DEEP LEARNING FOR DETECTING TREES IN THE URBAN ENVIRONMENT FROM LIDAR

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Julian Rice

August 2022

COMMITTEE MEMBERSHIP

TITLE: Deep Learning for Detecting Trees in the Urban Environment from LIDAR

AUTHOR: Julian Rice

DATE SUBMITTED: August 2022

COMMITTEE CHAIR: Jonathan Ventura, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: G. Andrew Fricker, Ph.D.
Professor of Geography

ABSTRACT

Deep Learning for Detecting Trees in the

Urban Environment from LIDAR

Julian Rice

Cataloguing and classifying trees in the urban environment is a crucial step in urban and environmental planning. However, manual collection and maintenance of this data is expensive and time-consuming. Algorithmic approaches that rely on remote sensing data have been developed for tree detection in forests, though they generally struggle in the more varied urban environment. This work proposes a novel method for the detection of trees in the urban environment that applies deep learning to remote sensing data. Specifically, we train a PointNet-based neural network to predict tree locations directly from LIDAR data augmented with multi-spectral imaging. We compare this model to numerous high-performant baselines on a large and varied dataset in the Southern California region. We find that our best model outperforms all baselines with a 75.5% F-score and 2.28 meter RMSE, while being highly efficient. We then analyze and compare the sources of errors, and how these reveal the strengths and weaknesses of each approach.

ACKNOWLEDGMENTS

Thanks to Dr. Ventura, for your guidance and insight throughout this project. Thanks also to Cami Pawlak, Cameron Gonsalves, Milo Honsberger, Skyler Han, Viet Nguyen and all other students who assisted with data acquisition, cleanup, and annotation, and worked on preceding and related projects which made my work easier. Last but not least, thank you all to my friends and family who made this possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Overview & Motivations

Trees hold a crucial place in our local and global ecosystems, no matter where we live. Even in the urban environment, where we tend not to take particular notice of the trees around us, they serve a variety of beneficial purposes, including mitigating pollution, improving home heating and cooling efficiency, and creating outdoor social spaces [1].

Identification and classification of trees is a crucial component of accurate, data-driven forestry and urban management. In forestry, this includes applications for tree biodiversity analysis and monitoring [59], and wildlife habitat assessment and protection [26]. In urban areas, tree inventories can be utilized in similar ways, and further assist with urban and environmental planning [69]. While rural forests have been researched extensively, urban forests have generally been less of a focus, and accurate monitoring of trees is key to improving our understanding especially as the climate changes [54]. Further, accurate tree inventories and monitoring is crucial for unlocking trees' own positive environmental impacts; research has shown that planting trees does not automatically result in a positive environmental impact (particularly in the urban environment), and that selection of ideal planting locations and monitoring throughout trees' lifetimes can be key to realizing their potential [44]. Yet, keeping an up-to-date record via manual surveying of trees is difficult given the prohibitive associated costs and the ever-changing nature of any forest [18]—a manual survey

of all street trees (i.e. not including trees in backyards or on private land) in Los Angeles required 18 months of effort and cost $2 million [6].

Because of these issues, the task of automating tree detection and classification has been of interest to researchers for decades [19]. This approach has been made possible by the advent and widespread collection of a wide variety of remote sensing data sources in recent decades, including multi-band and multi-perspective imaging, LIDAR, and RADAR, usually collected from aircraft, UAV, or satellite [36]. Alone as well as in combination, these sources capture a wide spectrum of features of trees and their environments. From these features, we can attempt to firstly identify where trees occur, and subsequently may be interested in determining attributes of them such species, size, health, foliage coverage, or growth over time [36].

Although hand-crafted algorithmic approaches to these problems have been developed [18], the size and novel nature of these data make data-driven machine learning approaches appealing. However, these new data sources bring with them new challenges. For example, aerial LIDAR captures surfaces in the form of an un-ordered cloud of points in XYZ space, a format which requires more sophisticated algorithms that tend to be computationally expensive [71][36] (compared to, for example, image analysis). Further challenges are introduced in the urban environment for all data sources, because of the relative sparsity, irregular distribution, and heterogeneity of its trees, not to mention the myriad of human-made objects and structures with which the trees coexist (and may appear similar to). Still, it is hypothesized that LIDAR represents the most promising path toward successfully detecting and analyzing individual trees in the urban environment, especially when combined with imagery sources [36].

Given the recent and dramatic rise in the effectiveness of deep learning neural networks for a variety of tasks [33][31], it is natural to question whether they could

be useful in this domain as well. Various deep learning methods have in fact been proposed that utilize various forms of imagery data, for both rural [19][51][59][47], and to a lesser extent urban [3] forests. However, with the advent of networks designed specifically to operate on pointclouds [56], new deep learning approaches to this problem that utilize raw LIDAR are appealing, but little-explored so far.

1.2 Objectives & Approach

In this work, we are interested in applying deep neural networks directly to LIDAR data augmented by multi-spectral imagery. Though many previous tree detection approaches are premised on generating some form of raster (image) from LIDAR [25][24][66], this is a destructive operation by nature which may obscure useful information contained in the data [71]. We instead evaluate approaches to deep learning for tree detection that operate directly on LIDAR from the urban environment.

1.2.1 Research Questions

- RQ1: How effectively can deep learning be applied directly to LIDAR pointclouds to detect trees in the urban environment?
- RQ2: What structures in the urban environment are difficult to distinguish from trees?
- RQ3: Can deep learning for LIDAR pointclouds provide information that is complementary to that from other methods?

1.2.2 Hypothesis

A sufficiently well-tuned deep neural network, learning directly from imagery-augmented LIDAR, will outperform other methods for tree detection in the urban environment including:

1. methods which do not utilize deep neural networks,
2. methods which only utilize spectral data, and
3. methods which utilize LIDAR in a summarized form (e.g. rasters).

1.3 Contributions

We develop a technique for urban tree detection that is efficient and effective. We find that:

- Our method outperforms our baselines in terms of predictive capability, achieving the best F-score of 75.5% on our test set.
- Our method outperforms our baselines in terms of localization accuracy, achieving a best root-mean-squared error of 2.28 meters on true positive predictions.
- Our method is significantly more efficient than the other deep learning baseline we compare against.
- Our method synthesizes deep learning techniques developed in a variety of disparate fields, including microscopy, crowd-counting, remote sensing, and forestry.

- Our method is capable of achieving high performance from publicly available datasets.

Chapter 2

BACKGROUND

This chapter will provide a brief overview of common background information, techniques, and terminology from forestry and deep learning that will be required for understanding future chapters. Those with a background in deep learning may find it useful to read the section on remote sensing for forestry, and vice versa.

Table 2.1: Common acronyms & terms

Term	Description
LIDAR	Light Detection And Ranging: pointcloud data
RGB	Red, Green, and Blue, the visible light channels
NIR	Near-Infrared: a non-visible light channel
NDVI	Normalized Difference Vegetation Index, a measure of foliage
CHM	Canopy Height Model, a raster of tree height
ANN	Artificial Neural Network, also abbreviated as NN
CNN	Convolutional Neural Network, an ANN specialized for images
Dense	Densely connected layer, a simple neural network layer
ReLU	Rectified Linear Unit, the function $f(x) = \max(0, x)$

2.1 Remote Sensing for Forestry

Remote sensing refers to the collection and use of geo-referenced data gathered remotely, usually by aircraft or satellite. Remote sensing data can assist in many tasks of interest to the forestry community, including assessing forest size and diversity, calculating urban green volume, and monitoring forests over time. Perhaps the two most common forms of remote sensing data utilized for forestry are imagery and LIDAR.

Remote sensing imagery can take many forms, usually captured from satellite. The most common is standard three-band RGB (Red-Green-Blue) imagery, which captures

the same spectrum of information that our eyes perceive. We will use the term "multi-spectral" (or "multi-band") to refer to imagery that contains additional wavelengths outside of the RGB channels; a common addition is near-infrared (NIR) channel. We will further use "hyper-spectral" to refer to imagery with a much larger number of bands, in the dozens or hundreds (although the literature is inconsistent in this terminology).

Although our eyes cannot perceive NIR wavelengths, they are quite useful because photosynthetic cells (the cells that make plant leaves green) tend to strongly reflect NIR, while strongly absorbing red and blue wavelengths. Given this unique trait of plant greenery, we can calculate a measure called the Normalized Difference Vegetation Index (NDVI), which utilizes the NIR and Red bands to give a metric of how much leafy vegetation is present in each pixel [30]:

$$\text{NDVI} = \frac{\text{NIR} - \text{Red}}{\text{NIR} + \text{Red}} \tag{2.1}$$

This metric ranges from -1 to 1, where values close to 1 signify leafy vegetation, and values close to or less than 0 tend to be other objects. It is of course not a foolproof method of detecting trees for a number of reasons: deciduous trees that have lost their leaves in the winter will not register high values; non-trees such as grass and shrubs will register high values; and interference from clouds, pollution, and shadows can result in misleading values.

Another common form of remote sensing data is LIDAR, or Light Detection and Ranging. LIDAR is analogous to radar in that is uses the response time of waves to determine the distance from the source to some reflective surface. The difference is that while RADAR uses radio waves, LIDAR uses light waves. LIDAR, which is usually captured by piloted or unpiloted aircraft, returns a nominally unordered set

of points in XYZ space where each point represents a location where a light ray was reflected from a surface back to the source. These points do not have even spacing or density (especially since the aircraft passes can overlap in some areas but not others). They are also subject to noise in measurement, as well as detecting transient objects such as birds or small surfaces such as telephone wires.

Because of the unwieldy nature of raw LIDAR, a common approach is to rasterize it, which converts it to an image-like format, where each pixel represents some property of the spatial extent of the pixel. Commonly generated rasters are the Digital Terrain Model (DTM), which measures the absolute height of the ground, the Digital Surface Model (DSM), which measures the absolute height of the tallest object, and the Canopy Height Model (CHM), which is the height of the tallest object relative to the ground, and is computed as the difference between the DSM and DTM. It is most commonly the CHM that is then used for further forestry applications.

Literature reviews [15, 36] note that the most multi-model approachs—those which combine multiple sources of data, particularly LIDAR and multi-spectral imagery—are most promising for challenging forestry problems.

2.2 Deep Learning

Although the purpose of this work is not to provide an in-depth explanation of the basis of deep learning, a general background is provided here. For a more involved discussion of the history and workings of deep learning, we recommend Lecun et al. 2015 [33].

Deep learning refers to the use of artificial neural networks that are able to learn their own parameters. Similar to the way a linear regression is able to approximate

the data to which it is fit, neural networks can learn to approximate functions, albeit usually much more complicated functions than a line of best fit. Artificial neural networks (ANNs) are called such because they mimic the way we understand the human brain's neurons to transmit information and learn, though the term "artificial" is often dropped in the literature since it is implied.

Given a set of numeric input features, and some numeric target labels, a typical deep learning approach will be to train a neural network to predict the correct label for each input. A typical neural network accepts a vector of real numbers as input, and applies a variety of operations with learned weights to generate an output vector. A commonly used operation is the matrix multiplication of the input vector by a matrix of learned weights, and the addition of a learned bias term. Each of these operations is then commonly followed by the application of a non-linear function element-wise to the output vector, called an "activation function" in analogy to the way neurons are activated in the human brain. The weight operation and nonlinearity together are generally called one "layer" in the network; when many such layers are applied one after another, the network is considered deep (though there is no well-defined threshold for when a network becomes deep). We will refer to this particular type of layer as a Dense layer, which gets its name from the fact that each output element is affected by every input element, and thus in a graph representation the network is very densely connected. This layer is alternately called a fully-connected layer, and many of them together is often called a multi-layer perceptron or feed-foward network in the literature. Another common type of neural network is convolutional neural network (CNN), which is primarily used on image data. These networks function by learning sliding window filters that identify patterns in gridded data, like RGB images.

