

# **Internship Report**

**Tree object detection using airborne images and LiDAR  
point clouds**

**Alexandre Bry**

**July 2024**

Computer Science Department  
École polytechnique,  
Palaiseau, France

Research Department  
Geodan B.V., Amsterdam,  
The Netherlands

# Abstract

The goal of the internship was to study the combination of LiDAR point clouds and aerial images in a deep learning model to identify individual trees, and in particular those covered by other trees. To do this, I modified a model capable of merging LiDAR and RGB data to feed it with more information about the geometry below the canopy surface. This required to create my own tree dataset, using publicly available data from the Netherlands. A few interesting results emerged and the model proved its ability to quickly learn to find large and medium trees, even with a small training dataset. However, this new pipeline should be evaluated on a larger dataset to precisely determine the influence of the modifications on the performance regarding small and covered trees.

The source code for this report can be found [here](#)<sup>1</sup> and the online version is [here](#)<sup>2</sup>. The source code for the project can be found [here](#)<sup>3</sup>.

---

<sup>1</sup><https://github.com/alexandre-bry/Geodan-internship-report>

<sup>2</sup><https://alexandre-bry.github.io/Geodan-internship-report>

<sup>3</sup><https://github.com/sogelink-research/tree-segmentation>

# Acknowledgments

Firstly I would like to express my gratitude to Tom van Tilburg, who helped me a lot at every step of the internship. From data management and processing to the architecture of the model, he brought a lot of new ideas and encouraged me to question my own. Brian de Vogel also deserves my heartfelt thanks for his ideas and his unconditional help regarding any practical and material matter. Together with Tom, they took from their time to supervise me every week, answer to any of my questions and introduce me to several geo-spatial topics that I knew nothing about beforehand.

Then I would like to thank the two people who made this internship possible in the first place. I am grateful to Henk Scholten, who supported my internship all along and enabled me to meet new people in several major institutions. A special thanks also to Rene Bruinink for his direct and sincere interest towards my spontaneous application, as well as for his transparency and honesty from the beginning of the internship discussion.

Finally I am sincerely thankful to the rest of the Research Team, Anne Blankert, Bert Temme, Kamiel Verhelst, Stefan de Graaf and Tim Ebben for warmly welcoming and integrating me to the team. Loïc de Jong also deserves a special thanks for his help regarding many practical matters and administrative tasks that were specific to living in the Netherlands. He also supported me a lot in my adaptation to this new country, even though it is not so different from France.

# Table of contents

<b>1 State-of-the-art</b>	<b>7</b>
1.1 Computer vision tasks related to trees . . . . .	7
1.2 Datasets . . . . .	8
1.2.1 Requirements . . . . .	8
1.2.2 Existing datasets with annotated trees . . . . .	8
1.2.3 Public data . . . . .	9
1.2.4 Dataset augmentation techniques . . . . .	11
1.3 Algorithms and models . . . . .	11
1.3.1 Images only . . . . .	11
1.3.2 LiDAR only . . . . .	12
1.3.3 LiDAR and images . . . . .	13
<b>2 Objectives and motivations</b>	<b>14</b>
2.1 General idea . . . . .	14
2.2 Covered trees . . . . .	14
2.3 Multiple layers of CHM . . . . .	15
<b>3 Dataset creation</b>	<b>16</b>
3.1 Definition and content . . . . .	16
3.2 Data annotation setup . . . . .	16
3.3 Challenges and solutions . . . . .	19
3.4 Augmentation methods . . . . .	21
<b>4 Model and training</b>	<b>23</b>
4.1 Model architecture . . . . .	23
4.2 Training pipeline . . . . .	24
4.2.1 Data preprocessing . . . . .	24
4.2.2 Training loop . . . . .	24
4.2.3 Output postprocessing . . . . .	25
<b>5 Results</b>	<b>28</b>
5.1 Training parameters . . . . .	28
5.2 CHM layers . . . . .	30
5.3 Covered trees . . . . .	31
5.4 Visual results . . . . .	33
<b>6 Discussion and improvements</b>	<b>35</b>
6.1 Dataset . . . . .	35
6.2 Instability . . . . .	36
6.3 Model performance . . . . .	36

# List of figures

3.1	One data instance with ground-truth bounding boxes . . . . .	17
3.2	Data sources used to annotate the trees . . . . .	18
3.3	A tree that was cut off and replaced . . . . .	19
3.4	Example of data misalignment . . . . .	20
3.5	Examples of data augmentations on an RGB image with the probabilities used when training the model. . . . .	22
4.1	Model architecture . . . . .	23
4.2	Example of three sortedAP curves with the results of a model with three different data combinations . . . . .	26
4.3	Example of the values of sortedAP depending on the confidence threshold	27
5.1	Results with different training parameters for all experiments . . . . .	29
5.2	Results with different training parameters for all evaluation data setups	30
5.3	Results with different CHM layers . . . . .	31
5.4	sortedAP curve with different CHM layers on training/validation/test set . . . . .	32
5.5	Ground-truth boxes and predictions from two models . . . . .	34

# Introduction

Individual detection of trees is an increasingly important topic in computer vision, because of its numerous applications and the recent major improvements in object detection. Trees are more and more monitored, from forest carbon stocks to urban tree inventories, to allow for a better assessment of their evolution and a better management of forests and urban trees. Most of the tasks related to trees either require or would benefit from a method allowing precise delineation and separation of trees as individual entities.

To identify trees, the two most common types of data used are LiDAR point clouds and visible/hyperspectral images. The two types of data are indeed complementary, as point clouds capture geometric information with multiple features, while images capture colors. However, combining them into a single pipeline is not an easy task because they inherently have very different spatial organization and encoding. This difficulty holds true for both analytical methods and deep learning models. However, some deep learning models still manage to merge point clouds and images by creating rasters from the surface of the point cloud, which can then be handled similarly to visible and hyperspectral images.

In this work, I focused on one specific deep learning model, and tried to improve it by extracting more information from the LiDAR point cloud, which also contains a lot of data below the canopy surface. With this modification, I also hoped to be able to experiment on the ability of the model to detect covered trees, which are trees invisible from above because of occlusion by a larger tree. This is a task that is completely impossible for the initial model before my modifications, because all the data it takes as input is limited to what is visible from above.

To experiment on this, I had to create my own tree annotations dataset, using high-quality data available on the whole of the Netherlands, and hand-annotations for the trees. I then implemented the whole training pipeline and the model from scratch, before conducting experiments to assess the impact of my modifications.

# 1 State-of-the-art

## 1.1 Computer vision tasks related to trees

Before talking about models and datasets, let's define properly the task that this project focused on, in the midst of all the various computer vision tasks, and specifically those related to tree detection.

The first main differentiation between tree recognition tasks comes from the acquisition of the data. There are some very different tasks and methods using either ground data or aerial/satellite data. This is especially true when focusing on urban trees, since a lot of street view data is available [1].

This leads to the second variation, which is related to the kind of environment that we are interested in. Papers in this field usually focus on one or two types of tree environments: urban areas [1, 2], tree plantations [3, 4, 5] and forests [3, 6, 7]. These types of environments influence, among other things, the organization of the trees in space. This is important, because the tasks and the difficulty depends on the type of environment. Tree plantations are much easier to work with than completely wild forests, while urban areas contain various levels of difficulty ranging from alignment trees to private and disorganized gardens and parks. For this project, we mainly focused on urban areas, but the pipeline and the model should still be applicable to tree plantations and forests.

Then, there are four fundamental computer vision tasks that have their respective applications when dealing with trees [4]:

- Classification, which consists in assigning one class label to an image, equivalent to putting it into a category given a list of possible categories. This is quite rare for airborne tree applications though since there are multiple trees on each image most of the time
- Detection, which consists in detecting objects and placing boxes around them
- Semantic segmentation, which consists in associating a class label to every pixel of an image,
- Instance segmentation, which consists in adding a layer of complexity to semantic segmentation by also differentiating between the different instances of each class

These generic tasks can be extended by trying to get more information from the data about the trees. The most common pieces of information are the species and the height, but some models also try to predict the health of the trees [4], or their carbon stock [8].

In this work, I focus on the detection of trees, with a classification between several labels related to the discrepancies between the different kinds of data.

## 1.2 Datasets

### 1.2.1 Requirements

Before presenting the different annotated trees datasets and the reasons why they were not fully usable for the project, let's enumerate the different conditions and requirements I was looking for to properly train the model:

- Multiple types of data:
  - Aerial RGB images
  - LiDAR point clouds (preferably aerial)
  - Aerial infrared (CIR) images (optional)
- Tree crown annotations or bounding boxes
- High-enough resolution:
  - For images, about 25 cm
  - For point clouds, about 10 cm

Here are the explanations for these requirements. As for the types of data, RGB images and point clouds are required to experiment on the ability of the model to combine the two very different kinds of information they hold. Infrared data can also improve tree detection, but it was optional for this work because RGB images are enough to study the combination. Regarding tree annotations, it is necessary to have a way to spatially identify them individually, using crown contours or simply bounding boxes. Since the model outputs bounding boxes, any kind of other format can easily be transformed to bounding boxes. Finally, the resolution has to be high enough to identify all individual trees, including the smallest ones. For the point clouds especially, the whole idea is to see if and how the topology of the trees can be learnt, using at least the trunks and even the biggest branches if possible. Therefore, even if they are not really comparable, this is the reason why the required resolution is more precise for the point clouds.

### 1.2.2 Existing datasets with annotated trees

As explained above, there are quite a lot of requirements to fulfill to have a complete dataset with annotated trees which is suitable for the task. In practice, almost all the available datasets with annotated trees are missing something, as they are mainly focusing on using one kind of raw data (either spectral/hyperspectral images or LiDAR point clouds) and try to make the most out of it, instead of trying to use all the types of data together.

The most comprehensive list of tree annotations datasets was published in OpenForest [9]. FoMo-Bench [10] also lists several interesting datasets, even though most of

them can also be found in OpenForest. Without enumerating all of them, there are multiple kinds of datasets that all have their own flaws regarding the requirements of this work.

Firstly, there are the forest inventories. TALLO [11] is probably the most interesting one in this category, because it contains a lot of spatial information about almost 500K trees, with their locations, their crown radii and their heights. Therefore, everything needed to localize trees is in the dataset. However, I didn't manage to find RGB images or LiDAR point clouds of the areas where the trees are located, making it impossible to use these annotations to train tree detection.

Secondly, there are the RGB datasets. Two examples of these datasets with a high quality of image are ReforestTree [8] and MillionTrees [12]. The only but major drawback of these datasets is obviously that they don't provide any kind of point cloud, which makes them unsuitable for the task.

Thirdly, there are the LiDAR datasets, such as [13] and [14]. Similarly to RGB datasets, they lack one of the data source for the task I worked on. But unlike them, they have the advantage that the missing data could be much easier to acquire from another source, since RGB aerial or satellite images are much more common than LiDAR point clouds. However, this solution was abandoned for two main reasons. First it is often quite challenging to find the exact locations where the point clouds were acquired. Then, even when the location is known, it is often in the middle of a forest where the quality of openly available satellite imagery very low.

Finally, I also found two datasets that had RGB and LiDAR components. The first one is MDAS [15]. This benchmark dataset encompasses RGB images, hyperspectral images and Digital Surface Models (DSM). There are however two major flaws. The obvious one is that this dataset was created with land semantic segmentation tasks in mind, so there is no tree annotations. The less obvious one is that a DSM is not a point cloud, even though it is some kind of 3D information and is often created using a LiDAR point cloud. As a consequence, this substantially limits the ability to experiment with the point cloud.

