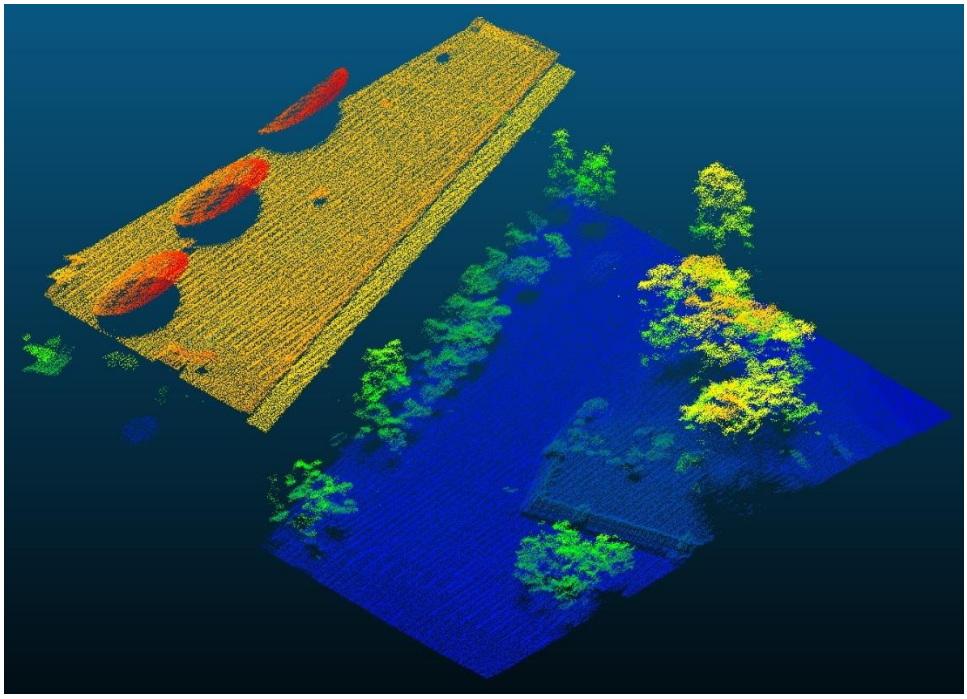


Figure 14. Front view of the example tile.

3.2.2 Field Collected Data

The data was obtained in the field in 2014, 2016 and 2017. Unfortunately, the database owned by the City of Toronto was not able to be used because of its incompleteness. Thus, fieldwork was conducted separately to generate a new single-tree inventory for this study. In the field, the location of the tree trunk was recorded with a handheld GPS. Items recorded include: 1) Common name; 2) Dbh [cm]; 3) Tree height to base of the crown [m]; 4) Total height [m]; 5) Location description; 6) Latitude, longitude; 7) Status and status date; 8) Tree crown width; 9) Ownership; 10) Tree conflicts with objects such as sidewalk, signs, trees, wires, other structures; and 11) Tree health such as broken branch, dead branches, a cavity in trunk or branches, and weak yellow foliage observed. 5717 trees with 142 species were recorded in total. Figure 15 shows the frequency distribution of the top 15 tree species and Figure 16 shows the distribution of species of labeled trees in the dataset. A summary of the statistics of the recorded trees is provided in Table 2.

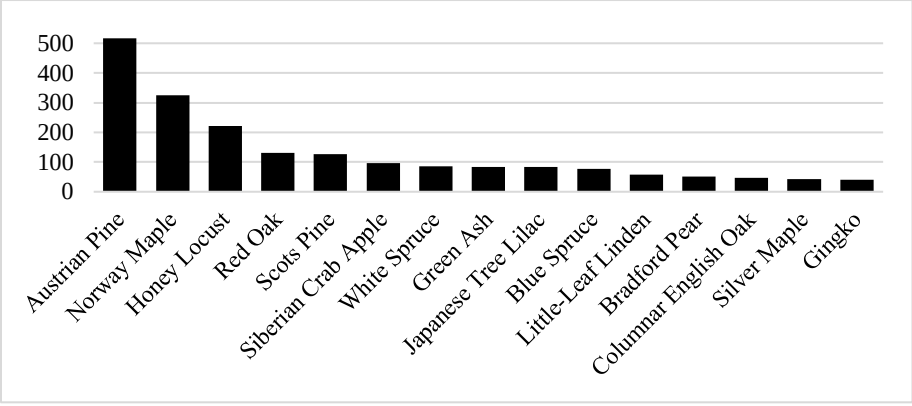


Figure 15. Frequency of the top 15 tree species. The frequency is collected from the field-collected data of York University, Toronto.

Figure 16. Species classification distribution of labeled Trees.

	Maximum	Minimum	Mean	Standard deviation
DBH [cm]	88.50	0.50	23.37	13.48
Tree Height [m]	27.97	0.07	10.35	4.30
Crown Width [m]	21.50	0.20	6.17	3.13

Table 2. Summary of the statistics of field-collected trees.

3.2.3 Data processing

As depicted in Figure 17, the procedure was developed to merge the tree inventory information with ALS tree data semi-automatically. Most of the workflow process was automated, but some components were preserved for human inspection at multiple points to correct any faults produced by automation. In the initial inspection process, following the automatic categorization of a point cloud, misclassified points were manually corrected using human eye. The second round of examination rectified mistakes in tree position produced by GPS in the field. This procedure employed both GSV and GE images. Also discovered were the trees in the tree inventory that no longer exist in the field. Consequently, they were eliminated from the dataset. The third part of the examination was to confirm that tree boundaries were

painted on the tree crowns and that the bounding box height was adequate. A final assessment of the generated 3D bounding boxes was conducted.

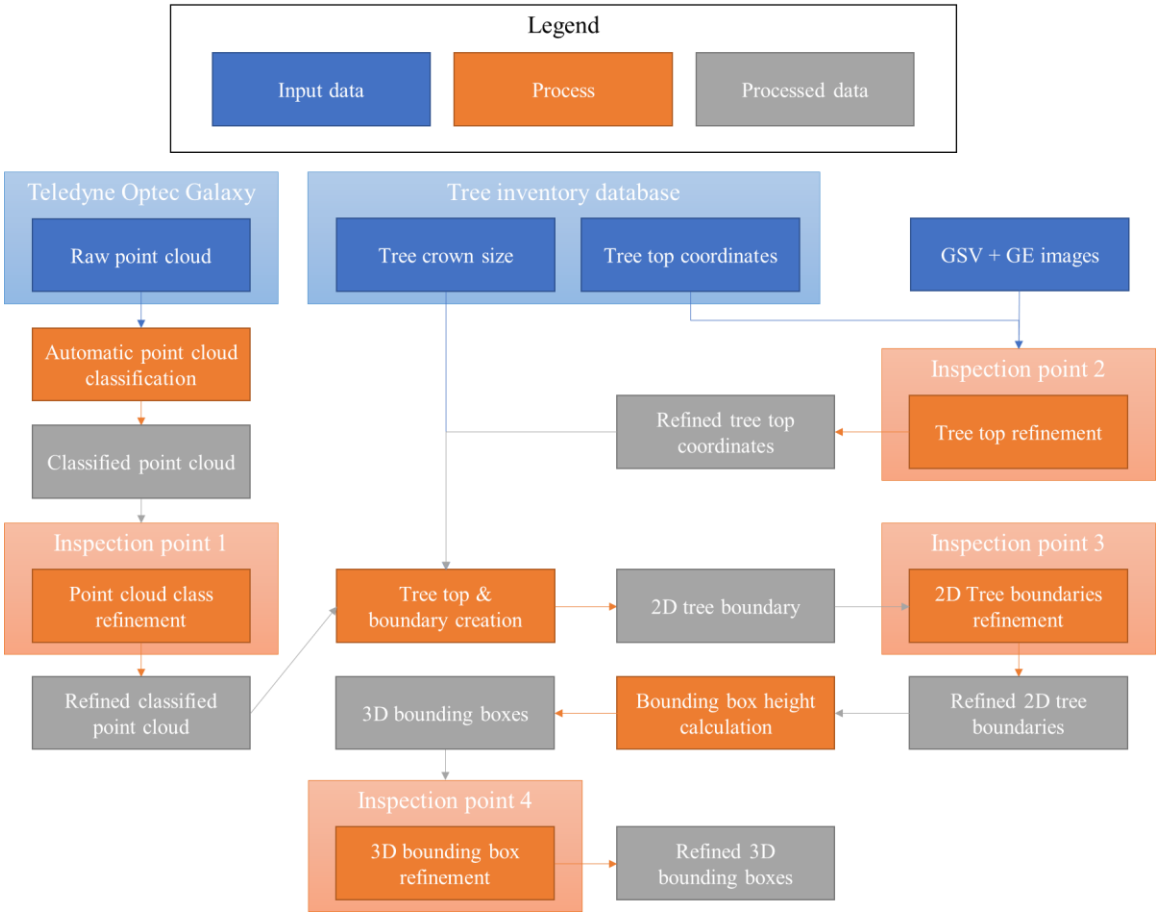


Figure 17. Workflow for semi-automatic tree benchmark creation. The processes that are marked with inspection points are the procedures that still done with the human eye inspections.

3.2.4 Point cloud classification

ALS point cloud classification (segmentation) was performed with two sets of software, LAStools [64] and TerraScan [65]. The initial automated point cloud classification was process with LAStools. Since the functions of LAStools software does not provide visualization of the

point cloud, after the automatic classification, the classification results were manually validated with TerraScan to correct all the misclassified points. The output of this process yields a four-classes dataset: ground, vegetation, building and other. The visualization of this outcome of the classified point cloud can be found in Figure 9.

3.2.5 Tree annotation with 3D bounding boxes

The semi-automatic production of tree annotations utilized:

1. ALS data
2. GPS locations
3. Tree crown widths

gathered during fieldwork. The reported field data were insufficiently precise to enable automatic tree annotation. The two most severe mistakes resulted from GPS and field measurements of tree crowns. To fix GPS errors, we carefully inspected each tree and used GSV and GE to correct the (x, y) coordinates of the GPS location to the observed treetop. We eliminated 633 out of 5,717 field-collected trees because they could not be identified in the ALS dataset during the second human assessment point of the tree top detection procedure depicted in Figure 9.

These trees may have existed during field trips but were removed before the 2018 collection of ALS data. There were 5,084 2D bounding boxes maintained in the data. The marker-controlled watershed segmentation technique [38] was applied to the ALS dataset to refine and get a more precise tree crown size. The Canopy Height Model (CHM) was initially constructed from categorized ALS data. The following stage was to apply a smoothing operator, in this case, a Gaussian smoothing, before identifying the local maximums as treetops and

outlining the tree crowns, also known as watersheds. Despite its efficiency and effectiveness, this approach relied on the magnitude of smoothing. If smoothing was too tiny, this procedure would generate a large number of false positives (FP), and if smoothing was too high, it would generate a large number of false negatives (FN). To overcome the ambiguity of smoothing, the sensitivity of local maximum detections was substantially improved. Thus, several FP would be formed for treetops, and a single tree's crown would be divided into numerous little watersheds. The minor watersheds that belonged to the same tree crown were subsequently combined. In this manner, a small watershed might be identified for multiple trees, allowing tree crown overlap detection. The watershed possibilities for a given tree crown were chosen based on the overlapping circle obtained from the field-recorded GPS location and tree crown width.

If a watershed overlapped in the area by more than 30 percent, it would be assigned to the same tree crown. This strategy permitted the identification of a small watershed by two or more trees, allowing the final tree boundary to contain overlapping portions. Finally, the width and length of the 3D bounding box were determined by subtracting the maximum x and y coordinates from the minimum x and y coordinates of the tree's boundary. For height, the maximum and minimum z-axis values of the 3D bounding box were determined using the ten biggest z values of points in the 16% center and the smallest z value of points over the 2D BEV bounding box, respectively. Figure 5d is an illustration of the tree bounding box visualization. In this final assessment stage, depicted in Figure 9 as the development of 3D bounding boxes, all bounding boxes with less than 97% overlap with another box in BEV were maintained. These highly overlapping bounding boundaries were maintained due to the close proximity of the understory and trees, and since just one tree crown was visible in the ALS data. For this version of benchmark data, 4,496 out of 5,084 trees were separated. 59% of the bounding boxes had less

than 40% overlap, 30% had 40 to 80% overlap, and 11% had greater than 80% overlap. The degree of overlap was read as the proportion of trees that were obscured by another tree. The visual examples of this different categories of overlapping are shown in Figure 18.

After this entire process, there was a human eye validation procedure to validate the overall results. All 434 tiles were visited individually, each ground truth bounding box was inspected to be checked if it has corrected annotation. During this process, minor changes made to correct the details of the annotations.

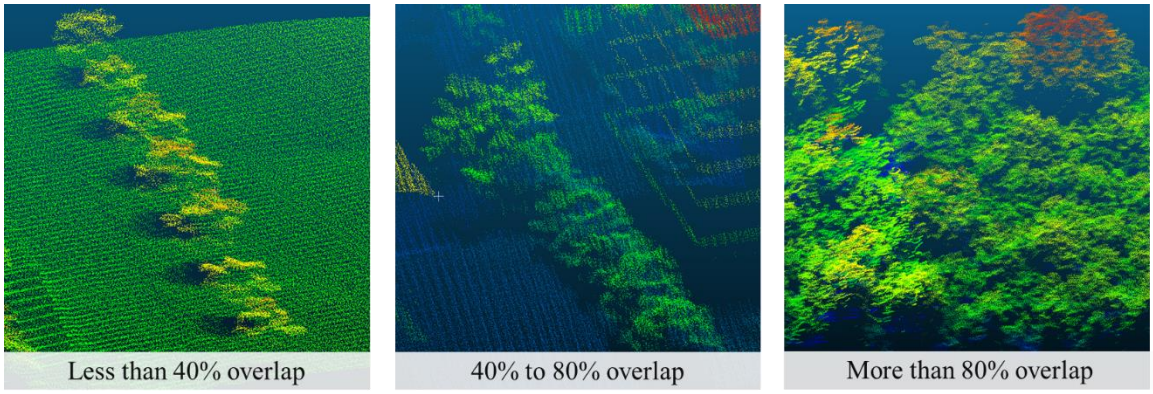


Figure 18. Examples of three categories overlaps. From the left, less than 40% overlap, 40% to 80% overlap and more than 80% overlap.

3.3 Dataset analysis

This section will describe this dataset's height, width, truncation, and occlusion features. The dataset contains 5,577 instances of 4,492 trees distributed across 434 tiles. Due to the spacing between tiles, there might be many instances of a tree, and hence, there are more trees than trees. Figure 19 depicts a tree that has been truncated into two tiles.

.

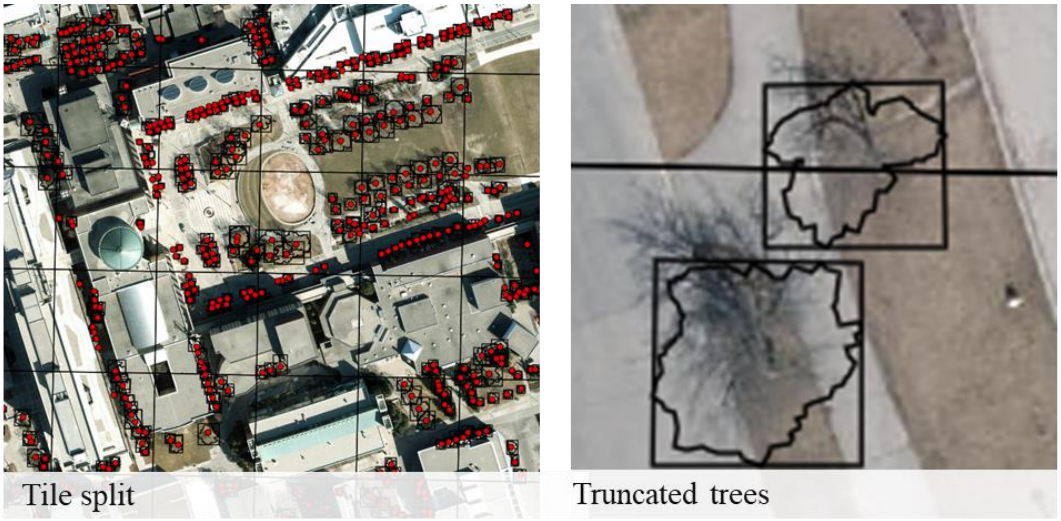


Figure 19. BEV image of truncated tree example. The tile splitting causes the trees to be divided as well into multiple tiles as can be seen on the left figure. The red dots are representing trees. The tree boundaries are captured in the boxes as can be seen on right with BEV image from the top. The horizontal line in the middle is the border for the tiles.