A neural network is trained to minimize the error between its predictions and the target labels. This is done via the Backpropogation algorithm [23], which determines

how much each individual weight contributed to the overall error by taking the partial derivative of the error with respect to each weight, and adjusting them accordingly. This is done repeatedly until the weights have converged to a (potentially local) minima in the loss space. Commonly, a subset of the data is held out as a validation set, and is evaluated after each iteration, to ensure that the model is not over-fitting to its training data. Note that the training process outlined here is a simplified abstraction; in practice, it is complicated by learning on mini-batches, using more advanced weight-update algorithms that incorporate adaptive learning rates and momentum, and other enhancements such as batch normalization and dropout.

Chapter 3

RELATED WORK

This chapter will provide an overview of relevant research conducted along two fronts: Remote Sensing for Trees (Section 3.1), and Pointcloud Deep Learning (Section 3.2). Lastly, we will introduce the methods that are bridging the gap between the two, in Section 3.3. Remote Sensing for Trees will cover various algorithmic, statistical, machine learning, and deep learning approaches to tasks such as tree detection (identifying which areas contain trees), delineation (identifying individual tree locations or tree crowns), and classification (identifying traits of individual trees, usually family or species, though attempting to identify size, age, or any other attribute is possible as well). In our work, we are primarily interested in tree detection and delineation, which is a necessary precursor to classification. Pointcloud Deep Learning will cover various approaches to deep learning applied pointcloud data.

3.1 Remote Sensing for Forestry

Remote sensing for forestry has a long history, dating back to at least the 1970's with the launch of Landsat-1 [8]. Boyd & Danson [8] in 2005 review the first three decades of forestry applications of satellite remote sensing, finding that it had enabled three primary avenues of research: "(1) the spatial extent of forest cover, which can be used to assess the spatial dynamics of forest cover; (2) forest type and (3) biophysical and biochemical properties of forests." More recent surveys find that improved spatial resolution has led to increased work on individual tree-level metrics (as opposed to focus on forests or stands)[18], and an increased prevalence of fusing

data from different sources and scales particularly in the urban environment [36]. We seek to continue these trends, by introducing a new method for individual tree detection in the urban forest.

This section covers research into techniques that utilize remote sensing data to achieve various tree detection, delineation, and classification tasks, in both urban and rural forests.

3.1.1 Algorithmic & Statistical Methods

Various algorithmic, statistical, and traditional machine learning methods have been utilized for forestry applications. In recent decades, the number of publications dealing with such tasks have been steadily increasing, suggesting an increased interest in these problems [18].

3.1.1.1 Methods for Imagery

One common approach is to predict tree features directly from multi-spectral imagery. The most basic method is to use spectral analysis to identify which pixels contain trees, and classify those trees' species based on their unique spectral properties; this is the approach taken by Xiao et al. [69], applied to relatively low spatial resolution (3.5m/pixel) imagery in urban Modesto, CA. More recently, Alonzo et al. [3] and Jensen et al. [27] applied statistical methods to various spectral properties to classify individual tree species in urban environments. Shang & Chisolm [59] instead compared multiple traditional machine learning techniques—support-vector machine (SVM), AdaBoost, and random forest—against traditional statistical methods, using multi-spectral imagery as input. They found these techniques outperformed statisti-

cal methods for species classification in the challenging region of the Australian rural forest, which contains many varieties of similar species.

A less common approach with imagery data is to instead use multi-view imagery; that is, imagery that contains multiple different perspectives of the same tree. Perhaps the most comprehensive dataset of this kind to date, from Beery et al. [6], combines Google Maps imagery with one or more Google Street View images of each tree.

3.1.1.2 Methods for LIDAR

Alternatively, LIDAR-based approaches seek to identify trees by structure, instead of spectral reflectance. Detection and classification based on LIDAR-derived CHMs is common. Early work 2000s by Popescu & Wynne [52, 53] found that LIDAR-derived CHMs are useful for the detection of trees, and estimation of individual tree properties such as height and crown size, as well as forest-scale metrics such as forest volume and biomass. The "watershed method" [45] has become a standard approach for tree detection and segmentation with CHMs [38, 60, 66]. Watershed methods involve finding local maxima in the CHM, segmenting or contouring them at saddle points and valleys, and combining or filtering these points with various algorithms to achieve final tree delineations. An efficient and well-performing modern example of the watershed method for tree detection is PyCrown [74].

However, it is obvious that rasterizing 3-dimensional LIDAR into a 2-dimensional CHM loses information that may be helpful, and thus researchers have been interested in techniques that operate more directly on LIDAR pointclouds. Li et al. [35] develop an approach that which claims to be the first to segment trees directly from a LIDAR pointcloud. They do so with a top-down method that works from the highest points down to the lowest, similar to the theory behind the watershed method. Jakubowski

et al. [25] evaluate this approach in comparison with a CHM watershedding method, and finds neither outperforms the other in all cases. Another technique that operates directly on pointclouds, by Ayrey et al. [4], is based on horizontally slicing the pointcloud at 1-meter intervals, clustering points within each slice, and aggregating the slices to generate delineated tree canopies. An obvious deficit of these LIDAR-only approaches is their struggle in delineating between trees and human-made structures, especially for non-learning methods like the watershed method. For this reason they are almost exclusively tested and utilized in rural instead of urban areas, where there are few buildings and thus nearly anything more than a few meters off the ground can be assumed to be a tree.

3.1.1.3 Methods for Multimodal Data

Perhaps most promising are the approaches that combine both LIDAR and imagery, especially in the urban environment where it cannot be assumed that high points represent treetops, as can be assumed in most rural forests. One of the simplest methods for combining LIDAR and spectral information is to filter a CHM by an aligned NDVI image. Huang et al. [24] utilize this approach, to detect trees for the calculation of urban green volume. Alternatively, instead of interpolating the LIDAR to the imagery, some researchers opt to go the other direction. Parmehr et al. [48] interpolate imagery-derived NDVI values to each LIDAR point, to assist in tree detection and delineation in the urban environment. Zhang et al. [71] develop a similar approach, first filtering LIDAR points by the closest associated NDVI pixel, then using a simple local maximum-based algorithm to determine treetops and crowns, which are further filtered. The obvious benefit of interpolating imagery to the LIDAR pointcloud is that the full 3-D structure of the LIDAR data is preserved [48].

3.1.2 Deep Learning Approaches

In recent years, with the remarkable success of deep learning in a wide variety of disciplines [33], a number of deep-learning approaches to tree detection, delineation, and classification have been developed. Specifically, researchers have been inspired to leverage the demonstrated success of convolutional neural networks (CNNs) [31, 33]. For example, Li et al. [34] applied CNNs to multi-spectral imagery for the purpose of palm oil tree detection and delineation. However, more approaches have been developed that leverage LIDAR in combination with this imagery. Fricker et al. [19] demonstrate the benefits of multi-spectral imagery over RGB imagery, by training CNNs to predict tree species on imagery filtered by a CHM to exclude pixels less than 5 meters above the ground; they find that the multi-spectral CNN outperforms the RGB model significantly. Onishi & Ise [47] also apply a CNN to RGB imagery, with a CHM being utilized in preprocessing. Pleșoianu et al. [51] create an ensemble of models trained on a variety of LIDAR raster-derived and imagery sources, finding that an ensemble of two different models often outperforms a model trained on just one data source (though why they did not compare this to a model trained on a stack of more than one data source is not exactly clear). In Weinstein et al. [65], LIDAR is used in a more inventive way: they use the CHM-based technique developed by Silva et al. [60] to generate tree crowns, which they treat as "noisy labels." They then pre-train a convolutional neural network on corresponding RGB images, with these noisy labels as the target, and finally fine tune the model on hand-annotated trees.

3.2 Pointcloud Deep Learning

3.2.1 Voxel Methods

The initial approaches to deep learning on pointclouds (aside from applying standard CNNs to rasters, as described in previous sections) was to convert the pointcloud to a 3D grid of "voxels" (the 3-dimensional extension of a pixel), where each voxel contains some feature of the pointcloud within its space, usually the point density or just a binary encoding of whether points are present in that voxel. These voxel grids could then be learned on by 3D convolutional neural networks [28, 43, 68]. However, and the 3D convolution is computationally expensive, and since these 3D voxel grids tend to be very sparse much of that computation is essentially wasted [56].

3.2.2 PointNet

Though deep learning had been applied to pointcloud-derived data (for example, LIDAR-derived CHMs) it was not until PointNet, by Qi et al. [56], that a deep learning technique was invented that was effective at operating directly on pointclouds. The main difficulty is that while neural networks had proven quite effective at learning from uniformly spaced and gridded information such as images, pointclouds contain spatial information but the points themselves are unstructured and ordered only by arbitrary collection order. This means any network that learns from them should be invariant to the order of the points. The key insight of PointNet is that this can be done by the use of symmetric functions; that is, functions like addition or maximum are invariant to the order of their inputs:

$$\max(a, ..., z) \equiv \max(z, ..., a)$$

PointNet thus learns by generating local feature independently for each input point, and then computing a global feature as the element-wise maximum of all the local feature vectors. To make their approach further invariant to orientation of the object described by the pointcloud, they included alignment sub-networks (also referred to as T-Nets) which learn to reorient pointclouds to a canonical orientation.

From this basic structure, the original PointNet authors propose two different output modes for two common tasks related to pointcloud learning:

- Classification, for classifying the scene as a whole. For example, we might want to know if a certain pointcloud represents a lamp or a chair. The PointNet authors propose applying a standard multi-layer perceptron network to the global feature vector followed by a softmax activation, to output a vector of representing a probability distribution over all possible classes.
- Segmentation, for classifying each point in the scene individually. For example, when there are multiple objects in a single scene, we may want to know which points represent the extents of which objects. The PointNet authors propose concatenating the global feature vector to each local feature, and then applying simple

PointNet++ (aka PointNet 2) [57] extended upon the original by iteratively grouping points, so that features could be learned for scales larger than single points but smaller than the scene as a whole. Specifically, three steps are used:

1. Sampling: Farthest-Point Samping (FPS) is used to select centroids in the pointcloud.
2. Grouping: Ball Query clustering is used to select all points in the neighborhood of each centroid. Ball Query clustering works by simply selecting all points

within a fixed distance of each centroid, which the authors find performs better than approaches like K-Nearest Neighbors.

3. Learning: A standard PointNet network is applied to each group, to learn a feature vector for each group.

These three stages, which together are called a Set Abstraction operation, can be repeated by taking the output features for each group as the new input points to the next layer. In the original PointNet++, they use two successive Set Abstraction layers before the final segmentation/classification output layers.

3.3 Pointcloud Deep Learning for Forestry

Only a few others have attempted to apply deep learning directly to LIDAR data (as opposed to LIDAR-derived rasters such as a CHM) for forestry purposes. Inspired by voxel-based methods, Ayrey et al. [5] voxelize LIDAR pointclouds, and apply 3D convolutional neural networks, testing this approach in rural forested areas in New England. However, they do note that training a single model takes upwards of one day, as might be expected given the computational complexity of 3-D convolutions. To the best of our knowledge, Chen et al. [10] is the only attempt to apply a PointNet-style network directly to tree detection and classification, which they do in both city parks and nearby rural forests. However, they do it in a somewhat odd way; they split the space it into a grid of large voxels in XY space, such that they are applying PointNet within a vertically-oriented rectangular prism roughly the size of one tree, and classify only whether a tree is present in that voxel or not. For the actual tree location and segmentation, another CHM-based local maxima algorithm was used, but only on the voxels that their PointNet labeled as containing trees. Thus, their method still does not exploit the full information contained in the pointcloud for tree

detection and segmentation. However, they do demonstrate the general ability of a PointNet-style network to discriminate between trees, buildings, and other objects. Methods operating directly on LIDAR have been applied to closely related tasks as well. For example, Liu et al. [37] detect trees using the watershed method and then use a novel vertically-segmented PointNet variant for species classification.