The only real dataset with RGB and LiDAR comes from NEON [6]. This dataset contains exactly all the data I was looking for, with RGB images, hyperspectral images and LiDAR point clouds. With 30975 tree annotations, it is also a quite large dataset, spanning across multiple various forests. The main reason why I decided not to use it in the end is the quality of the data, which is not bad but not as great as the one from the data available for the Netherlands, which I will talk about in the next section Section 1.2.3.

### 1.2.3 Public data

After rejecting all the available datasets I had found, the only remaining solution was to create my own dataset. I won't dive too much in this process that I will explain in Chapter 3. I just want to mention all the publicly available raw data that I used or could have used to create this custom dataset.

For practical reasons, the two countries where I mostly searched for available data are France and the Netherlands. I was looking for three different data types independently:

- RGB (and if possible CIR) images
- LiDAR point clouds
- Tree annotations

These three types of data are available in similar ways in both countries, although the Netherlands have a small edge over France. RGB images are really easy to find in France with the BD ORTHO [16] and in the Netherlands with the Luchtfotos [17], but the resolution is better in the Netherlands (8 cm vs 20 cm). Hyperspectral images are also available in both countries, although for those the resolution is only 25 cm in the Netherlands.

As for LiDAR point clouds, the Netherlands have a small edge over France, because they have already completed their forth version covering the whole country with AHN4 [18], and are working on the fifth version. In France, data acquisition for the first LiDAR point cloud covering the whole country started a few years ago [19]. It is not yet finished, even though the data is already available for half of the country. The other advantage of the data from the Netherlands regarding LiDAR point clouds is that all flights are performed during winter, which allows light beams to penetrate more deeply in trees and reach trunks and branches. This is not the case in France, were data is acquired during the whole year, adding a level of variation and therefore more difficulty.

The part that is missing in both countries is related to tree annotations. Many municipalities have datasets containing information about all the public trees they handle. This is for example the case for Amsterdam [20] and Bordeaux [21]. However, these datasets cannot really be used as ground truth for a custom dataset for several reasons. First, many of them do not contain coordinates indicating the position of each tree in the city. Then, even those that contain coordinates are most of the time missing any kind of information allowing to deduce a bounding box for the tree crowns. Finally, even if they did contain everything, they only focus on public trees, and are missing every single tree located in a private area. Since public and private areas are obviously imbricated in all cities, it means that any area we try to train the model on would be missing all the private trees, making the training process impossible because we cannot have only a partial annotation of images.

The other tree annotation source that we could have used is the Boomregister [22]. This work covers the whole of the Netherlands, including public and private trees. However, the precision of the masks is far from perfect, and many trees are missing or incorrectly segmented, especially when they are less than 9 m high or have a crown diameter smaller than 4 m. Therefore, even though it is a very impressive piece of work, I decided that it could not be used as training data for deep learning models due to its biases and imperfections. Therefore, the only remaining solution was to annotate trees by myself, to create my own dataset of annotated trees using the available data.

### 1.2.4 Dataset augmentation techniques

When a dataset is too small to train a model, there are several ways of artificially enlarging it.

The most common way is to randomly apply deterministic or random transformations to the data, during the training process, to be able to generate several unique and different realistic data instances from one real data instance. There are a lot of different transformations that can be applied to images, divided into two categories: pixel-level and spatial-level [23]. Pixel-level transformations modify the value of individual pixels, by applying different filters, such as random noise, color shifts and more complex effects like fog and sun flare. Spatial-level transformations modify the spatial arrangement of the image, without changing the pixel values. In other words, these transformations move the pixels in the image. These transformations range from simple rotations and cropings to complex spatial distortions. In the end, all these transformations are simply producing one artificial image out of one real image.

Another way to enlarge a dataset is to instead generate completely new input data sharing the same properties as the initial dataset. This can be done using Generative Adversarial Networks (GAN). These models usually have two parts, a generator and a discriminator, which are trained in parallel. The generator learns to produce realistic artificial data, while the discriminator learns to discriminate between real data and artificial data produced by the generator. If the training is successful, we can then use the generator and random seeds to generate random but realistic artificial data similar to the dataset. This method has for example been successfully used to generate artificial tree height maps [24].

However, training GANs can be very unstable, and I haven't found any paper applying this technique to generate LiDAR and RGB data at the same time. The artificial instances would need to be consistent between the two types of data, which might be very difficult. Therefore, I only used the random image transformations during the training process, because the chances of training of successful GAN seemed too low to be worth it.

## 1.3 Algorithms and models

In this section, the different algorithms and methods are grouped according to the type of data they use as input.

### 1.3.1 Images only

First, there are methods that perform tree detection using only visible or hyperspectral images or both. Several different algorithms have been developed to analytically delineate tree crowns from images, by using the particular shape of the trees and its effect on images [3]. Without diving into the details, here are a few of them. The watershed algorithm identifies trees to inverted watersheds using the grey-scale image

and tree crowns frontiers are found by incrementally flooding the watersheds [25]. The local maxima filtering uses the intensity of the pixels in the grey-scale image to identify the brightest points locally and use them as treetops [26]. Reversely, the valley-following algorithm uses the darkest pixels which are considered as the junctions between the trees since shaded areas are the lower part of the tree crowns [27]. Another interesting algorithm is template matching. This algorithm simulates the appearance of simple tree templates with the light effects, and tries to identify similar patterns in the grey-scale image [28]. Combinations of these techniques and others have also been proposed.

But with the recent developments of deep learning in image analysis, deep learning models are increasingly used to detect trees using RGB images. In some cases, deep learning is used to extract features that can then be the input of one of the algorithms described above. One example is the use of two neural networks to predict masks, outlines and distance transforms which can then be the input of a watershed algorithm [29]. In other cases, a deep learning model is responsible of directly detecting tree masks or bounding boxes, often using CNNs, given the images [30].

### 1.3.2 LiDAR only

Reversely, some of the methods to identify individual trees use LiDAR data only. There are a lot of different ways to use and analyze point clouds, but the one that is mostly used for trees is based on height maps, or Canopy Height Models (CHM).

A CHM is a raster computed as the subtraction of the Digital Terrain Model (DTM) to the Digital Surface Model (DSM). What it means is that a CHM contains the height above ground of the highest point in the area corresponding to each pixel. This CHM can for example be used as the input raster for the watershed algorithm, as it contains the height values that can be used to determine local maxima [31]. The other analytical methods described in the previous section (Section 1.3.1) also have their equivalents using the CHM.

A lot of different analytical methods and variations of the simple CHM were proposed to perform individual tree detection, but in the end, most of them still use the concept of local maxima [32, 33]. A CHM can also be used as the input of any kind of convolutional neural network (CNN) because it is shaped exactly like any image. This allows to use a lot of different techniques usually applied to object detection in images.

Then, even though I finally used an approach similar to the CHM, I want to mention other kinds of deep learning techniques that exist and could potentially leverage all the information contained in a point cloud. These techniques can be divided in two categories: projection-based and point-based methods [34]. The main difference between the two is that projection-based techniques are based on grids while point-based methods take unstructured point clouds as input. Among projection-based methods, the most basic method is 2D CNN, which is how CHM can be processed. Then, multiview representation tries to tackle the 3D aspect by projecting the point cloud in multiple directions before merging them together. To really deal with 3D

data, volumetric grid representation consists in using 3D occupancy grids, which are processed using 3D CNNs. Among point-based methods, there are methods based on PointNet, which are able to extract features and perform the classical computer vision tasks by taking point clouds as input. Finally, Convolutional Point Networks use a continuous generalization of convolutions to apply convolution kernels to arbitrarily distributed point clouds.

### 1.3.3 LiDAR and images

Let's now talk about the models of interest for this work, which are machine learning pipelines using both LiDAR point cloud data and RGB images.

One example is a pipeline which uses a watershed algorithm to extract crown boundaries, before extracting individual tree features from the LiDAR point cloud, hyperspectral and RGB images [35]. These features are then used by a random forest classifier to identify which species the tree belongs to. This pipeline therefore makes the most out of all data to identify species, but sticks to an improved variant of the watershed algorithm for individual tree segmentation, which only uses a CHM raster.

Other works focused on using a model from end to end that is able to take both the CHM and the RGB data as input and combine them to make the most out of all the available data. Among other examples, there are ACE R-CNN [36], an evolution of Mask region-based convolution neural network (Mask R-CNN), and AMF GD YOLOv8 [7], an evolution of YOLOv8. These two models have proven to give much better results when using both the images and the LiDAR data as a CHM than when using only one of them.

In this work, I focus on AMF GD YOLOv8, which looks promising and flexible thanks to the flexibility of YOLOv8 [37]. Indeed, YoloV8 has different variants allowing the model to deal with different computer vision tasks.

## 2 Objectives and motivations

In this section, I will explain the objectives that I set for this internship and the motivations that led to them.

### 2.1 General idea

The basis for this internship was to look at deep learning models to detect trees using LiDAR and aerial images. In four months, it would have been difficult to dive into the literature, think about a completely new approach and develop it. Therefore, I wanted to find an interesting and not too complicated deep learning model, and try a few changes that would hopefully improve the results.

This idea was also reinforced by the decision to create my own dataset, which stemmed from two reasons. The first reason was the small number of openly available tree annotation datasets which contained both LiDAR and RGB data. I therefore thought that creating a new dataset and making it available could be a great contribution. The second reason was to have more control over the definition and the characteristics of the dataset, to be able to experiment on the detection of specific trees.

### 2.2 Covered trees

The main thing that I wanted to experiment on was the possibility to make better use of the LiDAR point cloud to be able to detect covered trees. Covered trees are the trees which are located partially or completely under another tree crown. This makes them impossible to completely delineate when using only data that is visible from above. These trees are not meaningless or negligible, because as demonstrated in this paper [33], they can represent up to 50% of the trees in a forest.

However, doing this implied being able to process them on the whole pipeline. In practice, covered trees are never annotated in all the datasets that are created using only RGB images, because they are simply not visible. This means that creating my own dataset was the only solution to have a dataset containing all trees including covered trees and be able to easily identify them.

## 2.3 Multiple layers of CHM

Being able to find covered trees meant finding a way to extract more information out of the LiDAR point cloud than what is contained in the CHM. In fact, the CHM only contains a very small part of the point cloud and doesn't really benefit from the 3D information that is contained inside the point cloud, only from its 3D appearance from above. This is particularly true when the point cloud is acquired in a season where trees don't have their leaves, because the LiDAR then goes deep into the tree more easily, and can find the trunk and many of the largest branches.

Therefore, getting information below the tree crown surface was mandatory to find covered trees. But it could also be helpful for the model to find better separations between each tree, thanks to having access to the branches and the trunks.

To do this, I wanted to stick to a simple solution that would integrate well with the initial model and wouldn't require too many changes. The idea I implemented is therefore very simple. Instead of having only one CHM raster, the model will take multiple layers, each focusing on a different height interval. One way to do this, which is used in the third method of this paper [32], would be to use the previous CHM by removing all the points that are in the interval between the CHM height and 0.5 m below, before computing an additional layer. Instead, I tried a more simple and straightforward way to do it, by removing all the points above a certain height threshold, and compute the CHM with the points that are left. Doing this for multiple height thresholds creates a more complete view of what the point cloud looks like at multiple levels, which gives a lot more information about the organization of the point cloud.

## 3 Dataset creation

### 3.1 Definition and content

As explained in Section 1.2.1, the main requirements of the dataset that I wanted to create were to contain at the same time LiDAR data, RGB data and CIR data, with simple bounding box annotations for all trees, including trees that are partially or completely covered by other trees (see Section 2.2).

Then, to make the most out of the point cloud resolution and the RGB images resolution, I decided to use a CHM resolution of 8 cm, which is also the resolution of the RGB images. However, the resolution of CIR images is 25 cm, which made it less optimal, but still usable.