3.3.1 Heights

As shown in Figure 20, 78.79% of the trees are within the range between 4 to 14 meters in height. There were only a small number of trees that are smaller than 2 meters. Overall, the distribution of tree heights was normally distributed. Considering most of them were artificially planted or well-managed by York university, it is understandable that there are not many trees that are outliers. However, it generates a different problem for DCNN based 3D object detection algorithms. As it is evident in Figure 20, there are not enough training sample in the dataset, that are shorter than 4 meters or taller than 18 meters. Hence, the network does not have enough training sample to learn about such trees.

For example, overall recall score for test set from PV-RCNN++ model with IoU threshold of 0.3 was 0.78. However, for the trees that are shorter than 4 meters, the recall score was 0.46. Also, recall score for the trees that are taller than 18 meters was 0.66. Therefore, the trees that are outliers were harder to detect.

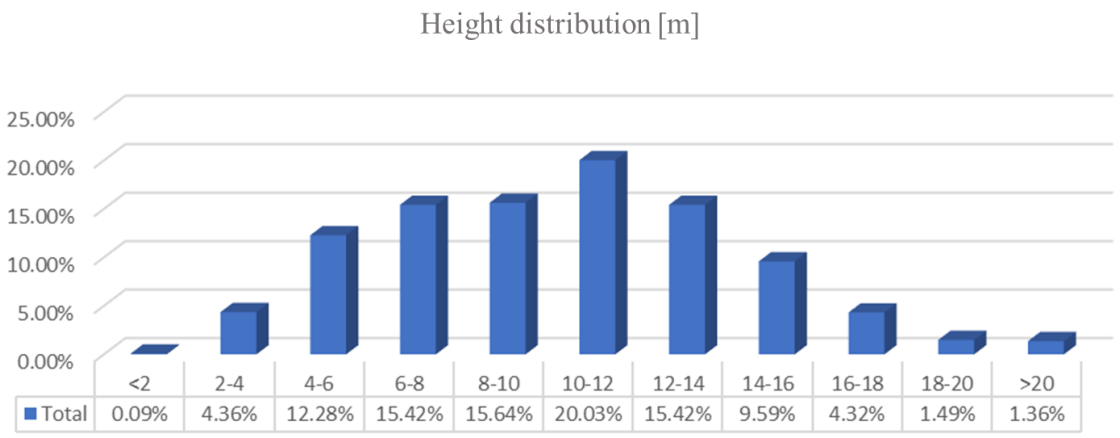


Figure 20. Tree height distribution graph.

3.3.2 Sizes

The size of a tree was calculated by the width and the length of the bounding box that covers a tree watershed from BEV, which generated the sizes of trees in m^2. Unlike the height distribution, it was more positively skewed in size as shown in Figure 21.

Also, as it was discussed in section 3.3.1, it is hard for the models to detect the trees that are outliers in the dataset. Hence, there was a need to figure if there is linearity between two characteristics of trees: height and size. It was hypothesized that the tree's size would also increase if a tree's height increased. To confirm this hypothesis, R^2 value was calculated between the height and size of each tree. The result can be observed in Figure 22. Unlike the hypothesis, there was not a strong enough relationship between the height and the size of trees; the R^2 value was only 0.26, which is not enough to confirm there is a relation between the two characteristics.

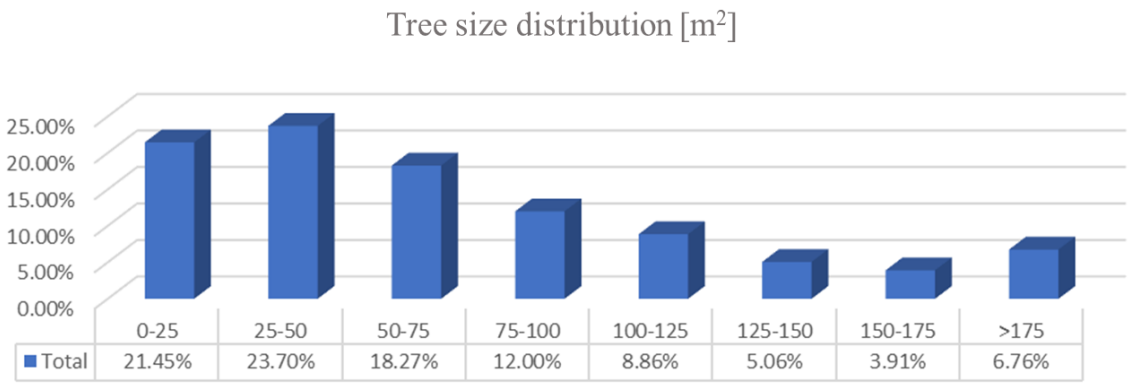


Figure 21. Tree size distribution graph.

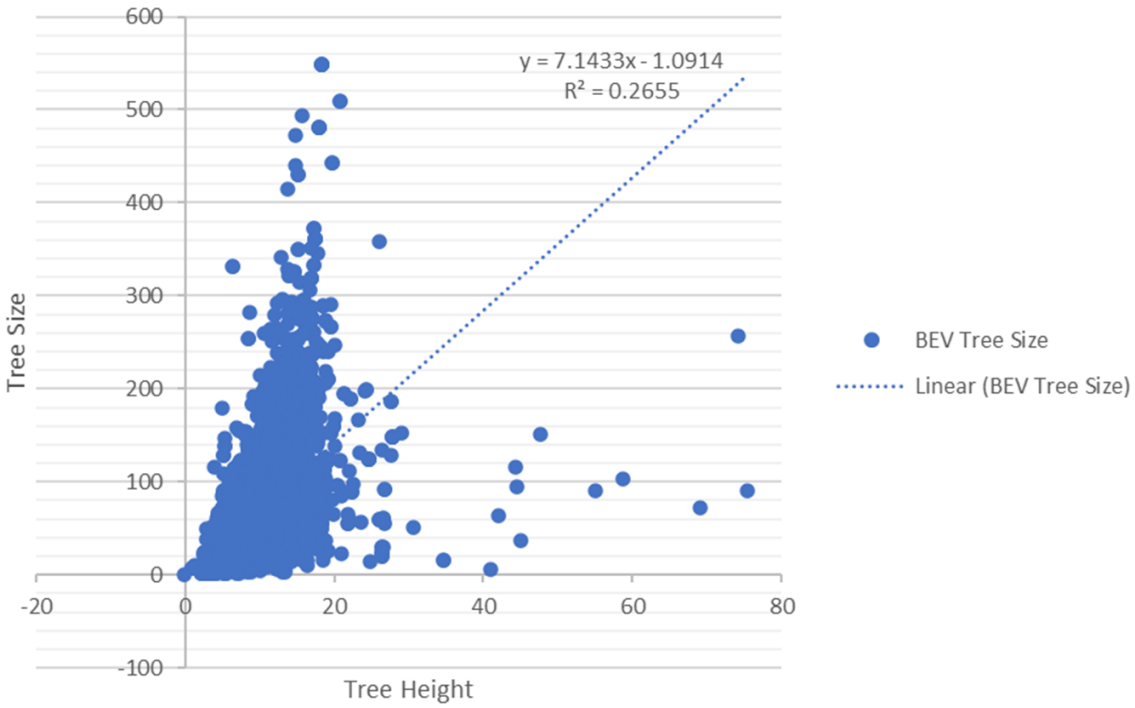


Figure 22. Tree size vs height distribution graph.

3.3.3 Occlusions

Occlusion of tree was calculated by dividing overlapped area of a tree by overall size of a tree. The range between 0% and 10% occlusion accounted for around 35% of all occurrences,

but the distribution of other ranges was uniform. As depicted in Figure 23, this indicates that most trees did not significantly obstruct each other.

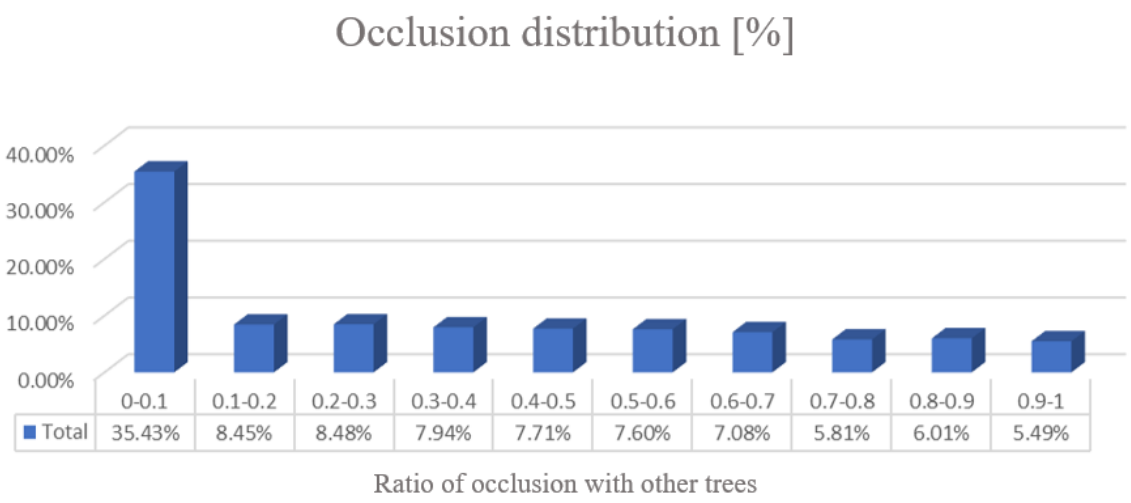


Figure 23. Tree occlusion distribution graph.

The initial hypothesis about the occlusion level was that if it a tree is more occluded, it would be harder to be detected. For example, the recall score for test set from PV-RCNN++ model with IoU threshold of 0.3 for the trees has less than 30% occlusion level was 0.70. However, for those trees that has more than 70% occlusion level, the recall score was 0.34 that is significantly lower than the prior case.

3.4 Summary

In this chapter, novel dataset called YUTO Tree-5000 was presented. Initially the reason why it must have been built was revisited, which was briefly discussed in the previous chapter.

YUTO Tree-5000 dataset was carefully designed to capture the most realistic scenes of urban environment. Hence, it utilized the most optimal ALS collected point cloud data and fused with the field collected data which was human collected. The process of amalgamating tree inventory information with ALS tree data was proceeded semi-automatically with multiple human inspection point still intact. Inevitably, the performance of DCNN based object detection algorithms are bounded to what it was given to be trained with. Hence, this novel dataset was analyzed in different aspects such as height, size, etc. for acknowledging how they may affect the predictions from the detection algorithms.

Chapter 4.

Baseline networks

As it was discussed in the previous chapters, two-stage method is often considered to have better accuracy than the One-stage method in regression and classification confidence. The reason we refer to these networks two-stage method is that in the first stage, it generates bounding box proposals. And in its second stage, it pools features, classifies them, and refines box regression through another network. There are two different ways of generating these features. One is to extract them from the feature maps of the backbone networks e.g., Voxel RCNN [66]. Another is to encode directly from the raw point cloud itself e.g., STD [49], PointRCNN [50].

There are different arguments from researchers to assert the necessity of the second stage. Among such claims, a couple are dominant. One of the most common arguments is that features may recover their lost positional information. The extracted features sometimes lose its positional information by various procedures such as voxelization, striding, etc. the claim is that the second stage may help it to retrieve. The other is the misalignment between the confidence

map and localization accuracy. It is because the classification and the regression are operated in the two different branches. Hence, they may not align correctly with another. Later argument is not often used in the ICTD, since the field is yet new to the method, the most concern is still at the detection of trees. Such arguments are often braced by the increase of detection accuracy which brought from the implementation of the second stage.

Some argues that the second stage is not necessary to achieve the optimal accuracy. For example, the authors of the Voxel RCNN assert that utilizing voxel-based features can achieve the same level of positioning accuracy without fusing the point-based features [67]. Also, some argues one stage structure can produce accurate enough localization to match the two-stage structure, since the benefit of having two-stage structure is that refinement of classification and enhancing regression [68].

4.1 One-stage networks

For the one-stage network testing, Pointpillars [69] and SECOND [62] were selected for the testing. Even though the VoxelNet [44] was not including for the testing in this specific paper, it needs to be mentioned. The later one-stage networks inherited many ideas from the general structure of the VoxelNet [44]. As its name suggests, it utilized voxels from the raw point cloud. It extracts features through 3D convolution. Since the domain is in 3D, the process is computationally expensive because point cloud is sparser than in 2D. Thus, SECOND [62] utilizes 3D sparse convolution to compensate the slower processing time of 3D convolution. To accelerate the encoding process, PointPillars [69] encodes given point cloud into pillars in BEV, then apply 2D CNN. The algorithm design of each network is shown in the Figure 24.

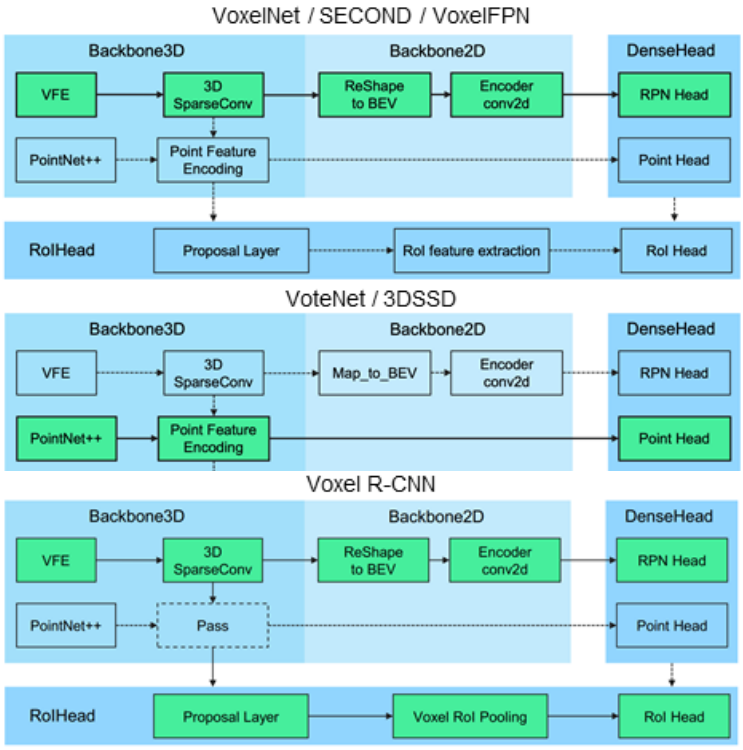


Figure 24. Algorithm diagrams of One-stage networks. In the design diagrams, the procedures described with green boxes are activated processes. The reason why the structures are similar is that they branched out from the same backbone model structure. This allowed the researchers could generate different models more easily and efficiently [44], [62], [66].

