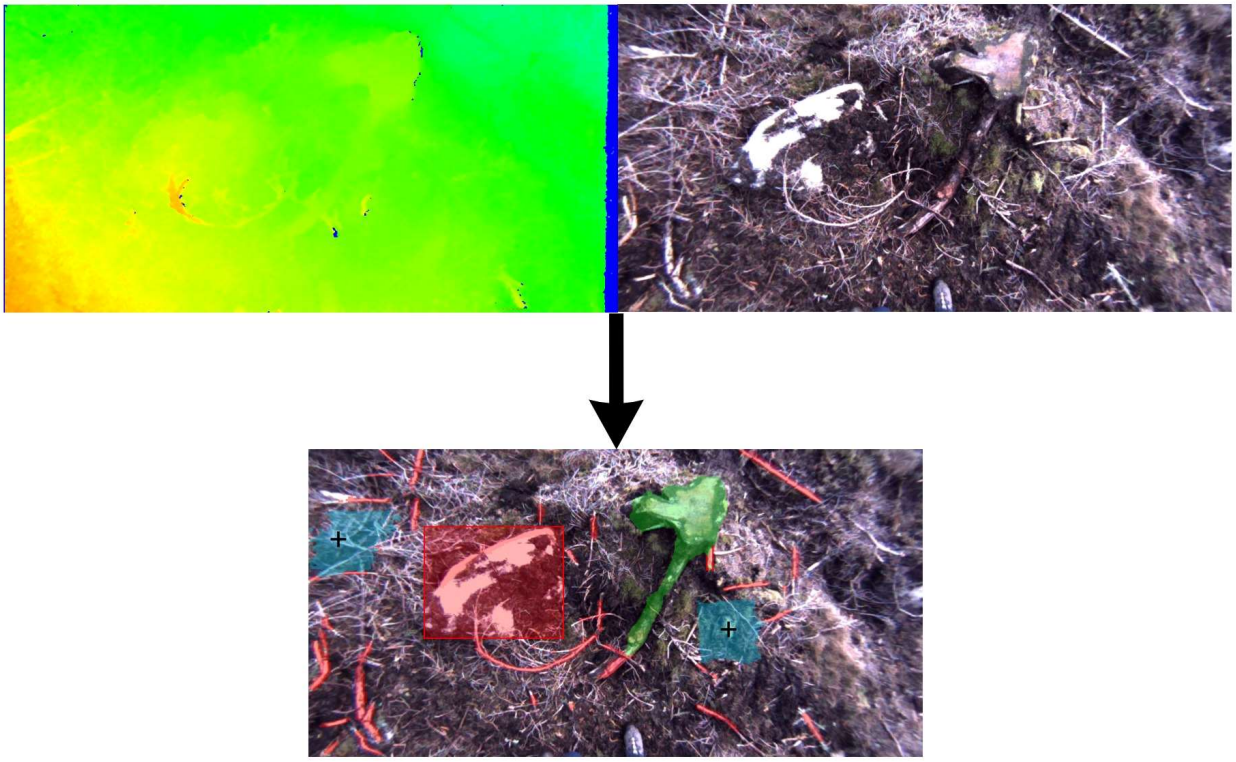
CHALMERS
UNIVERSITY OF TECHNOLOGY



Building a Computer Vision System for Autonomous Tree Planting

Using YOLO and U-Net to Find Planting Spots in Clear-Felled Areas

Master's thesis in Complex Adaptive Systems

Olle Christenson
Jens Lundgren

DEPARTMENT OF Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2022
www.chalmers.se

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

Building a Computer Vision System for Autonomous Tree Planting

Using YOLO and U-Net to Find Planting Spots in Clear-Felled Areas

OLLE CHRISTENSON
JENS LUNDGREN

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2022

Building a Computer Vision System for Autonomous Tree Planting
Using YOLO and U-Net to Find Planting Spots in Clear-Felled Areas
OLLE CHRISTENSON
JENS LUNDGREN

Master's thesis 2022:32
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
The figure shows an example of an input, consisting of a disparity map and an RGB image, together with the corresponding output from the final model. The detected obstacles are colored red, the detected stump is colored green, and the proposed planting spots are colored blue.

Chalmers Reproservice
Göteborg, Sweden 2022

Building a Computer Vision System for Autonomous Tree Planting
Using YOLO and U-Net to Find Planting Spots in Clear-Felled Areas
Master's thesis in Complex Adaptive Systems
OLLE CHRISTENSON
JENS LUNDGREN
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

Planting new saplings at clear-felled areas is an expensive and demanding task for forest owners. To improve efficiency and consistency, Södra Skogsägarna has initiated a project to develop an autonomous vehicle to plant new saplings in clear-felled areas. A crucial function of the system is how to select the planting spots.

This thesis aims to create a deep learning-based computer vision model to locate favorable planting spots. A stereo camera that provides RGB-D data from different scenes, where a sapling should be planted, will be used. The created model takes this data as input and returns the coordinates of two proposed planting spots. The model is based on a YOLO network for object detection and two different implementations of U-Net networks for segmentation. The algorithm was able to find good planting spots in 81% of the test cases.

A discussion of the most common reasons why the model occasionally proposes invalid planting spots and suggestions on how to solve these problems are given. Suggestions are also given on how the project group could proceed with the project and improve the system. The main conclusions are that a better suited camera than the one used in this thesis should be used and that more data should be collected to increase the models robustness.

The code for the final system can be found in the repository `https://github.com/Birken666/Master-Thesis`.

Keywords: Computer Vision, Stereo Vision, Machine Learning, Object Detection, Semantic Segmentation, Forestry, Tree Planting

Preface

This report is the result of our master's thesis project carried out at Bit Addict AB in Gothenburg, and the last part of our M.Sc. degree at Chalmers University of Technology. Bit Addict AB is a consulting company that currently, among other projects, assists in the development of an autonomous tree-planting robot. The aim is for an early stage prototype robot to be available and tested in the field during planting season 2023. The company is in charge of the control system, which involves navigating the robot at clear-felled areas and selecting favorable spots to plant a sapling.

Acknowledgements

We would like to thank Krister Wolff, our supervisor and examiner at Chalmers University of Technology. We also would like to thank our supervisor at Bit Addict AB, Emil Gustafsson, who together with the other employees Karl Svensson, Axel Löf, Marcus Hilding Södergren, and Niclas Wikström contributed with relevant feedback and discussions throughout the project. Finally, we would like to thank all other Bit Addict AB employees for making us feel welcomed.

Thank you.

Olle Christenson Jens Lundgren, Gothenburg, June 2022.

Nomenclature

RGB-D	Red Green Blue - Depth
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
Pixel	Picture Element
kNN	k-Nearest Neighbour
IoU	Intersection over Union
NMS	Non-Max Suppression
API	Application Programming Interface
YOLO	You Only Look Once

Contents

1 Introduction

Sweden has a vast forest cover with abundant forest resources, where almost 70% of the land area is covered by woodland [1]. Since early industrialization, forests have played an important role in Sweden's economic development, and, among many different purposes, trees can be used to produce paper, clothing, bio-fuels and energy [2]. The largest forest owners association in Sweden is called Södra, an international forest industry group with more than 5000 forest owners [3]. In 2018 the members collectively owned 2.5 million hectares of forest, and their trees are regularly felled and replanted to make use of the renewable material [4].

After clearing an area of trees, Södras current method for planting new trees involves using a machine to prepare the soil and then manually placing the saplings on the prepared ground. While this method provides good conditions for saplings to grow, both present difficulties for both Södra and the vegetation. For example, common methods for soil preparation, such as harrowing, often prepare an excessive amount of ground and cause unnecessary damage to surrounding vegetation, significantly reducing the presence of lichens [5]. Manual planting of saplings, on the other hand, does not damage the area, but is dependent both on the availability of workers and on their ability to identify good planting spots [6]. To solve these problems, Södra initiated a project called BraSatt in 2020, whose objective is to develop an autonomous planting vehicle. The planting vehicle, called *E-Beaver*, is an 8-wheeled machine, split into a front and rear half, using what is called an articulated steering system to navigate. A large excavator is mounted on the back of the E-Beaver, from where the preparation and planting of the saplings will be carried out. The goal of the project is for the vehicle to autonomously navigate through the terrain, select a valid planting spot, prepare the specific patch, and lastly place a sapling at the prepared area.

Among the six other companies working on the complete project [7], the engineering and consult agency Bit Addict AB is in charge of developing the control system and automating the navigation and planting aspects of the vehicle. The purpose of this report is to, under the supervision of Bit Addict AB, develop a method that autonomously identifies good planting spots that offer the saplings good conditions to grow.

1.1 Purpose and Scope

The E-Beaver uses an excavator mounted on the robot to dig planting holes and then plant the saplings. The planting spot is prepared by pressing the excavator bucket down into the ground, and if the resistance passes a certain threshold, the machine stops and assumes that it cannot dig in this particular area. Trying to dig in a spot can be both time and energy consuming, and it is therefore desirable to make as few failed attempts as possible. To reduce the risk of encountering invalid spots, a stick can be inserted into the ground to perform a small-scale test before trying to dig. Although this saves some time and energy, this system can be taken one step further. Instead of blindly inserting the stick into the ground at different locations, the system can gather and analyze image data from the scene before selecting a spot. For areas where it is possible to see common obstacles or other undesired obstructions from above ground, a camera can be used to directly infer these areas as invalid.

This is where this thesis project comes in - by using computer vision methods, the goal is to identify, locate, and propose valid planting spots to be validated by the stick. The purpose is to make the system more efficient in time and energy than inserting the stick into blindly selected areas. Furthermore, using computer vision, the system can also try to find optimal planting spots and not just valid planting spots, in an attempt to optimize the growing conditions for the sapling. For example, convex areas usually indicate a lower risk of drowning the plant, and therefore planting a sapling in such an area can increase the probability of it surviving.

1.2 Limitations

The ambition from Södra is for the entire planting cycle to last approximately 30 seconds on average per sapling. At each location where the E-beaver stops, the goal is to plant two saplings. This means that a total of one minute can be spent moving from one location to the next, selecting two planting spots, and finally planting two saplings. Furthermore, each planting spot should be at least 0.45x0.80 meters, as these are the dimensions of the excavator bucket needed to prepare the area. The two planting spots also need to be at least

one meter apart from each other, to prevent the trees from interfering with each other when they eventually grow.