To be able to get results even with a small dataset, I decided to focus on one specific area, to limit the diversity of trees and environments to something that should still be learnt with a small dataset. Therefore, the whole dataset is currently inside of a  $1 \text{ km} \times 1 \text{ km}$  square around the Geodan office in Amsterdam. It contains 2726 annotated trees spread over 241 images of size 640 px  $\times$  640 px i.e. 51.2 m  $\times$  51.2 m. All tree annotations have at least a bounding box, and some of them have a more accurate polygon representing the shape of the crown. There are four classes, which I will detail in the next section Section 3.3, and each tree belongs to one class.

On Figure 3.1, you can see what the dataset finally looks like, with all data sources and bounding boxes.

### 3.2 Data annotation setup

Annotating all these trees took me about 100 hours, with a very high variation of the time spent per tree depending on the complexity of the area. To make the most out of the different data types and fully use them, I mainly used four data sources and tools.

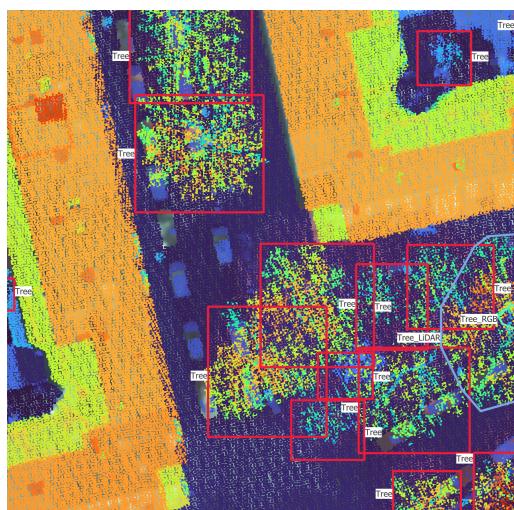
The most important tool I used is QGIS [38]. QGIS allowed me to visualize all types of data on the same map: images, point clouds, CHM rasters, boxes and polygons. I therefore used it to annotate the trees, switching between RGB images and LiDAR point clouds to detect individual trees. One very useful feature of QGIS is the possibility to filter the point cloud based on some properties of the points. I mainly used it to filter out all the points that were classified as buildings and to adapt the height interval to make the most out of the color gradient and get rid of the ground or the large trees. This way of removing part of the point cloud using



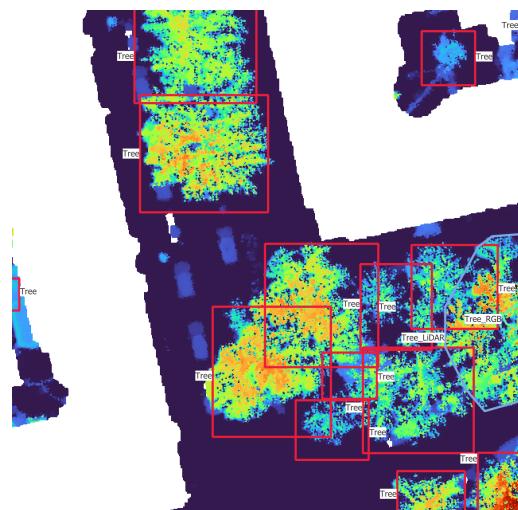
(a) RGB image



(b) CIR image



(c) LiDAR point cloud



(d) CHM raster

Figure 3.1: One data instance with ground-truth bounding boxes

height thresholds is also related to the idea of having multiple CHM layers fed into the model using height intervals of the point cloud.

However, QGIS is mostly limited to 2D views from the top, which doesn't benefit from the 3D aspect of the point cloud. To visualize the point cloud, I used the official viewer provided with the AHN4, called AHN-puntenwolkenviewer, which means AHN point cloud viewer [39]. This viewer allows to navigate through the point cloud, see it from different angles and color the points according to different attributes. I found the intensity (intensiteit) to be a convenient way to display the point cloud, as it sometimes allow to differentiate between leaves and branches, and it breaks the homogeneity of using the height as color attribute. This viewer enabled me to make sense out of the 2D view of the point cloud I had in QGIS.

Finally, I also used Google Street View [40] to have visual information about the trees. This was mainly useful for the most difficult parts of the dataset, but it was only available for trees that were not too far from a street. Google Street View images helped me in dense areas with thin trees, where the point cloud doesn't capture very well the structure of the trees and trees are mostly indistinguishable from above. Having the possibility to switch between different dates also enabled me to see the evolutions and have more chances to find a clear picture.

In Figure 3.2, you can see what each data source looks like.

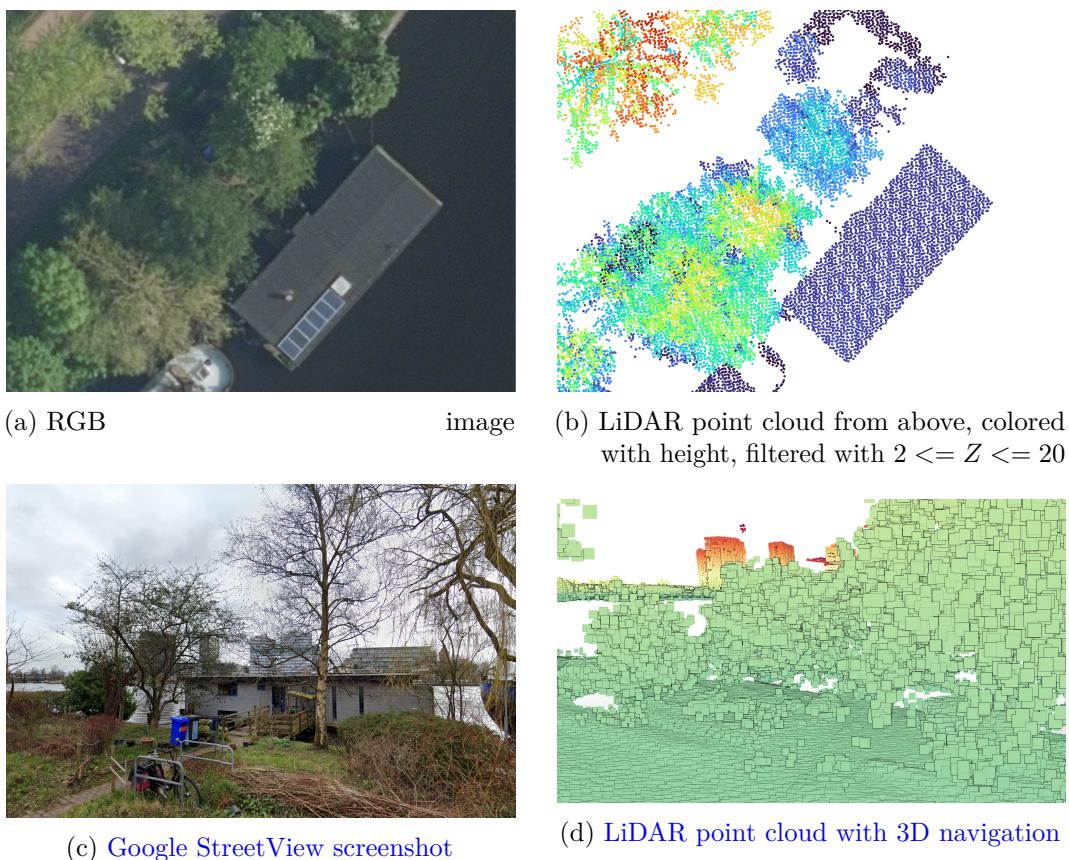


Figure 3.2: Data sources used to annotate the trees

### 3.3 Challenges and solutions

The creation of this dataset raised a number of challenges. The first one was the interval of time between the acquisition of the different types of data. While the point cloud data dated from 2020, the RGB images were acquired in 2023. It would have been possible to use images from 2021 or 2022 with the same resolution, but the quality of the 2023 images was much better, in the sense that trees were much more distinguishable. Consequently, there were a certain amount of changes regarding trees between these two periods of acquisition. Some large trees were cut, while small trees were planted, sometimes even at the position of old trees that were previously cut in the same time frame. For this reason, a non negligible number of trees were either present only in the point cloud, or only in the images. An example of such a situation can be found in Figure 3.3. To try to handle this situation, I created two new class labels corresponding to these situations. This amounted up to 4 class labels:

- “Tree”: trees which are visible in the point cloud and the images
- “Tree\_LiDAR”: trees which are visible in the point cloud only but would be visible in the images if they had been there during the acquisition
- “Tree\_RGB”: trees which are visible in the images only but would be visible in the point cloud if they had been there during the acquisition
- “Tree\_covered”: trees that are visible in the point cloud only because they are covered by other trees.

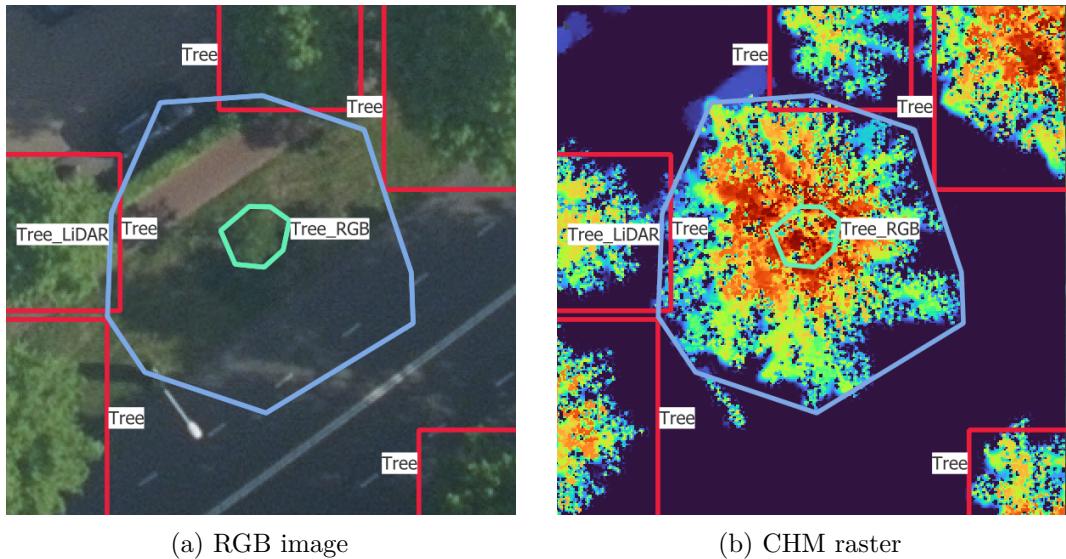


Figure 3.3: A tree that was cut off and replaced

The next challenge was the misalignment of images and point cloud. This misalignment comes from the images not being perfectly orthogonal. Point clouds don't have this problem, because the data is acquired and represented in 3D, but objects in images have to be projected to a 2D plane after being acquired with an angle that is not perfectly orthogonal to the plane. Despite the post-processing that was surely performed on the images, they are therefore not perfect, and there is a shift

between the positions of each object in the point cloud and in the images. This shift cannot really be solved, because it depends on the position of the object relative to the sensor. Because of this misalignment, a choice had to be made as to where tree annotations should be placed, using either the point clouds or the RGB images. I chose to the RGB images as it is simpler to visualize and annotate, but there was not really a perfect choice.

On Figure 3.4, you can see two of the issues. First, you can see that a bounding box that is well-centered around the tree in the RGB image is completely off on the CIR image, and also not really centered on the CHM raster. Then, you can see that the bounding box is much smaller on the CHM, mainly for two reasons: the tree grew between the acquisition of the LiDAR point cloud and the RGB image and small branches on the outside of the tree are hard to capture for LiDAR beams.