There are other well-acknowledged networks in this category, that were not tested with YUTO Tree-5000 dataset. Confident IoU-Aware Single-Stage object Detector (CIA-SSD) [70] utilizes IoU prediction branch and executes the post-processing to include localization accuracy to confident scores. Also, there is AFDet [59] network, which was the Waymo dataset [21] contest winner of 2020.

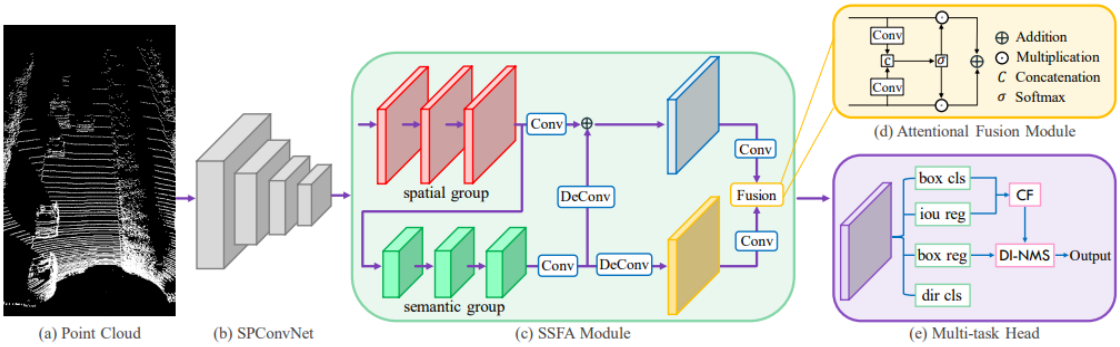


Figure 25. Algorithm diagram of CIA-SSD. This shows this pipeline of CIA-SSD model, the model has different approach in the frameworks comparing to the models described in the Figure 24. The procedure is to encode the input, feature aggregation for feature extraction and then feeds it to the multitask head for the object classification and localization. Hence, in the essence the approach of One-stage is evident, which is the same as others mentioned previously [70].

4.2 Two-stage networks

For the two-stage network testing, PointRCNN [50], Part-A2 [45], Pyramid R-CNN [71], VoxelRCNN [66], PV-RCNN [72] and PV-RCNN++ [73] were selected.

As it was discussed in the previous sections, there are three popular streams based on the representation of points of interest: point based, voxel based, point and voxel based. Point based methods treat the sampled point clouds as Points of Interest. e.g., PointRCNN [50], STD [49] Voxel based methods use the voxel points from 3D CNNs as Points of Interest e.g., Part-A2 [45], Voxel R-CNN [66]. And lastly, point and voxel based methods use the key points that encode the whole scene as Points of Interest. e.g., PV-RCNN [72], PV-RCNN++ [73].

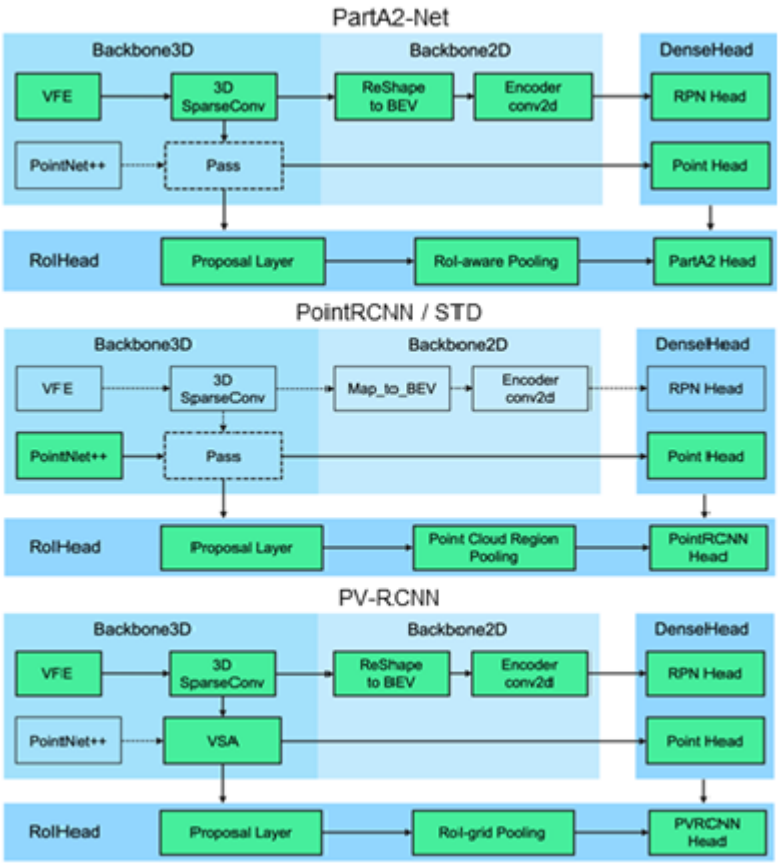


Figure 26. Algorithm diagrams of two-stage networks. The way they are described follows the same rule as in Figure 24 [50], [72], [73].

Many concepts and ideas were naturally inherited from 2D object detection algorithms to 3D object detection algorithms. Fast R-CNN [53], which later evolves to Faster R-CNN [54], suggested a concept of two-stage detection that the network generates RoIs in the first stage, then in the second, it refines what was predicted in the first stage.

PointRCNN [50] brings the Region-based Convolutional Neural Networks [74] method, which is predominantly utilized in 2D domain, to 3D domain. Point RCNN utilizes PointNet++ [63] with multi-scale grouping as a backbone network to generate 3D bounding box proposals then such proposals are pooled to the second stage to be refined. Point RCNN [50] countered the problem of pooling more than one proposal for one area of point clouds, meaning generating

different proposals for one points group. Because of this problem, the encoding of the geometric information of these proposals would not be possible.

Hence, Part-A2 [50] proposed a point cloud pooling module to solve this problem. The proposed module eliminates the ambiguity while conducting region pooling. The same problem was handled differently by PV-RCNN [72]. It summarizes voxel-wise feature volumes at multiple neural layers with learned points into sets of key points. These sampled key point features are aggregated to the RoI-grid points.

Unlike PV-RCNN [72], Pyramid R-CNN [71] utilizes the RoI-grid pyramid to eliminate the sparsity of point cloud problem by extensively collecting points of interest for each RoI. PV-RCNN++ [73], which is a more improved version of the original PV-RCNN [72], operates vector-pooling procedure to collect the key point features from various orientations. Voxel RCNN [66] suggested the voxel RoI pooling method, which directly extracts RoI features from the 3D feature volumes instead of point features.

There are other popular two-stage networks that were not tested with our dataset. For example, CenterPoint [61] proposed a new pooling module, which samples five key points from BEV features using bilinear interpolation. RSN [60] proposed a different two-stage method that suggests foreground segmentation as a first stage to sparsify the point clouds, boosting the efficiency of the second stage sparse convolution.

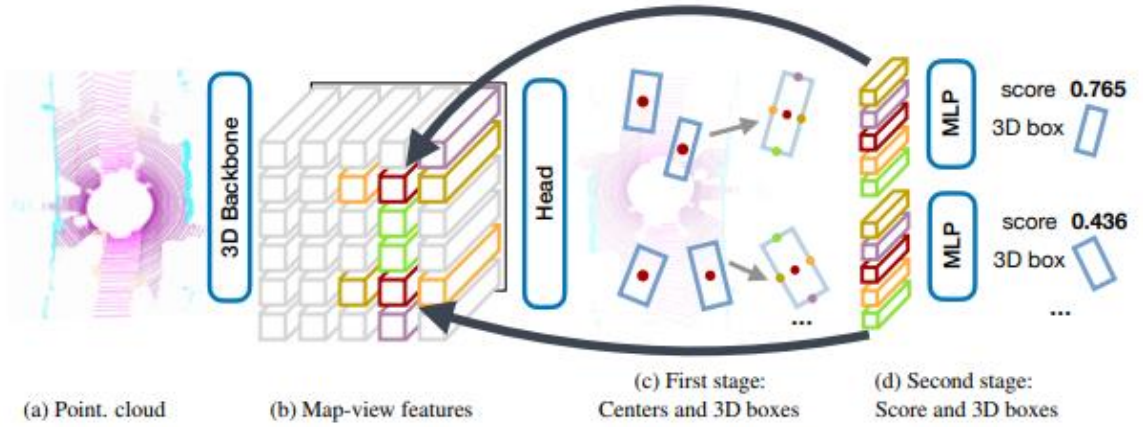


Figure 27. Algorithm diagrams of CenterPoint. It uses a standard 3D backbone for extracting features, then, a 2D CNN detection head finds the object centers and regress them to 3D bounding boxes. The prediction is for extracting point features at the 3D centers of each face of the estimation that is pushed to Multilayer Perceptron (MLP) procedure. This is a classic example of two-stage style detection algorithm. [61]

Network	Structure	Point representation	Anchor utilization
SECOND [62]	One-stage	Voxel	Anchor
PointPillars [69]	One-stage	Voxel	Anchor
PointRCNN [50]	Two-stage	Point	Anchor Free
Part-A2 [45]	Two-stage	Point & Voxel	Both
PV-RCNN [72]	Two-stage	Point & Voxel	Anchor
Pyramid R-CNN [71]	Two-stage	Point & Voxel	Anchor
VoxelRCNN [66]	Two-stage	Voxel	Anchor
PVRCNN++ [73]	Two-stage	Point & Voxel	Both

Table 3. List of baseline networks. For baseline network testing, eight SOTA networks were selected, and they were utilized for validation and testing of the YUTO Tree-5000 dataset.

4.3 Ablation studies

As is shown in Table 3, the selected networks were carefully selected due to their characteristics that were discussed in previous chapters such as the existence of the region proposal stage, pre-defined anchor boxes, etc.

Also, many of them could be trained upon the same platform of OpenPCDet [75], which reduced the time of environmental setups for each network requires. For the evaluation of the dataset and comparison among different types of these eight networks, the settings of the networks such as the maximum number of voxels or number of proposals were set to be the same. This will be discussed more thoroughly in the following.

4.3.1 Performance metrics

Before discussing more details of ablation studies, the performance measuring techniques that were utilized to compute the accuracy must be described first. For analyzing the results from each test, either Average Precision (AP), F1 score, or both were computed. To understand these metrics, confusion matrix needs to be referred first. Confusion matrix one of the most popular measures to solve classifications. It is often utilized not only for binary classification problems, also for multiclass classification problems as well. In the case of the individual single tree detection, binary classification method is used, and an example of a confusion matrix is as shown in Table 4.

		Predicted	
		Negative	Positive
Ground Truth	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

Table 4. Confusion matrix

Confusion matrices is based on the values of predicted and ground truth which often considered as actual reality. The TN represents the number of negative examples classified as negative, which are correct predictions. Similarly, TP indicates the positive examples that are classified positive, which are again correct predictions. Then, FP is for the number of negative ground truth object that are classified as positive, meaning prediction is made wrongly. FN stands for the positive ground truth objects classified as negative, meaning no prediction was made for the existing ground truth object. While establishing these measures, they are often elaborated to more advanced measures of precision, recall, AP and F1 scores.

Equation 3. Precision, Recall and F1 score equations

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}, \qquad F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The methods to calculate the precision and recall values are as in Equation 3. Precision is defined as the number of TP over the sum of the numbers of TP and FP, answering the question of out of all predictions, how many of them are correct. Recall is defined as the number of TP over the sum of the numbers of TP and FN, answering the question of out of all ground truths, how many of them are predicted. Then, F1 score is defined as in Equation 3. The F1 score presents the harmonic mean of Precision and Recall, calculating the average of precision and recall values. Hence, the F1 score would be high when both precision and recall are high and vice versa, when only one of them are high, the F1 score would remain as a medium.

To define the term AP, the precision-recall curve needs to be discussed first. When a prediction is made, it generates the conditional probability. The greater the probability, the more

confidence the model have towards its predictions. To generate the end predictions, an algorithm needs a threshold of this probability to decide if the generated prediction is valid enough to be the final prediction. This is where the user defines the threshold of how confident it should be to make end prediction. The smaller this threshold is, the higher the number of predictions will be made, and the lower the chances that the ground truths will be missed, resulting there will be higher recall values. In the opposite cases, when this confidence threshold is set as high, the predictions will be made with higher confidence, resulting there will be higher precision values. As mentioned in the previous paragraphs, it is ideal for the algorithm to achieve both high precision, and high recall. However, a detection algorithm, in most cases, cannot have both. Hence, there exists a tradeoff between precision and recall based on the confidence threshold that was decided by the user.

Here is where a precision-recall graph is often utilized. It plots the value of precision against recall for different confidence threshold values to help the user. A precision-recall curve is a graph with Precision values on the y-axis and Recall values on the x-axis. A good curve is when it has greater area under curve. As in the examples in Figure 28, the detection algorithm with the blue line has better performance than the one with the green line.

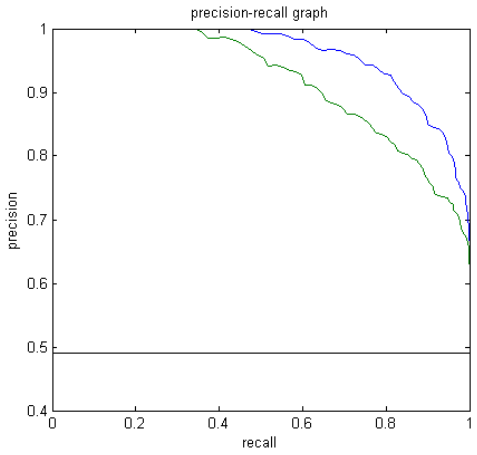


Figure 28. Precision-recall curve example. Retrieved from [76].

Therefore, the choice of what confidence value would be utilized is completely up to the user, which can be difficult and subjective.

Equation 4. Average precision equation

$$Average\ Precision\ (AP) = \sum_{k=0}^{k=n-1} [Recall(k) - Recall(k+1)] \times Precision,$$

$$\text{n} = \text{confidence threshold value}$$

Here is where average precision performs an important role as an indicator which removes the dependency of selecting the confidence threshold value. Like the area under the precision-recall graph, which was discussed in the previous paragraph, AP is another way to summarize this curve into a single value between 0 and 1. AP score is calculated as in Equation 4.

4.3.2 Dataset setting

First, the data from multiple flight paths were merged, then split into 60 m by 60 m data tiles. Ideally, relaying the entire scene as a one feed would be the best, however it is not an executable option due to hardware capacity. Thus, the optimal size of tile was calculated to feed the data to the algorithms, which was 60 m by 60 m. Second, truncated trees at the edge of the tiles were included in the datasets to allow the detection of partial trees. Also, any bounding boxes that were less than 1 m^2 from BEV POV were removed, which was generated during the tiling process. In the end, 5577 3D bounding boxes were generated, including the partial trees. Third, vegetation points classified but not included in the bounding box in were removed. This was caused since there were missed trees during the field survey. It was necessary not to include these non-labeled trees in the dataset because it may confuse the networks. Forth, we normalized the height of the data with LAStools [64] to fulfil the assumption that the base bounding box height is equal to 0 for the networks. Finally, we split our data tiles randomly into training (80%), validation (10%), and test (10%) datasets. The ratio for the training set would be increased to have better accuracy, however the total amount of annotated data that we had was not enough to increase the training set ratio further than 80%. Hence, the ratio was chosen as 80:10:10.

All experiments were either conducted on the OpenPCDet [75] or a modified version of the platform. Unless specified, the default hyperparameters were applied as the KITTI dataset [19] using the OpenPCDet toolbox [75].

4.3.3 Maximum number of voxels

Most baseline models assume that point cloud data is sparse because it would be acquired from a mobile platform, e.g., the KITTI dataset. Hence, many detection networks seek effective sampling methods for boosting performance.