The aim of this thesis is to provide a proof of concept, which means that the proposed method can be optimized further. As no suitable preexisting data sets was found, a custom data set was made to train the machine learning models. A major limitation with the custom data set is that the data, collected in clear-felled areas, could only be collected during two different days and at two different locations. Therefore, the weather and lighting conditions are very limited in the data set, only containing images from those two days.

1.3 Optimal Planting Spots

The task of finding good planting spots in clear-felled areas can be difficult. The areas are often filled with many different visible obstacles, such as stumps, rocks, and felled trees. Additionally, there can also be more objects below the ground that are not visible from above, which also restricts the ability to dig planting holes. Although the presence of these underground obstructions can be difficult or even impossible to determine using only computer vision, visible objects can be identified and ruled out to create a list of possible valid planting spots: a list of places where it should be *possible* to plant a sapling. However, if the location and types of obstacles in the scene can be determined, it may be possible to compare these spots not only to find a possible planting spot, but also an *optimal* spot; a planting spot that provides conditions that increase the long-term survival of the sapling. An optimal planting spot can be defined by following a few rules:

Planting close to a stump is a good rule of thumb that can overrule many other cues. If there is a stump at a certain location, it means that there has been a tree that has grown at the same spot earlier. Naturally, this leads to the conclusion that the conditions for growing a tree are good enough at that location. Evidence that this is a good planting spot is already there, and no guessing or inference should be made about whether the planting spot is good or not. With this information in mind, although stumps need to be avoided directly, it is still desirable to plant close to them. However, a problem with stumps is that even if the body of the stump is avoided and the excavator tries to plant close to the stump, there can be large roots below ground that belong to the stump but are not visible. Sometimes, the beginning of these roots can be seen on the stump as they descend into the ground; see Figure 1.1. In these cases, it is possible to locate them and then try to plant in an area close to the stump, for example, between two visible root legs.

Figure 1.1: *An example of a stump with visible roots above ground. These roots continue down underground, and can not be seen in the image. While they need to be avoided, it is still favorable to plant a sapling close to the stump.*

There are also a few common obstacles that usually lie above the ground and that simply need to be

avoided. The first are rocks that lie above the ground. They can vary in size, and it is important to distinguish between small rocks and large rocks. The former will not be a problem for the excavator, whereas the larger ones are too massive to dig through and, therefore, need to be avoided. In addition to rocks, there are also large pieces of wood, usually sticks or felled trees. Sticks tend to be smaller, whereas fallen trees tend to be larger. These are handled in the same way as rocks; the large ones need to be avoided, as they can hinder the excavator from digging, whereas the smaller ones can be possible to dig through.

As mentioned above, finding possible planting spots is only part of the problem. It is also important to find a good environment for the sapling to grow. Young saplings can be weak, and, for example, planting one where there is too much water could end up drowning the sapling. In places where it has been raining, small pools of water can gather if there are cavities in the ground. These pools of water should be avoided, as they otherwise could drown the sapling. However, the E-Beaver will sometimes be used during a sunny day when there is not much water in the clear-felled area. In these cases, a sapling could accidentally be planted in a cavity or a low spot in general, and then, when it rains, the water could gather in these spots and end up drowning the plant. To avoid this, an easy but effective rule of thumb is to choose a planting spot that is as high as possible.

All of these rules can be summarized in the following list of bullet points. These have been the guidelines for identifying an optimal planting spot throughout the project.

- Avoid rocks
- Avoid large sticks
- Avoid fallen trees
- Avoid stumps - but try to plant close to them
- Avoid water
- Try to plant as high up as possible

1.4 Related work

There has been a lot of research in the broad field of computer vision, and some of the work relates to this project. One project that had a similar goal is the paper *Stereo Vision Based Tree Planting Spot Detection*. That project has a great similarity to this thesis project in some areas; however, the approach was very different. In that paper, the planting spots are inferred using a 3D Implicit Shape Model (IMS), where the 3D shape is constructed using a Stereo Camera. Following this method, a 3D point cloud using stereo vision is created, and then a good planting spot can be found by detecting mounds in the point cloud. Fast Point Feature Histograms (FPFHs) are used to extract features at each position in the point cloud; then an algorithm is trained based on these features to create a contextual system to choose the planting spot. Shortly, this works by training a *codebook*, which consists of several *codewords*, where a codeword is a centroid of k-means clustering of the FPFH features. All of these codewords will cast a vote on where it thinks the best planting spot is. Predefined planting spots are used as data to train the codewords on how they should propose a planting spot, relative to their own position. In the inferring stage, the FPFH will be calculated for all feasible points in the point cloud, and the points will vote for the new planting spot according to the codeword to which they are assigned. This system will then directly infer a decision for the planting spot, which will be fully based on the shape of the 3D point cloud [8].

The ambition for this thesis, however, is to infer the planting spots by avoiding obstacles that could prevent the machine from planting a sapling. The work in *Implementation of a System for Real-Time Detection and Localization of Terrain Objects on Harvested Forest Land* investigates the possibilities of detecting objects in a forest environment. The article was helpful at the beginning of the project, as it showed that object detection is effective for the kind of objects that are relevant for this project [9].

The PhD-Thesis of Ahmed Ostovar is another relevant work. The topic of this thesis is object detection and recognition in outdoor environments. It provides a good summary of many different techniques and methods

that can be used to detect objects in cluttered environments, similar to the forest environment that the researchers used in these projects [10].

1.5 Contribution

While the project *Stereo Vision Based Tree Planting Spot Detection* exists, which had a similar goal as this one, this project attempts to contribute what that project was missing. One of the conclusions in that paper was that *"Additionally, most of the errors in the detection were explicable and the detection performance could be enhanced by removing the most obvious error sources, such as tree stumps, before attempting to plant"*. Although that project tries to find an optimal planting spot only based on 3D IMS, it does not take into account any visual obstacles. Therefore, this project solves what this paper is lacking without re-inventing what has already been done in that project. Although *Stereo Vision Based Tree Planting Spot Detection* was the most similar project found, it is still difficult to compare the results. The reason is that the results shown in that project describe the performance of a model that detects mounds created by a continuously operating spot mounder [8]. This thesis, however, shows a model that finds planting spots in clear-felled areas where the excavator can then dig.

Although a model is created that is able to detect and locate many obstacles in the forest, the model is not necessarily perfect. However, since we contribute a large data set relevant to the task, this can be used to train new models and then create even better algorithms. Therefore, the contribution of the data set, which can be used to train object detection models, segmentation models, and testing of arbitrary models, is also an important contribution.

Suggestions on how to continue the project are also provided. From the results of this project, indications are shown on which methods could work in the future to create a model that is supposed to locate an optimal planting spot used in the conclusive BraSatt project. A tool was also created to create segmentation data with a four-channel input. The tool can be used to annotate new data if the images are taken with a similar camera and one wants to create RGB-D data as input to a single-class semantic segmentation network.

The final model can be seen as a proof of concept. It shows that the method proposed in this thesis could be used to solve the task of finding planting spots. Together with the results, the method provides an important step toward autonomous tree planting. Although the results are not perfect, using the proposed method as a framework, an even better model could be created by using more data and better sensors.

2 Theory

In this project, deep learning based computer vision methods are the main subject. All the methods and results are somehow related to the field of computer vision. Therefore, the most important concepts are explained in this chapter to introduce some concepts to the reader, so that it will be easier to understand the rest of the report. In this chapter, the theory behind the most important techniques used in this project will be presented.

2.1 Computer Vision

For many years, researchers have been working on trying to make computers able to see. This task has shown to be much more difficult than what was initially expected, since the way humans and other animals perceive reality was thought to be less complicated than it actually is. Research has now been going on for many years on the subject, and the field has expanded into multiple subcategories, that can be collected under the term of computer vision [11]. To sum up the term, computer vision is a field of research regarding problems that involves images or image data in general. Many different kinds of problems can be included under the term, such as object detection, person tracking and stereo matching [12]. In recent years, there has been a rapid development and the understanding of the field has grown a lot. Tasks that seemed impossible earlier can now be solved with modern methods.

Because of the greatly increased computing power and popularity of artificial intelligence, deep learning (DL) has been introduced in many different fields, especially computer vision. Many state-of-the-art methods used in the field are nowadays based on deep learning. These methods, however, often use concepts from the traditional field of computer vision, and combine them with for example neural networks [13].

Before diving in to the modern methods of computer vision using neural networks, a few important concepts from traditional computer vision will be explained.

2.1.1 Image Representation

A color image can be represented as three matrices of equal size, corresponding to the red, green and blue (R,G,B) values of the image. Usually, the pixel values have an 8-bit depth, and therefore each matrix element can consist of any integer value between 0 and 255, denoting the intensity of each color that should be displayed in each pixel. If a *depth map* is provided, one can also produce an RGB-D representation of an image. Here, the fourth D-channel represents the distance (or disparity) value for that pixel, with respect to the camera.

2.1.2 Stereo Vision

When using a single camera, it is not possible to infer the 3D world coordinates of the points in the image. When using two cameras, however, it is possible to use *stereo matching* to calculate the position of the point and thus receive a depth map [12]. The principle is that, by using two cameras mounted close to each other, two slightly different images of the view can be produced. By finding corresponding points in the two images, one can calculate the distance to that point using triangulation, see Figure 2.1. One of the most difficult tasks of stereo vision is point matching, i.e. finding the points in the two images that are corresponding.

2.1.3 Fisheye Lens

Another important concept in computer vision is the phenomenon that occurs when the camera lens has a convex shape. This leads to an effect described as radial distortion, often called the fisheye effect, see Figure 2.2. This occurs because the convex shape of the lens bend the light differently depending on where the light rays hit the lens. The lens breaks the light with a larger angle closer to the edge, and smaller angle closer to the middle, which leads to a distorted projection of the image. A major benefit of using a convex lens is that the lens can acquire a very wide angle of view.