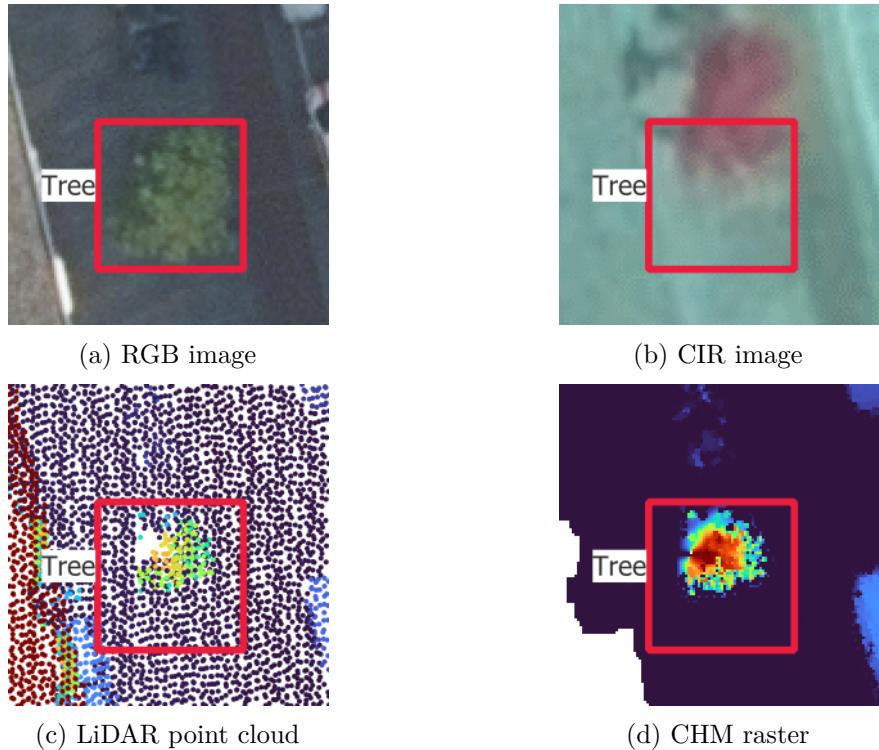


Figure 3.4: Example of data misalignment

Finally, the last challenge comes from the definition of what is considered as a tree and what is not. There are two main sub-problems. The first one comes from the threshold to set between bushes and trees. Large bushes can be much larger than small trees, and sometimes have a similar shape. Therefore, it is hard to keep coherent rules when annotating them. The second sub-problem comes from multi-stemmed and close trees. It can be very difficult to see, even with the point cloud, if a there is only one tree with two or more trunks dividing at the bottom, or multiple trees which are simply close to one another. This challenge is also mentioned in another paper [41]. In the end, it was just an unsolvable problem for which the most important was to remain consistent over the whole dataset.

## 3.4 Augmentation methods

Dataset augmentation methods are in the middle between dataset creation and deep learning model training, because they are a way to enhance the dataset but depend on the objective for which the model is trained. Their importance is inversely proportional with the size of the dataset, which made them very important for my small dataset of annotated trees.

As it was already explained in Section 1.2.4, I used Albumentations [23] to apply two types of augmentations: pixel-level and spatial-level.

Spatial-level augmentations had to be in the exact same way to the whole dataset, to maintain the spatial coherence between RGB images, CIR images and the CHM layers. I used three different spatial transformations, applied with random parameters. The first one chooses one of the eight possible images when flipping and rotating by angles that are multiples of 90°. The second one adds a perspective effect to the images. The third one adds a small distortion to the image.

On the contrary, pixel-level augmentations must be applied differently to RGB images and CHM layers because they represent different kinds of data, and the values of the pixels do not have the same meaning. In practice, a lot of transformations were conceived to reproduce camera effects on RGB images or to shift the color spectrum. Among others, I used random modifications of the brightness, the gamma value and added noise and a blurring effect randomly to RGB images. For both types of data, a channel dropout is also randomly applied, leaving a random number of channels and removing the others. A better way to augment the CHM data would have been to apply random displacements and deletions of points in the point cloud, before computing the CHM layers. However, these operations are too costly to be integrated in the training pipeline without consequently increasing the training time, so this idea was discarded.

On Figure 3.5, you can see an RGB image and 15 random augmentations of this image, generated with the transformations and the probabilities used during training. The most visible change happens when one or two color channels are dropped, which completely changes the color of the image. But other effects are also visible such as luminosity changes in images n°4 and 14, perspective changes in n°5, 8 and 13, blurring in n°1 and 14, and distortions in n°10 and 12. All these effects and some other less identifiable augmentations (like noise), are randomly combined to produce many different images, with bounding boxes being modified accordingly.

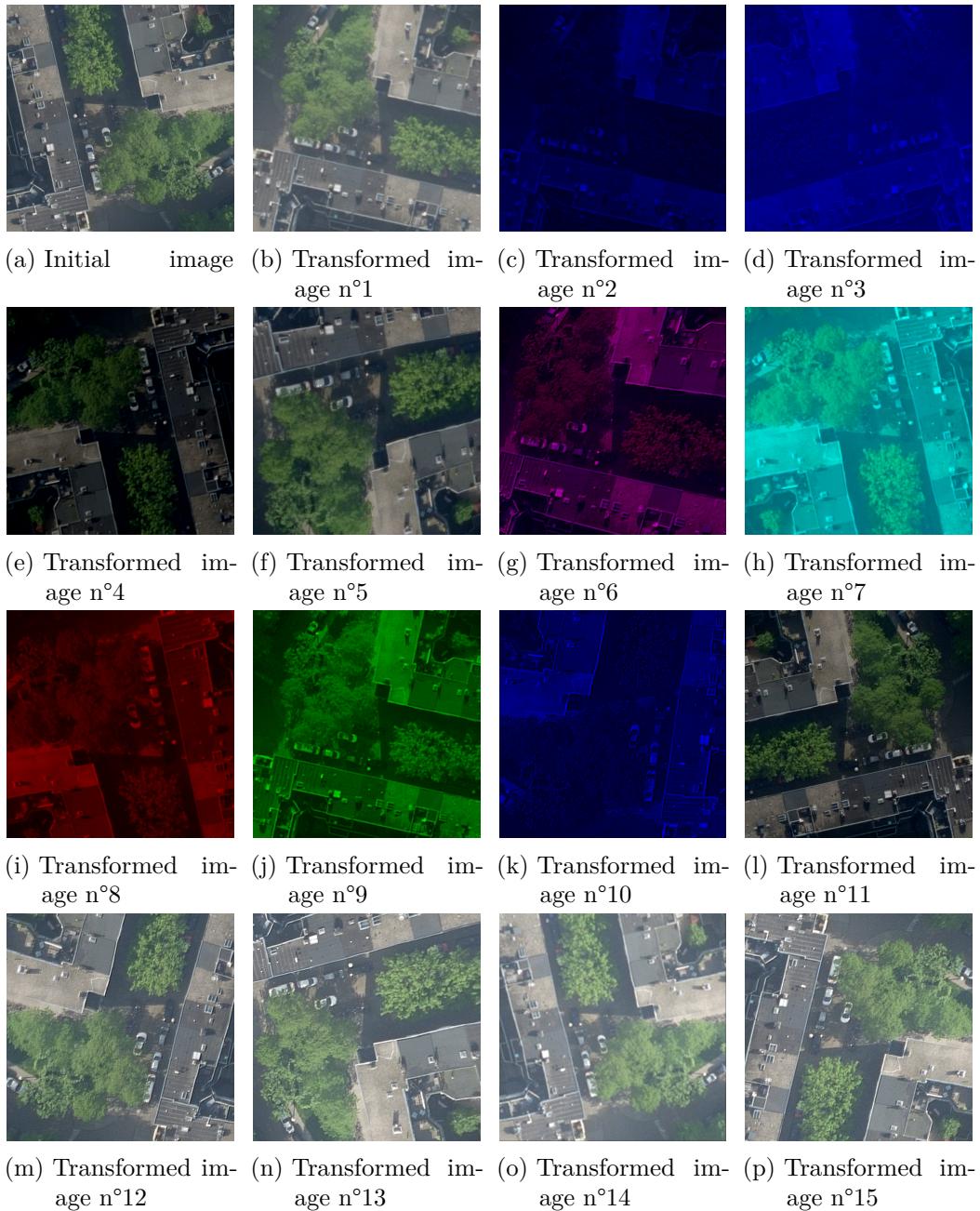


Figure 3.5: Examples of data augmentations on an RGB image with the probabilities used when training the model.

# 4 Model and training

The deep learning model that is used is based on AMF GD YOLOv8, the model proposed in this paper [7].

## 4.1 Model architecture

The general architecture of the model is conceptually simple. The model takes two inputs in the form of two rasters with the same height and width. The two inputs are processed using the backbone of the YOLOv8 model [37] to extract features at different scales. Then Attention Multi-level Fusion (AMF) layers are used to fuse the features of the two inputs at each scale level. Then, a Gather-and-Distribute (GD) mechanism is used to propagate information between the different scales. This mechanism fuses the features from all scales before redistributing them to the features, two times in a row. Finally, the features of the three smallest scales are fed into detection layers responsible for extracting bounding boxes and assigning confidence scores and class probabilities to them.

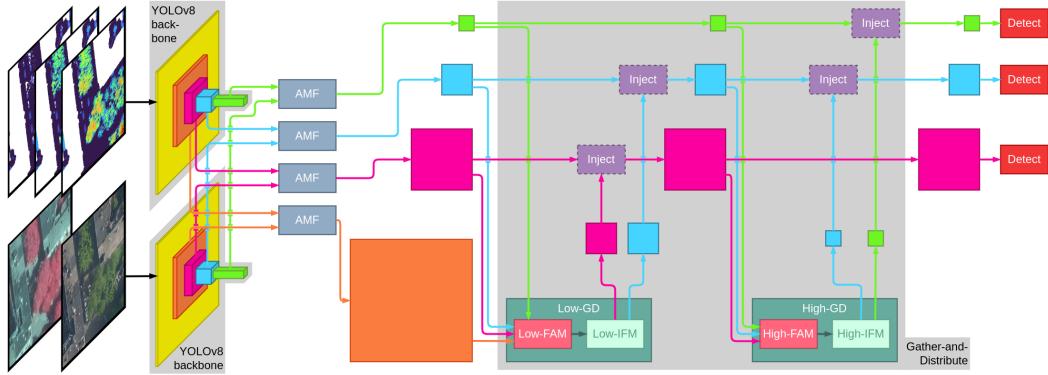


Figure 4.1: Model architecture

In practical terms, the input rasters have a shape of  $640 \times 640 \times c_{\text{RGB}}$  and  $640 \times 640 \times c_{\text{CHM}}$ , where  $c_{\text{RGB}}$  is equal to 6 when using RGB and CIR images, and 3 when using only one of them, and  $c_{\text{CHM}}$  is the number of CHM layers used for the model. Since the resolution that is used is 0.08 m, this means that each image spans over 51.2 m.

The only real modification that I made to the architecture compared to the initial paper is adding any number of channels in the CHM input, while there was only one

originally ( $c_{\text{CHM}} = 1$ ). Using CIR images in addition to RGB images is also new, but this is a less important modification.

## 4.2 Training pipeline

The training pipeline consists of three steps. First, the data is pre-processed to create the inputs to feed into the model. Then, the training loop runs until the end condition is reached. Finally, the final model is evaluated on all the datasets.

### 4.2.1 Data preprocessing

Data pre-processing is quite straightforward. The first step is to divide the dataset into a grid of  $640 \times 640$  tiles. Then, all these tiles are placed into one of the training, validation and test sets.

As for RGB and CIR images, preprocessing only contains two steps: tiling the large images into small  $640 \times 640$  images, and normalizing all images along each channel. When both data sources are used, RGB and CIR images are also merged into images with 6 channels, which will be the input of the model.