Conversely, YUTO Tree-5000 dataset was acquired through an airborne platform, which makes the dataset much denser. To illustrate the differences, Figure 29a shows an example of KITTI data in BEV and Figure 29 b shows an example of YUTO Tree-5000 in BEV angle. In this example, 5.3% and 96.6% of the pixels contain ALS points in Figure 29a and Figure 29b. Due to this difference, the more voxels being utilized for the voxel-based networks, the better results were generated. Table 5 illustrates the differences between the default and the maximum number of voxels resulting in different IoU. Experiments were set to utilize the maximum number of voxels that GPU memory allows, 57600 for PointPillars [69] and 150000 for others.

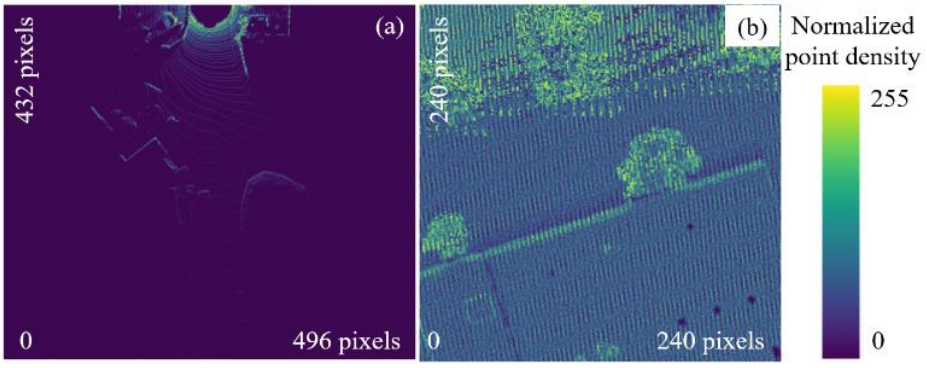


Figure 29. Point density visual comparison. Point cloud visualization from each dataset: (a) an example of BEV of KITTI dataset (b) an example of YUTO Tree-5000.

Maximum number of voxels		AP	AP
Train	**Test**	**(IoU = 0.3)**	**(IoU = 0.5)**
4300 (7.5%)	10700 (18.7%)	0.695	0.532
57600 (100%)	57600 (100%)	0.756	0.594

Table 5. Voxel utilization comparison. Validation Results from the default set number of maximum number of voxels used and 100% utilization of voxels.

4.3.4 Voxel size

As it was discussed in the previous section, utilizing well-optimized voxels is a critical aspect for certain networks which has voxels as the points of interest representation. As it is important to utilize the most effective number of voxels, utilizing optimal voxel size is critical as well. At the stage where the network transforms points into voxels, there are several parameters that can be manipulated: voxel size, the maximum points per voxel, and the maximum number of voxels. Four scenarios were set to be tested:

1. Case with the identical setting as KITTI [19]
2. Case with initial custom configurations
3. Case that elaborated from the initial custom configuration with a larger voxel size
4. Case that elaborated from the initial custom configuration with a smaller voxel size

The specifics of the configurations are illustrated in the Table 6. For each case, three experiments were executed. As the results are shown in the Table 7, the best results are from the third case which utilized all pixels in the pseudo image. Hence, the network testing inherited such a configuration from this experiment.

Case	Voxel size	Max points per voxel	Maximum number of voxels		Average non zeros	Pseudo image size
			Train	Test		
1	[0.1875, 0.1875, 32]	32	16000	40000	93616.40	320 x 320
2	[0.25, 0.25, 32]	250	20000	40000	54423.20	240 x 240
3	[0.3, 0.3, 32]	32	36864	36864	35742.20	320 x 320
4	[0.0625, 0.0625, 32]	15	300000	300000	342315.70	320 x 320

Table 6. Voxel configurations for each case. 60 m by 60 m lidar tile size were utilized for all cases, pseudo image sizes are in pixel.

Case		BEV			3D		
	IoU	0.3	0.5	0.7	0.3	0.5	0.7
1	Mean	0.667	0.594	**0.447**	0.528	0.392	0.108
	SD	0.008	0.010	0.114	0.123	0.016	0.013
2	Mean	0.660	0.579	0.339	0.608	0.411	0.122
	SD	0.012	0.015	0.004	0.008	0.022	0.009
3	Mean	**0.699**	**0.607**	0.339	**0.639**	**0.465**	**0.130**
	SD	0.004	0.005	0.011	0.017	0.006	0.005
4	Mean	0.570	0.502	0.325	0.539	0.373	0.098
	SD	0.012	0.008	0.010	0.010	0.010	0.012

Table 7. Results for different voxel configurations. The results shown is the mean AP for each case. For each POV, IoU thresholds of 0.3, 0.5 and 0.7 were utilized for each BEV and 3D POV.

4.3.5 Number of Proposals

As it was discussed in the beginning of this chapter, most two-stage methods use the top 100 of the RPN results when inferencing. However, with the default settings of hyperparameters, the two-stage networks performed worse than the one-stage networks in the initial preliminary result-generating tests. These results did not suit the expectations from the theoretical point of view. Furthermore, from the results, it was observed that all two-stage networks had very high precision and low recall at the default values. Naturally, it was hypothesized that if more RPN proposals are utilized, the better the result would be. Later tests were done accordingly to this hypothesis. Hence, RPN proposal settings were changed from top-100 to top-500, to match the

training phase. As a result, voxel based two-stage networks could achieve better AP scores, compared to one-stage networks. Table 8 shows the test results of voxel-based 2-stage methods improved drastically.

Network	Before: Top 100			After: Top 500		
	AP	Precision	Recall	AP	Precision	Recall
Part-A2	0.53	0.88	0.55	0.71	0.65	0.77
PV-RCNN	0.56	0.90	0.57	0.73	0.70	0.77
Pyramid R-CNN	0.57	0.87	0.58	0.79	0.77	0.83
VoxelRCNN	0.60	0.86	0.61	0.75	0.64	0.80
PV-RCNN++	0.55	0.89	0.57	0.72	0.62	0.78

Table 8 Results from each network for RPN proposal adjustment. AP, precision, and recall scores recorded with IOU threshold of 0.3.

4.3.6 Optimization

For the deep neural network algorithms, the most way of optimization methods is often considered those which are based on stochastic gradient descent (SGD). The shortcoming of the utilization of SGD as a hyperparameter is the learning rate. The magnitudes of different parameters vary, resulting in the adjustment being demanded throughout the entire training process, which makes the tuning difficult. Different adaptive variants of SGD are being utilized in various situations, such as Adam [77], Adagrad [78], Adadelta [79] and RMSprop [80]. The learning rate adaptations from these algorithms to different parameters are automatic, based on the gradient statistics. Among these algorithms, the Adam algorithm was chosen for the optimization which is for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments [77]. Adam optimizer with an initial learning rate of 0.0003 and weight decay of 0.0001 was utilized for the testing of all networks. According to the learning speed of the baseline networks, the training was stopped learning up to 100 or 200 epochs.

4.3.7 Postprocessing

Non Maximum Suppression (NMS) procedure functions as the eliminator of the overlapping bounding boxes. During NMS procedure, it compares the confidence value of each prediction candidates. If the prediction that does not have the maximum confidence level has more value of IoU with the prediction with the maximum confidence than the set threshold, it would be considered as duplicates. Therefore, these duplicates would be deleted from the list of prediction.

The default NMS value was set for the objects that generally do not overlap. Hence, the value was set to be low for generating accurate predictions for such object. However, in the YUTO Tree-5000 dataset, many overlapped trees were gathered as a cluster, making the default NMS threshold was too low for trees. Tree clusters were detected as 1 TP at default NMS, resulting in many FNs. Hence, we increased the NMS to 0.4 to retain some of these overlapping trees.

4.3.8 Anchor size

The trees in YUTO Tree-5000 dataset varies in size. This size variation directly affected determining the number of anchors and their sizes. The initial hypothesis was that if the network utilizes more anchors, the better detection results would be achieved. Hence, 1 to 5 anchors were tested for the dataset where anchor sizes were derived from K-means++.

In computational geometry, the K-means clustering [81] is one of the oldest and most used approach. It describes the question of when there is a given an integer K and a set of n data points in R and decide which integer K would be the optimal to minimize the total squared distance between each point and its closest center. The K-means++[82] slightly differs from this original K-means. Instead of randomly selecting the initial value of the K centroids, it rather

selects one initial centroid randomly and successively chooses centers until K has been finally chosen. Then the data points are chosen as a new centroid with a probability, which is determined by a potential function that depends on the distance from already chosen centers.

For this testing, sizes of anchors were set as:

- K=1: [7.18, 6.86, 9.07]
- K=2: [5.08, 4.79, 6.58], [9.65, 9.29, 11.99]
- K=3: [4.59, 4.21, 5.51], [7.24, 7.09, 10.47], [11.78, 11.22, 12.81]
- K=4: [4.39, 3.79, 5.31], [6.96, 7.55, 8.68], [12.06, 11.75, 12.52], [7.84, 6.34, 13.27]
- K=5: [4.02, 3.46, 4.69], [6.03, 6.00, 7.93], [8.98, 9.66, 10.19], [7.71, 6.21, 13.50], [13.24, 12.37, 13.62]

The test results in the Table 1Table 9 show that two anchor sizes [5.1,4.8,6.6] and [9.7,9.3,12.0] should be used. Then these values were applied to all anchor-based networks.

Number of K	**BEV**			**3D**		
	AP	**Precision**	**Recall**	**AP**	**Precision**	**Recall**
Default (2)	0.62	0.88	0.63	0.60	0.86	0.61
1	0.77	0.78	0.81	0.74	0.76	0.78
2	0.75	0.74	0.79	0.73	0.72	0.77
3	0.71	0.75	0.75	0.70	0.74	0.74
4	0.71	0.75	0.73	0.68	0.73	0.71
5	0.68	0.73	0.71	0.66	0.72	0.70

Table 9. Results from each number of K. Voxel-RCNN was selected for this testing because it had best results from the preliminary testing. IoU threshold of 0.3 was applied for this testing.

4.4 Experimental results

From the Table 10, Table 11 and Figure 30, it was observed that the difference between BEV AP and 3D AP was small, indicating it was easier to predict values according to the z-axis and, therefore, more essential to find the methods to predict accurate BEV bounding box.

The only point-based network, PointRCNN [50], performed the worst for tree detection for both BEV AP and 3D AP. This was because it draws sample points randomly, making the input data structure very sparse. Unlike instances such as cars and bicycles in the dataset created for autonomous driving, trees often overlap. To compare the prediction performance concerning overlap (occlusion), Figure 31 shows that the higher the overlap ratio, the FN is being predicted, that is, a drop in recall.

IoU threshold	SECOND	PointPillars	PointRCNN	Part-A2	PV-RCNN	Pyramid R-CNN	VoxelRCNN	PV-RCNN++
0.1	0.78	0.78	0.49	0.80	0.79	**0.91**	0.82	0.81
0.2	0.76	0.74	0.47	0.78	0.78	**0.89**	0.80	0.79
0.3	0.72	0.72	0.45	0.75	0.76	**0.84**	0.77	0.75
0.4	0.66	0.67	0.41	0.69	0.71	**0.74**	0.73	0.69
0.5	0.60	0.60	0.33	0.62	0.64	0.64	**0.67**	0.63
0.6	0.48	0.49	0.24	0.49	0.54	0.51	**0.56**	0.51
0.7	0.34	0.35	0.14	0.39	0.41	0.39	**0.43**	0.37
0.8	0.21	0.19	0.05	0.25	0.24	0.24	**0.29**	0.22
0.9	0.07	0.04	0.00	0.08	0.07	0.08	**0.10**	0.07

Table 10. BEV AP scores from each network. The best results were bolded.

IoU threshold	SECOND	PointPillars	PointRCNN	Part-A2	PV-RCNN	Pyramid R-CNN	VoxelRCNN	PV-RCNN++
0.1	0.77	0.76	0.48	0.79	0.79	**0.90**	0.81	0.81
0.2	0.73	0.73	0.47	0.76	0.76	**0.86**	0.79	0.78
0.3	0.69	0.68	0.42	0.71	0.73	**0.79**	0.75	0.72
0.4	0.62	0.60	0.34	0.65	0.67	0.68	**0.70**	0.64
0.5	0.53	0.52	0.27	0.56	0.58	0.56	**0.60**	0.55
0.6	0.41	0.39	0.17	0.44	0.47	0.45	**0.49**	0.43
0.7	0.29	0.24	0.08	0.33	0.35	0.33	**0.36**	0.31
0.8	0.15	0.08	0.01	0.20	0.18	0.19	**0.21**	0.17
0.9	0.03	0.00	0.00	0.03	0.03	0.04	**0.01**	0.02

Table 11. 3D AP scores from each network. The best results were bolded.

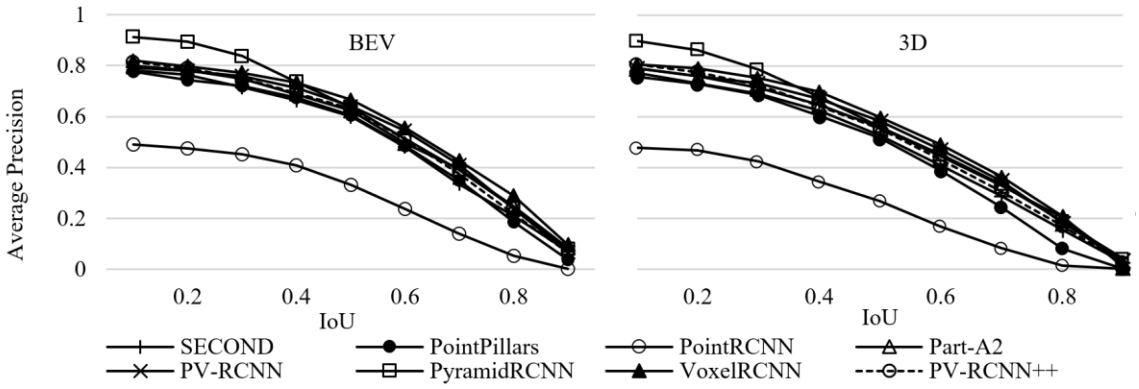


Figure 30. BEV and 3D results graphs. AP at different IoU threshold for 8 networks at (a) BEV and (b) 3D.