To deal with the radial distortion of the images provided by the fisheye lens, one can perform a *rectification* of the image. By remapping the pixels in the raw image data, a rectilinear perspective of the image can

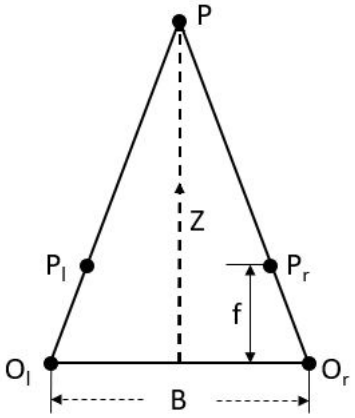


Figure 2.1: *Illustration of how triangulation works between two images, to obtain the Z-coordinate of a point. P is the point in the real world, and the objective is to find the coordinates of P. P_l and P_r are the projections of the point P in the left and right camera, respectively. O_l and O_r are the camera centers of the left and right camera. B is the baseline, which corresponds to the horizontal distance between the two camera centers. The equation to find the Z-coordinate is then described as $Z = \frac{fB}{B-c}$, where f is the focal length of the cameras and c is the distance between P_l and P_r.*

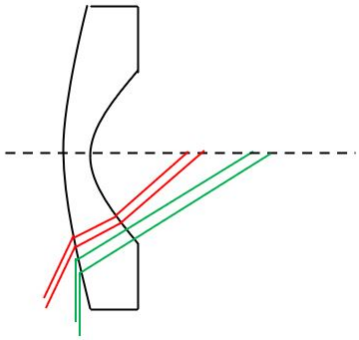


Figure 2.2: *Illustration of how the fisheye lens breaks the light differently depending on where it hit the lens. In practice, the light usually go through a few more lenses before projecting the image. Note that the rays that enter from a very wide angle are still able to hit the lens, and break into the receptive field, due to the convex shape of the lens.*

be acquired. The radial distortion is governed by the following infinite sum:

$$\Delta x = x(k_1 r^2 + k_2 r^4 + ...)$$

$$\Delta y = y(k_1 r^2 + k_2 r^4 + ...)$$

where $r^2 = x^2 + y^2$ and $k_1, k_2, ...$ are radial distortion parameters: One or two are usually enough to model distortion for real-life applications. These equations can then be used to correct the distortion using interpolation [14].

2.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational network that, in a gross manner, attempts to mimic the inner workings of a mammals central nervous system [15]. As the name implies, it is a network of multiple *neurons*, through which the data is propagated. A network can have multiple layers of neurons, called *hidden layers*, with a varying number of neurons per layer. A standard ANN propagates in a linear fashion, where

each neuron receives its input from the output of all of the neurons in the layer before it. This raw input is then multiplied by an individual *weight*, which in turn are summarized, and a *bias* is added. Lastly, this value is passed through a non-linear *activation function*. A schematic overview of how a single neuron works can be seen in Figure 2.3. Figure 2.4 depict a schematic overview of a *fully connected deep* neural network, with 3 input neurons, 2 hidden layers with 5 neurons each and 2 output neurons. The network is described as fully connected, clarifying the general structure of all neurons in a layer being connected to all layers in the previous layer. It is also described as deep, meaning that there are multiple layers between the input and output layer [16].

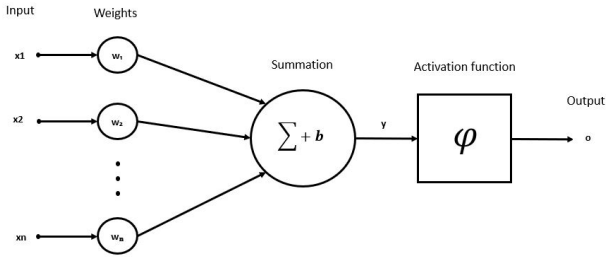


Figure 2.3: *Schematic overview of the mathematical operations being executed when propagating input through a neuron.*

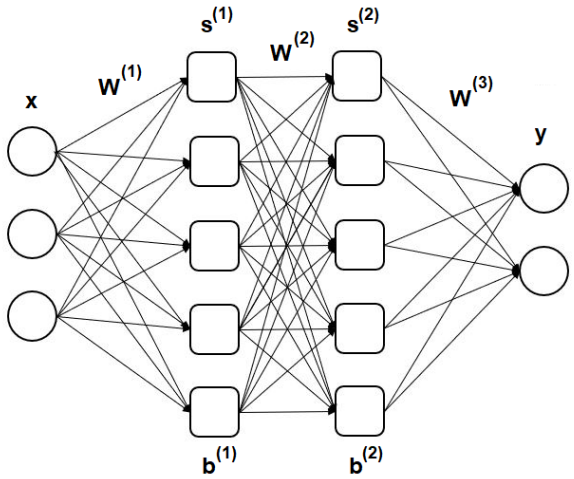


Figure 2.4: *Schematic overview of an example of a neural network with 3 input neurons, 2 hidden layers, with 5 neurons each, and 2 output neurons.* $\mathbf{x}$ *and* $\mathbf{y}$ *are the inputs and outputs, respectively, while* $\mathbf{W^{(n)}}$ *and* $\mathbf{b^{(n)}}$ *corresponds to the weights and biases for layer* $\mathbf{s^{(n)}}$.

The activation function, denoted φ in Figure 2.3, is needed to introduce non-linearity to the equation. Without it, the neural network could essentially be reduced to a linear regression model that would not be able to capture non-linear patterns. There are many different activation functions, but some of the most common are the hyperbolic function *tanh*, the sigmoid functions, and Rectified Linear Unit (ReLU); see Figure 2.5.

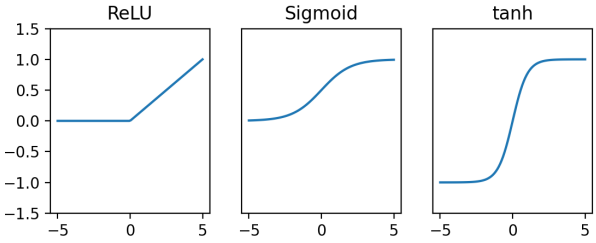


Figure 2.5: *Comparison between three common activation functions: ReLU* ($max(0, x)$)*, Sigmoid* ($\frac{1}{1+e^{-x}}$) *and* $tanh(x)$.

In total, the value of a neuron s is defined as:

$$s = \varphi\left(\sum_j W_j x_j + b\right) \tag{2.1}$$

Unlike other conventional algorithms, ANNs can solve a wide variety of complex problems, such as non-linear or stochastic problems, while using simple computational operators, such as addition and multiplication [17]. The correctness of the network predictions is calculated using a *loss function*, a metric to quantify the difference between a known ground truth value $\mathbf{t}$ and the output of the network $\mathbf{y}$. A common choice of loss function is Mean Squared Error (MSE), defined as

$$L = \frac{1}{2}\sum_i (t_i - y_i)^2 \tag{2.2}$$

The *weights* between the neurons, and the *biases* of each neuron, are often randomly initialized and then iteratively updated to converge towards a local minimum of the loss function using some sort of optimization method. This update process is called *backpropagation*, and the weights are updated with respect to the loss function. A common optimization method is gradient descent, which increments the weights in the following manner:

$$\delta W_{mn}^{(k)} = -\eta \frac{\partial L}{\partial W_{mn}^{(k)}} \tag{2.3}$$

Here, the small parameter $\eta > 0$ is a predetermined learning rate that scales the magnitude of each descent. The derivatives of the loss function are evaluated using the chain rule: for the last weight $\mathbf{W^{(3)}}$ in Figure 2.4, the chain rule is applied once to obtain:

$$\frac{\partial L}{\partial W_{mn}^{(3)}} = -\sum_i (t_i - y_i)\frac{\partial y_i}{\partial W_{mn}^{(3)}}, \tag{2.4}$$

And then once again:

$$\frac{\partial y_i}{\partial W_{mn}^{(3)}} = \frac{\partial}{\partial W_{mn}^{(3)}}\varphi\left(\sum_j W_{ij}^{(3)} s_j^{(2)} + b_i\right) \tag{2.5}$$

Which can be written as:

$$\frac{\partial y_i}{\partial W_{mn}^{(3)}} = \varphi'\left(\sum_j W_{ij}^{(3)} s_j^{(2)} + b_i\right)\delta_{im} s_n^{(2)} \tag{2.6}$$

Here, δ_{im} is the Kronecker delta: $\delta_{im} = 1$ if $i = m$, and zero otherwise. The biases are updated in a similar way:

$$\delta b_m^{(k)} = -\eta \frac{\partial L}{\partial b_m^{(k)}} \tag{2.7}$$

For $\mathbf{b^{(2)}}$, this yields:

$$\frac{\partial L}{\partial b_m^{(2)}} = (t_m - y_m)\varphi'\left(\sum_j W_{ij}^{(3)} s_j^{(2)} + b_i\right) \tag{2.8}$$

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of ANNs, often used for object recognition and pattern detection in image data. While regular fully connected ANNs can be applied to solve a broad variety of problems, its major drawback is the large number of parameters, often leading to excessive complexity which, in turn, can penalize the overall learning of the network. To combat this, CNNs are mainly based on a different type of layer, namely convolutional layers. The purpose of the convolutional layer is to extract specific features of the input image and reduce its dimensionality, often applying a fully connected layer in the end to perform the actual task at hand.

Through the convolutional layer, the input image is gradually exposed to *kernels*, (sometimes called *filters*), and the dot product between the image patch and a kernel is calculated. Using backpropagation, each kernel learns to recognize certain patterns or features, such as edges or squares. Since these features occur in multiple parts of the images, the same kernel and its corresponding values are used on all parts of the input, leading to a greatly reduced complexity compared to a fully connected ANN. Figure 2.6 illustrates how a kernel is gradually applied to an input image.