As for CHM layers, there are more steps. The first step is to compute a sort of flattened point cloud, by computing the DTM, which represents the height of the ground, and removing this height to the point cloud. Then, for each CHM layer, if the height interval is  $[z_{\text{bot}}, z_{\text{top}}]$ , all the points which have a height  $h$  such that  $z_{\text{bot}} \leq h \leq z_{\text{top}}$  are extracted, and the DSM is computed for this smaller point cloud. Since the ground height was already removed from the point cloud, this DSM is the CHM. Then, all the layers are merged into one raster with multiple channels and the whole raster is normalized with the average and the standard deviation over all channels. Finally, these rasters are tiled exactly like RGB and CIR images, which gives the inputs of the model.

All these operations are conceptually simple, but they can be computationally expensive. Therefore, I put effort into accelerating with different methods. First, I made sure to save the most important and generic elements to avoid useless computations every time the model is trained again, without saturating the memory. Then, I also implemented multi-threading for every possible step to improve the raw speed of preprocessing. Finally, performance is also the reason why normalization is performed during preprocessing instead of during the initialization of the data in the training loop.

### 4.2.2 Training loop

The training loop is very generic, so I will only mention the most interesting parts. First, an Adam optimizer and a basic learning rate scheduler with a multiplier at each epoch  $i$  which is  $1/\sqrt{i+2}$  are used.

Then, since the batch size cannot be very large because of the GPU memory space required by all the images, there is the possibility to perform gradient accumulation, which means that backward propagation won't be performed with each batch, but instead every two or more batches. The idea behind this is to add more stability to the training, since back-propagating on only a few images is prone to overfitting on a set of examples which are not representative of the whole dataset.

As for the criterion to stop the training session, the loss on the validation set is used. Once this loss hasn't improved for 50 iterations over the whole dataset, the training stops and the model that had the best validation loss is saved.

Besides these details, the training loop is very generic. We loop over the entire training set with batches to compute the loss and perform gradient back-propagation. Then the loss is computed over the validation set and stored as the metric that decides when to stop.

### 4.2.3 Output postprocessing

Regarding postprocessing of the output of the model, there are a few things to mention. First, the model outputs a lot of bounding boxes, which have to be cleaned using two criteria. The first criterion is the confidence score, which is used by setting a threshold below which bounding boxes are discarded. The second criterion is the intersection over union (IoU) with other bounding boxes. IoU is a metric used to quantify how similar two bounding boxes are. It is a value between 0 and 1, which formula is:

$$\text{IoU}(A, B) = \frac{\text{area}(A \cap B)}{\text{area}(A \cup B)}$$

This metric can be used to detect bounding boxes which are too similar to each other. This allows to simply keep the bounding box with the highest confidence score when two bounding boxes have an IoU larger than a certain threshold. This process based on IoU to eliminate duplicate bounding boxes is called non-maximum suppression.

For the evaluation, the process is a little different, because non-maximum suppression is performed but the confidence threshold used to filter predicted boxes is 0. The main metric that is computed is called sortedAP [42], which is an evolution of the mean (point) average precision (mAP). mAP is defined as follows:

$$\begin{aligned} \text{mAP} &= \frac{1}{N} \sum_{t \in T} \text{AP}_t \\ \text{AP}_t &= \frac{TP_t}{TP_t + FP_t + FN_t} \end{aligned}$$

where  $T = \{t_1, t_2, \dots, t_N\}$  is a list of IoU threshold values,  $TP_t$  are the true positives when the IoU threshold is  $t$ ,  $FP_t$  are false positives and  $FN_t$  are false negatives. The reason why  $TP$ ,  $FP$  and  $FN$  depend on  $t$  is that a bounding box is considered to be true if its IoU with one of the ground-truth bounding boxes is larger than  $t$ .

sortedAP is an improvement over this method because there is no need to select a list of IoU threshold values. Predicted bounding boxes are sorted according to their confidence score which allows to compute AP incrementally for any value of  $t$ . Then, the area of the curve of the AP with respect to the IoU threshold is used as a metric, between 0 and 1, 1 being the best possible value. You can see in Figure 4.2 an example of the output of the sortedAP method. The curves represent the AP at any IoU threshold value. Then, the integral of this curve gives the value of sortedAP, which is displayed in the legend here.

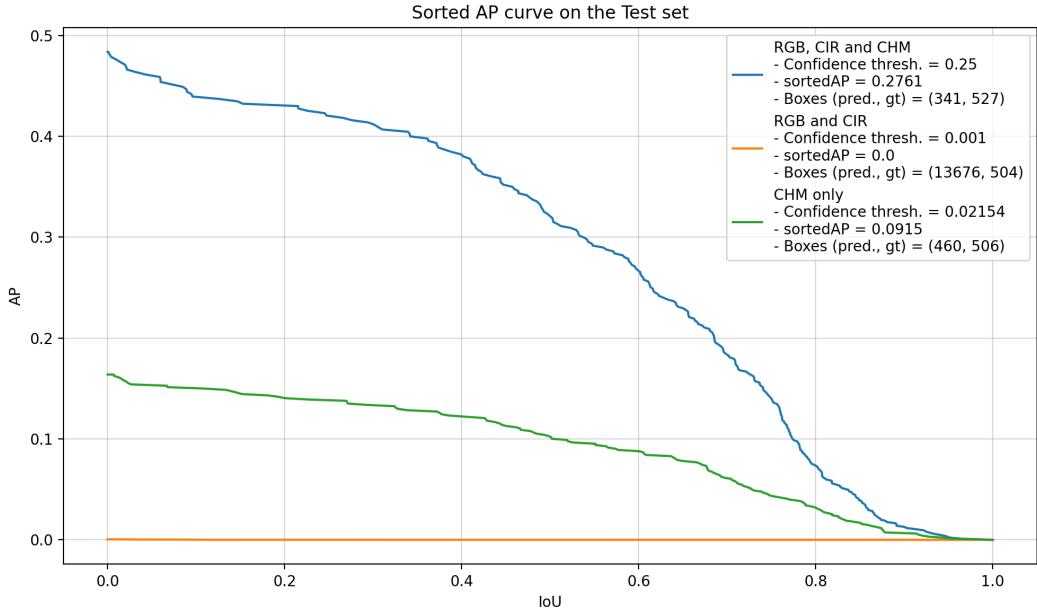


Figure 4.2: Example of three sortedAP curves with the results of a model with three different data combinations

Finally, it is necessary to choose a fixed confidence threshold to compute the curve, because this value decides which boxes will be kept and how the predicted and ground-truth boxes will be matched. Therefore, the value of the best confidence threshold has a very high impact over the value of sortedAP, and cannot easily be determined during the training, as the confidence of the model evolves quickly. Therefore, computing sortedAP for different confidence threshold allows to always estimate the best performance it can have. Figure 4.3 shows how the value of sortedAP evolves and how the confidence threshold values in the previous figure (Figure 4.2) were chosen.

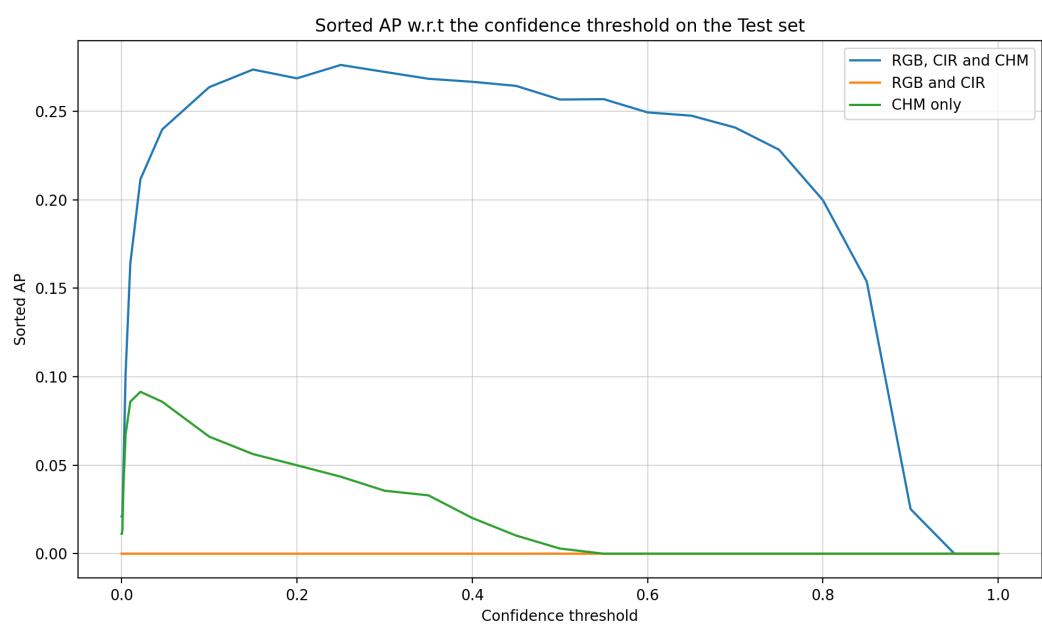


Figure 4.3: Example of the values of sortedAP depending on the confidence threshold

# 5 Results

In this section are the results of the experiments performed with the model and the dataset presented before.

## 5.1 Training parameters

The first experiment was a simple test over the different parameters regarding the training loop. There were two goals to this experiment. The first one was to find the best training parameters for the next experiments. The second one was to see if randomly dropping one of the inputs of the model (either RGB/CIR or CHM) could help the model by pushing it to learn to make the best out of the two types of data.

The different parameters that are tested here are:

- “Learn. rate”: the initial learning rate.
- “Prob. drop”: the probability to drop either RGB/CIR or CHM. The probability is the same for the two types, which means that if the displayed value is 0.1, then all data will be used 80% of the time, while only RGB/CIR and only CHM both happen 10% of the time.
- “Accum. count”: the accumulation count, which means the amount of training data to process and compute the loss on before performing gradient back-propagation.

As you can see on Figure 5.1, sortedAP reaches at best values just above 0.3. Since the dataset is small, the training process overfits quickly, and the model doesn’t have enough training steps to have confidence scores which reach very high values. As a consequence, it is difficult to know beforehand which confidence threshold to choose. Therefore, the sortedAP metric is computed over several different confidence thresholds, and the one that gives the best value of sortedAP is kept. This is why the name of the column is “Best sortedAP”, as the highest sortedAP value over several confidence thresholds is displayed.

This experiment shows that a learning rate of 0.01 seems to make the training too much unstable, while 0.001 doesn’t give very high score. Moreover, the training process seems very unstable in general, which mostly comes from the dataset being too small. However, a learning rate between 0.0025 and 0.006 seems to give the most stable results, when the drop probability is 0. This seems to show that the idea of randomly dropping one of the two inputs doesn’t really help the model to learn.