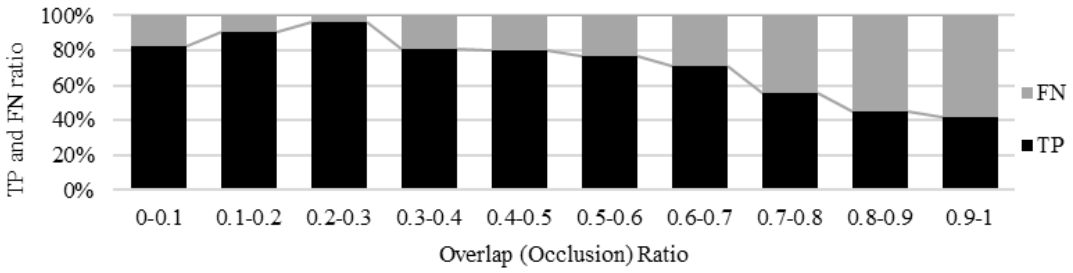
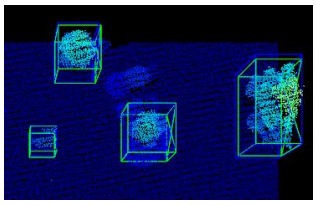
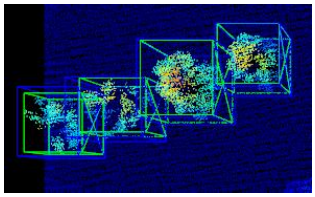
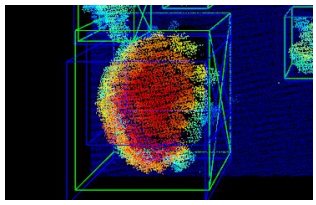
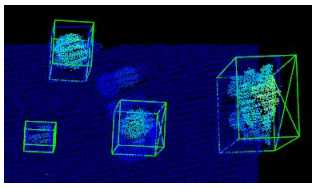
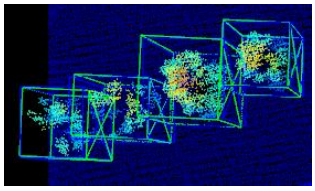
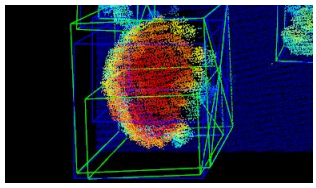
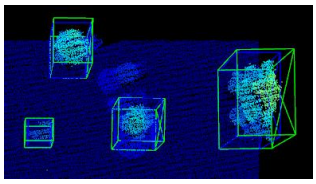
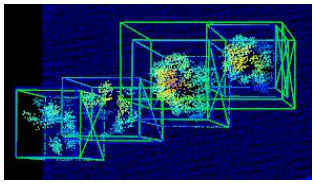
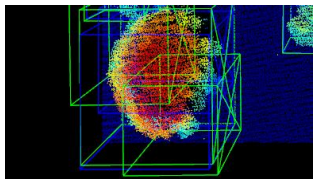
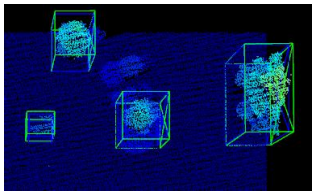
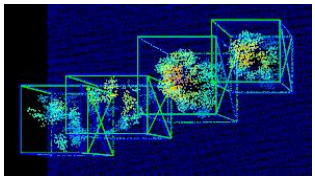
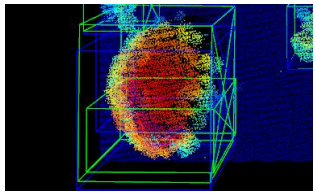


Figure 31. TP and FN distribution among different levels of IoU. True positive (TP), false negative (FN) distribution with different tree overlap (occlusion) ratio at IoU >0.3. This specific result was extracted from the PointPillars [69] results.

Table 12 is showing visualizations of the predictions that were generated from each network. It shows for all three different categories of occlusions which are less than 40%, 40% to 80% and more than 80%. Trees in less than 40% example are easy be detected because they are well separated while there are no external objects such as buildings or no elevation among trees. 40% to 80% example contains 4 trees that are moderately overlapping each other. Lastly, more than example for 80% has 3 trees mangled with each other, there is a large tree in the middle and 2 other trees are under the canopy of the large one.

	Occlusion levels		
	Less than 40%	40% to 80%	More than 80%
PointPillars			
SECOND			
Part-A2			
VexelRCNN			

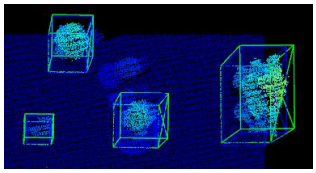
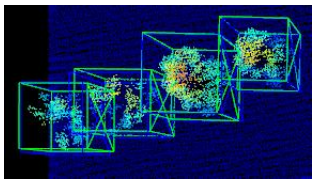
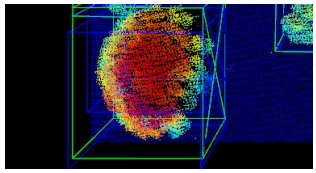
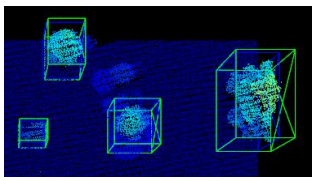
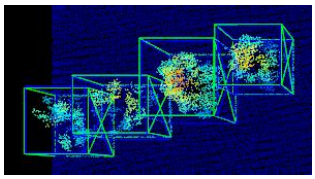
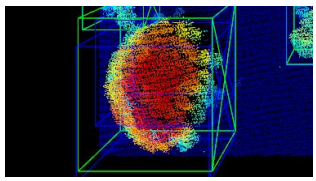
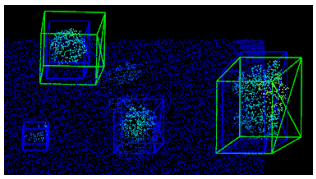
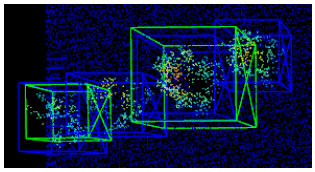
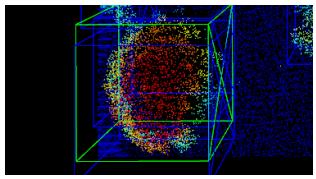
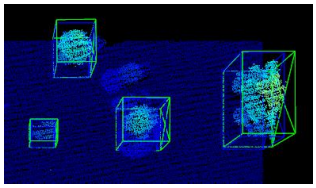
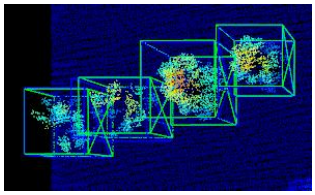
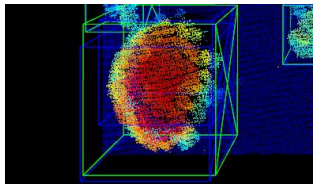
PV-RCNN			
PV-RCNN++			
PointRCNN			
Pyramid R-CNN			

Table 12. Prediction visualization from each network at different occlusion. The green boxes are the generated predictions, and the blue boxes are the ground truth.

4.5 Summary

YUTO Tree-5000 dataset is unique in many ways. It includes annotations for trees with varying degrees of occlusion. Occlusion ranges from zero intersections to a high degree with

other objects and intersect with other trees. Also, it is much denser than other popular MLS collected point cloud dataset. Therefore, the structures of baseline networks were reviewed carefully in terms of one-stage and two-stage networks. Also, many ablation studies were proceeded to optimize the performance of each network to compare the quality of predictions of the models correctly. Finally, the benchmark was properly assessed using eight detection networks. The experiments identified two best performing networks, Pyramid R-CNN and VoxelRCNN, under different IoU thresholds. Also, two-stage detectors generally performed better than one-stage due to the refinement of the second stage. This results directly connected to the following chapter of developing custom network module for improved prediction results.

Chapter 5.

Volumetric module

5.1 Motivation

As was discussed in the previous sections of Ablation study, specifically under the anchor size, the proper utilization of anchor boxes is critical to certain detection networks. As it was proven in the experimental results section of the previous chapter, a network with two-stage structure with an anchor box approach generated better detection results than the others. Also, it was proven that the utilization of many different sizes of anchor boxes does not benefit the results, which was shown in the Anchor size part of the Ablation studies section (Table 9. Results from each number of K of section 4.3.8 Anchor size). Hence utilizing a well-designed anchor box size is the optimal way for the given structures of the networks. For example, in this paper, K means ++ was utilized to calculate the most efficient anchor box size. This approach is logical for the networks that have been tested due to the detection goal for them being detecting objects for autonomous driving applications.

However, as it was discussed throughout this entire paper, the object that I pursue to detect is very different than those of the conventional autonomous driving applications. Applying single or a few accurately selected sizes of anchors are adequate for those objects such as cars, humans, or traffic objects since such targets do not vary much in their sizes. As was shown in section 3.3, the trees in YUTO Tree-5000 dataset vary from 2 meters to 20 meters in height. While 63.42% of trees are under 75 square meters in size, but also 3.76% of trees are larger than 175 square meters.

The limitation of anchor size may seem to be simple to engineer to solve, which can be utilizing anchor free network rather than anchor based. Hence the hypothesis of anchor free algorithm would generate better predictions than the counterpart was set to be proven. However, the best score was not from the anchor-free algorithm, nor it did show any other benefits of being anchor-free. These results initiated the motivation of this chapter specifically.

5.2 Preliminary testing

Firstly, conducting tests for proving the existence of a such problem was necessary. Pyramid R-CNN [71] was selected as the baseline network for this testing, because of its superior performance among other networks which was proven from the previous test of the YUTO Tree-5000 dataset, as shown in both Table 10 and Table 11. Also, it is with two-stage structure and utilizes anchored based methods which perfectly suits the purpose of the testing. This testing of different anchor sizes further than just generating detection scores was outlined to prove that there are differences among the detection results from different anchor sizes for different sizes of trees. The initial detection results from different anchor sizes are shown as in Table 13. The difference among scores from different anchor sizes was notable for further

analysis. Overall AP scores decreased as the anchor size increased, while precision scores increased, and recall scores decreased. On the other hand, F1 score sustained its level throughout the different sizes.

Anchor size	BEV				3D			
	AP	Precision	Recall	F1	AP	Precision	Recall	F1
5,5,15	0.67	0.65	0.71	0.68	0.56	0.57	0.62	0.60
5,5,20	0.65	0.64	0.70	0.67	0.57	0.58	0.63	0.60
5,5,30	0.65	0.63	0.71	0.67	0.57	0.57	0.64	0.61
9,9,15	0.62	0.67	0.66	0.67	0.53	0.59	0.58	0.59
9,9,20	0.60	0.74	0.64	0.68	0.52	0.66	0.57	0.61
9,9,30	0.62	0.68	0.66	0.67	0.53	0.60	0.58	0.59
13,13,15	0.56	0.75	0.60	0.66	0.51	0.70	0.56	0.62
13,13,20	0.56	0.76	0.59	0.67	0.50	0.70	0.54	0.61
13,13,30	0.54	0.74	0.57	0.65	0.47	0.66	0.51	0.58

Table 13. Detection results from different anchor sizes. Pyramid R-CNN was selected for this testing because it had best results from the preliminary testing. IoU threshold of 0.5 was applied for this testing.

However, it was not evident enough to conclude that there were meaningful differences among the results. Hence, the results were analyzed further regarding the heights of the tested trees and the predicted IoU of each. The Figure 32 and Figure 33 show the outcome. As the figures present, there are disparities among results from different anchor sizes. When the selected anchor size gets larger, the prediction IoU increased as the testing tree heights get increase. In other words, if the set anchor size is too big, the network experienced difficulties in detecting smaller trees and vice versa.

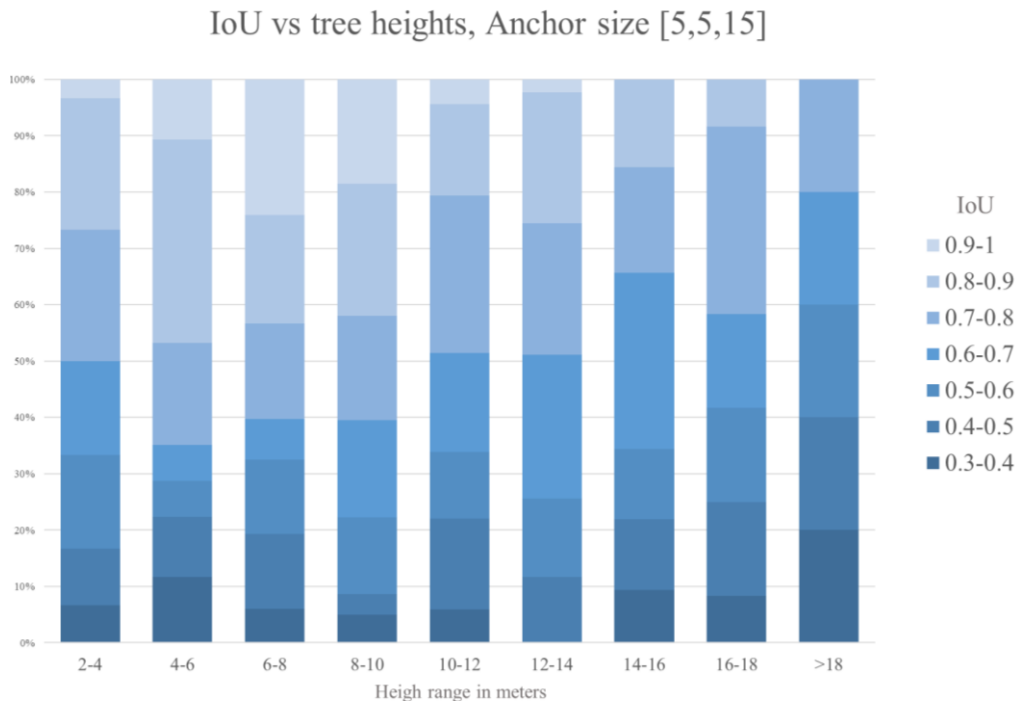


Figure 32. Tree heights vs detection IoU with anchor size [5,5,15]. The tree sizes are divided into 9 ranges and the number of detected instances is recorded in the chart within.

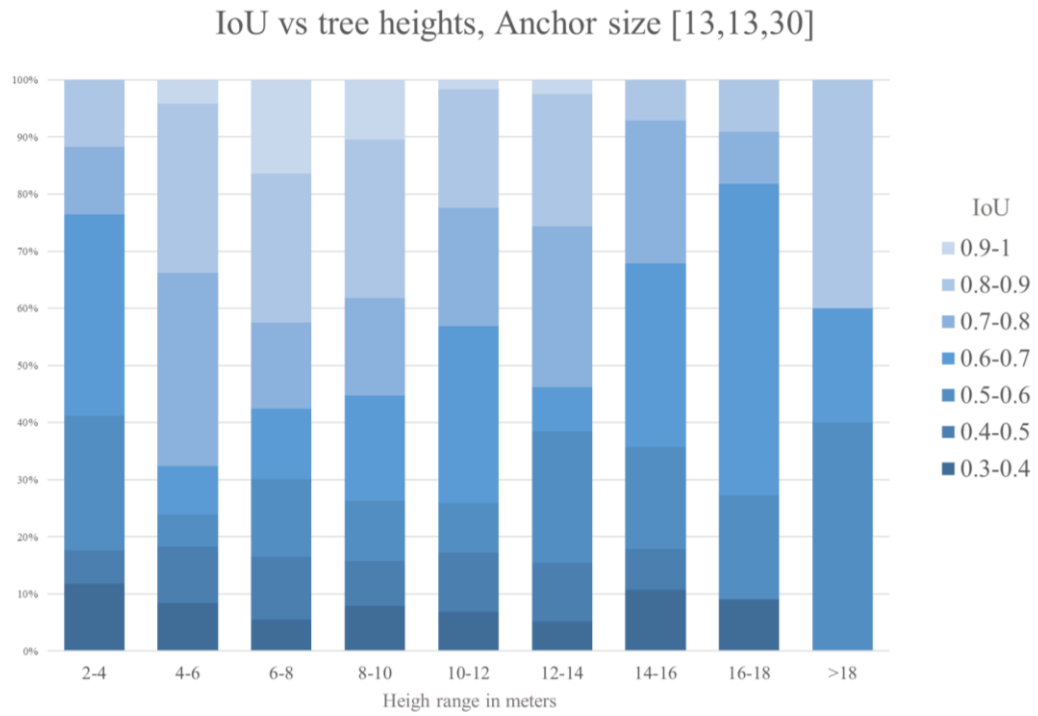


Figure 33. Tree heights vs detection IoU with anchor size [13,13,30]. Same criteria as the previous figure.

5.3 Methods

With problem acknowledgment of anchor sizing proven by the previous testing, it is evident that improving the regression method would also help the overall results. Instead of overhauling the entire regression procedure, improving the loss function was chosen as a proof of concept. Before portraying the improved loss function design, the original method that was used for the baseline algorithm must be discussed.