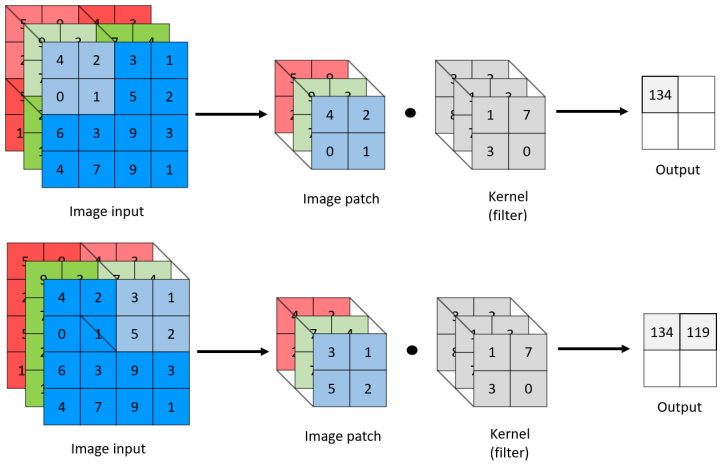


Figure 2.6: *A 4x4x3 RGB input image is being passed through a 2x2x3 kernel, yielding a single value as output per patch. A pre-determined stride determines how much the kernel will shift each step. Here, the stride is set to 2 and the complete output will be a 2x2 matrix.*

One of the most straightforward applications of a CNN is to classify an image. For example, it can be used to predict whether an image contains a cat or a dog. However, classification only takes into consideration a single class for the entire input image. If there are multiple objects in the same image that need to be identified, one needs to use a method called object detection instead. Object detection is the task of detecting one or multiple objects in an image by drawing a bounding box around them, and then classifying these boxes. The bounding boxes both classify what object lies inside them, as well as finding the location of the objects in the image.

2.2.2 YOLO

You Only Look Once (YOLO) is an object detection algorithm that gained a lot of attention when it was released in 2016, due to its speed compared to other methods at the time. As the name implies, the algorithm only passes through the image once. It is a neural network that takes an image as input and outputs bounding box coordinates with class probabilities for each bounding box. The first step of the YOLO algorithm is to divide the image into a grid of S x S cells.

Each cell tries to detect an object by proposing B bounding boxes with the corresponding confidence scores. The proposed bounding boxes contain x- and y-coordinates of all bounding box centers, *width* and *height* of the boxes, and a *confidence score* that an object appears in the box. Furthermore, each cell predicts *conditional probabilities* for each class. The confidence score indicates both the confidence that the predicted bounding box

contains an object, but also how accurate the box is. The confidence is defined as $Pr(object) * IoU^{truth}_{prediction}$, where IoU is the *intersection over union*, illustrated in Figure 2.7.

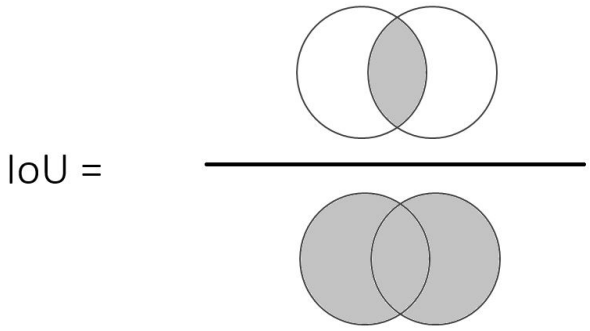


Figure 2.7: *Intersection over Union illustrated. As the name implies, the intersection is the shared area between both sections, while the union is the total area of both sections.*

While the confidence score estimates a general certainty that **any** object is inside the box, the class probabilities determine what particular class the objects is most likely to be in the bounding box. For a model with C potential classes, each cell therefore returns a prediction of $B * 5 + C$ values [18]. The predictions are then processed using *non-maximum suppression* (NMS), an algorithm to filter out the best predictions among all proposed bounding boxes [19]. The NMS algorithm can be summarized by the following steps, given a list of all proposed boxes B_{tot} and an initially empty list of filtered proposals D:

- Select the proposal with the highest confidence score from B_{tot} and move it to D.
- Calculate the IoU between the selected box and all other proposed boxes. If the IoU is greater than a preset threshold value, the other prediction is removed from B_{tot} as it is assumed to represent the same object, but with less accuracy.
- Repeat this process until B_{tot} is empty.

The end result acquired from NMS is a filtered list of predicted bounding boxes, where only the best-fitting bounding box per object is saved; see Figure 2.8. Additionally, a preset confidence threshold is often used to filter out predictions with low confidence scores.

Figure 2.8: *Proposed bounding boxes before (left) and after (right) non-maximum suppression is applied. Only the box with the highest accuracy score is kept, while the others are discarded.*

2.2.3 YOLOv5

Ever since the first version of YOLO was published, the general method has been improved and new versions of YOLO have been released. The latest major release is called YOLOv5, which is an open source repository where constant changes are steadily being pushed. As of April 2022, the latest version of YOLOv5 is v6.1, which is the version used for the final algorithm in this thesis. Among different improvements from the original YOLO, YOLOv5 also allows a selection of different network sizes, ranging from a small-scaled nano to a large-scaled XL version. The main difference between the sizes is the amount of kernels in the convolutional layers. Larger scaled versions generally provide more accurate predictions, but take longer to train and infer [19].

Anchor Boxes

A feature that was not used in the original YOLO version is the use of *anchor boxes*. The idea behind the anchor boxes is that instead of the model creating a bounding box from nothing, it uses predefined anchor boxes for each cell, which can be seen as default bounding boxes. The neural network then has to predict a small offset that slightly changes the anchor boxes, rather than predicting the geometrical features of the bounding box.

Since every anchor box in YOLOv5 generates a bounding box, the output size of every cell is increased to $n_{anchors} * (n_{classes} + 5)$. Furthermore, the class and *objectness* is predicted for every anchor box in every cell instead of just one class prediction per cell. Objectness represents how accurate the bounding box created is according to the model, where the target objectness is defined as the IoU between the predicted bounding box and the ground-truth box in training.

To initialize the anchor boxes, K-means clustering is used on the training data, where K is the amount of desired anchor boxes. Instead of creating the clusters using Euclidean distance, the distance is calculated using:

$$d(box, centroid) = 1 - IoU(box, centroid) \tag{2.9}$$

After clustering the bounding boxes from the training data into K clusters, the centroid of each cluster is used as the initial guess for the anchor boxes. After initialization, an evolutionary algorithm is used to fine-tune the anchors according to the loss function used in training. By using the loss function from training as the objective function in the evolutionary algorithm, the anchor boxes are tuned to yield a smaller loss for the bounding box prediction priors, which should lead to smaller offsets on average to fit the boxes. This recalculation of the anchor boxes will be executed if the best possible recall for the anchor boxes is less than 98% for the data [19].

2.2.4 The Network Architecture of YOLOv5

To understand the full structure of the YOLOv5 model, it is a good idea to begin by trying to understand the smaller building blocks of the model, which can be seen as modules.

The Conv Module

The Conv Module consists of a normal convolutional layer, followed by a *batch normalization*, and lastly a *Sigmoid Linear Unit* (SiLU) activation function [19]. The purpose of batch normalization is to achieve faster and more stable training, by normalizing the mean to zero and the variance to one for each mini batch, i.e. all the data that the loss was calculated for[20]. The SiLU activation function is a modified combination of the previously mentioned sigmoid and ReLU functions, see Figure 2.5. In practice, SiLU looks like a continuous approximation of ReLU, which for values with large magnitude are very similar. The SiLU function is computed by multiplying the sigmoid with its input: $x\sigma(x)$ [21]. See Figure 2.9 for a schematic overview of the Conv Module.

The C3 Module

The C3 Module is a CSP bottleneck (from "Cross Stage Partial Networks[22]) with 3 convolutions. This module is built using the formerly described Conv Module and the concept of a CSP bottleneck. The idea is that

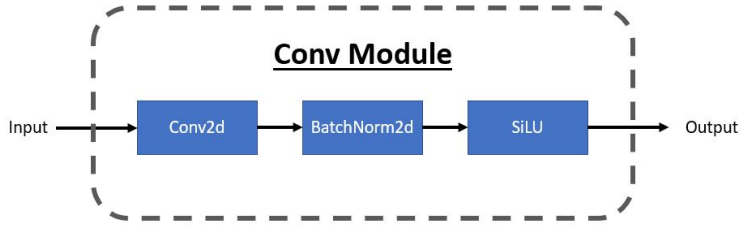


Figure 2.9: *The Conv Module. The return function from this module is computed in the following way: y = SiLU(BatchNorm(Convolution(x)))*

when going from c_{in} channels to c_{out} channels in this module, two different paths are used to create the output channels from the input. Some, usually 50%, of the feature maps are created by only going through one Conv module, while the other feature maps are created by propagating through a longer path consisting of sequential bottlenecks. Both of the parts are then concatenated to get the full amount of output channels. Then, all of the data goes through one final Conv module. See Figure 2.10 for an overview of the C3 Module. To briefly mention the workings of the bottlenecks, they propagate the input through two Conv Modules that return the same amount of channels as the input[19].

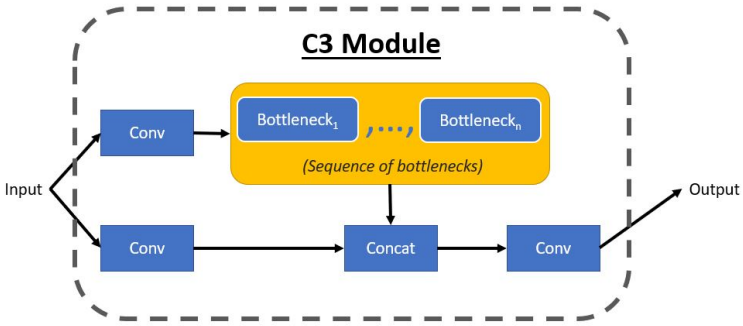


Figure 2.10: *The C3 Module. The return function from this module is computed in the following way: y = Conv(Concatenate(BottleneckSequence(Conv(x)), Conv(x)))*

The SPPF Module

The Spatial Pyramid Pooling (SPPF) Module is, like the C3 Module, based on the Conv Module, but also combined with the Max-Pooling layers. The purpose of this module is to find features of different scales. With every Max-Pooling layer, the feature map becomes coarser and the objects or features of larger sizes are more likely to be found. In the end, the created feature maps of all different scales are concatenated to find objects or features of all different sizes [19].

Using these modules as building blocks, the entire architecture of the YOLOv5 model, which can be seen in Figure 2.12, can be split up in to three parts: the Backbone, the Neck and the Head.