The next graph (Figure 5.2) displays more results for the same experiments. Here, the results are colored according to the data that used to evaluate the model. In

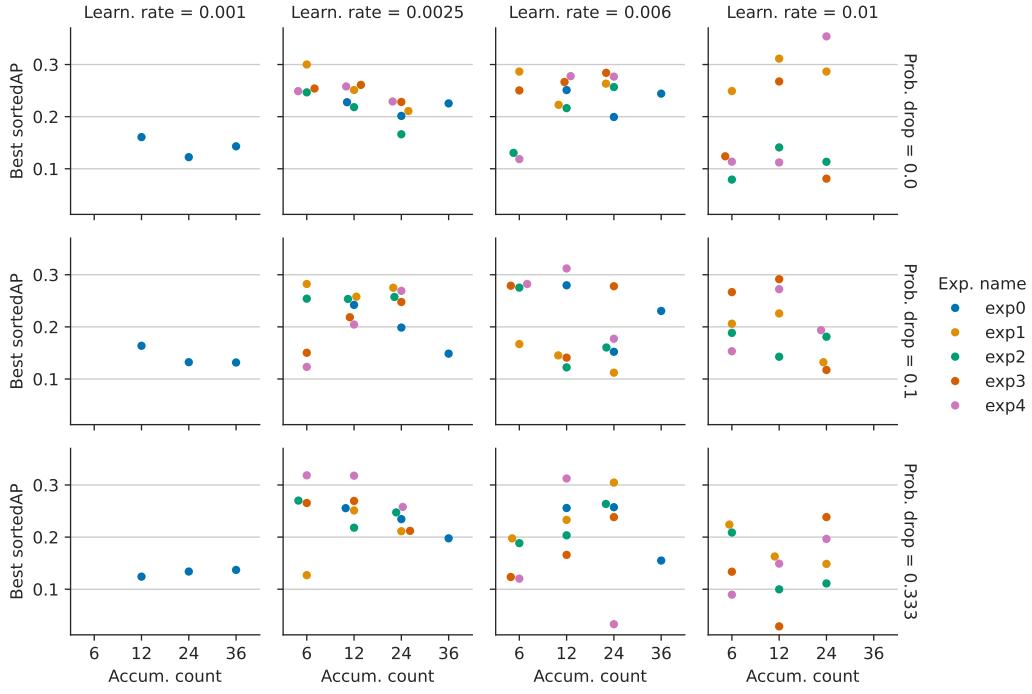


Figure 5.1: Results with different training parameters for all experiments

blue are the values of sortedAP when the model is evaluated with the CHM layers data and dummy zero arrays as RGB/CIR data. These dummy arrays are also those that are used as input when one of the channel is dropped during training, when we have a drop probability larger than 0. Not many conclusions can be drawn from this plot. One interesting observation is that randomly dropping one of the two inputs with the same probability seems to have a much larger influence over the results using RGB/CIR than CHM. While CHM gives better results than RGB/CIR when always training using everything, RGB/CIR seems to perform better alone when also trained alone, even outperforming the combination of both inputs in certain cases. This behavior is not desired, as having mode data should not make the results worse. The explanation is probably related to the speed of the training, which might be much quicker for RGB only than for CHM only or both together. Then, if the training stops early, which is the case when the learning rate is high, RGB only can output the best results.

From the results of this experiment, I decided to pick the following parameters for the next experiments:

- Initial learning rate: 0.004
- Drop probability: 0
- Accumulation count: 10

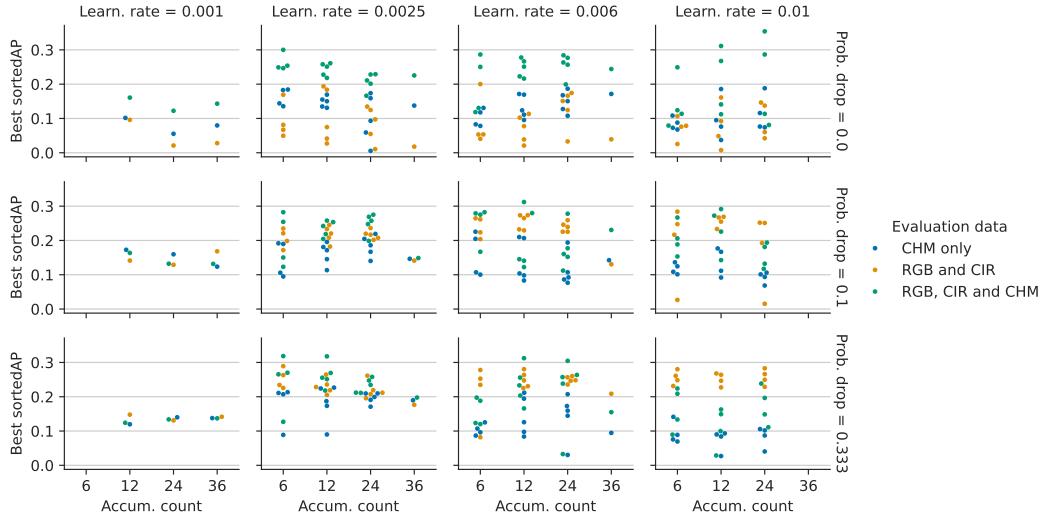


Figure 5.2: Results with different training parameters for all evaluation data setups

## 5.2 CHM layers

The goal of the second experiment was to test the model with different layers of CHM, to see whether more layers can improve the results. I tried two different ways to cut the point cloud into intervals defining these layers. In both cases, we first define the height thresholds  $[t_1, t_2, \dots, t_n]$  that will be used. Then, there are two possibilities for the height intervals to use to compute the CHM layers:

$$\begin{aligned} \text{Disjoint} = \text{True: } & [[-\infty, t_1], [t_1, t_2], [t_2, t_3], \dots, [t_{n-1}, t_n], [t_n, \infty]] \\ \text{Disjoint} = \text{False: } & [[-\infty, t_1], [-\infty, t_2], \dots, [-\infty, t_n], [-\infty, \infty]] \end{aligned}$$

The results of this experiment can be found in Figure 5.3. To experiment on other parameters, half of the models were trained to be agnostic while the other half was not, to see if the way trees are separated in the four classes has an impact on performance, either facilitating the learning or hindering the generalization. On this plot, the borders correspond to the list of height thresholds  $[t_1, t_2, \dots, t_n]$ .

These results are difficult to interpret. The first clear effect that we can notice for the agnostic models is that in the disjoint method (first row starting from the top), when separating the LiDAR point cloud in too many layers, the model doesn't manage to make extract information from the CHM. When using only one CHM layer on the whole point cloud (which corresponds to ( $\text{Borders} = []$ )), we can see that the model performs poorly when using only one data type, but as well as the other models when using both. The explanation for this may be related to the data augmentation pipeline, because there is random channel dropout during the training. When there are multiple channels in the input, almost all of them can be randomly dropped out, with the limitation that at least one channel will always remain. This is the reason why there are red, green and blue (two channels dropped) but also purple and cyan (one channel dropped) images in Figure 3.5. But when there is only one channel, this

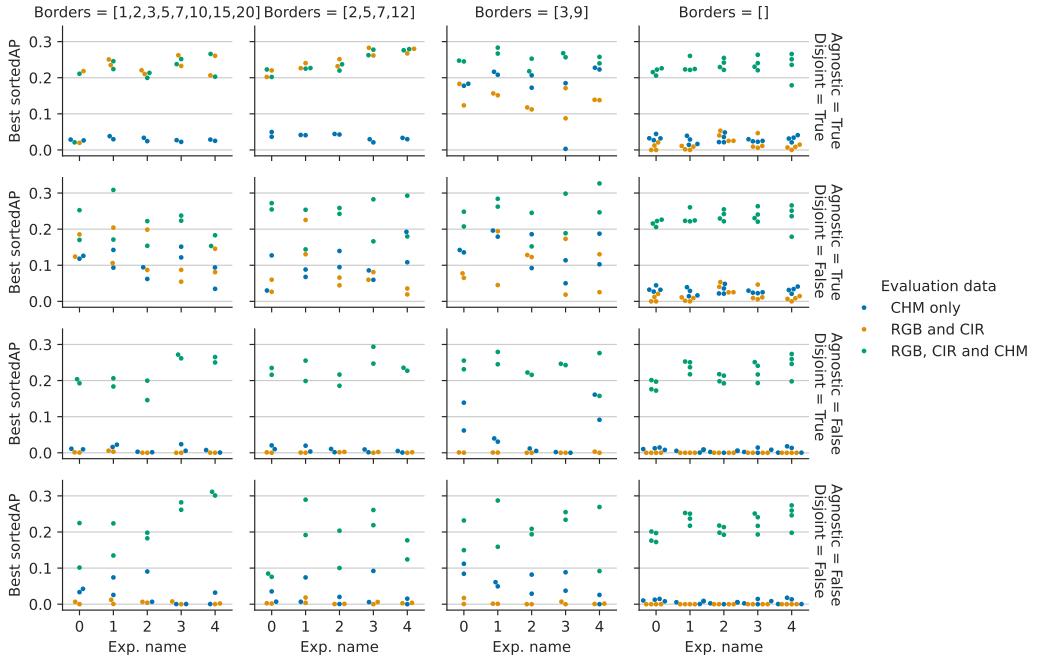


Figure 5.3: Results with different CHM layers

channel is never dropped, which doesn't force the model to learn how to use only part of the data.

The other correlation that is visible relates to how models perform using only RGB/CIR or CHM when being agnostic or not. It is logical for this parameter to have an impact on the results, since teaching the model to either make a difference between some of the trees or not will have an impact on what it learns and what it has to focus on to identify the right class when not being agnostic. From what we see here in Figure 5.3, it looks like having to make a difference between the different classes is significantly hinders its performance whatever combination of input data is used, even though it is more prevalent when using only one input.

Besides these observations, it is hard to draw anymore conclusion from the rest of the experiments. There is too much variation in the results, which shows how unstable the training process is. From these results and the previous paragraph, one interpretation could be that the augmentation pipeline is related to this instability, as the results with the basic CHM layer have a much smaller variance.

### 5.3 Covered trees

Then, if we try to look at the performance of the models on the covered trees, which are called "Tree\_low\_hard" in Figure 5.4, it is also difficult to draw any conclusion. The models have mainly learnt to find trees with the generic "Tree" label, and they are seemingly equally bad at finding the other classes of trees. In Figure 5.4, we can

see the performance of two models trained with the same repartition of the data into training, validation and test set. Both models were not agnostic, which means that they learnt to detect each class of trees and label them properly. The one of the left uses the largest number of CHM layers (with Borders = [1, 2, 3, 5, 7, 10, 15, 20] and Disjoint = False), whereas the one on the right only uses the default CHM layer (Borders = []).

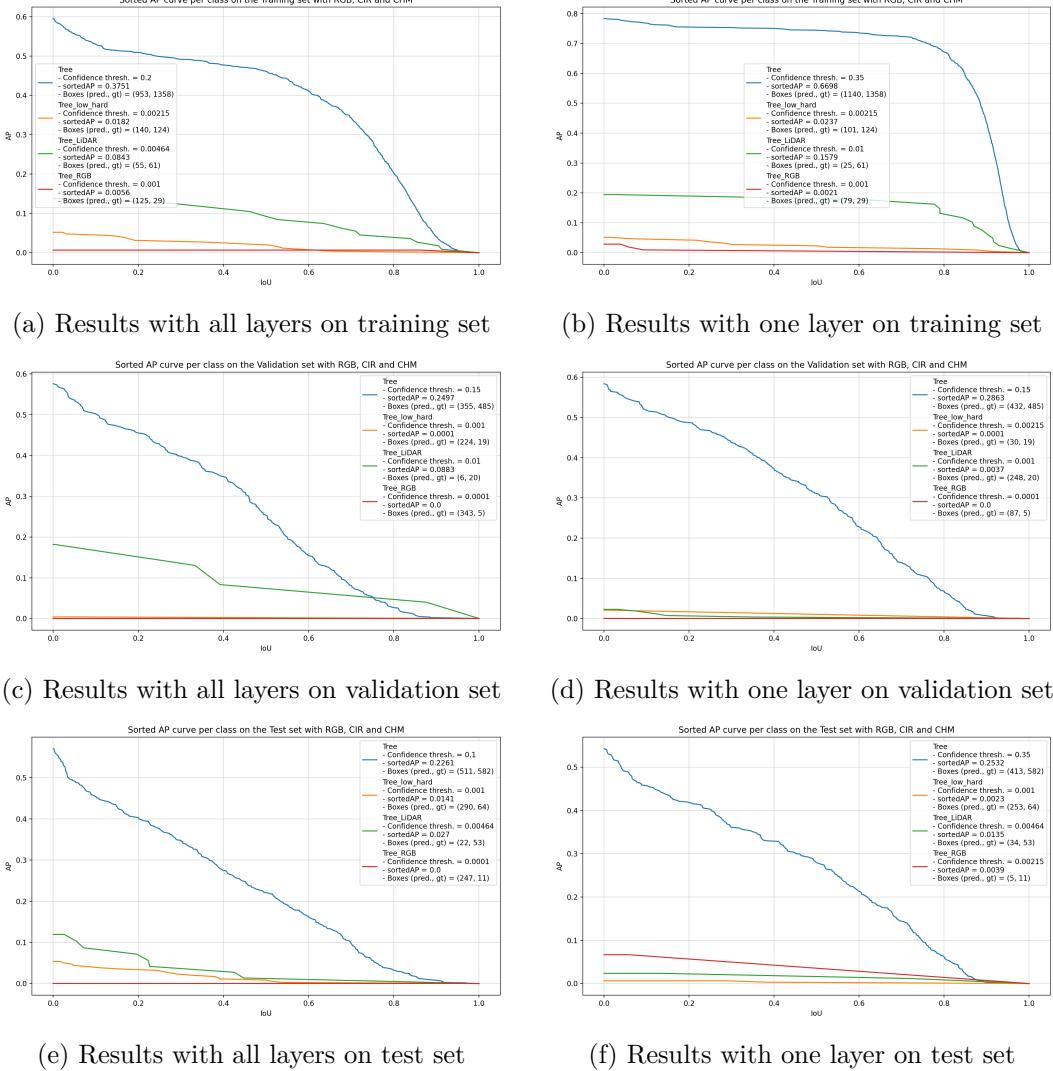


Figure 5.4: sortedAP curve with different CHM layers on training/validation/test set

These two examples are really representative of the results of the other models that were trained. There are only 207 covered trees in the dataset, which is too small to get the models to learn to identify them and get solid results when comparing different configurations of CHM layers. Most of the differences that we see come from the normal random variation during the training.