The selected baseline algorithm, Pyramid R-CNN [71] utilizes the smooth L1 loss as its loss function. The smooth L1 loss concept is one of the most widely adopted methods for regression for object detection algorithms, regardless of whether the domain is based on 2D or 3D. The smooth L1 loss was originally proposed with the Fast R-CNN [53], making its bounding box regression more robust compared to the L2 Loss which is much stricter than the L1.

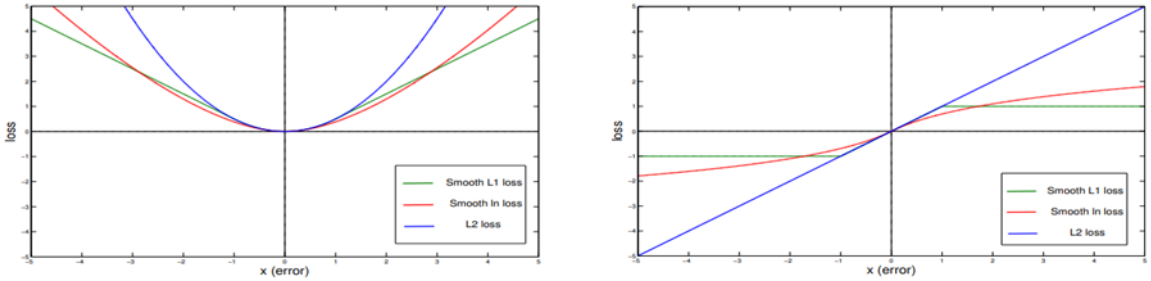


Figure 34. The visualizations of different loss functions. L2, smooth L1, and smooth Ln are shown, and the L2 function uses the same coefficient of 0.5 with smooth L1 [83]

Even though it was proposed almost a decade ago, it is still being utilized by many state-of-the-art networks. Among 2D detection algorithms, Faster RCNN [53] and Detectron2 [84] which is a library of the Facebook AI Research team that provides state-of-the-art detection and segmentation algorithms that succeeded Detectron [84] and Mask R-CNN benchmark [27].

Among 3D detection algorithms, Pyramid R-CNN [71] and PV-RCNN++ [73] still utilize smooth L1 loss and many other networks such as CIA-SSD [70] include this smooth L1 loss as a part of the total loss function.

As stated earlier, Smooth L1 Loss for object detection was originally proposed in Fast R-CNN [53]. Its role was to induce bounding box regression more robust by replacing the L2 Loss, which is much stricter. Smooth L1-loss can be interpreted as a mixture of traditional L1-loss and L2-loss. Its behavior is dictated by the absolute value of the argument. When it is high, it functions as the L1 loss. When the absolute value gets near zero, it behaves as L2. This can be proven in the following equations in the Equation 5.

Equation 5. Smooth L1 loss equation

$$L_{1;smooth} = \begin{cases} |x|, & if\ |x| > \beta; \\ \frac{1}{|\beta|}x^2, & if\ |x| \leq \beta \end{cases} , L_{\delta}(\beta) = \begin{cases} \frac{1}{2}\beta^2, & if\ |\beta| \leq \delta; \\ \delta\left(|\beta| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$

Left function represents smooth L1 loss function while the right represents Huber loss function. β is a hyper-parameter, which generally taken as 1. 1/β appears near x^2 term to make it continuous.

Hence, the focus of this implementation is to develop and design a regression algorithm that can expand and shrink the set anchor boxes more effectively and efficiently by add feature of the learnable weight for regression algorithms. The objective would not be to replace the existing smooth L1 loss implementation entirely. Rather, modification of it would be more focused. While executing this, the results of implementation would be the proof of the concept that allow this concept would expand further.

5.4 Implementation

Before implementation, the previous implementation of L1 loss function needs to be shown. The way to implement the smooth L1 loss is countless, since it is not a distinctive method which allow a few ways to execute.

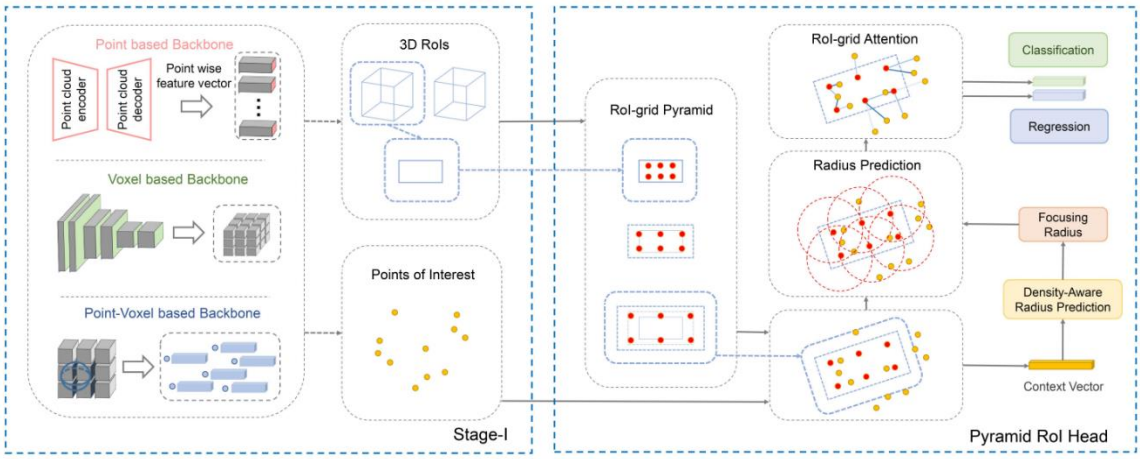


Figure 35. Algorithm diagrams of Pyramid R-CNN. The framework of Pyramid R-CNN. It claims to that it can adapt to diverse backbones such as point based, or voxel based. It takes the form of two-stage detection algorithm. In the first stage it uses its backbone for generating proposals and Point of interest, then in the second stage, it proposes the pyramid RoI head which can be applied upon the 3D proposals and point of interest.

There is the original implementation from the Fast RCNN [53], but it is generally considered as outdated since it was first implemented nearly a decade ago. One of the most popular ways of implementing this smooth L1 loss function is from the AI team of the Facebook group [84], also loss functions of the OpenPCDet borrows this method as well [75]. The following code-wise example shown in figure 36 is the original smooth L1 loss function from the Pyramid R-CNN network [71].

```
class WeightedSmoothL1Loss(nn.Module):

    def __init__(self, beta: float = 1.0 / 9.0, code_weights: list = None):

        super(WeightedSmoothL1Loss, self).__init__()
        self.beta = beta
        if code_weights is not None:
            self.code_weights = np.array(code_weights, dtype=np.float32)
            self.code_weights = torch.from_numpy(self.code_weights).cuda()

    @staticmethod
    def smooth_l1_loss(diff, beta):
        if beta < 1e-5:
            loss = torch.abs(diff)
        else:
            n = torch.abs(diff)
            loss = torch.where(n < beta, 0.5 * n ** 2 / beta, n - 0.5 * beta)

        return loss

    def forward(self, input: torch.Tensor, target: torch.Tensor, weights: torch.Tensor = None):

        target = torch.where(torch.isnan(target), input, target)  # ignore nan targets

        diff = input - target
        # code-wise weighting
        if self.code_weights is not None:
            diff = diff * self.code_weights.view(1, 1, -1)

        loss = self.smooth_l1_loss(diff, self.beta)

        # anchor-wise weighting
        if weights is not None:
            assert weights.shape[0] == loss.shape[0] and weights.shape[1] == loss.shape[1]
            loss = loss * weights.unsqueeze(-1)

        return loss
```

Figure 36. The original smooth L1 loss implementation. The original smooth L1 loss function that is utilized in Pyramid R-CNN network [71], which is shared for the other numerous networks that utilize this loss function.

As it was discussed previously, the source code for this module was based on fvcore.nn.smooth_l1_loss from the fvcore of the Facebook research lab [85]. As it is shown in the Figure 36, the smooth L1 loss is to be calculated as in the Equation 6

Equation 6. Smooth L1 loss equation from code breakdown

$$\text{smoothl1(x)} = \begin{cases} \frac{1}{2}x^{\frac{2}{\beta}}, & if\ abs(x) < \beta; \\ abs(x) - \frac{1}{2}\beta, & \text{otherwise,} \end{cases} \quad ,where\ x = input - target.$$

In the testing environment, for the default value of β, 1.0 / 9.0 was used.

If the β shifts towards 0, then the smooth l1 loss converges to the L1 loss. On the other hands, if the β increases towards the positive infinite, the smooth L1 converges to a constant 0. As the value of the β varies, the L1 segment of the loss has a constant slope of 1 as shown in the Figure 36. The input is the predicted location of the designated objects, while the target represents the regression target. The difference is calculated by subtracting the target from the input. The most critical part of this calculation is the last part where it calculates the difference and applies the set weights. In this original module, it utilizes the code-wise weighting meaning that the pre-defined weight is utilized to calculate the loss. Therefore, when the network generates the prediction which is significantly larger or smaller than the ground truth bounding box, it still utilizes the same weight for generating the loss. Such fact affects the overall efficiency and effectiveness of the module. Hence, the goal of this modification is to penalize such behavior to rectify the weights and make it learnable.

```
class WeightedSmoothL1Loss(nn.Module):

    def __init__(self, beta: float = 1.0 / 9.0, code_weights: list = None):

        super(WeightedSmoothL1Loss, self).__init__()
        self.beta = beta
        if code_weights is not None:
            self.code_weights = np.array(code_weights, dtype=np.float32)
            self.code_weights = torch.from_numpy(self.code_weights).cuda()

    @staticmethod
    def smooth_l1_loss(diff, beta):
        if beta < 1e-5:
            loss = torch.abs(diff)
        else:
            n = torch.abs(diff)
            loss = torch.where(n < beta, 0.5 * n ** 2 / beta, n - 0.5 * beta)

        return loss

    def forward(self, input: torch.Tensor, target: torch.Tensor, weights:
torch.Tensor = None):

        target = torch.where(torch.isnan(target), input, target)
        # ignore nan targets

        diff = input - target

        # minimum iterating over the last dimension
        axis = -1
        # obtain the minimal values
        min_values = diff.min(dim=axis).values
        # reshape the minimal values for comparing itself with diff tensor
        min_values_shape_corrected = min_values.unsqueeze(axis)
        # generate the mask of the non-minimal elements
        mask = (diff != min_values_shape_corrected)

        diff = diff * mask
        diff_min = diff * mask

        if self.code_weights is not None:
            diff = diff * diff_min * self.code_weights.view(1, 1, -1)

        loss = self.smooth_l1_loss(diff, self.beta)

        # anchor-wise weighting
        if weights is not None:
            assert weights.shape[0] == loss.shape[0] and weights.shape[1] ==
loss.shape[1]
            loss = loss * weights.unsqueeze(-1)

        return loss
```

Figure 37. The modified smooth L1 loss implementation.

Figure 37 is the proposed module that is after the modification of the original smooth L1 loss function. Up to calculating the difference procedure is identical with the original function. However, after such procedure it saves the minimum iterating over the last dimension. Then, it obtains the minimal values form the calculated difference by taking the last iteration. The minimum values are reshaped to be compared with the difference tensor to generate the mask of the non-minimal elements.

By implementing this feature creating a weighted regression loss function, it replaces the traditional approach. It compares the given anchor box size with the prediction box size. Therefore, it can be calculated that how much it is adjusting to generate the prediction bounding box from the pre-defined anchor box size. With such calculation, it can learn how much it should have adjusted to make the correct size of prediction rather than adjusting rigorously.

5.5 Results

The results from the modified smooth L1 loss function are shown in the Table 14. Overall, there was significant improvement of minimum AP and F1 score increase of 1%. Naturally, as the IoU threshold increases the boost from the modification decreases. However, it is evident that there was a noticeable escalation in the detection accuracy. The testing was executed with Pyramid R-CNN [71] which was the most effective algorithm to detect individual trees, that was proven from the section 4.4 experimental results of the ablation studies. Since the network was already producing the best accuracy among other two-staged detectors, the increment that this module may provide to other two-staged networks is well envisaged.

IoU	Original				Modified			
	BEV AP	**BEV F1**	**3D AP**	**3D F1**	**BEV AP**	**BEV F1**	**3D AP**	**3D F1**
0.1	0.93	0.91	0.91	0.90	0.96	0.94	0.94	0.92
0.2	0.90	0.89	0.86	0.86	0.93	0.91	0.89	0.88
0.3	0.85	0.84	0.79	0.79	0.85	0.84	0.80	0.80
0.4	0.76	0.76	0.69	0.70	0.77	0.76	0.71	0.71
0.5	0.67	0.68	0.56	0.60	0.68	0.69	0.59	0.61
0.6	0.54	0.56	0.46	0.50	0.56	0.57	0.47	0.50
0.7	0.41	0.46	0.35	0.40	0.42	0.45	0.36	0.40
0.8	0.26	0.32	0.19	0.26	0.27	0.32	0.16	0.22
0.9	0.09	0.15	0.03	0.07	0.06	0.11	0.02	0.06

Table 14 Results from the original smooth L1 and its modified version. The test was operated with the predefined anchor box size of [5, 5, 15] for both versions.

As it is proven in the Table 14, there was a clear improvement in the detection accuracy. The improvement was more evident and observable when the scope was toward the individual results from each tile. Figure 38 and Figure 39 are showing the same tile, which was ran through both detection algorithms. There were visible proofs of improvement such as better accuracy and less false predictions. This result shows how accurate the modified algorithm is with the tree object. Since there is no exact comparable dataset to the YUTO Tree-5000 dataset, the most similar dataset would be the pedestrian moderate category of the KITTI dataset. It has much less occlusion than the trees in the YUTO Tree-5000, but it remains with fair amount of it. The best detection algorithm for such category now is the 3D-FCT [86] with 3D AP score of 0.58 with IoU threshold of 0.5. It has lower detection accuracy than our modified network with less complicated dataset.

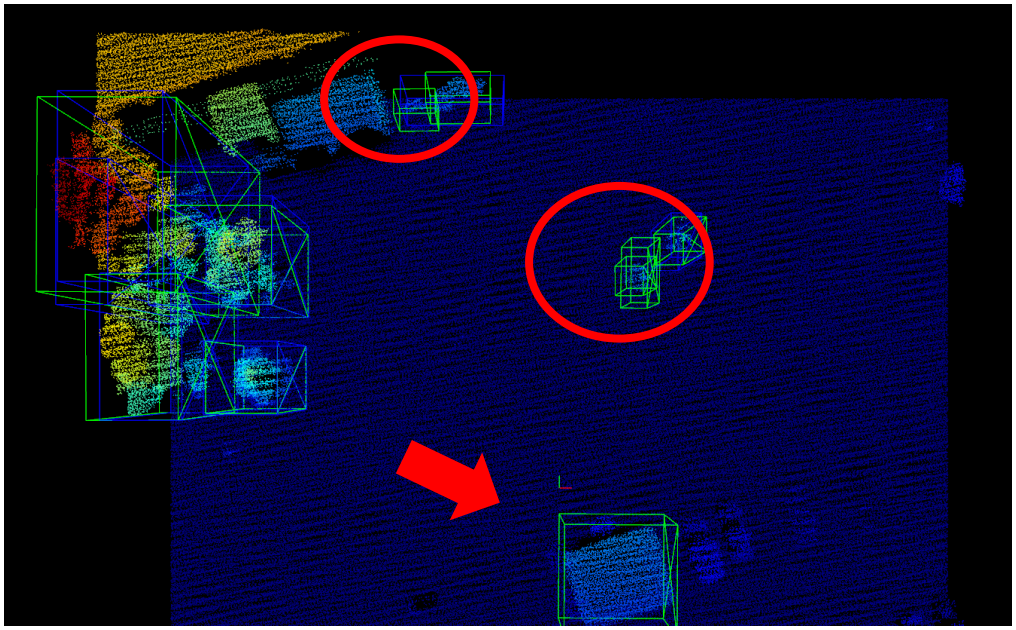


Figure 38. Detection result from the original Pyramid R-CNN. The blue boxes are the ground truth bounding boxes and the green boxes are the predictions. The top two red circles show the mispredictions of a single tree detected as multiple. Bottom arrow shows the parking booth detected as a tree.