The Backbone Structure

Backbone is a term commonly used in the field of deep machine learning, which refers to the part of the network that extracts features. The backbone of YOLOv5 is based on CSPNet, mentioned in the C3 Module. The purpose of creating the CSPNet was to deal with three problems: strengthening the learning capacity of a CNN, removing computational bottlenecks (evening out the amount of computations between layers) and reducing memory cost [22]. The backbone is built by alternating between Conv Modules and C3 Modules, and finishing

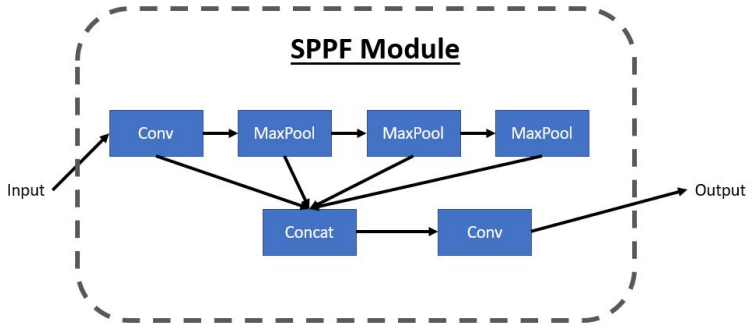


Figure 2.11: *The SPPF Module. The return function from this module is computed in the following way: y = Conv(Concatenate((x, MaxPool(x), 2*MaxPool(x), 3*MaxPool(x))*

off with an SPPF Module.

The Neck Structure

The neck is used to create *feature pyramids*, whose purpose is to identify objects on different scales. In YOLOv5, Path Aggregation Network (PANet) is used as the neck of the model [19]. This method downsamples with convolutional layers and then *upsamples* (increases the x-y-dimension of the image) the image data back and forth between high resolution and low resolution to find objects/features on different scales. In other words, this method tries to use the information from all feature levels in the network rather than just using the information from the last feature state. The motivation for the architecture is that early layers are usually able to extract simple patterns and textures, while succeeding layers usually are able to extract more intricate features representing specific objects. In the end, all kinds of features help classify the data; therefore, the information from the early layers is also utilized when making the predictions. The neck uses ResNet as the basic structure[19], where $\{P_2, P_3, P_4, P_5\}$ denotes feature levels of different resolution generated by the feature pyramid, with P_2 as the highest resolution and P_5 as the lowest. The idea is that larger objects will be found in the coarser structures, i.e. larger P-levels, and smaller objects in the finer grids, i.e. smaller P-levels.

By looking at the structure in Figure 2.12, one can see that the neck begins by stepwise upsampling, taking the 8x8 (P5) resolution output from the backbone and then upsampling it to 32x32 (P3). It then uses this P3 state, concatenates it with the P3 state from the backbone, and proposes bounding boxes on this latent state. After this, the network downsamples the resolution with convolutions down to P4 and concatenates it with the other P4 state in the neck, and proposes bounding boxes on this latent state. Lastly, it downsamples the resolution with convolutions down to P5 and concatenates it with the other P5 state in the neck, and proposes the last bounding boxes. This means that, prior to the bounding box proposals for every resolution stage, two collections of feature maps of the same resolution are concatenated. The feature maps from a state that was achieved by upsampling get concatenated with a state that was achieved from downsampling. The idea is that the feature maps created by the downsampling will have more detailed information since the features are created from convolutions on a higher resolution. On the contrary, the feature maps that are achieved from the upsampling hopefully contain more contextual information, since they are created by upsampling coarser and richer feature maps.

The Head structure

The Head calculates the final detections of the model. This is where the anchor boxes are applied, together with the features to create the final output. As can be seen in Figure 2.12, the head detects bounding boxes at three different scales: P4, P4, and P5.

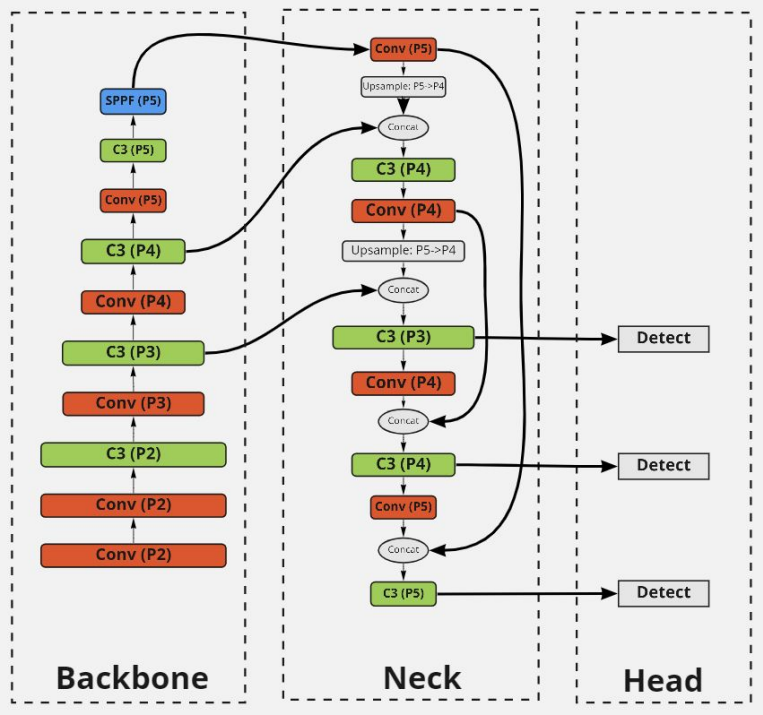


Figure 2.12: *Structure of the YOLOv5 network, built up by the Conv, C3 and SPPF modules. The numbers P2 - P5 represent the size of the feature maps at that stage, where feature maps at P5 are the smallest, consisting of 8x8 matrices, and feature maps at P2 are the highest, consisting of 64x64 matrices. The bounding box predictions are proposed at three different scales: P3, P4, and P5, from the last three C3 Modules.*

2.2.5 Semantic Segmentation

While object detection is an efficient way to localize an object in an image, it does so by proposing a bounding box around the item. Semantic segmentation, on the other hand, attempts to locate the exact position of an item by classifying each pixel in an image as belonging to either a specific class or the background. An illustration of the difference between semantic segmentation and object detection can be seen in Figure 3.1.

2.2.6 U-Net

U-Net is a well-established model architecture used for semantic segmentation [23]. The initial U-Net architecture was introduced in 2015, but the original structure is often slightly modified depending on the application. An example of the network architecture can be seen in Figure 2.13. The general architecture of the network can be split into two main parts: a contracting (or encoding) part on the left side, followed by an almost symmetrical expansive (or decoding) part on the right side, yielding a U-shaped network. In the contracting part, the input image is gradually being convoluted to a smaller latent space, resembling the architecture of a typical convolutional network by applying regular convolutional layers with ReLU activation, followed by max-pooling layers. In each step, the x-y-dimension of the input is reduced, while the z-dimension, or the number of feature channels, is doubled. In the expansive part, the input is gradually passed through an upsampling layer, a concatenation with the corresponding output from the contracting path, and convolutional layers with ReLU activation. At each step in the expansive path, the x-y-dimension of the input is increased using the upsampling layer, while the number of feature channels is halved using the convolutional layer. In Figure 2.13, the final layer of the network uses a 1x1 convolution to map each 64-component feature vector to a single class integer. Therefore, the final output of the network has the same x-y-shape as the input, but with a single integer value denoting the pixels class index.
A major difference between U-Net and similar networks is the use of the copy and concatenating part, represented by gray arrows in Figure 2.13. This connection provides information from multiple scales of the image size, allowing the successive convolutional layer to assemble a more precise output based on this information [23]. Additionally, instead of assembling and training the network from scratch, a pre-trained CNN classifier is often

used for the encoding part, to help extract features to a latent space. Popular choices for encoder architectures include state-of-the-art classifiers such as Inceptionv3 [24], different variations of ResNet [25], or VGG-16 [26]. They are often pre-trained on large-scale datasets, such as ImageNet [27]. The final classification layer of these networks is removed, so that the output of the encoding part only contains the feature extraction, rather than the class prediction. The aim of the pre-trained classifier, often called *backbone*, is to produce a better feature extraction than an encoder trained from scratch and thus, by the use of transfer learning, increase the performance of the U-Net network.

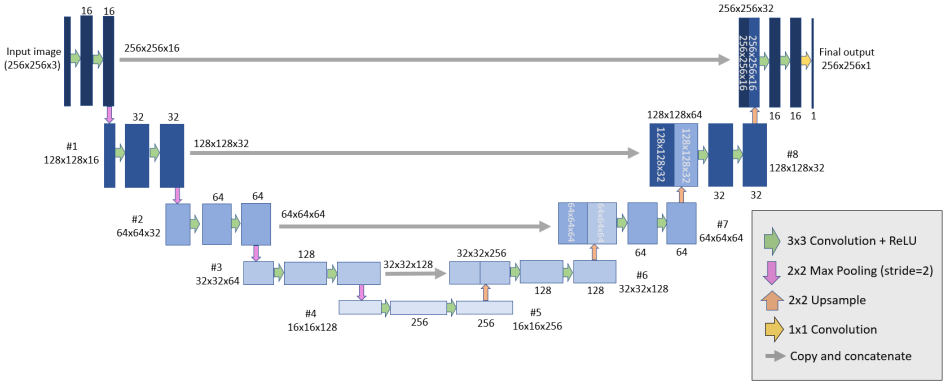


Figure 2.13: *An example overview of how an U-Net architecture can look like. The input image is gradually convoluted to a smaller x-y-size, while the depth (number of feature maps, denoted at the top or bottom of each box), gradually increases on its way to the bottle neck. The different operations are described in the bottom right corner. In this example, the input of each convolution is padded so that the input and output have the same x-y-shape. Adaptation of original image in [23].*

3 Method

The aim of the thesis can be divided into two main subtasks: (1) collecting and annotating a data set for training and evaluating data-driven methods and (2) developing a method to identify valid planting spots based on the collected data.

To create a custom data set for this project, it was important to keep all the potential obstacles in mind when creating the data set. The approach to solve the subtasks was to begin by conceiving a rough idea of what methods could be used to locate valid planting spots, then create a data set suited for these methods, and finally train and implement all parts of the model. Note that there were still many opportunities to test different architectures and models, although the type of method was decided. For example, images annotated with bounding boxes are limited to models that predict bounding boxes, but there are many different methods and model architectures that can be applied to make such predictions.