## 5.4 Visual results

In Figure 5.5 are the outputs of trained models on one instance from each of the training, validation and test sets. The models used are those which results are shown in Figure 5.4.

The confidence thresholds used to remove low confidence predictions are the ones that gave the best sortedAP values in Figure 5.4. One aspect that is partly visible here is that all the categories except the basic “Tree” have confidence scores which are so low that they are rejected after the selection with the confidence threshold. This is the reason why there is no predicted bounding box from another class than “Tree”, even though the models do predict boxes with other labels. This is the case on the whole dataset, including the training set, and shows how the models learns more slowly to find the other categories.

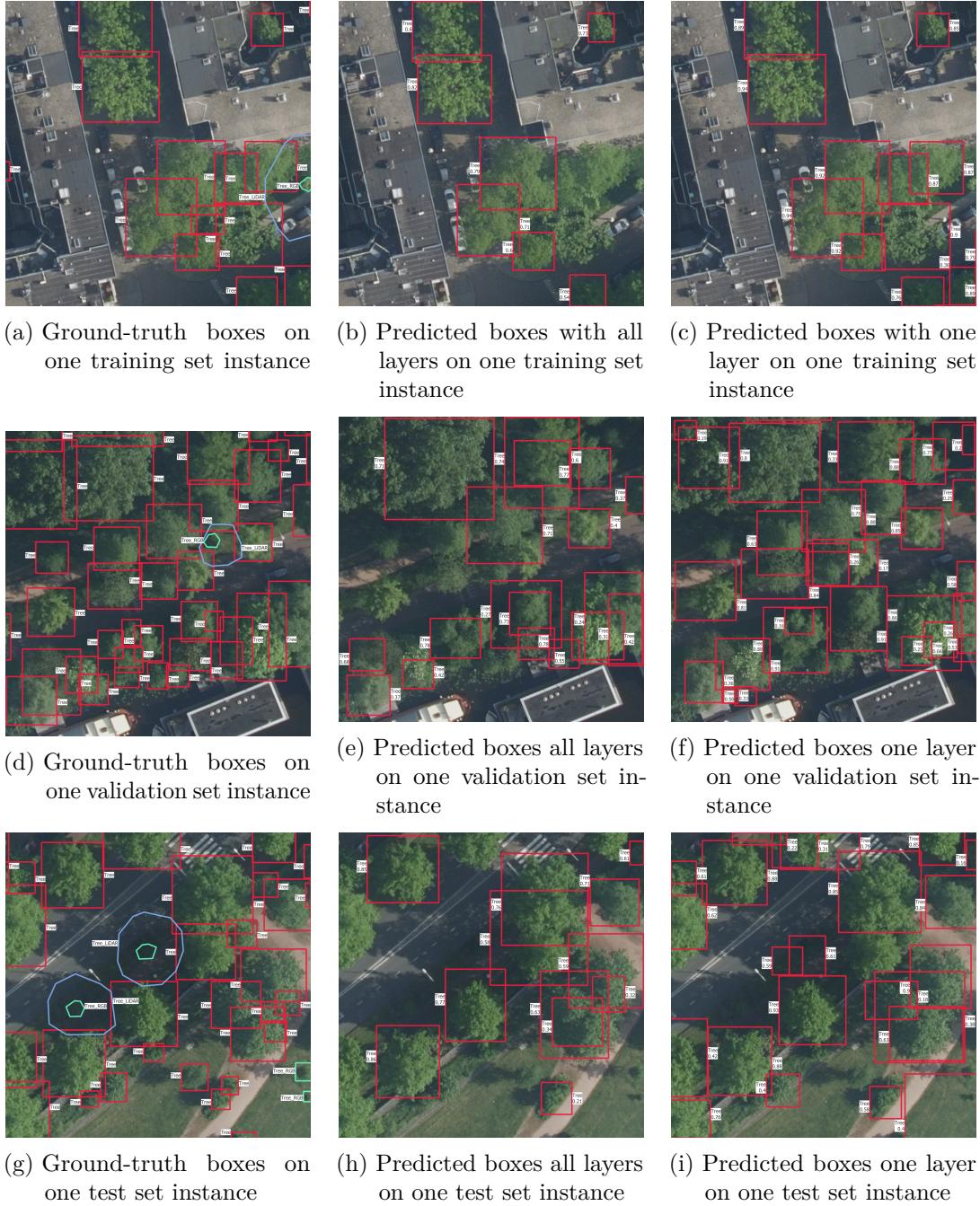


Figure 5.5: Ground-truth boxes and predictions from two models

# 6 Discussion and improvements

## 6.1 Dataset

From the results of this work, the only certitude is that more training data would be necessary to confirm any conclusion. Even with augmentation techniques, the dataset is too small to completely train a model and really experiment with the small changes applied to it. Since the training loop quickly reaches overfitting, we don't really get to see how the model could perform in the most interesting cases, which are the small, covered or hardly visible trees.

Therefore, the biggest and conceptually simplest improvement that could be done to this work would be to improve and extend the dataset. Improving with more diversity, covering a larger part of the Netherlands (or even beyond, but consistency over images and point clouds can only be ensured in this country), and extending with more images and more trees. These improvements could also include adding species information to the dataset, to train models that are capable of differentiating species.

At the time of writing this report, the newest version of the point cloud has also been released for one third of the Netherlands, including the area of the current dataset. This new data could be better for this project, because it was acquired in 2023, the same year as the images that are used in the dataset.

Another approach to generate a larger dataset could be to create a large artificially annotated dataset, like it was done in another paper [30]. Their approach was to create a very large dataset with medium-quality data which can be used to pre-train the model. Then, they use hand-annotated data to finish the training. They created this large dataset using classical non-machine learning techniques, using only the point cloud. This approach could solve some of the issues, but it is not sure whether this would not introduce problems related to covered trees, which might be missing a lot more in the automatic dataset. Furthermore, using only the point cloud would leave behind all the trees only visible on the images.

Finally, it would also be interesting to see if the NEON tree dataset [6] can be used to train a model and make some experiments. This would not also allow to experiment on covered trees, but it could still show whether multiple layers of CHM can help delineating individual trees.

## 6.2 Instability

As explained in the previous section Section 6.1, the size of the dataset is probably the main reason for the instability of the training pipeline. But other reasons might also be responsible for this instability. As explained in Section 5.2, the random channel dropouts might also destabilize the training by creating inputs with very different repartition of the information, since some channels are randomly removed. The value to use as a replacement of these channels is also not easy to choose, as on the normalized CHM rasters, a value of 0 might be equivalent to a height above ground of 10 m for example, and a flat surface at 10 m is not exactly no data and might also be misleading if the CHM usually goes from 0 to 4 m.

More generally, it is possible that other factors are responsible for the instability of the training pipeline, and it would be useful to find and correct them.

## 6.3 Model performance

Even though the results displayed in Chapter 5 are far from perfect, they are still promising for several reasons.

First, the overall results of the model on the general trees are quite impressive, given that only about 1600 trees are used to train each model, with only about 150 images covering about 0.4 km<sup>2</sup>. Even though the whole dataset is quite homogenous, this still shows that the model is capable of generalizing relatively well.

Then, a few correlations have been noticed regarding some parameters or combinations of parameters, even though more robust experiments would be necessary to confirm them. This is for example the case between agnostic and non-agnostic models, where the former seem to perform better in general.

However, the models don't perform good enough in the difficult areas to be able to draw conclusion on the modifications of the architecture. The overall performance of the model, which rarely gets a sortedAP score above 0.3, is also relatively low. But this low score mostly comes from the amount of small trees which have a relatively high importance in the computation of sortedAP because each tree has the same weight, no matter their size. In contrast, the models perform quite well on the large trees, finding most of them in situations that are not too much complex.

Finally, it could be interesting to try another method to extract multiple CHM layers, which consists in removing only the points of the point cloud which corresponds to the previous CHM and all those which are close below them, and repeating the process to scan downwards. This has the advantage of being more adaptive than using fixed height thresholds for the whole dataset.

# Conclusion

All in all, the results of this internship are interesting and promising, even if not decisive.

Regarding datasets, the new dataset that was created is promising for several reasons. First, its spatial extent can easily be extended since the raw data that is used is publicly available and covers the whole of the Netherlands. Then, its quality will also surely keep increasing in the future with the different iterations of the images and the point clouds, at the cost of small annotations modifications to update the dataset with new and cut-off trees, as well as tree growth. The diversity of trees and environments from the Netherlands is obviously not even close from what can be found globally, which wouldn't make it a great dataset to train a global model, but it has the potential to be a perfect playground for testing new methods. Finally, the main drawback of this dataset are the spatial and temporal shifts between each type of raw data. But these shifts have at least proven to be manageable by the deep learning models that were trained here. Having these shifts is also interesting because counting on having perfectly aligned RGB images and point clouds is even less likely than having both of them available in the first place.

Regarding the model, it is unclear whether having multiple layers of CHM really improves the results. This is because these layers would have the biggest impact in the detection of covered trees, which are a specific case that is harder than the other trees. And since the training dataset was too small, the model overfitted quickly and could really reach the state when it start learning to find these harder trees. Therefore, more experiments on a bigger dataset, maybe using better augmentation techniques, would be required to get an answer. Besides that, the architecture in itself proved to provide great performance and is quickly able to learn to detect the medium and large trees. Some interesting improvements could easily be added to the model, such as the prediction of mask instead of bounding boxes, which only requires to change the detection heads, or the prediction of species. However, these changes would require the dataset to be substantially with species and precise delineations for all trees.