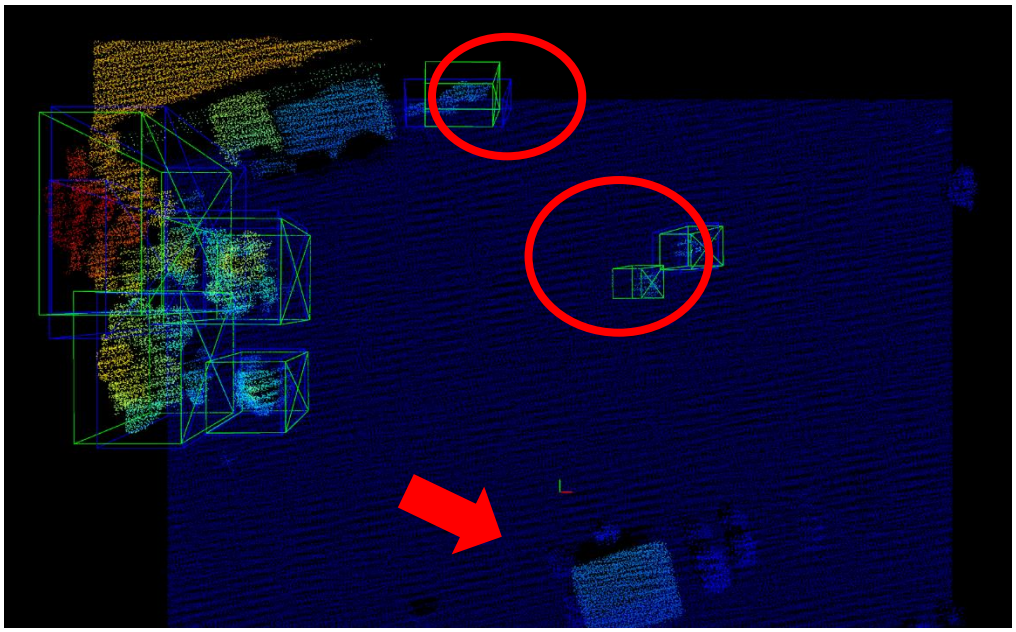


Figure 39. Detection result from the modified Pyramid R-CNN. Both circles have right amount of tree detected and no false prediction upon the parking booth.

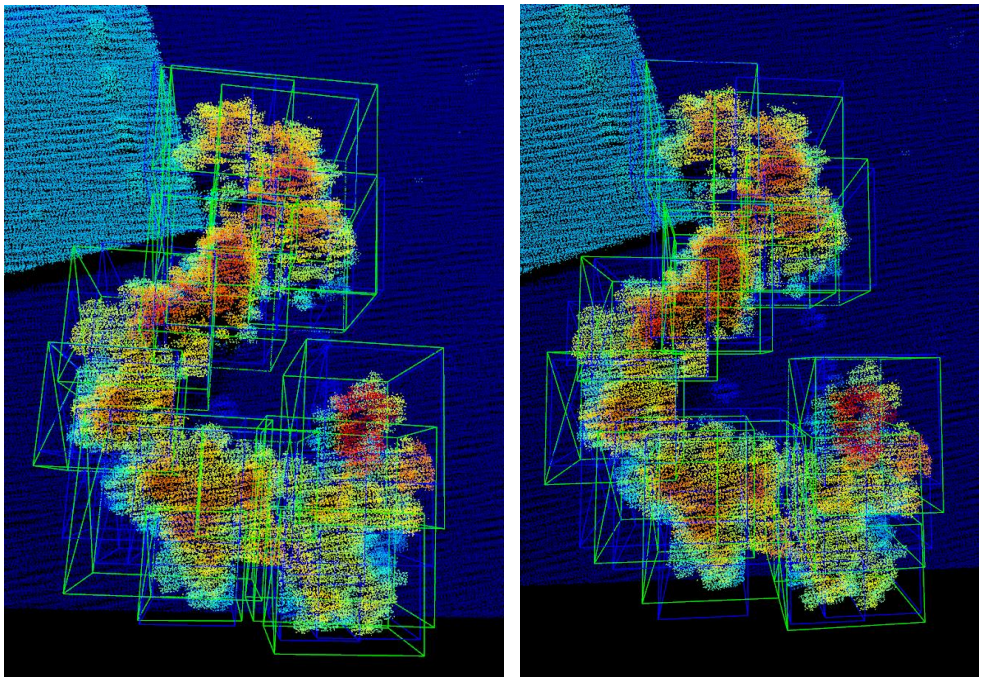


Figure 40. Detection result comparison. The detection results are far better when the objects are more clustered/occluded. The left is from the original and the right is from the modified. On the left, the predictions are overly populated, while on the right, predictions are much more accurate and precise. Even though it still was not able to capture all the objects in the tile.

5.6 Summary

In this chapter, a novel volumetric module that improves the detection results of the baseline model. From the previous sections of Ablation study of the last chapter, the proper anchor box utilization is one of the keys to improve the detection accuracy. There was multiple preliminary testing of different sizes of anchor box utilization to prove the results from each different size of anchor boxes are generating different accuracy according to the size of the object. Hence, the objective was to implement a regression algorithm which expand and shrink the set anchor boxes more flexibly by proposing the learnable weight concepts. As it was shown in the results section, the overall prediction accuracy had significant improvement from the implementation of this volumetric module.

Chapter 6.

Conclusion and Perspective

6.1 Conclusion and main contribution

Traditionally, the task of detecting or categorizing individual trees in a certain area was done by human eyes with the manual process that is often time-consuming, laborious, and sometimes dangerous. There were many attempts to automate the detection process of trees in different domains in forestry. Since the 1990s, when the ALS collected point cloud data was introduced to the field, many scientific studies found the potential of accurate tree detection of its height and size e.g.,[87], [88]. However, the approaches relied on rather mathematical and statistical studies and mostly focused on the domains of forests. In the 2010s, the emergence of deep learning algorithms, especially deep learning on 3D point clouds, attracted much attention and the strong surge of development of self-driving vehicles. Several publicly available datasets are also released, such as Semantic3D [23], ApolloCar3D [89], and the KITTI Vision Benchmark Suite [19], [25]. Naturally, there were attempts of utilizing point clouds data and deep learning algorithms to detect individual trees [90], but the overall number is small.

Individual trees are one of the most complex objects for 3D detection as natural objects. First, they vary in size and shape as it was discussed in the section 3.3.1 Heights and the Figure 20. Tree height distribution graph. For example, there is a tree that is 25 meters high while other trees that smaller than the bushes. Secondly, trees can have different occlusion levels, from standing alone without intersection with other objects to heavily occluded with other street furniture or other trees. Last but most importantly, the availability of the LiDAR tree benchmark dataset is extremely limited, making training more difficult.

Almost all research on this matter uses individual datasets gathered and the process by themselves, including the study of Brolly et al [90] was acknowledged previously. Traditionally speaking, on top of the watershed method, there were two main-stream point cloud-based methods. The older k-means approach was a clustering method with an iterative partitioning approach that usually requires seed points and then partitions [91]. A newer voxel-based method generally regards the approach that estimates crown base height after using local maximum focal filtering from a canopy height model [92] With the voxel-based method, the loss of a portion of geometric features would be inevitable.

This thesis presents two major achievements while pursuing the objective that stated. Firstly, it presents a workflow for creating a single-tree benchmark dataset named as YUTO Tree-5000 for tree detection, which was evaluated using eight networks. Existing benchmark datasets for urban object detection do not include tree detection, and existing forest benchmark datasets do not have publicly accessible 3D bounding box annotations. The development and availability of this dataset not only established a critical common ground for training networks suited for updating urban tree inventories in the community. Eight SOTA 3D detectors were tested to verify the applicability of existing networks designed for MLS applied to ALS.

Secondly, it presents the volumetric module with learnable weights which helps the network improve its detection accuracy significantly. With the help from the dataset which specifically built for the individual tree detection, it was able to be built and tested to verify its effectiveness. Also, the modification was based upon the existing smooth L1 loss function, meaning it can be applied to any algorithm utilizes such loss function to improve its accuracy.

6.2 Research challenges and limitations

Due to the popularity of the application towards the autonomous driving in the industry, it is inevitable that most of the 3D object detection research tends to gear toward such specific domain. As it was discussed in section 1.1 in the previous chapter, the domain for the autonomous driving and for individual tree detection remote from each other, which generates two major obstacles for this project.

First, there is a significant lack of the existing 3D LiDAR dataset which are collected with ALS. This leads to the problem of the incapability of comparing the YUTO Tree-5000 dataset with other dataset to validate its quality and specialty There are numerous dataset which were collected for the forestry applications, which differs from the main goal of this project, hence, not able to be utilized to test. There are somewhat similar datasets which were collected with TLS [40], but these datasets are not completely available to the public. Therefore, it is difficult to test the validity of the YUTO Tree-5000 dataset, which triggers the following problem as well.

Second, there is a significant lack of the existing 3D object detection networks that are either designed, or at least tested with the ALS collected dataset. This issue is directly connected with the previous issue. It is an established fact that the design of detection network is strongly

affected by its application and its target dataset to test with. If there is no dataset which support certain domain, naturally, there would not be a network which is geared towards it. This lack of networks leads to the fact that there is no availability to reference for developing a new module of a network or an overall network. As it was stated in the chapter 5, the baseline networks that this project is referencing has fundamental problem of them being to be expected to utilize the MLS collected dataset. These networks still function adequately with the YUTO Tree-5000 dataset and generates appropriate prediction results. However, the usual designs and point density boosting modules are clearly geared towards the application of autonomous driving. Therefore, who are interested in such specific 3D object detection algorithm develop their own dataset to test and develop their own dataset causing the inability of correlation among the algorithm to expand and proceed.

6.3 Perspectives and future works

The YUTO Tree-5000 dataset is unique that it includes annotations for trees with varying degrees of occlusion. It ranges from trees that have no intersections with other trees or other things to trees with a high degree of occlusion with other objects and intersect with other trees. Although including highly overlapped trees (up to 97% overlapped in BEV) inevitably degraded the detection performances of baseline networks, it is essential to include these problematic cases because they are prevalent in urban green spaces such as parks and woodlots. Akin to pedestrian detection in crowded scenes, occlusion is one of the most complicated problems to solve for tree detection. The decreasing detection accuracy with an increasing percentage of the overlap was observed in the testing of this dataset. The problem solving of this occlusion can be further pursued to accurate tree detection in areas where trees are overlapped.

Also, this dataset is particularly valuable because it includes field data with additional information about the trees' species, DBH, and other conditions. The annotation is built based on fieldwork. The disadvantage of generating annotations solely based on remotely sensed data is that the annotated bounding boxes will be biased on what can be perceived from the data rather than capturing the reality. Especially if trees grow close to each other, prior information such as the number and location of treetops through fieldwork will increase annotation accuracy. This information can contribute to generate classification of the 3D object detection in the future. The presented workflow may be applied to larger-scale annotation projects, and future studies will include the enlargement of the YUTO 5000 Tree.

The testing results proved that the two-stage detectors perform better than One-stage due to the refinement of the second stage. However, to achieve this, it was necessary to adjust the NMS settings at the test stage, allowing more proposal boxes to be proposed for the second refining stage. To improve these results, as it was presented in the chapter 5, an additional network module was designed to improve tree detection. However, the module can be more elaborated, for example, the smooth L1 loss function may be replaced with entirely new and more effective loss function with other method such as focal loss [93] or Gaussian based pooling method [94] can be utilized to add a layer.

Bibliography

[1] I. S. Alemdag, “Estimating ovendry mass of trembling aspen and white birch using measurements from aerial photographs,” *https://doi.org/10.1139/x86-030*, vol. 16, no. 1, pp. 163–165, 2011, doi: 10.1139/X86-030.

[2] D. G. Pitt and G. R. Glover, “Large-scale 35-mm aerial photographs for assessment of vegetation-management research plots in eastern Canada,” *https://doi.org/10.1139/x93-269*, vol. 23, no. 10, pp. 2159–2169, 2011, doi: 10.1139/X93-269.

[3] W. Zhang, Y. Ke, L. J. Quackenbush, and L. Zhang, “Using error-in-variable regression to predict tree diameter and crown width from remotely sensed imagery,” *https://doi.org/10.1139/X10-073*, vol. 40, no. 6, pp. 1095–1108, Jun. 2010, doi: 10.1139/X10-073.

[4] S. C. Popescu, “Estimating biomass of individual pine trees using airborne lidar,” *Biomass Bioenergy*, vol. 31, no. 9, pp. 646–655, Sep. 2007, doi: 10.1016/J.BIOMBIOE.2007.06.022.

[5] D. G. Leckie, F. A. Gougeon, S. Tinis, T. Nelson, C. N. Burnett, and D. Paradine, “Automated tree recognition in old growth conifer stands with high resolution digital imagery,” *Remote Sens Environ*, vol. 94, no. 3, pp. 311–326, Feb. 2005, doi: 10.1016/J.RSE.2004.10.011.

[6] K. Dralle and M. Rudemo, “Stem number estimation by kernel smoothing of aerial photos,” *https://doi.org/10.1139/x26-137*, vol. 26, no. 7, pp. 1228–1236, 2011, doi: 10.1139/X26-137.

[7] M. Hirschmugl, M. Ofner, J. Raggam, and M. Schardt, “Single tree detection in very high resolution remote sensing data,” *Remote Sens Environ*, vol. 110, no. 4, pp. 533–544, Oct. 2007, doi: 10.1016/J.RSE.2007.02.029.

[8] J. Hyyppä, H. Hyyppä, D. Leckie, F. Gougeon, X. Yu, and M. Maltamo, “Review of methods of small-footprint airborne laser scanning for extracting forest inventory data in boreal forests,” *https://doi.org/10.1080/01431160701736489*, vol. 29, no. 5, pp. 1339–1366, Mar. 2008, doi: 10.1080/01431160701736489.

[9] C. Song, M. B. Dickinson, L. Su, S. Zhang, and D. Yaussey, “Estimating average tree crown size using spatial information from Ikonos and QuickBird images: Across-sensor and across-site comparisons,” *Remote Sens Environ*, vol. 114, no. 5, pp. 1099–1107, May 2010, doi: 10.1016/J.RSE.2009.12.022.

[10] J. P. Ardila, W. Bijker, V. A. Tolpekin, and A. Stein, “Context-sensitive extraction of tree crown objects in urban areas using VHR satellite images,” *International Journal of Applied Earth Observation and Geoinformation (JAG)*, vol. 15, no. 1, pp. 57–69, 2012, doi: 10.1016/J.JAG.2011.06.005.

[11] B. Ruzena and J. J. Gibson, “Active, perception,” *Proceedings, of, the, IEEE,, volume*, vol. 76, pp. 996–1005, 1988.

[12] D. H. Ballard, “Animate vision,” *Artif Intell*, vol. 48, no. 1, pp. 57–86, Feb. 1991, doi: 10.1016/0004-3702(91)90080-4.

[13] J. Hyyppä and M. Inkinen, “DETECTING AND ESTIMATING ATTRIBUTES FOR SINGLE TREES USING LASER SCANNER,” 2006.

[14] E. Naesset, “Airborne laser scanning as a method in operational forest inventory: Status of accuracy assessments accomplished in Scandinavia,” *Scand J For Res*, vol. 22, no. 5, pp. 433–442, 2007, doi: 10.1080/02827580701672147.