3.1 Locating a Valid Planting Spot

The task of identifying valid planting spots can be defined in two different ways: (1): Create a model that directly recognizes an area as a valid planting spot, or (2): create a model that recognizes an invalid planting area and then classifies all other areas in the scene as valid planting areas. A decision was made to use the latter method, since it is more straightforward to define recurring obstacles than valid spots immediately. Later, two more steps were added to find the optimal planting spot among the valid ones. In the end, the general approach consisted of the following three steps:

1. Identify all visible obstacles and areas where the excavator cannot dig, such as large rocks, stumps, and other common objects in the forest. These areas are ruled out, while the rest remain as possible planting spots.

2. If there is a stump at the current location, try to locate an area close to it while still avoiding its exact position, as it is expected that there are favorable growing conditions close to stumps as described in Section 1.3. If a large enough area is found, without interfering with other obstacles, choose this as an optimal planting spot.

3. The next step after trying to plant between the root stems is to spread the rest of the proposed planting spots over the valid planting area. These proposals are ranked based on their altitude; spots that are higher up are better, as explained in Section 1.3.

The goal is to plant two saplings at each location. Since it is likely that some proposed planting spots will be invalid in practice, the model also needs to propose and rank additional planting spots as backup, in cases where the excavator fails to plant at the proposed spot.

3.2 Non-Deep Learning Approaches

Many of the state-of-the-art deep learning methods can be computationally expensive to train and infer. Because they are so complicated, it can be difficult to analyze if something goes wrong, or why the method isn't working. Therefore, using an interpretable model where one can understand what happens is usually preferable, both because it is possible to understand the inner workings and because it is usually computationally cheaper. Initially, a lot of time was spent trying to use different interpretable machine learning models without neural networks and other more straightforward techniques. An example of a non-deep learning method that was investigated was clustering. By grouping the RGB-D data into different clusters, based on similarities in color or height above ground, it could be possible to group all pixels into different subsets. The intention was for each of these subsets to correspond to a specific type of obstacle. Another method that was explored was thresholding: Since the stumps usually stick out a bit, the idea was that thresholding using the depth data would be sufficient to find stumps. However, none of these methods yielded any good results. Furthermore, no previous work showing successful results was found for similar tasks. Therefore, this project turned to deep

learning methods since earlier work was found that showed successful results when using deep learning to solve similar tasks.

3.3 Selecting Deep Learning Models

Detecting and locating objects in an image is a common task in computer vision. Two common types of output that can be obtained from deep learning models are bounding boxes and segmentation masks, see Figure 3.1. Bounding boxes are rectangular and provide general information about the position of an object. However, they do not provide detailed information about the exact pixels that the object covers. This means that it is unknown where the contours and edges of the object are located; it can only be concluded that the object is located somewhere inside the bounding box. The semantic segmentation output, on the other hand, attempts to provide much more precise information and classifies each pixel in an image to belong to a specific object type.

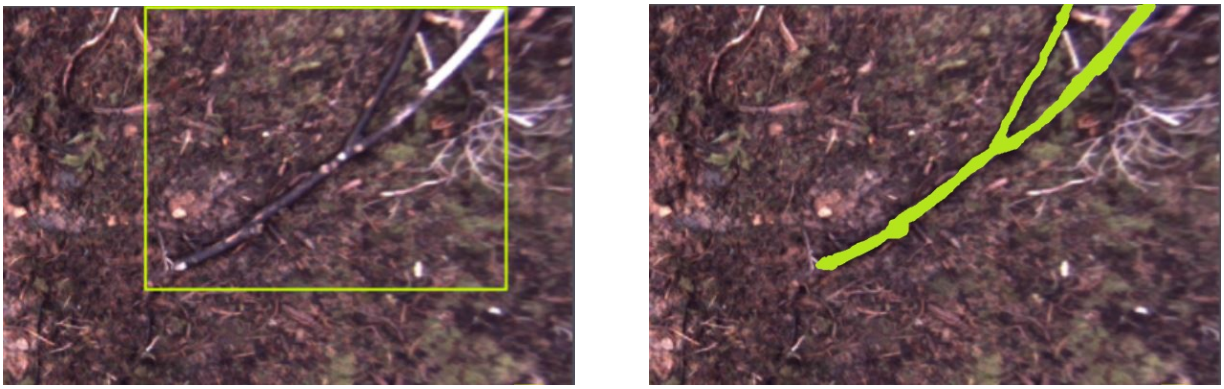


Figure 3.1: *Example of detecting a stick with a bounding box compared to segmenting the stick with a semantic segmentation network. As seen, the area of the segmentation mask is much smaller than the area of the bounding box.*

Both types of models have their advantages. An obvious perk of using a semantic segmentation model is that it yields more exact information about the location of an object in the input image. If bounding boxes are predicted, all pixels inside the bounding box must be classified as invalid planting spots, while the segmentation model only rules out the exact pixels as invalid. Bounding boxes, on the other hand, are much more time-efficient to annotate when creating a training set for the model. Drawing a bounding box around an object is generally much faster and easier than making a detailed trace around an object.

In the end, bounding boxes were considered sufficient to detect rocks and pools of water, since these obstacles generally fit well within a bounding box without the bounding box occupying a larger area than necessary. Additionally, obstacles such as rocks and pools of water often occupy or affect an additional area beneath the ground, not visible to the camera. Therefore, having an additional margin on the location of these objects can be beneficial.

On the contrary, large sticks and fallen trees usually have a different kind of geometry: they tend to be elongated and completely above ground. A bounding box around these obstacles tends to cover a lot of area to fit around the stick, with a lot of empty space inside the box, especially if the stick lies diagonally in the image as shown in Figure 3.1. With this in mind, a decision was made to use semantic segmentation to handle this type of obstacle.

As mentioned above, it is favorable to plant saplings close to a stump since it yields a good nutritional source and, in turn, good growing conditions for the tree. However, stumps often incorporate large roots above and below the ground, which pose obstacles to the excavator. Since the camera is unable to visually detect anything below ground, the position of the below ground roots have to be estimated based on the information above ground. A straightforward approach is to identify where the stump transitions from the stump stem

to the above ground roots, which can usually be seen from above ground almost as legs slithering down in the ground from the stump; see Figure 1.1. Since identifying areas close to stumps is a crucial part of the project, it is important to detect the correct area. To increase the accuracy of the segmentation and reduce the risk of incorrectly classifying other obstacles as stumps, a combination of object detection and semantic segmentation was used. First, the general position of a stump is predicted using a bounding box. The isolated area is then passed through a segmentation network that classifies each pixel individually. Additionally, this method reduces the risk that the roots of the stump are classified as sticks despite their similarities, since the object detection algorithm will be trained to find all parts of the stump. To minimize the risk of encountering roots below ground while still planting close to a stump, it is best to place the plant between two root legs.

Object Detection

Since the project has a time constraint on the algorithm, a fast model was needed for the object detection task. In the end, a YOLOv5 model was used for this task. YOLO has a good trade-off between inference time and performance, which was suitable for this project.

Semantic segmentation

Many different segmentation models were trained, using different backbones, loss functions, optimizers, and other parameters; however, they all had a U-Net-inspired architecture. The reason for using a U-Net architecture for the segmentation models was that this has shown good performance for many other segmentation tasks [23].

3.4 Gathering Images

The camera used in this project is called Omega and was developed by Arcure. It is a stereo camera made for tough and harsh environments [28]. This camera was chosen because the E-Beaver, to which the camera is attached, will be used outdoors in a forest environment. Therefore, the camera must withstand rain, minor shocks, and vibrations. The use of a stereo camera also made it possible to infer 3D coordinates of points in the images using stereo matching; see Section 2.1.2. This is necessary to provide the excavator with information about where the planting spots are located in the E-Beavers global coordinate system.

To use the Omega camera, an Application Programming Interface (API) was provided that enabled communication with, for example, a computer. The stereoscopic camera makes several different types of data available via the API, such as different camera parameters and different kinds of images. The programming interface is written in C++ and consists of several functions that were utilized to create a small program that saved all the necessary image data by pressing a button on the computer.

The camera has two fisheye lenses, and as mentioned previously, each call to the API can return several types of data. The data types used in this project were the following three: (1) an unmodified image produced by the fisheye lens without any manipulation, (2) a rectified version of the fisheye image and (3) a disparity map of every pixel in the rectified image, see Figure 3.2. By closely examining Figure 3.2, one can see that the quality of the unmodified fisheye image is higher than the slightly more blurred rectified version. The reason is that the data in the rectified image is the result after applying the interpolation to rectify the image, which in turn creates artificial data; see Section 2.1.3. Since the non-rectified image data is of higher quality, one could hypothesize that this data would yield better results for training a neural network to recognize objects. However, since the non-rectified image has a radial distortion, all of the objects will show less consistency in their geometry. For example, if a stump is close to the edge of the image, it will look different compared to how it would look in the middle of the image. With this in mind, the rectified images were used instead.

The images were supposed to look as similar as possible to the images that the camera will obtain when it is mounted on the E-beaver. To mimic the real situation as closely as possible, the same camera was used for all images. When taking the images, the camera also had the same geometrical conditions as it is planned to have on the robot. All images were acquired at locations where the robot will be used, that is, clear-felled areas on fields owned by Södra Skogsägarna. The range of the excavator is about 3 meters. Therefore, it is not necessary to capture an image that covers a range much larger than 3x3 meters, since the excavator will not reach further than that. However, some additional margin in each direction can be