Chapter 4

METHODS

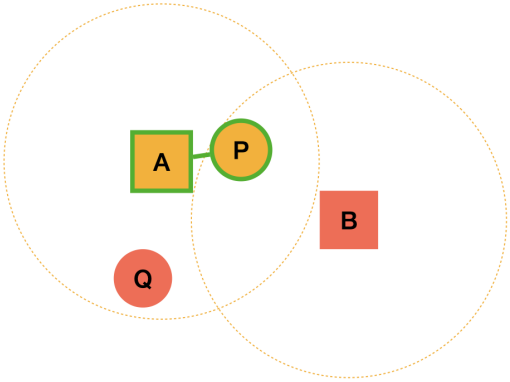
4.1 Overview

First, we define our evaluation metrics (Sec. 4.2) and dataset (Sec. 4.3). Then, we introduce our proposed method: a PointNet-based architecture for learning directly from spectrally-augmented LIDAR data, detailed in section 4.4. Section 4.5 details various competing methods which we compare against.

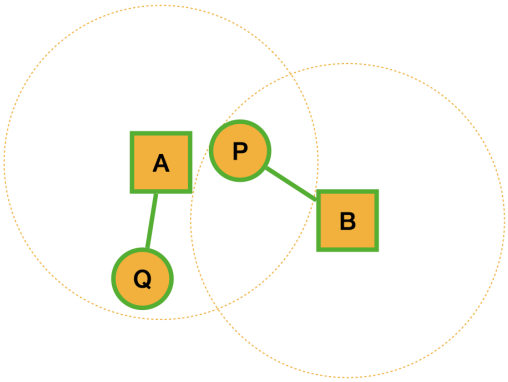
4.2 Point-Matching: A Method-Agnostic Scoring Metric

In order to fairly compare any two methods for urban tree detection, we define a simple metric that can be used to evaluate how well a set of predicted locations corresponds to actual tree locations. The goal is to find a 1-to-1 mapping between as many ground-truth and predicted trees as possible. We want a 1-to-1 mapping so that each predicted tree is matched with exactly one ground-truth tree; otherwise, a predicted tree could be interpreted as predicting the existence of greater than or less than one trees in its vicinity, making the predictions difficult to interpret. Our approach is inspired by similar metrics used in forestry [17] and crowd-counting [62]. To ensure predictions are sufficiently localized to the trees they are matched with, matches are only allowed within a certain Euclidean distance, for which we use 6 meters. This threshold is somewhat arbitrary; a larger threshold will in general give better matching scores, and a smaller threshold worse scores.

Informally, the algorithm seeks to find the most optimistic interpretation of a method's predictions. As an illustrative example, consider Fig. 4.1, where we assume two ground-truth trees, A and B (squares), and two predictions, P and Q (circles). If we use a sub-optimal matching scheme (Fig. 4.1a) by matching each prediction with its nearest ground-truth, both P and Q are assigned to A, with P being the only one matched (to preserve the 1-to-1 property) since it is closest. An optimal matching scheme on the other hand (Fig. 4.1b) matches P with B, since Q can be matched with A. Finding the optimal solution to this problem turns out to be a version of the linear assignment problem, for which efficient algorithms are already known [12].



(a) A sub-optimal matching. Predictions are circles, squares are ground-truth, and the dashed rings show the maximum match distance threshold for each ground-truth. Matches (true positives) have green borders and connecting lines, while incorrect predictions (false negative for B and false positive for Q) are red.



(b) An optimal matching. All are now true positive. Note that even though P is closer to A than B, matching it with B allows for a more optimal matching overall.

Figure 4.1: Illustration of Point-Matching

Formally, consider a set of ground-truth tree locations $Y = \{y_1, ..., y_m\}$, and predicted tree locations $\hat{Y} = \{\hat{y}_1, ..., \hat{y}_n\}$. Let D be the maximum distance threshold for matches, and d_{ij} be the distance between y_i and $\hat{y}_j$; thus we consider any assignment between two points where $d_{ij} > D$ to be "unmatched". We compute the optimal linear assignment—via the algorithm as described in [12] and implemented in Scikit-Learn

[50]—by evaluating the cost of matching y_i to $\hat{y}_j$ with the following function:

$$\text{Cost} = \begin{cases} d_{ij} & \text{if } d_{ij} \leq D \\ L & \text{otherwise, where } L \text{ is a large constant} \gg D \end{cases} \tag{4.1}$$

This cost function will always maximize the number of matches, since any reduction in the cost of unmatched points—case #2 in the cost function—will always outweigh any costs accumulated in case #1. We use $L = 10^{10}$, but anything a few orders of magnitude larger than D should be sufficient.

4.2.1 Derived Metrics

The point-matching method naturally lends itself to a common set of machine learning metrics—precision, recall, and F-score—by taking the following definitions:

- True Positives (TP): The number of matched predictions, or equivalently the number of matched ground truth since matching is 1-to-1 (count of green-bordered circles in Fig. 4.1).
- False Positives (FP): The number of unmatched predictions (count of red circles in Fig. 4.1).
- False Negatives (FN): The number of unmatched ground-truth trees (count of red squares in Fig. 4.1).

The metrics are computed in the standard manner:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F}_1\text{-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Recall measures the proportion of ground-truth trees that are matched to a prediction, while precision measures the proportion of predictions that are are matched to a ground-truth. To illustrate the difference, imagine a scenario in which the model greatly over-predicts, such as predicting there is a tree on every pixel; in this case, every ground-truth will be positively matched to a prediction, so recall will be 100%. However, precision will be very low, since only a small fraction of predictions are matched to a ground-truth. F_1-score is harmonic mean of the two. For all three of these metrics, values closer to 1.0 (or 100% when expressed as a percentage) are better.

We also define a metric that determines the extent to which a method consistently over- or under-estimates the number of trees, which we will call bias. It essentially measures the percentage difference between the number of predictions and the number of ground-truth trees, which can be expressed in multiple ways (including in terms of precision and recall):

$$\text{Bias} = \frac{\mathit{count}(\text{predictions})}{\mathit{count}(\text{ground-truth})} - 1 = \frac{TP + FP}{TP + FN} - 1 = \frac{\text{Recall}}{\text{Precision}} - 1$$

Subtracting one gives us the difference between that ratio and an unbiased estimator's expected ratio of one. Bias is positive when a prediction over-estimates the number of trees, and negative when it under-estimates; values closer to zero are better. Another

useful feature of bias is that it can be used to compute an unbiased estimate of the true tree count for an area:

$$\text{count}_{\text{unbiased}} = \text{count}_{\text{biased}} \cdot \frac{1}{1 + \text{bias}}$$

For example, consider a model which we believe has a bias of -15%. We expect it to predict 15% fewer trees than actually exist. To get an unbiased count, we multiply by $(1 - 0.15)^{-1} = 1.176$, or increase our biased count by 17.6%. Of course, this method does not tell you *where* the trees are that you over or under counted; only their expected number.

Lastly, we compute the Root-Mean-Squared Error (RMSE) between matches. This is a measure of the average distance between correctly matched ground-truth and predictions, and thus serves to measure how well a model is able to localize the exact location of the tree trunk. Given the set of all distances between matched pairs $\{d_1, ..., d_k\}$:

$$\text{RMSE} = \sqrt{\frac{1}{k} \sum_{i=1}^{k} d_i^2}$$

It should be noted that this metric is not entirely independent of precision and recall. If a model is extremely cautious, and only predicts a tree when it is almost certain it knows the tree trunk is, it would have a very good RMSE and precision, but a quite low recall. The converse example, where the model always predicts a huge number of trees everywhere (low precision, high recall), results in at least one prediction being very close to each ground-truth by chance, and thus a good RMSE since the point-matching metric will generally choose the closest candidate. With this in mind, RMSE is only truly comparable when models have the same precision and recall, which rarely occurs. Thus, comparisons of RMSE should be seen only as approximations of actual localization skill.

Table 4.1: Dataset statistics

Region	Patches	LIDAR Pts	Trees	Trees Per Patch			
				Avg	Median	Min	Max
Santa Monica	92	17.7 M	4109	44.66	46	8	71
Long Beach	100	10.3 M	5845	58.45	60	6	140
Claremont	92	20.2 M	4678	50.85	46.5	6	115
Riverside	90	9.9 M	4087	45.41	41.5	0	111
Total	374	58.1 M	18719	50.05	48	0	140

4.3 Dataset

Three primary sources of data are used: satellite imagery, aerial LIDAR, and manual ground-truth tree annotations. We select four regions in the greater Los Angeles region as a testbed, for their quantity of trees, variety of tree phenotypes and human-made structures, and availability of data: Santa Monica, Long Beach, Riverside, and Claremont. See Table 4.1 for summary statistics of our dataset.

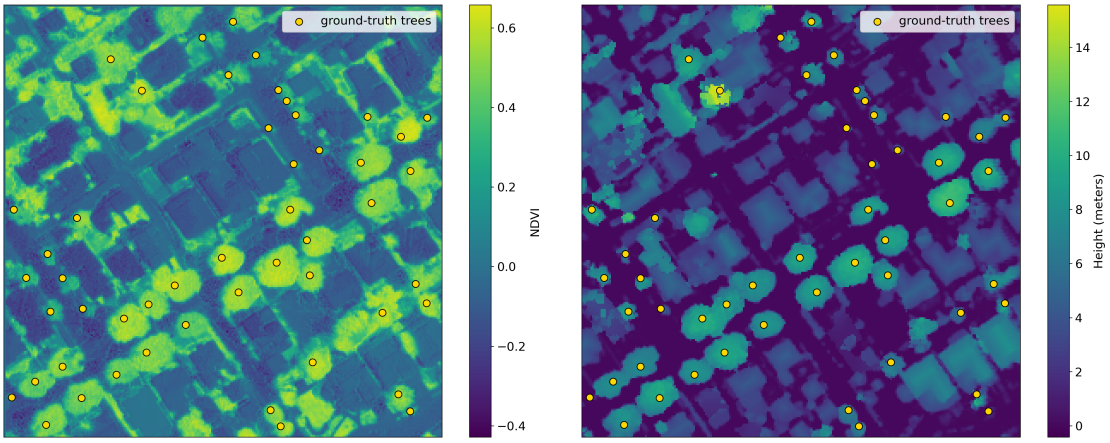
Patches are selected throughout each region as training tiles, and are annotated with ground-truth tree locations. We randomly select patches with the following split: 80% training, 10% validation (for applicable methods, otherwise it is grouped in with training), and 10% for testing. Each set is stratified so that a proportional number of patches are selected from each of the four dataset regions. Every method uses the same training and testing set, making their results directly comparable.

4.3.1 NAIP Imagery

Spectral data (imagery) is collected by the National Agricultural Imagery Program (NAIP), and is publicly accessible at no charge[1]. It is collected at 60 cm resolution,

[1]https://fsa.usda.gov/programs-and-services/aerial-photography/imagery-programs/naip-imagery/index

(a) RGB Image



(b) NDVI Image

(c) Canopy Height Model (CHM)

Figure 4.2: Spectral and raster visualizations for an example patch (Santa Monica #88), with ground-truth tree markings.

and contains a Near-Infrared (NIR) band in addition to the standard Red-Green-Blue bands, from which we can further calculate NDVI. We use NAIP imagery from 2018.

We select a number of patches from each region, in 153.6 x 153.6 meter (256 x 256 pixel) squares aligned with NAIP imagery pixels. Patches are selected throughout the area of interest to provide a variety of tree appearances and counts.