[15] B. Koch, U. Heyder, and H. Welnacker, “Detection of individual tree crowns in airborne lidar data,” *Photogramm Eng Remote Sensing*, vol. 72, no. 4, pp. 357–363, 2006, doi: 10.14358/PERS.72.4.357.

[16] S. Solberg, E. Naesset, and O. M. Bollandsas, “Single Tree Segmentation Using Airborne Laser Scanner Data in a Structurally Heterogeneous Spruce Forest,” *Photogramm Eng Remote Sensing*, vol. 72, no. 12, pp. 1369–1378, 2006, doi: 10.14358/PERS.72.12.1369.

[17] K. Calders *et al.*, “Terrestrial laser scanning in forest ecology: Expanding the horizon,” *Remote Sens Environ*, vol. 251, Dec. 2020, doi: 10.1016/j.rse.2020.112102.

[18] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” Jul. 2022, doi: 10.48550/arxiv.2207.02696.

[19] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, 2012, doi: 10.1109/CVPR.2012.6248074.

[20] A. Patil, S. Malla, H. Gang, and Y. T. Chen, “The H3D dataset for full-surround 3D multi-object detection and tracking in crowded urban scenes,” *Proc IEEE Int Conf Robot Autom*, vol. 2019-May, pp. 9552–9557, May 2019, doi: 10.1109/ICRA.2019.8793925.

[21] P. Sun *et al.*, “Scalability in Perception for Autonomous Driving: Waymo Open Dataset,” *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 2446–2454, 2020. Accessed: Aug. 25, 2022. [Online]. Available: http://www.waymo.com/open.

[22] H. Caesar *et al.*, “nuScenes: A multimodal dataset for autonomous driving,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 11618–11628, Mar. 2019, doi: 10.48550/arxiv.1903.11027.

[23] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, “Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, no. 1W1, pp. 91–98, May 2017, doi: 10.5194/ISPRS-ANNALS-IV-1-W1-91-2017.

[24] X. Roynard, J.-E. Deschaud, and F. Goulette, “Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification,” *Int J Rob Res*, vol. 37, no. 6, pp. 545–557, 2018.

[25] J. Behley *et al.*, “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV*, 2019, pp. 9296–9306.

[26] W. Tan *et al.*, “Toronto-3D: A Large-scale Mobile LiDAR Dataset for Semantic Segmentation of Urban Roadways,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 797–806.

[27] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans Pattern Anal Mach Intell*, vol. 42, no. 2, pp. 386–397, Mar. 2017, doi: 10.48550/arxiv.1703.06870.

[28] S. Lumnitz, T. Devisscher, J. R. Mayaud, V. Radic, N. C. Coops, and V. C. Griess, "Mapping trees along urban street networks with deep learning and street-level imagery," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 175, pp. 144–157, May 2021, doi: 10.1016/J.ISPRSJPRS.2021.01.016.

[29] S. Lumnitz, "Mapping urban trees with deep learning and street-level imagery," 2019, doi: 10.14288/1.0387513.

[30] H. Kaartinen and J. Hyyppä, "EuroSDR/ISPRS Project, Commission II 'Tree Extraction'; Final Report; Official Publication 53; EuroSDR (European Spatial Data Research)," *Dublin, Ireland*, 2008.

[31] H. Kaartinen *et al.*, "An International Comparison of Individual Tree Detection and Extraction Using Airborne Laser Scanning," *Remote Sens (Basel)*, vol. 4, no. 4, pp. 950–974, 2012.

[32] J. Vauhkonen *et al.*, "Comparative testing of single-tree detection algorithms under different types of forest," *Forestry: An International Journal of Forest Research*, vol. 85, no. 1, pp. 27–40, Jan. 2012, doi: 10.1093/FORESTRY/CPR051.

[33] K. Connie, J. Yeonjeong, L. Hyungju, and S. Gunho, "YUTO Tree-5000 Benchmark," in *43rd Canadian Symposium on Remote Sensing, 10th International Conference on Agro-Geoinformatics*, Jul. 2022.

[34] L. Eysn *et al.*, “A Benchmark of Lidar-Based Single Tree Detection Methods Using Heterogeneous Forest Data from the Alpine Space,” *Forests 2015, Vol. 6, Pages 1721-1747*, vol. 6, no. 5, pp. 1721–1747, May 2015, doi: 10.3390/F6051721.

[35] X. Liang *et al.*, “International benchmarking of terrestrial laser scanning approaches for forest inventories,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 144, pp. 137–179, Oct. 2018, doi: 10.1016/J.ISPRSJPRS.2018.06.021.

[36] B. G. Weinstein, S. Marconi, M. Aubry-Kientz, G. Vincent, H. Senyondo, and E. P. White, “DeepForest: A Python package for RGB deep learning tree crown delineation,” *Methods Ecol Evol*, vol. 11, no. 12, pp. 1743–1751, Dec. 2020, doi: 10.1111/2041-210X.13472.

[37] S. Schmohl, A. N. Vallejo, and U. Soergel, “Individual Tree Detection in Urban ALS Point Clouds with 3D Convolutional Networks,” *Remote Sensing 2022, Vol. 14, Page 1317*, vol. 14, no. 6, p. 1317, Mar. 2022, doi: 10.3390/RS14061317.

[38] L. Najman and M. Schmitt, “Watershed of a Continuous Function,” 1994, doi: 10.1016/0165-1684(94)90059-0ï.

[39] L. Jing, B. Hu, T. Noland, and J. Li, “An individual tree crown delineation method based on multi-scale segmentation of imagery,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 70, pp. 88–98, Jun. 2012, doi: 10.1016/J.ISPRSJPRS.2012.04.003.

[40] H. Huang, X. Li, and C. Chen, “Individual tree crown detection and delineation from very-high-resolution UAV images based on bias field and marker-controlled watershed segmentation algorithms,” *IEEE J Sel Top Appl Earth Obs Remote Sens*, vol. 11, no. 7, pp. 2253–2262, Jul. 2018, doi: 10.1109/JSTARS.2018.2830410.

[41] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, “Occlusion-aware R-CNN: Detecting Pedestrians in a Crowd.” pp. 637–653, 2018.

[42] B. Li, “3D Fully Convolutional Network for Vehicle Detection in Point Cloud,” *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-September, pp. 1513–1518, Nov. 2016, doi: 10.48550/arxiv.1611.08069.

[43] B. Yang, W. Luo, and R. Urtasun, “PIXOR: Real-Time 3D Object Detection From Point Clouds.” pp. 7652–7660, 2018.

[44] Y. Zhou and O. Tuzel, “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection.” pp. 4490–4499, 2018.

[45] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, “From Points to Parts: 3D Object Detection from Point Cloud with Part-Aware and Part-Aggregation Network,” *IEEE Trans Pattern Anal Mach Intell*, vol. 43, no. 8, pp. 2647–2664, Aug. 2021, doi: 10.1109/TPAMI.2020.2977026.

[46] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum PointNets for 3D Object Detection from RGB-D Data,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 918–927, Nov. 2017, doi: 10.48550/arxiv.1711.08488.

[47] Y. Chen, S. Liu, X. Shen, and J. Jia, “Fast Point R-CNN.” pp. 9775–9784, 2019.

[48] C. R. Qi, O. Litany, K. He, and L. J. Guibas, “Deep Hough Voting for 3D Object Detection in Point Clouds”.

[49] Z. Yang, Y. Sun, S. Liu, X. Shen, J. Jia, and Y. Lab, “STD: Sparse-to-Dense 3D Object Detector for Point Cloud.” pp. 1951–1960, 2019.

[50] S. Shi, X. Wang, and H. Li, “PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud.” pp. 770–779, 2019. Accessed: Aug. 25, 2022. [Online]. Available: https://github.com/sshaoshuai/PointRCNN.

[51] A. Jaiswal, Y. Wu, P. Natarajan, and P. Natarajan, “Class-agnostic Object Detection,” *Proceedings - 2021 IEEE Winter Conference on Applications of Computer Vision, WACV 2021*, pp. 918–927, Nov. 2020, doi: 10.48550/arxiv.2011.14204.

[52] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)”, Accessed: Nov. 14, 2022. [Online]. Available: http://www.cs.berkeley.edu/˜rbg/rcnn.

[53] R. Girshick, “Fast R-CNN,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. ICCV 2015, pp. 1440–1448, Feb. 2015.

[54] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *Adv Neural Inf Process Syst*, vol. 28, 2015, Accessed: Aug. 25, 2022. [Online]. Available: https://github.com/

[55] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, Accessed: Nov. 14, 2022. [Online]. Available: http://pjreddie.com/yolo/

[56] L. Fan, X. Xiong, F. Wang, N. Wang, and Z. Zhang, “RangeDet: In Defense of Range View for LiDAR-Based 3D Object Detection.” pp. 2918–2927, 2021. Accessed: Aug. 25, 2022. [Online]. Available: https://github.com/TuSimple/RangeDet.

[57] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, “LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving.” pp. 12677–12686, 2019.

[58] A. Bewley, P. Sun, T. Mensink, G. Research, D. Anguelov, and C. Sminchisescu, “Range Conditioned Dilated Convolutions for Scale Invariant 3D Object Detection,” May 2020, doi: 10.48550/arxiv.2005.09927.

[59] R. Ge *et al.*, “AFDet: Anchor Free One Stage 3D Object Detection,” Jun. 2020, doi: 10.48550/arxiv.2006.12671.

[60] P. Sun *et al.*, “RSN: Range Sparse Net for Efficient, Accurate LiDAR 3D Object Detection.” pp. 5725–5734, 2021.

[61] T. Yin, X. Zhou, and P. Krähenbühl, “Center-based 3D Object Detection and Tracking,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 11779–11788, Jun. 2021, doi: 10.1109/CVPR46437.2021.01161.

[62] Y. Yan, Y. Mao, and B. Li, “SECOND: Sparsely Embedded Convolutional Detection,” *Sensors 2018, Vol. 18, Page 3337*, vol. 18, no. 10, p. 3337, Oct. 2018, doi: 10.3390/S18103337.

[63] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” *Adv Neural Inf Process Syst*, vol. 30, 2017.

[64] C. Hug, P. Krzystek, and W. Fuchs, “ADVANCED LIDAR DATA PROCESSING WITH LASTOOLS”.

[65] Terrasolid, “TerraScan.” Accessed: Aug. 25, 2022. [Online]. Available: https://terrasolid.com/products/terrascan

[66] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, “Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection,” *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, vol. 2A, pp. 1201–1209, Dec. 2020, doi: 10.48550/arxiv.2012.15712.

[67] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. V. R.-C. N. N. Li, “Towards High Performance Voxel-based 3D Object Detection.”

[68] Y. Hu *et al.*, “AFDetV2: Rethinking the Necessity of the Second Stage for Object Detection from Point Clouds,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, pp. 969–979, Jun. 2022, doi: 10.1609/AAAI.V36I1.19980.

[69] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “PointPillars: Fast Encoders for Object Detection From Point Clouds.” pp. 12697–12705, 2019. Accessed: Aug. 26, 2022. [Online]. Available: https://github.com/nutonomy/second.pytorch

[70] W. Zheng, W. Tang, S. Chen, L. Jiang, and C.-W. Fu, “CIA-SSD: Confident IoU-Aware Single-Stage Object Detector From Point Cloud,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 4, pp. 3555–3562, May 2021, Accessed: Aug. 26, 2022. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16470

[71] J. Mao, M. Niu, H. Bai, X. Liang, H. Xu, and C. Xu, “Pyramid R-CNN: Towards Better Performance and Adaptability for 3D Object Detection.” pp. 2723–2732, 2021.

[72] S. Shi *et al.*, “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection.” pp. 10529–10538, 2020.

[73] S. Shi *et al.*, “PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection,” Jan. 2021, doi: 10.48550/arxiv.2102.00463.

[74] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” pp. 580–587, 2014. Accessed: Aug. 26, 2022. [Online]. Available: http://arxiv.

[75] OpenPCDet Development Team, “OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds.” 2020. Accessed: Aug. 26, 2022. [Online]. Available: https://github.com/open-mmlab/OpenPCDet

[76] “Precision-Recall Curve | ML - GeeksforGeeks.” https://www.geeksforgeeks.org/precision-recall-curve-ml/ (accessed Dec. 18, 2022).

[77] D. P. Kingma and J. L. Ba, “Adam: A Method for Stochastic Optimization,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Dec. 2014, doi: 10.48550/arxiv.1412.6980.

[78] J. Duchi and Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[79] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” Dec. 2012, doi: 10.48550/arxiv.1212.5701.

[80] T. Tieleman, G. Hinton, and others, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[81] K. Krishna and M. N. Murty, “Genetic K-means algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 29, no. 3, pp. 433–439, 1999, doi: 10.1109/3477.764879.

[82] D. Arthur and S. Vassilvitskii, “k-means++: The Advantages of Careful Seeding”.

[83] Y. Liu and L. Jin, “Deep Matching Prior Network: Toward Tighter Multi-Oriented Text Detection.” pp. 1962–1969, 2017.

[84] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2.” 2019. Accessed: Oct. 10, 2022. [Online]. Available: https://github.com/facebookresearch/detectron2

[85] “GitHub - facebookresearch/fvcore: Collection of common code that’s shared among different research projects in FAIR computer vision team.” https://github.com/facebookresearch/fvcore (accessed Oct. 24, 2022).

[86] N. Sharma, “3D-FCT: Simultaneous 3D object detection and tracking using feature correlation”.

[87] S. Magnussen and P. Boudewyn, “Derivations of stand heights from airborne laser scanner data with canopy-based quantile estimators,” *https://doi.org/10.1139/x98-078*, vol. 28, no. 7, pp. 1016–1031, 2011, doi: 10.1139/X98-078.

[88] J. E. Means, S. A. Acker, B. J. Fitt, M. Renslow, L. Emerson, and C. J. Hendrix, "Predicting Forest Stand Characteristics with Airborne Scanning Lidar," 2000, Accessed: Oct. 24, 2022. [Online]. Available: http://www.sbgmaps.com/lidar_technologies.htm

[89] X. Song *et al.*, "ApolloCar3D: A Large 3D Car Instance Understanding Benchmark for Autonomous Driving." pp. 5452–5462, 2019.

[90] G. Brolly, G. Király, M. Lehtomäki, and X. Liang, "Voxel-Based Automatic Tree Detection and Parameter Retrieval from Terrestrial Laser Scans for Plot-Wise Forest Inventory," *Remote Sensing 2021, Vol. 13, Page 542*, vol. 13, no. 4, p. 542, Feb. 2021, doi: 10.3390/RS13040542.

[91] F. Morsdorf, E. Meier, B. Kötz, K. I. Itten, M. Dobbertin, and B. Allgöwer, "LIDAR-based geometric reconstruction of boreal type forest stands at single tree level for forest and wildland fire management," *Remote Sens Environ*, vol. 92, no. 3, pp. 353–362, Aug. 2004, doi: 10.1016/J.RSE.2004.05.013.

[92] S. C. Popescu and K. Zhao, "A voxel-based lidar method for estimating crown base height for deciduous and pine trees," *Remote Sens Environ*, vol. 112, no. 3, pp. 767–781, Mar. 2008, doi: 10.1016/J.RSE.2007.06.011.

[93] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *IEEE Trans Pattern Anal Mach Intell*, vol. 42, no. 2, pp. 318–327, Aug. 2017, doi: 10.48550/arxiv.1708.02002.

[94] T. Kobayashi, "Gaussian-Based Pooling for Convolutional Neural Networks", Accessed: Oct. 24, 2022. [Online]. Available: https://github.com/tk1980/GaussianPooling.