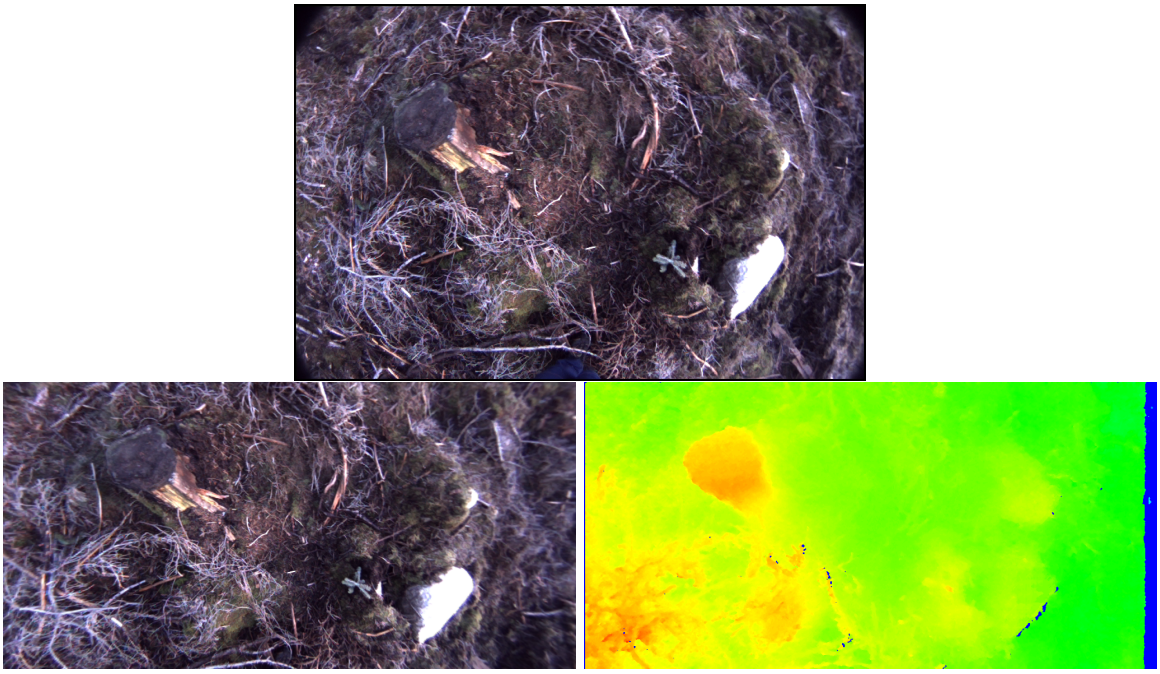


Figure 3.2: *The three data types obtained from the camera. At the top is the raw fisheye image, the lower left the rectified version of the same image, and to the right the disparity map of the rectified image.*

beneficial, since it will help analyze the area near the edge-cases of the excavators reach. The camera is angled so that the principal axis of the camera is parallel to the direction of the gravitational field. This is important for two reasons: First, since the cameras use fisheye lenses, the angle of view is very large. Because of this, a slight tilt of the camera causes the camera to suddenly capture points that are very far away. When this happens, the image is filled with unnecessary data, and the model has to handle irrelevant data points. When the covered area is too large, it can also cause problems with the exposure time of the camera and in turn yield an overexposed image. This happens because the probability that there are diverse light conditions in the image increases because points that are farther away from each other are less likely to be equally bright. The other important reason for having the principal axis parallel to the gravitational field is the depth information from the camera. The disparity data represent the distance along the principal axis; if the axis is parallel to the gravitational field, the disparity linearly relates to the altitude of the points in the image.

The images were collected during two different days. From this point forward, the data collected on the first day will be referred to as "subset 1", while the data from the second day will be referred to as "subset 2". Naturally, the union of these two subsets will be referred to as "subset 1+2", representing all the image data from both days. More information on the differences between the two sets is provided in 4.

3.5 Creating the Object Detection Data Set

As previously mentioned, two important parts of the model are object detection with bounding boxes and semantic segmentation. Therefore, after specifying the machine learning methods, the next step in building the data set was to annotate the data for these tasks. First, the data were annotated to train an object detection model. Annotation was done using Roboflow's built-in tools to draw bounding boxes around all items that should be detected by the object detection model; see Figure 3.3 [29]. By doing this, text files containing labels were generated for all images. Using these labels as ground truth, an object detection model can be trained to detect these objects.

When detecting rocks, the larger ones are more important to identify, as they pose immediate obstructions to the excavator. Small rocks, on the other hand, can often be moved by the excavator without complication.

Figure 3.3: *Example of an annotated image: Purple bounding boxes mark out stumps, pink bounding boxes mark out water, and yellow bounding boxes mark out stones.*

However, when annotating the images, the decision was made to annotate *all* rocks, regardless of their size. There are two reasons for this: Firstly, it can be difficult to create a model that generalizes the appearance of only large rocks. If only large rocks are annotated, the model would be punished during training if it predicted the location of a small rock. Secondly, since the object detection model is trained and inferred exclusively on RGB data, it is impossible for the model to derive the actual size of an object. For these two reasons, the object detection model was trained to detect all rocks of all different sizes. The smaller ones can then be filtered out in a later stage of the full algorithm, before selecting a planting spot.

After annotating all objects with bounding boxes, the images were resized by stretching them to a size of 640x640. The images were also augmented in a few different ways. First, the images were flipped both horizontally and vertically. Then the hue was changed using random values between -20 and +20 degrees; the purpose of this was to simulate warmer and colder days. The images were also augmented by manipulating the saturation to simulate a wider range of humidity in the scene. Lastly, the brightness was also changed to simulate brighter or darker days. The augmentations were carried out using Roboflow's built in tools [29].

3.6 Training the Object Detection Model

To evaluate how YOLOv5 performed depending on the data, three models based on the same YOLOv5 class were created and trained, using three different data sets. The first model was trained exclusively on subset 1, i.e. data from the cloudy day. The second model was trained on subset 2, i.e. data from the sunny day, and the third model was trained on the combined data set of both subset 1 and subset 2. Throughout all training sets, a pre-trained YOLOv5 XL model was used and trained for 30 epochs.

When training a model, the training script automatically saves two instances of the model weights: the weights from the last training epoch, as well as the weights that produced the lowest validation loss across all epochs. By always saving the weights that yield the lowest loss on the validation data, the risk of overfitting the final model is greatly reduced, since a backup of the best weights is saved. The approach was therefore to run enough epochs to be confident that a new, better set of weights would not be found by running additional epochs, i.e., when the loss values had stagnated, and then restore the weights from the epoch with the lowest validation loss.

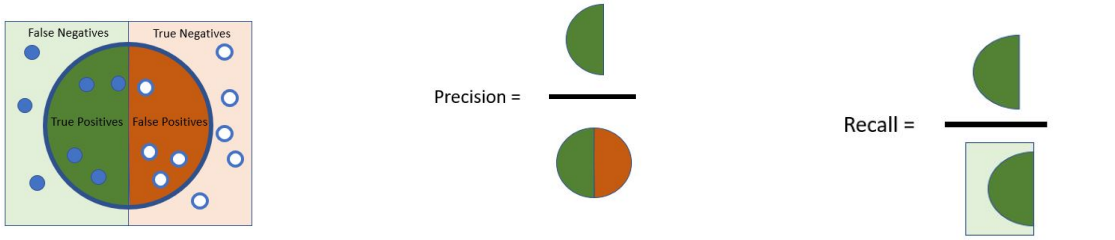
3.6.1 Loss Function

The YOLO loss function is a linear combination of three losses: a *box loss*, an *objectness loss* and a *class loss*. The box loss is used to ensure that the network learns to predict the correct coordinates of the bounding boxes. The purpose of objectness loss is to train the network to correctly estimate whether or not there is an object in

the proposed bounding box, where the target value is the IoU between the ground-truth bounding box and the predicted bounding box. Finally, the class loss is used to train the network to predict the correct class of the object inside the bounding box. Two important metrics that are used to evaluate object detection models are *recall* and *precision*, which will be briefly described below.

3.6.2 Precision and Recall

When talking about precision in the context of object detection, one usually refers to how many of the proposed bounding boxes are correct. This is defined as the ratio between the number of correct bounding boxes and the number of all proposed bounding boxes, as illustrated in Figure 3.4b. The quantity is described as $\frac{TP}{TP+FP}$, where TP denotes the number of true positives, and FP denotes the number of false positives; see Figure 3.4a. Recall, on the other hand, is a measure of how many objects were found, that is, how many ground truth boxes the model predicted, as illustrated in Figure 3.4c. The definition of recall can be written as $\frac{TP}{TP+FN}$, where FN denotes the number of false negatives; again visualized in Figure 3.4a.



(a) *The four outcomes of a prediction: True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN).*

(b) *The precision is measured by calculating the number of true positive predictions divided by the number of all the positive predictions.*

(c) *Recall is measured by taking the number of true positives divided by the number of all the ground truth positives.*

Figure 3.4: *Visualisation of the precision and recall metrics. Adaptation of original image in [30].*

3.7 Creating the Segmentation Data Set

As explained in 2.2.6, semantic segmentation is the task of classifying each individual pixel in an input image into a class. Both stumps and sticks are segmented using such a model. In practice, the two obstacles could be trained and predicted using the same semantic segmentation model. However, instead of combining the two obstacles into a single model, the two obstacles are trained on individual models. There are many reasons to use separate networks; as previously mentioned, the approach to segment stumps is two-fold, as a bounding box of its general location is first predicted using YOLOv5, before the isolated bounding box area is passed on to the segmentation network. On the contrary, the entire frame is used as input to segment sticks. The aim of isolating the location of stumps prior to segmenting the obstacle is to increase precision and decrease the risk of classifying root stems as sticks, despite their similar shapes. Therefore, the desired output of each semantic segmentation model is a WxHx1 mask, where W and H denote the width and height of the input image. Each pixel in the output mask acts as a binary response variable with only two classes, namely (1) the object at hand for the specific model, i.e., stump or stick, and (2) background/other. To train such a model, a substantial amount of training data needs to be created that matches the desired output for each training input.

Training data for the stick segmentation model was created using the AC295-final-project-JWI library [31]. This library offered tools that made it easy to trace around the edges of objects, thereby classifying the correct pixels; see Figure 3.5. This library was used on large pieces of wood, such as felled trees and large sticks. After annotating an image, the library creates a segmentation mask with the same dimensions, where each pixel value in the mask is either 0, representing the background, or 1, representing a large stick/felled tree. A U-Net based semantic segmentation model can then be trained on the data, where the resized RGB image is used as input and the created mask is used as ground truth.