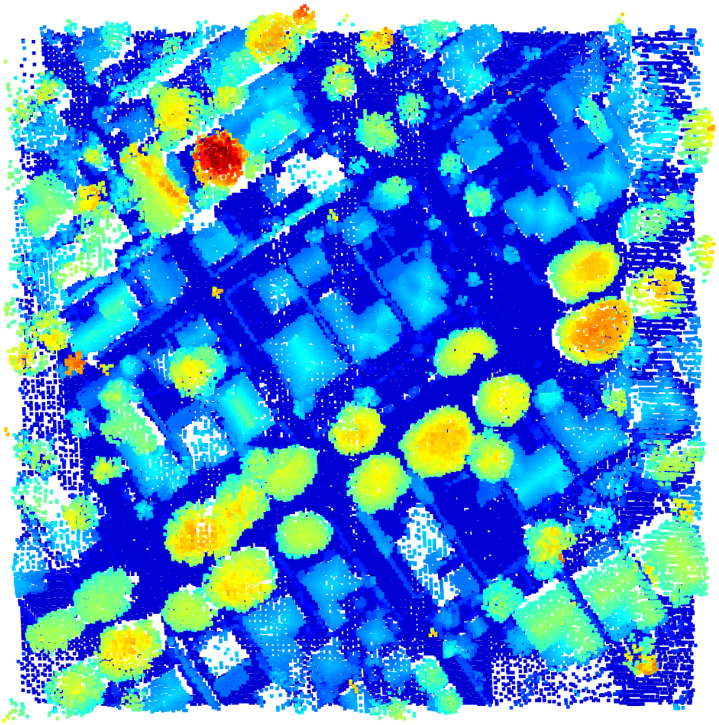
4.3.2 LIDAR

LIDAR is provided freely to the public by the U.S. Geological Survey's 3-D Elevation Program (USGS 3DEP)[2]. It is captured overhead by aircraft, with the data we selected being gathered between 2016 and 2019. The 3DEP standards specify that the root mean-squared error in the vertical direction (RMSEz) will be less than or equal to 10 cm, and the density of points will be at least 2 per square meter. However, we measure a density of closer to 6.58 points per square meter in our final data patches. Density does vary substantially throughout the dataset, caused by the fact that each pass of the aircraft overlaps partially with previous passes in some areas but not others, and the fact that LIDAR was captured on different dates and under different conditions in each region. A visualization of an example LIDAR patch is shown in Fig. 4.3 (visualization generated with Open3D [72]).

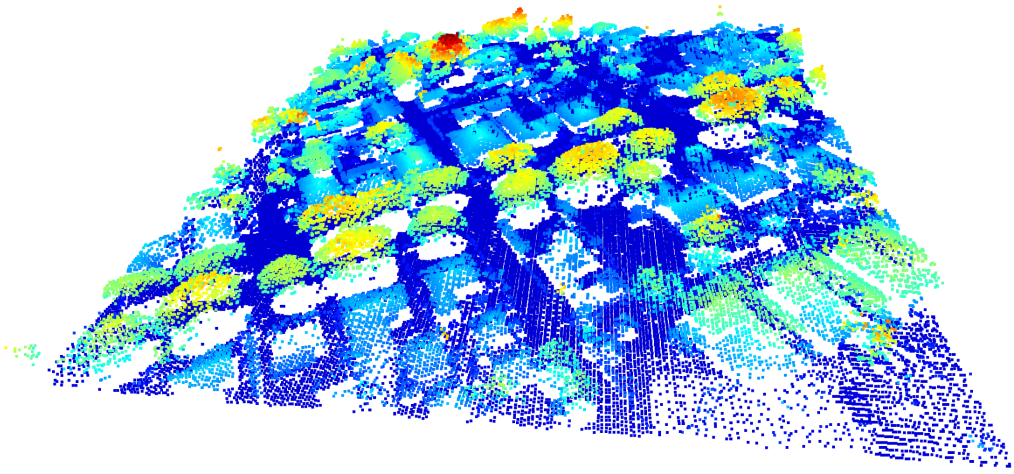
4.3.3 Ground-Truth Tree Annotations

To develop a consistent inventory of ground-truth tree locations, we manually annotated tree locations as they appeared in the NAIP imagery, and utilized additional sources of data (such as higher resolution imagery and NDVI images) to verify locations. Inventories gathered by the cities were used as the starting point for our ground-truth tree location annotations, though these inventories did not include trees in yards and on private property, and we made adjustments to the majority of points to better align with the imagery. Tree annotations were placed at the main trunk, or the peak of the canopy when the trunk location was not possible to determine.

[2]https://usgs.entwine.io/



(a) Overhead view



(b) Side view

Figure 4.3: Visualization of raw LIDAR for an example patch (Santa Monica #88), colored by height

4.3.4 Preprocessing

The data processing pipeline is as follows:

1. Once for each LIDAR data source:
   (a) The raw LIDAR point cloud is processed using PDAL [49] to generate a HeightAboveGround (HAG) dimension, a normalized version of the raw z dimension, and reproject to a common coordinate reference system (EPSG:26911).
   (b) Processed LIDAR is separated into the grid squares which have been identified for training, validation, or testing.
   (c) Red, Green, Blue, NIR, and NDVI are added as additional features to each LIDAR point, by nearest-neighbor interpolation of the NAIP values.
2. Once during model instantiation:
   (a) Remove points with spurious HeightAboveGround values, which we define to be less than -10 meters or than greater 50 meters.
   (b) Patches are subdivided. Assuming a user-provided subdivision parameter of k, subpatches are generated with a height and width equal to the full-sized patch's side length divided by k. These subpatches are designed to be overlapping exactly halfway with each of its cardinal neighbors, such that the total number of subpatches is $(2k-1)^2$.
   (c) For each subpatch, if there are fewer than n points in the patch (where n is the number of input points to the model, a user parameter), one of three actions is taken as determined by another user parameter:
      - The subpatch is skipped, or
      - The subpatch is filled with "no data" points (with -1000 HeightAboveGround values) until it reaches n total points, or

- The subpatch is filled by randomly re-sampling existing points until n total points is reached.

3. For each example generated during training or inference:

   (a) Read n points randomly from the subpatch (where n is a user parameter).

   (b) Scale the x and y dimensions to be between zero and one.

   (c) Scale the input HAG dimension so that the range 0 to 50 meters in the input correspond to the range 0 to 1 in the output.

   (d) If this sample is for training, the following augmentation may be applied:

   - Add zero-centered Gaussian noise to the input points, with the standard deviation being a user parameter.
   - Randomly rotate the input and ground-truth xy locations around the center of the patch by a multiple of 90 degrees.

4.4 Model Architecture

We seek a deep-learning architecture capable of accepting an un-ordered set of points—$\{p_1, ..., p_N\}$, each with C features representing spatial and spectral information—and outputting a set of points representing predictions for the locations of trees found within that scene, $\{\hat{y}_1, ..., \hat{y}_n\}$. Each output point $\hat{y}_i$ will have three features: predicted x and y location, and confidence in the closed interval $[0, 1]$. We allow the network to output many more predictions than the expected number of trees in a given scene, and thus we expect it will output many predictions with confidence close to zero, which can be considered non-predictions. The methodology for determining the final predicted tree locations from the PointNet's outputs is explored in a 4.4.3.

4.4.1 Modified PointNet

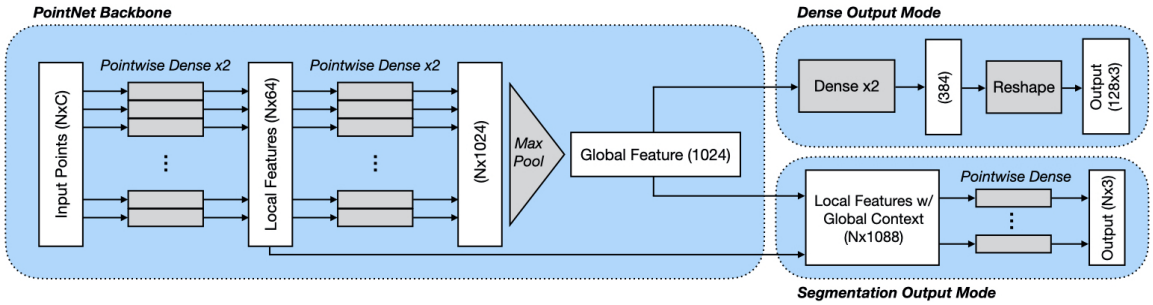


Figure 4.4: Modified PointNet architecture, with example tensor shapes

The backbone of our design is the PointNet [56] or PointNet++ [57] architecture, as described in 3.2.2. To this backbone we make two major modifications.

The first modification is derived from the fact that we can make some stronger assumptions about our input data than the PointNet authors could; we do not need to be invariant to arbitrary 3-D rotation. The PointNet authors were interested in classication of arbitrary objects; a lamp that is upside-down (or in any other orientation) is still a lamp, and should be recognized as such. We, however, know we will never see trees in any orientation other than vertical in the z dimension. Thus, we may drop the Alignment Networks (a.k.a. Transformation Nets or T-Net) in their original formulation, which were present to allow the network to learn to apply an arbitrary transformation to the pointcloud.

The second modification is derived from our desire for a different output format. The initial formulation of PointNet proposed two output modes for two different problems: classification and segmentation. We seek neither to classify the pointcloud as a whole, or classify each point within it. Instead, we want to output a set of confidence-weighted points which represent predicted tree locations. Thus we define two new output modes for this task:

- Segmentation mode: Though not actually performing segmentation, this method is named so because it is essentially the same as the original segmentation mode, except that the final output is three values for each point (x, y, and confidence) instead of a probability distribution over classes.
- Dense mode: This method generates predictions from only the global feature vector, by using a simple multi-layer perceptron.

To constrain the final outputs of the model, predicted (x, y) tree locations are constrained to be between zero and one by clipping (recall that the model input and target coordinate system is scaled to be between zero and one). The predicted confidences are also constrained to be between zero and one by either sigmoid activation or clipping.

4.4.2 Loss Functions

Two loss functions are evaluated, originally published as a component of approaches called DeepLOCO [9] and DeepSTORM [46]. These are borrowed from single molecule localization in microscopy, which similarly requires predicting a confidence-weighted set of coordinates, and comparing them to ground truth coordinates. Let Y will represent a set of m ground truth coordinates, $\{\theta_1, ..., \theta_m\}$, while $\hat{Y}$ represents a set of n predicted coordinates with confidence weights, $\{(\hat{\theta}_1, \omega_1), ..., (\hat{\theta}_n, \omega_n)\}$.

The concept behind both losses is very similar, as noted in the DeepLOCO publication [9]. Both consider predicted points to be expressing high confidence in their immediate vicinity, and increasingly less confidence as one moves further and further away. They model this confidence as a Gaussian curve, with which they can then evaluate how similar the predicted confidence space is to the actual tree location space. A simplified illustration of this concept is presented in Figure 4.5, which

demonstrates the losses working on a toy example in one spatial dimension. In that figure, note that both losses lead to similar optimization conclusions: the leftmost prediction is over-confident, while the rightmost two are under-confident. A more technical description of the loss functions follows.

4.4.2.1 DeepSTORM Loss

The first loss, from DeepSTORM [46], blurs each point (predicted or ground-truth) to a fine grid (we chose 128 x 128) by convolution with a Gaussian kernel, the standard deviation of which is a hyperparameter. The predicted location grids are each scaled by their confidence. These grids for predictions and ground-truth are then respectively aggregated, either by taking the element-wise maximum or the element-wise sum. The loss is defined as the mean squared error between these final two aggregated grids:

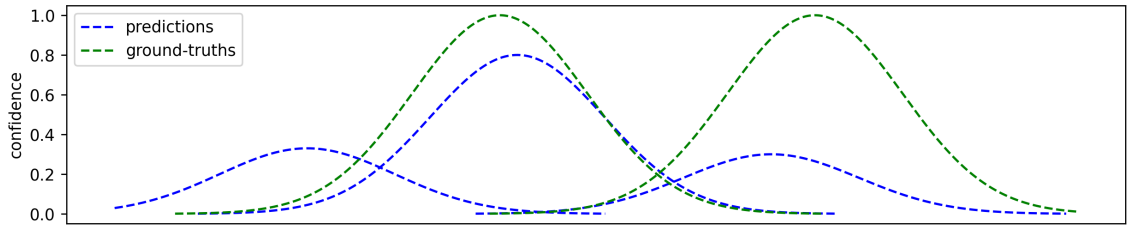
$$\text{loss}_{DeepSTORM}(Y, \hat{Y}) = \left\| \Big(\sum_{i=1}^{m} g \circledast \theta_i \Big) - \Big(\sum_{j=1}^{n} \omega_j (g \circledast \hat{\theta}_j) \Big) \right\|_2^2 \tag{4.2}$$

where g is the Gaussian kernel, and $\circledast$ represents the convolution operation, and the sums are elementwise operations. For max aggregation, apply the element-wise maximum function in the sums' place.