# Bibliography

- [1] Tito Arevalo-Ramirez et al. “Challenges for computer vision as a tool for screening urban trees through street-view images”. In: *Urban Forestry & Urban Greening* 95 (2024), p. 128316. ISSN: 1618-8667. DOI: [10.1016/j.ufug.2024.128316](https://doi.org/10.1016/j.ufug.2024.128316).
- [2] Jonathan Ventura et al. “Individual tree detection in large-scale urban environments using high-resolution multispectral imagery”. In: *International Journal of Applied Earth Observation and Geoinformation* 130 (2024), p. 103848. ISSN: 1569-8432. DOI: <https://doi.org/10.1016/j.jag.2024.103848>. URL: <https://www.sciencedirect.com/science/article/pii/S1569843224002024>.
- [3] Marilia Ferreira Gomes and Philippe Maillard. “Detection of Tree Crowns in Very High Spatial Resolution Images”. In: *Environmental Applications of Remote Sensing*. Ed. by Maged Marghany. Rijeka: IntechOpen, 2016. Chap. 2. DOI: [10.5772/62122](https://doi.org/10.5772/62122).
- [4] Anastasiia Safonova et al. “Olive Tree Biovolume from UAV Multi-Resolution Image Segmentation with Mask R-CNN”. In: *Sensors* 21.5 (2021), p. 1617. ISSN: 1424-8220. DOI: [10.3390/s21051617](https://doi.org/10.3390/s21051617).
- [5] Zhenbang Hao et al. “Automated tree-crown and height detection in a young forest plantation using mask region-based convolutional neural network (Mask R-CNN)”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 178 (2021), pp. 112–123. ISSN: 0924-2716. DOI: <https://doi.org/10.1016/j.isprsjprs.2021.06.003>.
- [6] Ben Weinstein, Sergio Marconi, and Ethan White. *Data for the NeonTreeEvaluation Benchmark (0.2.2)*. 2022. DOI: [10.5281/zenodo.5914554](https://doi.org/10.5281/zenodo.5914554).
- [7] Hao Zhong et al. “Individual Tree Species Identification for Complex Coniferous and Broad-Leaved Mixed Forests Based on Deep Learning Combined with UAV LiDAR Data and RGB Images”. In: *Forests* 15.2 (2024). ISSN: 1999-4907. DOI: [10.3390/f15020293](https://doi.org/10.3390/f15020293).
- [8] Gyri Reiersen et al. *ReforeSTree: A Dataset for Estimating Tropical Forest Carbon Stock with Deep Learning and Aerial Imagery*. 2022. arXiv: [2201.11192 \[cs.CV\]](https://arxiv.org/abs/2201.11192).
- [9] Arthur Ouaknine et al. *OpenForest: A data catalogue for machine learning in forest monitoring*. 2023. arXiv: [2311.00277 \[cs.CV\]](https://arxiv.org/abs/2311.00277).
- [10] Nikolaos Ioannis Bountos, Arthur Ouaknine, and David Rolnick. “FoMo-Bench: a multi-modal, multi-scale and multi-task Forest Monitoring Benchmark for remote sensing foundation models”. In: *arXiv preprint arXiv:2312.10114* (2023). URL: <https://arxiv.org/abs/2312.10114>.

- [11] Tommaso Jucker et al. “Tallo: A global tree allometry and crown architecture database”. In: *Global Change Biology* 28.17 (2022), pp. 5254–5268. DOI: [10.1111/gcb.16302](https://doi.org/10.1111/gcb.16302).
- [12] Ben Weinstein. *MillionTrees*. 2023. URL: <https://milliontrees.idtrees.org/> (visited on 07/08/2024).
- [13] Ekaterina Kalinicheva et al. *Multi-Layer Modeling of Dense Vegetation from Aerial LiDAR Scans*. 2022. arXiv: [2204.11620 \[cs.CV\]](https://arxiv.org/abs/2204.11620).
- [14] Stefano Puliti et al. *FOR-instance: a UAV laser scanning benchmark dataset for semantic and instance segmentation of individual trees*. 2023. arXiv: [2309.01279 \[cs.CV\]](https://arxiv.org/abs/2309.01279).
- [15] J. Hu et al. “MDAS: a new multimodal benchmark dataset for remote sensing”. In: *Earth System Science Data* 15.1 (2023), pp. 113–131. DOI: [10.5194/essd-15-113-2023](https://doi.org/10.5194/essd-15-113-2023).
- [16] Institut national de l’information géographique et forestière (IGN). *BD ORTHO*. Accessed: 2024-07-09. 2021. URL: <https://geoservices.ign.fr/bdortho>.
- [17] Beeldmateriaal Nederland. *Luchtfoto’s (Aerial Photographs)*. Accessed: 2024-07-09. 2024. URL: <https://www.beeldmateriaal.nl/luchtfotos>.
- [18] Actueel Hoogtebestand Nederland. *AHN4 - Actual Height Model of the Netherlands*. Accessed: 2024-07-09. 2020. URL: <https://www.ahn.nl/>.
- [19] Institut national de l’information géographique et forestière (IGN). *LiDAR HD*. Accessed: 2024-07-09. 2020. URL: <https://geoservices.ign.fr/lidarhd>.
- [20] Gemeente Amsterdam. *Bomenbestand Amsterdam (Amsterdam Tree Dataset)*. Accessed: 2024-07-09. 2024. URL: [https://maps.amsterdam.nl/open\\_geodata/?k=505](https://maps.amsterdam.nl/open_geodata/?k=505).
- [21] Bordeaux Métropole. *Patrimoine arboré de Bordeaux Métropole (Tree Heritage of Bordeaux Metropole)*. Accessed: 2024-07-09. 2024. URL: [https://opendata.bordeaux-metropole.fr/explore/dataset/ec\\_arbre\\_p/information/?disjunctive.insee](https://opendata.bordeaux-metropole.fr/explore/dataset/ec_arbre_p/information/?disjunctive.insee).
- [22] Coöperatief Boomregister U.A. *Boom Register (Tree Register)*. Accessed: 2024-07-11. 2014. URL: <https://boomregister.nl/>.
- [23] Alexander Buslaev et al. “Albumentations: Fast and Flexible Image Augmentations”. In: *Information* 11.2 (2020). ISSN: 2078-2489. DOI: [10.3390/info11020125](https://doi.org/10.3390/info11020125).
- [24] Chenxin Sun et al. “Individual Tree Crown Segmentation and Crown Width Extraction From a Heightmap Derived From Aerial Laser Scanning Data Using a Deep Learning Framework”. In: *Frontiers in Plant Science* 13 (2022). ISSN: 1664-462X. DOI: [10.3389/fpls.2022.914974](https://doi.org/10.3389/fpls.2022.914974).
- [25] L. Vincent and P. Soille. “Watersheds in digital spaces: an efficient algorithm based on immersion simulations”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.6 (1991), pp. 583–598. DOI: [10.1109/34.87344](https://doi.org/10.1109/34.87344).
- [26] Mike Wulder, K.Olaf Niemann, and David G. Goodenough. “Local Maximum Filtering for the Extraction of Tree Locations and Basal Area from High Spatial Resolution Imagery”. In: *Remote Sensing of Environment* 73.1 (2000), pp. 103–114. ISSN: 0034-4257. DOI: [10.1016/S0034-4257\(00\)00101-2](https://doi.org/10.1016/S0034-4257(00)00101-2).

- [27] François A Gougeon et al. “Automatic individual tree crown delineation using a valley-following algorithm and rule-based system”. In: *Proc. International Forum on Automated Interpretation of High Spatial Resolution Digital Imagery for Forestry, Victoria, British Columbia, Canada*. Citeseer. 1998, pp. 11–23. URL: <https://d1ied5g1xfgpx8.cloudfront.net/pdfs/4583.pdf>.
- [28] Richard James Pollock. “The automatic recognition of individual trees in aerial images of forests based on a synthetic tree crown image model”. AAIINN14815. PhD thesis. 1996. ISBN: 0612148157. URL: <https://dx.doi.org/10.14288/1.0051597>.
- [29] Maximilian Freudenberg, Paul Magdon, and Nils Nölke. “Individual tree crown delineation in high-resolution remote sensing images based on U-Net”. In: *Neural Computing and Applications* 34.24 (2022), pp. 22197–22207. ISSN: 1433-3058. DOI: [10.1007/s00521-022-07640-4](https://doi.org/10.1007/s00521-022-07640-4).
- [30] Ben G. Weinstein et al. “DeepForest: A Python package for RGB deep learning tree crown delineation”. In: *Methods in Ecology and Evolution* 11.12 (2020), pp. 1743–1751. DOI: [10.1111/2041-210X.13472](https://doi.org/10.1111/2041-210X.13472).
- [31] Doo-Ahn Kwak et al. “Detection of individual trees and estimation of tree height using LiDAR data”. In: *Journal of Forest Research* 12.6 (2007), pp. 425–434. ISSN: 1610-7403. DOI: [10.1007/s10310-007-0041-9](https://doi.org/10.1007/s10310-007-0041-9).
- [32] Lothar Eysn et al. “A Benchmark of Lidar-Based Single Tree Detection Methods Using Heterogeneous Forest Data from the Alpine Space”. In: *Forests* 6.5 (2015), pp. 1721–1747. ISSN: 1999-4907. DOI: [10.3390/f6051721](https://doi.org/10.3390/f6051721).
- [33] Yunsheng Wang et al. “International Benchmarking of the Individual Tree Detection Methods for Modeling 3-D Canopy Structure for Silviculture and Forest Ecology Using Airborne Laser Scanning”. In: *IEEE Transactions on Geoscience and Remote Sensing* 54.9 (2016), pp. 5011–5027. DOI: [10.1109/TGRS.2016.2543225](https://doi.org/10.1109/TGRS.2016.2543225).
- [34] Ahmed Diab, Rasha Kashef, and Ahmed Shaker. “Deep Learning for LiDAR Point Cloud Classification in Remote Sensing”. In: *Sensors (Basel)* 22.20 (Oct. 2022), p. 7868. DOI: [10.3390/s22207868](https://doi.org/10.3390/s22207868).
- [35] Haiming Qin et al. “Individual tree segmentation and tree species classification in subtropical broadleaf forests using UAV-based LiDAR, hyperspectral, and ultrahigh-resolution RGB data”. In: *Remote Sensing of Environment* 280 (2022), p. 113143. ISSN: 0034-4257. DOI: [10.1016/j.rse.2022.113143](https://doi.org/10.1016/j.rse.2022.113143).
- [36] Yingbo Li et al. “ACE R-CNN: An Attention Complementary and Edge Detection-Based Instance Segmentation Algorithm for Individual Tree Species Identification Using UAV RGB Images and LiDAR Data”. In: *Remote Sensing* 14.13 (2022). ISSN: 2072-4292. DOI: [10.3390/rs14133035](https://doi.org/10.3390/rs14133035).
- [37] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91).
- [38] QGIS Development Team. *QGIS Geographic Information System*. QGIS Association. URL: <https://www.qgis.org>.

- [39] Actueel Hoogtebestand Nederland. *AHN-puntenwolkenviewer* (*AHN point cloud viewer*). Accessed: 2024-07-31. URL: <https://www.ahn.nl/ahn-puntenwolkenviewer>.
- [40] Google. *Google Street View*. Accessed: 2024-07-31. 2024. URL: <https://www.google.com/streetview/>.
- [41] Ben G. Weinstein et al. “Individual Tree-Crown Detection in RGB Imagery Using Semi-Supervised Deep Learning Neural Networks”. In: *Remote Sensing* 11.11 (2019). ISSN: 2072-4292. DOI: [10.3390/rs11111309](https://doi.org/10.3390/rs11111309).
- [42] Long Chen et al. “SortedAP: Rethinking Evaluation Metrics for Instance Segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. Oct. 2023, pp. 3923–3929. URL: [https://openaccess.thecvf.com/content/ICCV2023W/BIC/html/Chen\\_SortedAP\\_Rethinking\\_Evaluation\\_Metrics\\_for\\_Instance\\_Segmentation\\_ICCVW\\_2023\\_paper.html](https://openaccess.thecvf.com/content/ICCV2023W/BIC/html/Chen_SortedAP_Rethinking_Evaluation_Metrics_for_Instance_Segmentation_ICCVW_2023_paper.html).