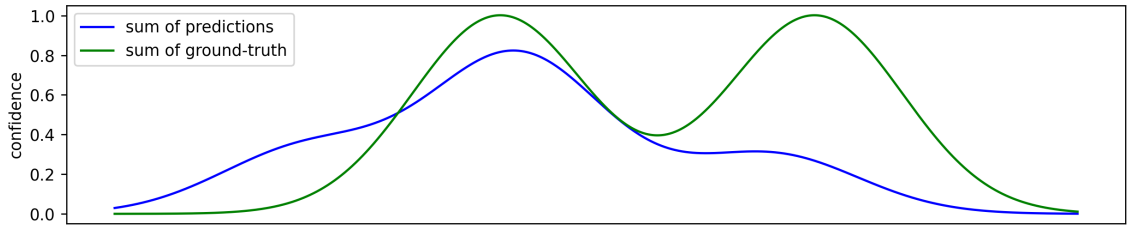
4.4.2.2 DeepLOCO Loss

The second loss is from DeepLOCO [9], which is similar to the above except that a set of points is rendered as an infinite-dimensional functional, instead of an image. For a weighted point set Y,

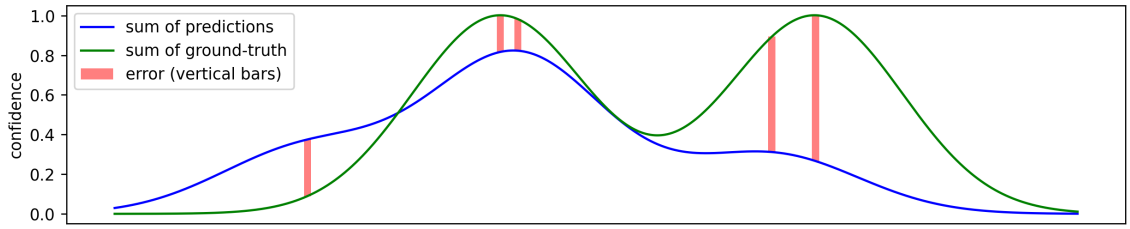
$$\mathcal{R}(Y) = x \mapsto \sum_i \omega_i \, g(\hat{\theta}_i - x) \tag{4.3}$$



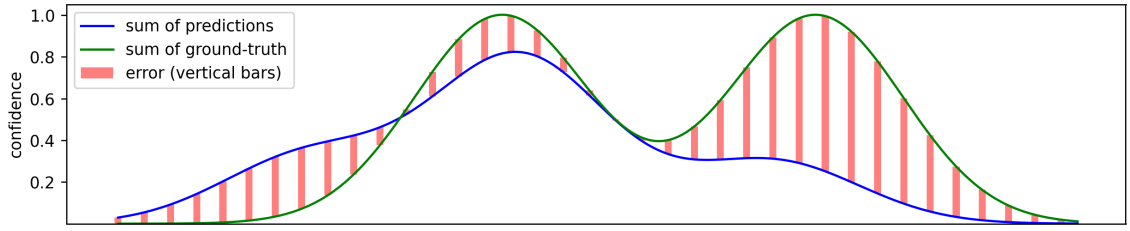
(a) Gaussian curves are placed at each of the predicted and ground truth location. Ground-truth trees are given a peak height of one, while predictions use their confidence as their peak height. All curves are given the same standard deviation (σ), which is a hyperparameter.



(b) The curves of predictions and ground-truth are respectively aggregated. This and the following figures use summing as that aggregation; taking the maximum is another option (tracing the highest of each color at every location).



(c) DeepLOCO Loss: Error is the sum of the absolute difference between the aggregated curves evaluated only at the locations of all predictions and ground-truth (the peaks of all curves in Fig. 4.5a).



(d) DeepSTORM Loss: Error is the sum of the absolute difference between the aggregated curves evaluated at fixed intervals (in practice, on a fixed-resolution grid since locations are in two-dimensional space).

Figure 4.5: Visual depiction of loss functions in one spatial dimension, for three predicted and two ground-truth tree locations

defines the functional. The loss is then computed as mean-squared error between these functionals evaluated at all prediction and ground-truth locations:

$$\text{loss}_{DeepLOCO}(Y, \hat{Y}) = \left\| (\mathcal{R}(Y) - \mathcal{R}(\hat{Y}))|_{Y \cup \hat{Y}} \right\|_2^2 \tag{4.4}$$

For expansions useful to the implementation of this loss, see Equations 4, 5, and 6 in [9].

4.4.3 Model Output Post-Processing

Once the network is sufficiently trained, we must have a way of converting the its predicted collection of confidence-weighted points into final predicted locations. We evaluate four methods for achieving this goal:

1. Raw: The simplest method; simply keep all points above a certain confidence threshold, and discard the rest.

2. k-means: Clustering by the classic k-means algorithm [40] (specifically, the MiniBatch variant described in [58]), with the points weighted by their predicted confidence. The centroids of the clusters are determined to be the new tree predictions, and are given a confidence of either the maximum or the mean confidence of their cluster. We apply confidence thresholding both to the model's outputs, and to k-means' final centroids. The rationale for the first threshold is that we expect that many predicted points will essentially non-predictions, with very low confidence, and thus we do not want them to influence clustering. The second confidence threshold after clustering is to eliminate clusters which do not have enough confidence to be final predictions.

3. DBSCAN: Same as the previous method, except clustering is done using the DBSCAN algorithm [16].

4. Peak-local-maxima (PLM) filtering: This method is similar in concept to the DeepSTORM loss. We blur each predicted point to a grid, with a Gaussian peak height equal to the predicted confidence for that point. After aggregating all these grids together (either element-wise sum or max), we apply a peak-local-maxima filter which identifies all points which are the local maxima in their neighborhood. Similar to the clustering methods, two confidence thresholds are applied both before and after filtering.

Each of these methods has one or more parameters, such as confidence thresholds or the numbers of clusters for K-means. These parameters are estimated on the validation set, as described in the next section.

4.4.4 Model Optimization

4.4.4.1 Training Procedure

The model is built and trained with Keras [11] and TensorFlow [42], on an NVIDIA V100 GPU. Training is executed in minibatches, with a whole pass through the training set being considered one epoch. After each epoch, the loss is evaluated on the validation set. If a new best validation loss has been achieved the model is saved. Otherwise, if three epochs have passed without validation loss improvement, learning rate is reduced by a factor of 0.2; if five epochs pass with no improvement, training is stopped.

Table 4.2: Post-processing methods & parameters

Method	Parameter	Description
All	Thresholds	Confidence thresholds on model outputs and final predictions.
k-Means	k	The number of clusters to create.
	Aggregation	How confidence of centroids is determined: average or max of cluster confidences.
DBSCAN	ϵ	Maximum distance allowed between directly connected points.
	Min Samples	Total confidence a cluster must achieve to be considered.
PLM	σ	Standard deviation of Gaussian blur.
	Min Dist	Minimum distance allowed between local maxima.
	Aggregation	How to aggregate all Gaussian blurs together: pixelwise sum or max.

4.4.4.2 Post-Processing Parameter Estimation

Once a model is trained, we must determine the optimal post-processing method and parameters (see Sec. 4.4.3). This is done by estimating the best parameters on the validation set: given the model's validation set predictions, we run a heuristic search over all reasonable method/parameter combinations, as summarized in 4.2. The parameters that result in the best validation F-score at the end of the search are kept as the final parameters. This search is implemented using the Tree of Parzen Estimators (TPE) algorithm [7] as implemented in Optuna [2], constrained to explore all methods equally. The search is run for 140 trials (25 are random startup trials) or 35 minutes, whichever comes first (these values are picked somewhat arbitrarily but we find they are more than sufficient for well-trained models). Each trial tends to take between a few seconds and one minute.

Table 4.3: Summary of important hyper-optimization parameters

Parameter	Possible Values
Optimizer	Adam [29], AdaMax [29], NAdam [14], Adadelta [70]
Batch size	8 to 128 by powers of 2
Initial learning rate	1e−5 to 1e−1, logarithmic scale
Output flow	"Dense" or "Segmentation"
Model size factor	2^n for n in $[-2, 2]$
Confidence activation	Sigmoid, ReLU
Loss function	DeepLOCO, DeepSTORM
Loss σ	0.5 to 6, in steps of 0.5 (units in meters)
Num inputs points	$100 \cdot 2^n$ for n in $[1, 4]$
Patch subdivision k	Integers 1 to 10

4.4.4.3 Hyper-Optimization

A number of hyper-parameters have to be set for training, including learning rate, batch size, and model size. To estimate the optimal parameters, we again utilize the TPE algorithm [7] implemented by Optuna [2]. For each trial, a model is fully trained and post-processing parameters are determined. The final validation-set F-score is returned as the optimization metric to Optuna. Training is limited at 500 epochs and 4 hours for each trial, though these limits are rarely reached; we find the best models use a fraction of each.

4.5 Alternative Methods

We compare our proposed solution against a number of baseline methods.

4.5.1 PyCrown

PyCrown [74] is a leading method for tree detection and tree-crown delineation. It uses a variation of the watershed method, described in Dalponte & Coomes [13], along

with enhancements that correct for trees on steep slopes. This approach accepts only LIDAR rasters as input and was developed for rural forests, so—like any purely watershed-based method—it is expected to misclassify buildings as trees at a high rate. Their code is publicly available[3].

The approach has a number of parameters, such as the amount of pre-smoothing applied to the CHM, size of the peak-local-maxima filter, and maximum tree-crown size; these are estimated with Optuna. We use the both the training and validation set for parameter estimation, as we find there is no noticeable over-fitting to account for.

4.5.2 PyCrown-Spectral

For urban applications, a common alteration to the typical watershed-based methods is to filter the Canopy Height Model so that all pixels with too low of an NDVI value are set to zero height. This is the exact approach used by [24] (albeit with a less sophisticated watershedding algorithm), and similar in concept to others such as [71]. Following their lead, we also evaluate a modification of PyCrown that incorporates this NDVI thresholding stage, which we will call PyCrown-Spectral. While these previous approaches have relied on fixed parameters, such as an NDVI threshold of zero or a pre-defined peak-local-max window size, we instead estimate them from the training set. This, combined with the fact that PyCrown is a sophisticated watershedding algorithm, leads us to believe that PyCrown-Spectral is a baseline as least as strong as the previously mentioned approaches that inspire it.

Parameters are estimated in the same manner as our standard PyCrown approach.

[3]https://datastore.landcareresearch.co.nz/dataset/pycrown, provided under the GNU GPLv3 license.

4.5.3 SFANet

For a purely imagery-based approach, we test against a sophisticated convolutional neural network (CNN) trained only on the multi-spectral imagery (RGB, NIR, and NDVI). We utilize SFANet [73]—originally developed for the task of croud-counting—which is itself build on top of a VGG-16 backbone [61]. SFANet's primary contribution is that the tasks of identification and localization are separated, and delegated to two separate portions of the network. SFANet uses a loss similar to DeepSTORM loss, i.e. the mean-squared error between the predicted and actual confidence rasters. The final network utilizes roughly 17 million parameters, and $\sim$12 hours to train on one NVIDIA V100 GPU.

A more thorough description of this network is included in a forthcoming manuscript by Ventura et al. [63].

Chapter 5

RESULTS

We evaluate each method on the 39 patches of the test-set, which no methods have been trained or otherwise calibrated on. In Section 5.1, we compute and analyze summary statistics of each model's performance, and qualitatively assess their performance by visual comparison. Then in Section 5.2, we make a more granular assessment of the causes of errors in each model. In Section 5.3, we conduct an ablation study to analyze the effect of various spectral channels' contributions to the network's performance. Lastly, we revisit the hypothesis and research questions in Section 5.4.

5.1 Overview of Results, by Method

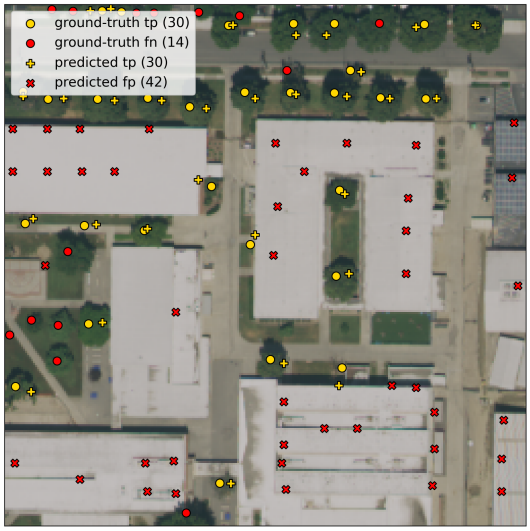
5.1.1 PyCrown & PyCrown-Spectral

As expected, the PyCrown approach—which was developed for non-urban areas—does not fair too well in the urban environment. Its precision and recall close to 50% suggest that only half of its predictions are correct, and that those correct predictions

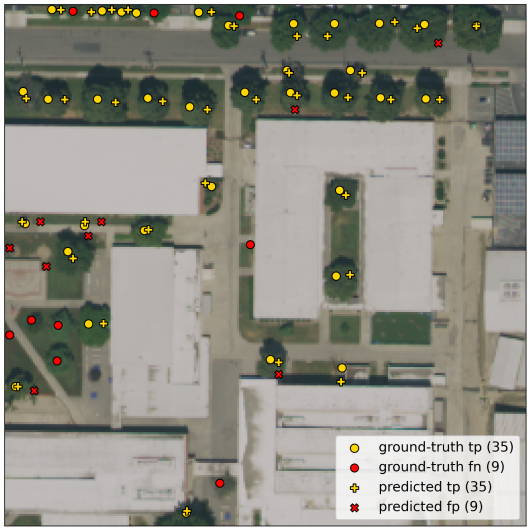
Table 5.1: Summary of test-set results

Method	Precision (%)	Recall (%)	F_1-score (%)	RMSE (m)	Bias (%)
PyCrown	52.4	51.1	51.8	2.72	-2.4
PC-Spectral	69.6	56.6	62.4	2.64	-18.6
SFANet	71.3	72.7	72.0	2.29	**1.9**
PointNet (ours)	75.9	71.0	73.4	2.38	-6.5
PointNet2 (ours)	**77.6**	**73.6**	**75.5**	**2.28**	-5.1

Metrics are reported for the parameterization of each model that achieved the best training/validation F-score. Best result in each column is bold.



(a) PyCrown: Long Beach 55



(b) PyCrown-Spectral: Long Beach 55

(c) PyCrown: Santa Monica 5

(d) PyCrown-Spectral: Santa Monica 5

Figure 5.1: Visual comparison of PyCrown and PyCrown-Spectral

only account for half of the ground-truth trees. A visual inspection (see the lefthand figures in Fig. 5.1) confirms that standard PyCrown predicts often at the peaks of buildings and other human-made structures.

PyCrown-Spectral does noticeably better, with a 17% improvement in precision, and 6% improvement in recall, which Figure 5.1 confirms is largely due to its ability to avoid predictions on human-made structures. This greater improvement in precision compared to recall comes with a corresponding increase in negative bias. It appears that while PyCrown had essentially no chance at distinguishing between trees and structures, PyCrown-Spectral is better able to, and with more "stringent" parameters can be more "careful" with its predictions; this trend is best depicted in Figs. 5.1 (c) and (d), where the best performing PyCrown-Spectral parameters end up achieving one fewer true-positive than PyCrown, but at the benefit of a 75% reduction in false-positives. Note that a more "balanced" set of parameters could be selected for PyCrown-Spectral that would result in lower bias and more even precision and recall, but these parameters would result in an overall decrease in F-score.

5.1.2 SFANet

Two things stand out about this approach's results. Firstly, it is the only method with a positive bias (or equivalently, the only method with a higher recall than precision), and has the smallest absolute bias of all methods. This bias term means the SFANet tends to predict around 2% more trees than actually exist. Secondly, it has very nearly the best RMSE, beat out by PointNet 2 by a margin of one centimeter. However, as is noted in Sec. 4.2.1, it is difficult to directly compare RMSEs, especially when one method has a positive bias and another has negative bias. We report RMSE as calculated, but it should not be concluded that one method is definitively better than the other at localization without further inquisition.

(a) SFANet

(b) PointNet 1

(c) PointNet 2

Figure 5.2: Visual comparison of SFANet and PointNet predictions, on an example patch (Santa Monica #5)

Table 5.2: Comparison of final PointNet 1 & 2 parameters

	PointNet 1	PointNet 2
Differing Params		
Loss	DeepLOCO	DeepSTORM
Output Mode	Dense	Segmentation
Learning Rate	1e−4	1e−2.5
Noise on Input Pts	$\sigma = 0.2$m	none
Aggregation	Sum	Max
Same/Similar Params		
Batch size	64	64
Confidence activation	Sigmoid	Sigmoid
Optimizer	Adam	Adam
Loss σ	3.5m	3.0m
Total Weights	12.4 M	250 K
Training Time	∼1 hr.	∼2 hr.

5.1.3 PointNet & PointNet 2

Our PointNet-based methods have the best overall performance in our experiments, with the best PointNet 2 achieving the best F-score of 75.5% and RMSE of 2.28 m (only 1 cm better than SFANet). It also achieves the best precision (77.6%) and recall (73.6%).

For PointNet 1 and 2, hyper-optimization settled on a fairly different set of parameters; the main differences are shown in Table 5.2. The fact that both losses and output modes are represented in the best parameters demonstrates that each is useful and effective for this problem. Overall the most striking difference between the two models is the number of weights; PointNet 1 settles on a model with around 50 times more trainable weights than PointNet 2. This is in part because of the sampling and grouping operations that PointNet 2 uses behind the scenes, which do not utilize any weights but do provide the model with significant information. Additionally, these operations can be computationally and memory intensive with large inputs, forcing the model toward smaller solutions; PointNet 2 ran in to memory limitations on our

GPUs when the model got too large. Still, that large of a difference suggests that the PointNet 2 model is able to use its weights more efficiently than PointNet 1.

In post-processing, both the best PointNet 1 and 2 found that the peak-local-max filtering method for determining final predicted tree locations was most effective. The DBSCAN clustering method was a close second.

5.2 Analysis of Errors

5.2.1 Methodology

For a more granular analysis of the causes of errors for each model, we qualitatively identify the cause of each error in five randomly selected test-set patches. For each false positive and negative, we identify what the most likely apparent cause is; in most cases this is fairly clear. In cases where multiple causes could be the culprit, we assign the error to the one we believe to be the most likely source. We identify the following causes of errors, in three categories: identification (determining which things are trees), delineation (separating out individual trees), localization (determining the location of the tree trunk):

1. Identification errors:
    - Structures: Mistaking human-made structures for trees.
    - Bushes/shrubs: Mistaking bushes or shrubs for trees.
    - Grass: Mistaking grass for trees.
    - Tree is small: Mistaking a tree for a non-tree because of its short height or small width.
2. Delineation errors:

- Over/under-predict dense trees: Predicting too many or few trees exist in an area where multiple tree canopies overlap.
- Mistake one for multiple: Predicting multiple trees on a canopy that is actually just one tree.

3. Localization error: a prediction is correctly placed on a tree canopy, but the canopy is large enough that the prediction falls more than 6 meters from the trunk, resulting in both a false positive and a false negative.

We also find a few rarer cases where inconsistencies in the data contribute to errors:

- LIDAR/image mismatch: When a tree appears in the image but not the LIDAR, or vice versa.
- Patch edge effects: Trees that appear only partially in the tile, leaving the model at a disadvantage.

Lastly, some errors have no obvious cause, and are included in the catch-all "Unclear" category.

5.2.2 Analysis

The results of this analysis are compiled in Tables 5.3 and 5.4. It should first be noted that there are certainly small-sample-size effects present in this data; if you read into individual numbers too closely you will be misled (e.g., it suggests that PointNet 1 outperforms PointNet 2, which is not is true on the test set as a whole), but it still illustrates general trends well.

Figure 5.3: The five patches randomly selected for source-of-error analysis, from Claremont, Claremont, Long Beach, Riverside, and Santa Monica respectively

Table 5.3: Sources of error

	PyCrown	PC-Spectral	SFANet	PointNet	PointNet2
False Positives (FP)	Count (Percentage of FP in Parentheses)				
1. Structures	91 (87%)	2 (5%)	0	1 (3%)	1 (3%)
2. Bushes/shrubs	4 (4%)	21 (57%)	26 (46%)	26 (81%)	30 (77%)
3. Grass	0	0	12 (21%)	0	0
4. Over-predict dense trees	3 (3%)	7 (19%)	11 (20%)	2 (6%)	3 (8%)
5. Mistake one for multiple	3 (3%)	7 (19%)	5 (9%)	2 (6%)	4 (10%)
6. Localization	1 (1%)	0	2 (4%)	0	0
7. Unclear	3 (3%)	0	0	0	1 (3%)
Total FP	105	37	56	32	39
False Negatives (FN)	Count (Percentage of FN in Parentheses)				
1. Tree is small	45 (64%)	57 (83%)	31 (74%)	26 (70%)	28 (78%)
2. Under-predict dense trees	16 (23%)	6 (9%)	7 (17%)	6 (16%)	4 (11%)
3. Localization	1 (1%)	0	1 (2%)	0	0
4. LIDAR/image mismatch	1 (1%)	1 (1%)	0	0	0
5. Patch edge effects	5 (7%)	5 (7%)	3 (7%)	5 (14%)	4 (11%)
6. Unclear	2 (3%)	0	0	3 (11%)	0
Total FN	70	69	42	37	36
Grand Total	175	106	98	69	75

Source of each error is qualitatively determined on a subset of 5 randomly selected test-set patches. Numbering order of error sources is only included to more easily make reference to specific rows.

Table 5.4: Sources of error by category

	PyCrown	PC-Spectral	SFANet	PointNet	PointNet2
Category	Count (Percentage of Total Errors in Parentheses)				
Identification	140 (80%)	80 (75%)	69 (70%)	**53** (77%)	59 (79%)
Delineation	22 (13%)	20 (19%)	23 (23%)	**11** (16%)	**11** (15%)
Localization	2 (1%)	**0**	3 (3%)	**0**	**0**
Data effects	6 (3%)	6 (6%)	**3** (3%)	5 (7%)	4 (5%)
Unclear	5 (3%)	**0**	**0**	**0**	1 (1%)
Total	175	106	98	**69**	75

Best result in each row is bolded.

5.2.2.1 Identification

One of the most noticeable trends in Table 5.3 is the difficulty many methods have distinguishing between bushes and small trees, which is understandable given that they can appear very similar (especially from an aerial view). The false positive of mistaking bushes for trees (row FP-2), and its twin false negative of not predicting trees because of their small size (row FN-1), make up a sizeable portion of most models' errors. It is along this bush vs. small tree axis that almost all identification errors occur (excepting the standard PyCrown approach which is easily fooled by buildings). Identification as a whole is the task where most errors occur for every model; it makes up at least 70% of all errors for every kind of model (see Tab. 5.4).

The SFANet is unique in that it mistakes grass for trees, an error which no other model makes. This is obviously caused by the fact that it does not have access to LIDAR data, making greenery at zero feet and 30 feet above ground difficult to distinguish between. The PointNet variants are overall best in this sub-task, with PointNet 1 slightly outperforming PointNet 2 in this sample.

5.2.2.2 Delineation

Generally the next-most common source of error was in delineation: determining how many trees are actually present in a region of thick overlapping canopies (FP-4 and FN-2 in Tab. 5.3), or alternately mistaking a single canopy for multiple overlapping canopies (FP-5). The PointNet variants are again the best at this sub-task.

5.2.2.3 Localization

Localization within the 6 meters threshold presents little challenge for most methods. This is partially because of the fact that a tree must be fairly large for a prediction to even be able to miss the trunk by at least 6 meters and still be on the canopy, so there are fewer chances for this sort of error to occur in the first place. Additionally, the center of the crown and the highest point of the tree tend to be good estimates of the tree trunk location (the latter is in fact the exact heuristic used by wastershed methods).

Though not a large portion of its overall errors, SFANet makes the most localization errors (while most other methods make none at all). This suggests that the structural information contained in LIDAR is greatly helpful particularly for the task of localizing the tree trunk.

5.2.2.4 Data Effects

Although we do find a few instances of errors potentially caused by the data itself (rows FN-4 and 5), we find that they do not make up a significant portion of any model's errors (around 5% of total errors), and contribute fairly equally across all methods.

A few more errors are caused by imperfections in the data, but of the sort that are essentially unavoidable and that a best-performing method will have to learn to avoid. One example is PointNet 2's single false positive on a structure: this occurred building that stood directly next to grass. Likely, a bit of the grass's spectral information got interpolated to the top of the building, which made it look a bit like a tree. While more perfect data could have avoided this error, the inherent spacial precision and

resolution limitations of this dataset are constraints that we consciously want to work within.

5.2.2.5 The "Unclear" Category

The unclear category is a catch-all to include errors with no clear cause. These errors occur very rarely.

5.3 Ablation Study

In order to test the importance of spectral information, we conduct an ablation study to test the importance of various spectral channels. We utilize the architecture of the best-performing PointNet 2 model, and only vary which spectral channels are added to the LIDAR points. We test four variants: removing NDVI, removing NIR & NDVI, removing all channels except NDVI, and removing all spectral information. For each variant, we run three trials and select the best-performing by validation F-score, to reduce the probability of selecting a bad network initialization by chance. The test-set results, shown in Table 5.5, indicate that the full spectral information we include is informative, and removing channels decreases performance in terms of both F-score and RMSE.

Table 5.5: Input channels ablation results

Channels	Precision	Recall	F-score	RMSE	Bias
R, G, B, NIR, & NDVI	**77.6**	**73.6**	**75.5**	**2.28**	-5.1
R, G, B & NIR	74.1	75.5	74.8	2.34	**2.0**
R, G, B	**77.6**	72.0	74.7	2.37	-7.3
NDVI only	75.7	73.2	74.5	2.35	-3.3
no spectral channels	73.1	66.7	69.7	2.59	-8.7

Notably, the model variant with no spectral information included does noticeably worse, with a 6-point drop in F-score. This would put it below the best-performing SFANet's results in our study, which utilizes *only* spectral information. This suggests that multi-band imagery information is on its own more informative than to LIDAR on its own—or is at least easier to learn from given its convenient gridded structure.

Our results here reaffirm the utility of NDVI as a metric; the methods with only NDVI as its spectral information performs very nearly as well as the method with all four mutli-band imagery channels (RGBN).

5.4 Summary

We find that our proposed networks, modifications of PointNet and PointNet 2, outperform all other methods in predictive ability as measured by F-score, and in localization accuracy as measured by RMSE. Our error analysis reveals that our networks outperform others in skill for the sub-tasks of identification, delineation, and localization. Our ablation study shows that the combination of spectral and LIDAR information is key to the network's success.

5.4.1 Hypothesis

Our hypothesis is that a sufficiently well-tuned deep neural network, learning directly from imagery-augmented LIDAR, will outperform other methods for tree detection in the urban environment, including non-neural-networks, those which do not utilize LIDAR, and those which use summarized LIDAR. Our experiments support this hypothesis, by demonstrating the existence of a neural network approach capable

of learning directly from LIDAR point-clouds that outperforms multiple other high-performing methods.

5.4.2 Research Questions

Our experiments directly address the research questions in multiple ways.

5.4.2.1 RQ1: How effectively can deep learning be applied directly to LIDAR pointclouds to detect trees in the urban environment?

We find that deep learning can be directly applied to LIDAR pointclouds with high effectiveness. As shown in our ablation study (Sec. 5.3), a well-tuned network learning from only LIDAR information (i.e., no spectral information) still performs well, achieving an F-score of nearly 70%. When multi-band spectral information and NDVI is included however, our network is able to achieve the best F-score with 75.5%, as well as the best RMSE of 2.28 meters.

5.4.2.2 RQ2: What structures in the urban environment are difficult to distinguish from trees?

When conceiving of the difficulties of tree detection in the urban environment compared to traditional forests, the primary difference seems to be the existence of human-made structures. However, as our error analysis (Sec. 5.2) reveals, most sophisticated methods rarely mistake buildings and other structures for trees. In fact, the most challenging "structures" tend to be bushes and shrubs, specifically in distinguishing them from smaller trees. Thus, the most challenging aspect of the urban environment

for the task of tree detection is not the existent of structures, but the great variety in appearance and structure of human-curated plants.

5.4.2.3 RQ3: Can deep learning for LIDAR pointclouds provide information that is complementary to that from other methods?

We found that our network was the clear best performer in our experiments. However, we do see more generally that the LIDAR-based approaches do seem to be somewhat complementary to the spectral-only SFANet approach. Our best PointNet 2 variant achieves the best RMSE and has a negative bias. SFANet conversely has positive bias, and while it achieves very nearly the same RMSE (Table 5.1), we do find that it makes more localization errors in our error analysis (Table 5.4). If a spectral-only method was found that outperformed ours in terms of F-score, the better localization skills of our method could be used to fine-tune the location of tree trunks.

Chapter 6

CONCLUSION

6.1 Overview

In this study, we propose a novel method for tree detection in the urban environment. Our method is build around a modified PointNet architecture [56][57], which learns directly from spectrally-augmented LIDAR data. To this backbone we add loss targets and post-processing methods specific to our task, utilizing and synthesizing techniques borrowed from numerous related fields including forestry, remote sensing, crowd-counting, and microscopy in the process. To validate our solution, we identify five regions in Southern California, and hand annotate tree locations for training and testing. Our results demonstrate the efficacy of our proposed method, with performance quantitatively and qualitatively compared to multiple baselines.

In Chapter 1, we introduce the problem at hand, our research questions, and hypothesis. Chapter 2 provides an overview of common approaches and terminology in forestry and deep learning that are crucial to understanding the remaining manuscript. In Chapter 3, we take a more detailed dive into specific works addressing the same or similar tasks, finding a wealth of related research but no approaches that utilize the same method as we do. Chapter 4 introduces that method, which is the use of modified PointNet architectures applied to spectrally-augmented LIDAR. That chapter also introduces the metrics we will use to evaluate our methods, and the baselines that we will compare it against. Chapter 5 does just that, analyzing the results of our experiments from multiple angles.

6.2 Contributions

We develop a technique for urban tree detection that is efficient and effective. Our contributions are that:

- Our method outperforms our baselines in terms of predictive capability, achieving the best F-score of 75.5% on our test set.
- Our method outperforms our baselines in terms of localization accuracy, achieving a best root-mean-squared error of 2.28 meters on true positive predictions.
- Our method is significantly more efficient than the other deep learning baseline we compare against, SFANet. Our method is 5x faster to train, and requires over 50x fewer trainable weights.
- Our method synthesizes deep learning techniques developed in a variety of disparate fields, including microscopy, crowd-counting, remote sensing, and forestry.
- Our method is capable of achieving high performance from publicly available datasets.

We believe the primary reason for the success of our method is that our network is capable from learning directly from un-modified, un-summarized LIDAR point clouds. This allows it access to the entirety of the spatial information captured in LIDAR—in contrast to a summarized form such as a raster or voxelization—while still being highly efficient.

Lastly, an important feature of our work is that we rely exclusively on publicly-available datasets. While more accurate and high-resolution data certainly exists in

some areas, any method that wants to have broad application must be capable of working with this quality of data and overcoming the deficiencies inherent to it.

6.3 Future Work

Most broadly, our results indicate that the most promising path forward for tree detection in the urban environment involves multi-modal data, particularly those which combine spectral and structural information. Although we propose one well-performing method for this sort of multi-modal data, further research can almost certainly uncover a more sophisticated one.

Perhaps the biggest drawback of our research is that it is geographically limited; that is not to say that our method would not perform well elsewhere, but at the very least without being trained on a more varied dataset it would likely struggle more in different climate zones. Further experiments in more varied regions can improve the model's ability to generalize, as well as perhaps identify new issues that were not apparent in our experiments. Particularly since all methods struggled with densely overlapping tree crowns, an area where this is more prevalent may provide the methods with more data to address this issue, or better highlight the fundamental reasons why this is a particularly difficult sub-task.

A possible avenue of improvement for our approach could be to experiment with losses that more directly optimize for the ground-truth targets. For example, since our losses treat predictions and ground-truths as Gaussian confidence curves, a since prediction can sometimes be used to match two ground-truth, or vice versa. We in fact did experiment with a loss—originally developed for dense crowd counting, as part of a network called Point To Point Net (P2PNet) [62]—that operates directly on a sets of predicted and ground-truth points such that the mapping between them

is guaranteed to be one-to-one. However we found the loss to be unstable in our experiments, and were not able to get it to train effectively. This could be because the ground-truth in our domain are much sparser, or because we simply did not find the right combination of hyper-parameters that suit this loss. More exploration of losses of this sort, that operate directly on point-sets instead of transforming them to an intermediate representation, is a worthwhile avenue of research.

Lastly, an obvious area of continuing research in this domain is to apply our network to downstream forestry tasks, including tree crown delineation (finding the exact borders of the tree crown, instead of just the center) and tree genus/species classification. Given the success of our model in the task of tree detection, and the fact that structural information likely becomes even more important for making these precise assessments, we believe our proposed approach is a promising one. However, these more challenging tasks naturally come with a requirement for more stringently annotated training data, which requires at the very least more time and expertise on the part of the annotator, or even in-person surveying by forestry experts. Thus, as is common in deep learning tasks, there is a need for better and more publicly accessible datasets, comprised not just of tree locations but accompanied by features such as crown size and species, and paired with LIDAR, satellite imagery, and other forms of collected temporally near the date of annotation.

BIBLIOGRAPHY

[1] Forest Inventory and Analysis National Program – Urban FIA Program, Oct. 2021. https://www.fia.fs.fed.us/program-features/urban/.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[3] M. Alonzo, K. Roth, and D. Roberts. Identifying Santa Barbara's urban tree species from AVIRIS imagery using canonical discriminant analysis. *Remote Sensing Letters*, 4(5):513–521, May 2013. https://doi.org/10.1080/2150704X.2013.764027.

[4] E. Ayrey, S. Fraver, J. A. Kershaw, L. S. Kenefic, D. Hayes, A. R. Weiskittel, and B. E. Roth. Layer Stacking: A Novel Algorithm for Individual Forest Tree Segmentation from LiDAR Point Clouds. *Canadian Journal of Remote Sensing*, 43(1):16–27, Jan. 2017. https://doi.org/10.1080/07038992.2017.1252907.

[5] E. Ayrey and D. J. Hayes. The Use of Three-Dimensional Convolutional Neural Networks to Interpret LiDAR for Forest Inventory. *Remote Sensing*, 10(4):649, Apr. 2018.

[6] S. Beery, G. Wu, T. Edwards, F. Pavetic, B. Majewski, S. Mukherjee, S. Chan, J. Morgan, V. Rathod, and J. Huang. The Auto Arborist Dataset: A Large-Scale Benchmark for Multiview Urban Forest Monitoring Under Domain Shift. pages 21294–21307, 2022.

[7] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[8] D. S. Boyd and F. M. Danson. Satellite remote sensing of forest resources: three decades of research development. *Progress in Physical Geography: Earth and Environment*, 29(1):1–26, 2005. https://doi.org/10.1191/0309133305pp432ra.

[9] N. Boyd, E. Jonas, H. Babcock, and B. Recht. DeepLoco: Fast 3D Localization Microscopy Using Neural Networks. Technical report, Feb. 2018.

[10] X. Chen, K. Jiang, Y. Zhu, X. Wang, and T. Yun. Individual Tree Crown Segmentation Directly from UAV-Borne LiDAR Data Using the PointNet of Deep Learning. *Forests*, 12(2):131, Feb. 2021.

[11] F. Chollet and others. Keras, 2015.

[12] D. F. Crouse. On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

[13] M. Dalponte and D. A. Coomes. Tree-centric mapping of forest carbon density from airborne laser scanning and hyperspectral data. *Methods in Ecology and Evolution*, 7(10):1236–1245, 2016. https://onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.12575.

[14] T. Dozat. Incorporating nesterov momentum into adam. 2016.

[15] R. O. Dubayah and J. B. Drake. Lidar Remote Sensing for Forestry. *Journal of Forestry*, 98(6):44–46, June 2000. https://academic.oup.com/jof/article-pdf/98/6/44/22558157/jof0044.pdf.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of*

the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, pages 226–231. AAAI Press, 1996.

[17] L. Eysn, M. Hollaus, E. Lindberg, F. Berger, J.-M. Monnet, M. Dalponte, M. Kobal, M. Pellegrini, E. Lingua, D. Mongus, and N. Pfeifer. A Benchmark of Lidar-Based Single Tree Detection Methods Using Heterogeneous Forest Data from the Alpine Space. *Forests*, 6(5):1721–1747, May 2015.

[18] F. E. Fassnacht, H. Latifi, K. Stereńczak, A. Modzelewska, M. Lefsky, L. T. Waser, C. Straub, and A. Ghosh. Review of studies on tree species classification from remotely sensed data. *Remote Sensing of Environment*, 186:64–87, Dec. 2016.

[19] G. A. Fricker, J. D. Ventura, J. A. Wolf, M. P. North, F. W. Davis, and J. Franklin. A Convolutional Neural Network Classifier Identifies Tree Species in Mixed-Conifer Forest from Hyperspectral Imagery. *Remote Sensing*, 11(19):2326, Jan. 2019.

[20] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *arXiv:1912.12033 [cs, eess]*, June 2020. arXiv: 1912.12033.

[21] Y. Guo, H. Zhang, Q. Li, Y. Lin, and J. Michalski. New morphological features for urban tree species identification using LiDAR point clouds. *Urban Forestry & Urban Greening*, Apr. 2022.

[22] I. Harmon, S. Marconi, B. Weinstein, S. Graves, D. Z. Wang, S. Bohlman, A. Zare, A. Singh, and E. White. Injecting Domain Knowledge Into Deep Neural Networks for Tree Crown Delineation. July 2022. Publisher: TechRxiv.

[23] R. Hecht-nielsen. III.3 - Theory of the Backpropagation Neural Network**Based on "nonindent" by Robert Hecht-Nielsen, which appeared in Proceedings of the

International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE. In H. Wechsler, editor, *Neural Networks for Perception*, pages 65–93. Academic Press, Jan. 1992.

[24] Y. Huang, B. Yu, J. Zhou, C. Hu, W. Tan, Z. Hu, and J. Wu. Toward automatic estimation of urban green volume using airborne LiDAR data and high resolution Remote Sensing images. *Frontiers of Earth Science*, 7(1):43–54, Mar. 2013.

[25] M. K. Jakubowski, W. Li, Q. Guo, and M. Kelly. Delineating Individual Trees from Lidar Data: A Comparison of Vector- and Raster-based Segmentation Approaches. *Remote Sensing*, 5(9):4163–4186, Sept. 2013.

[26] G. Jansson and P. Angelstam. Threshold levels of habitat composition for the presence of the long-tailed tit (Aegithalos caudatus) in a boreal landscape. *Landscape Ecology*, 14(3):283–290, June 1999.

[27] R. R. Jensen, P. J. Hardin, and A. J. Hardin. Classification of urban tree species using hyperspectral imagery. *Geocarto International*, 27(5):443–458, Aug. 2012. https://doi.org/10.1080/10106049.2011.638989.

[28] S. Ji, W. Xu, M. Yang, and K. Yu. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

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

[30] F. J. Kriegler, W. A. Malila, R. F. Nalepka, and W. Richardson. Preprocessing Transformations and Their Effects on Multispectral Recognition. In *Remote Sensing of Environment, VI*, Jan. 1969.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[32] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[33] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[34] W. Li, H. Fu, L. Yu, and A. Cracknell. Deep Learning Based Oil Palm Tree Detection and Counting for High-Resolution Remote Sensing Images. *Remote Sensing*, 9(1):22, Jan. 2017.

[35] W. Li, Q. Guo, M. K. Jakubowski, and M. Kelly. A New Method for Segmenting Individual Trees from the Lidar Point Cloud. *Photogrammetric Engineering & Remote Sensing*, 78(1):75–84, Jan. 2012.

[36] X. Li, W. Y. Chen, G. Sanesi, and R. Lafortezza. Remote Sensing in Urban Forestry: Recent Applications and Future Directions. *Remote Sensing*, 11(10):1144, Jan. 2019.

[37] M. Liu, Z. Han, Y. Chen, Z. Liu, and Y. Han. Tree species classification of LiDAR data based on 3D deep learning. *Measurement*, 177:109301, June 2021.

[38] T. Liu, J. Im, and L. J. Quackenbush. A novel transferable individual tree crown delineation model based on Fishing Net Dragging and boundary classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 110:34–47, Dec. 2015.

[39] Z. Liu, H. Tang, Y. Lin, and S. Han. Point-Voxel CNN for Efficient 3D Deep Learning. *arXiv:1907.03739 [cs]*, Dec. 2019. arXiv: 1907.03739.

[40] S. Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982. Publisher: IEEE.

[41] H. Lu and H. Shi. Deep Learning for 3D Point Cloud Understanding: A Survey. *arXiv:2009.08920 [cs]*, May 2021. arXiv: 2009.08920.

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

[43] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015.

[44] E. G. McPherson and A. Kendall. A life cycle carbon dioxide inventory of the Million Trees Los Angeles program. *The International Journal of Life Cycle Assessment*, 19(9):1653–1665, 2014.

[45] F. Meyer. Topographic distance and watershed lines. *Signal Processing*, 38(1):113–125, July 1994.

[46] E. Nehme, L. E. Weiss, T. Michaeli, and Y. Shechtman. Deep-STORM: super-resolution single-molecule microscopy by deep learning. *Optica*, 5(4):458–464, Apr. 2018. Publisher: OSA.

[47] M. Onishi and T. Ise. Explainable identification and mapping of trees using UAV RGB image and deep learning. *Scientific Reports*, 11(1):903, Jan. 2021.

[48] E. G. Parmehr, M. Amati, and C. S. Fraser. Mapping urban tree canopy cover using fused airborne LiDAR and satellite imagery data. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(7), 2016.

[49] PDAL Contributors. PDAL Point Data Abstraction Library, Nov. 2018.

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

[51] A.-I. Pleșoianu, M.-S. Stupariu, I. Șandric, I. Pătru-Stupariu, and L. Drăguț. Individual Tree-Crown Detection and Species Classification in Very High-Resolution Remote Sensing Imagery Using a Deep Learning Ensemble Model. *Remote Sensing*, 12(15):2426, Jan. 2020.

[52] S. C. Popescu and R. H. Wynne. Seeing the Trees in the Forest. *Photogrammetric Engineering & Remote Sensing*, 70(5):589–604, 2004.

[53] S. C. Popescu, R. H. Wynne, and R. F. Nelson. Measuring individual tree crown diameter with lidar and assessing its influence on estimating forest volume and biomass. *Canadian Journal of Remote Sensing*, 29(5):564–577, 2003. https://doi.org/10.5589/m03-027.

[54] H. Pretzsch, P. Biber, E. Uhl, J. Dahlhausen, G. Schütze, D. Perkins, T. Rötzer, J. Caldentey, T. Koike, T. v. Con, A. Chavanne, B. d. Toit, K. Foster, and B. Lefer. Climate change accelerates growth of urban trees in metropolises worldwide. *Scientific Reports*, 7(1):15403, Nov. 2017.

[55] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. Nov. 2017.

[56] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv:1612.00593 [cs]*, Apr. 2017. arXiv: 1612.00593.

[57] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *arXiv:1706.02413 [cs]*, June 2017. arXiv: 1706.02413.

[58] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.

[59] X. Shang and L. A. Chisholm. Classification of Australian Native Forest Species Using Hyperspectral Remote Sensing and Machine-Learning Classification Algorithms. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6):2481–2489, June 2014. Conference Name: IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

[60] C. A. Silva, A. T. Hudak, L. A. Vierling, E. L. Loudermilk, J. J. O'Brien, J. K. Hiers, S. B. Jack, C. Gonzalez-Benecke, H. Lee, M. J. Falkowski, and A. Khosravipour. Imputation of Individual Longleaf Pine (Pinus palustris Mill.) Tree Attributes from Field and LiDAR Data. *Canadian Journal of Remote Sensing*, 42(5):554–573, Sept. 2016. https://doi.org/10.1080/07038992.2016.1196582.

[61] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Sept. 2014.

[62] Q. Song, C. Wang, Z. Jiang, Y. Wang, Y. Tai, C. Wang, J. Li, F. Huang, and Y. Wu. Rethinking counting and localization in crowds: A purely point-

based framework. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3365–3374, 2021.

[63] J. Ventura, M. Honsberger, C. Gonsalves, J. Rice, C. Pawlak, N. L. S. Han, V. Nguyen, K. Sugano, J. Doremus, G. A. Fricker, J. Yost, and M. Ritter. Individual tree detection in large-scale urban environments using high-resolution multispectral imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2022. Note: Forthcoming publication.

[64] M. S. Warren, S. P. Brumby, S. W. Skillman, T. Kelton, B. Wohlberg, M. Mathis, R. Chartrand, R. Keisler, and M. Johnson. Seeing the Earth in the Cloud: Processing one petabyte of satellite imagery in one day. In *2015 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 1–12, Oct. 2015. ISSN: 2332-5615.

[65] B. G. Weinstein, S. Marconi, S. Bohlman, Alina Zare, and E. White. Individual Tree-Crown Detection in RGB Imagery Using Semi-Supervised Deep Learning Neural Networks. Technical report, Feb. 2021.

[66] B. Wu, B. Yu, Q. Wu, Y. Huang, Z. Chen, and J. Wu. Individual tree crown delineation using localized contour tree method and airborne LiDAR data in coniferous forests. *International Journal of Applied Earth Observation and Geoinformation*, 52:82–94, Oct. 2016.

[67] W. Wu, Z. Qi, and L. Fuxin. PointConv: Deep Convolutional Networks on 3D Point Clouds. *arXiv:1811.07246 [cs]*, Nov. 2020. arXiv: 1811.07246.

[68] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. pages 1912–1920, 2015.

[69] Q. Xiao, S. L. Ustin, and E. G. McPherson. Using AVIRIS data and multiple-masking techniques to map urban forest tree species. *In-*

ternational Journal of Remote Sensing, 25(24):5637–5654, Dec. 2004. https://doi.org/10.1080/01431160412331291224.

[70] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[71] C. Zhang, Y. Zhou, and F. Qiu. Individual Tree Segmentation from LiDAR Point Clouds for Urban Forest Inventory. *Remote Sensing*, 7(6):7892–7913, June 2015.

[72] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847*, 2018.

[73] L. Zhu, Z. Zhao, C. Lu, Y. Lin, Y. Peng, and T. Yao. Dual path multi-scale fusion networks with attention for crowd counting. *arXiv preprint arXiv:1902.01115*, 2019.

[74] J. Zörner, J. R. Dymond, J. D. Shepherd, S. K. Wiser, and B. Jolly. LiDAR-Based Regional Inventory of Tall Trees—Wellington, New Zealand. *Forests*, 9(11):702, Nov. 2018.

APPENDICES

Appendix A

ESTIMATED CO2 EMISSIONS RELATED TO EXPERIMENTS

Experiments were conducted using private infrastructure, which has an estimated carbon efficiency of 0.2376824 kgCO_2eq/kWh. 1144 cumulative GPU-hours of computation were performed on hardware of type Tesla V100-PCIE-16GB (TDP of 300W).

Total emissions are estimated to be 81.57 kg of CO_2 equivalent. This is approximately the emissions caused by 366 km (227 mi) driven with an average internal combustion engine car.

Estimations were conducted using the MachineLearning Impact calculator[1] presented in [32].

[1]https://mlco2.github.io/impact#compute