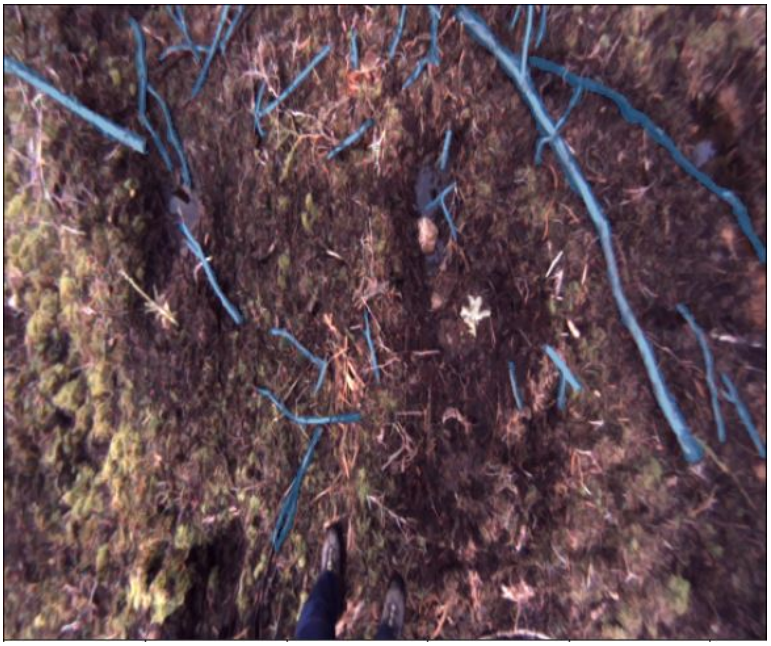


Figure 3.5: *Example of an annotated image for segmentation of larger pieces of wood using the AC295 library. The pixels that are marked with blue illustrate large sticks. The acquired mask for this image will have the same x-y-shape as the image, where the blue parts will have a mask value of 1 (corresponding to class index of large sticks) while the rest of the values will be 0 (corresponding to background).*

To create training data for the stump segmentation model, a different approach was used. Since the desired input is an isolated stump and not the entire image, the images needed to be cropped before creating a mask. Additionally, since the main part of a stump is often elevated from the ground, the disparity map generally holds additional information on its location. Instead of using only the RGB image as input for the segmentation model, the U-Net architecture can be modified to accommodate four-channel (RGB-D) input with the intent of the disparity supplying additional information. Due to this, the AC295-final-project-JWI library was deemed insufficient to create training data, since it only considers full RGB images. Instead, a custom annotation graphical user interface (GUI) tool was developed to create the data. The custom tool works in the following way:

- Given a full RGB image, and the corresponding RGB representation of the disparity map, select a region of interest, i.e. a bounding box of the object; see Figure 3.6.
- Save the isolated RGB area and isolate the same area in the RGB disparity map.
- Convert the RGB disparity map pixels to single integers using the provided Look Up Table (LUT) from the cameras user manual.
- Resize the disparity integers, as well as the isolated RGB area, to a pre-set dimension of WxH to fit model requirements.
- Draw a free-hand marking on the selected area; see Figure 3.7.
- Create a $WxHx1$ segmentation mask of the drawn area, where each pixel value in the mask is either 0, denoting the background, or 1 where the user has drawn, denoting stump.
- Concatenate the selected RGB area and the disparity integers into a single $WxHx4$ array.

The resulting $WxHx4$ RGBD array can then be used as input for a segmentation model, while the created segmentation mask is used as the ground truth.

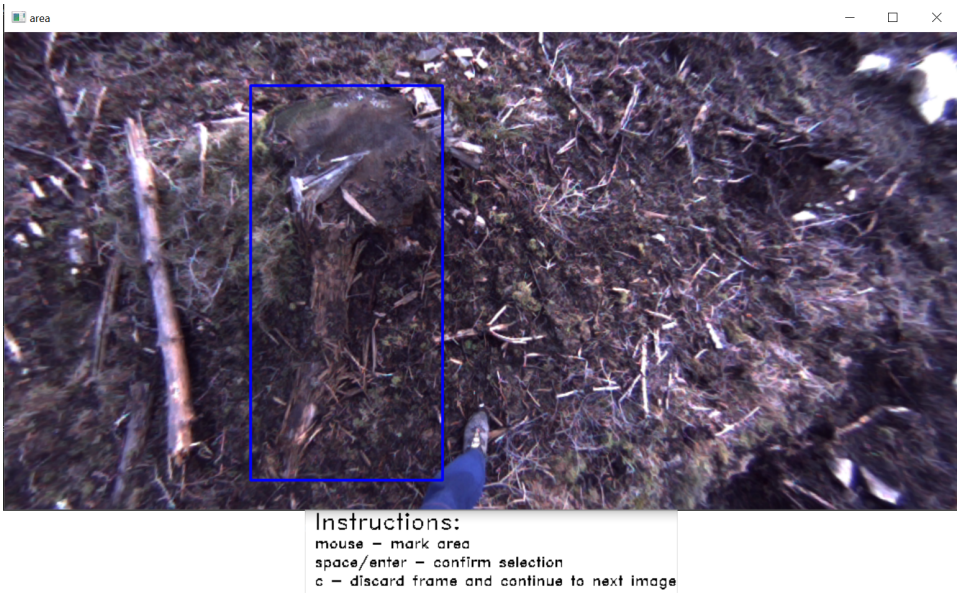
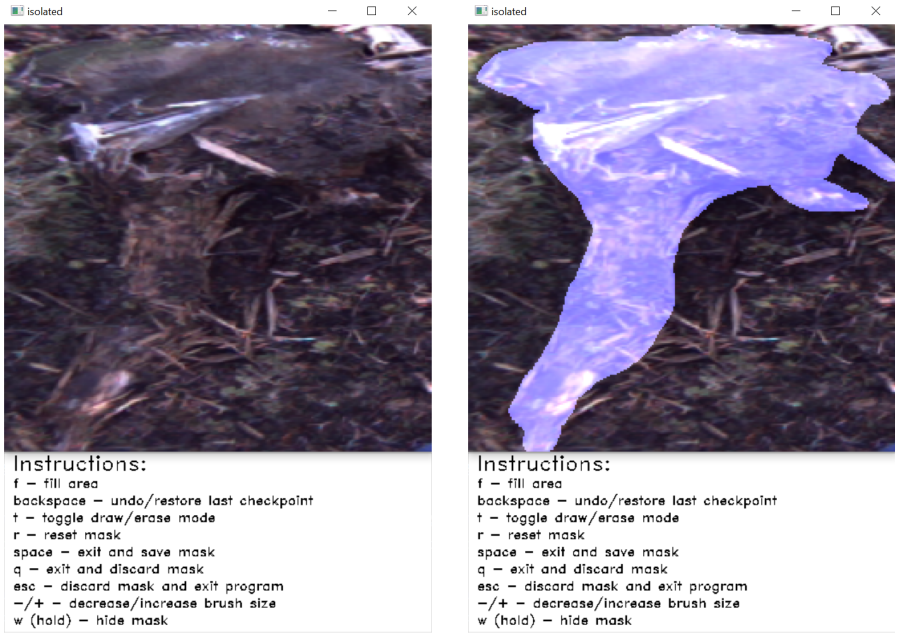


Figure 3.6: *Example of how the custom GUI is used to isolate certain areas in the data. All images in the data set are displayed in series. If the user manually selects a region of interest, i.e. a stump, the selected area will be reshaped to fit the model requirements and then passed on to the mask creating tool seen in Figure 3.7. A list of instructions can be seen at the bottom of the image, displaying all possible actions.*



(a) *The previously selected area is reshaped to fit model requirements.* (b) *The same image with a complete mask.*

Figure 3.7: *Example of how the custom GUI is used to create a mask.* ***(a)*** *shows the selected and resized area before drawing a mask.* ***(b)*** *displays the same image after the user has drawn the mask. A list of instructions can be seen at the bottom of the two images, displaying all possible actions the user can perform to help draw the mask.*

3.8 Training the Segmentation Models

As mentioned above, two different segmentation models were developed: one was developed to detect large sticks and fallen trees and one to detect stumps. The general structure of the two segmentation models was similar: both models were trained using a U-Net architecture, and Inceptionv3 was used as backbone for the two models. The weights of the backbones of the two models were both pre-trained on the ImageNet dataset - see chapter 2.2.6 for more information. The networks were implemented using the Segmentation Models library [32], a Keras-based framework that allows users to easily initialize different semantic segmentation networks, such as U-Net. All images were resized to a shape of 256x256. Before training both models, the training and validation data was augmented to increase the number of training samples. The augmentations included rotating the images, increasing or decreasing their brightness, and flipping the images. Since the stick segmentation model uses standard RGB images as input, augmentation for that data set was straightforward. However, since the stump segmentation uses RGB-D as input, the augmentation had to be done with caution, as it was important to augment the correct parts of the data; for example, rotating the input affects the entire RGB-D array as well as the segmentation mask, while changing brightness only affects the RGB part of the input, and the disparity and segmentation mask is left unchanged. Although the general structure of the two models was similar, there was slight variation in the training process for the two models, which will be discussed below.

3.8.1 Stick Segmentation Model

The stick segmentation model uses standard RGB images as input. Since the selected backbone was pre-trained on RGB images from the ImageNet data set, the loaded U-Net model could be used without alteration while still making use of the pre-trained weights from the backbone. A common choice of loss function for semantic segmentation models is binary cross entropy, which is a measure of the difference between two probability distributions for a given random variable or set of events. Mathematically, binary cross-entropy is defined as:

$$L_{BCE}(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right). \tag{3.1}$$

where $\hat{y}$ denotes the predicted pixel value and y denotes the ground truth value. Binary cross-entropy works best for data sets with equal distribution among classes [33]. However, when the images were annotated, it became apparent that the sticks in the data set are thin and sparsely located. As a result, most of the segmentation masks contained zeros, representing the background. According to the ground truth segmentation masks created for the training set, an average of approximately 5.4% of the pixels were classified as sticks, while 94.6% were classified as background. This makes the data set very imbalanced and, in practice, if the network were to always predict all pixels to be background, it would still have an accuracy of 94.6%. To combat this, dice loss was used instead of the standard binary cross-entropy loss. Dice loss is defined as:

$$DL(y, \hat{y}) = 1 - \frac{2y\hat{y} + \gamma}{y + \hat{y} + \gamma} \tag{3.2}$$

When applied to the stick segmentation model, the numerator corresponds to the sum of correctly predicted stick pixels, and the denominator corresponds to the sum of total stick pixels of both the prediction and the ground truth. γ is added to both the numerator and the denominator to ensure that the function is not undefined in edge cases, such as when $y = \hat{y} = 0$. When the stick segmentation model was trained, γ was set to 10^{-6}. Dice loss generally works better than binary cross-entropy for class-unbalanced data sets, which is why it was used when training the stick segmentation model [33]. All hyperparameters for training were selected by trial and error: The learning rate was initially set to $lr_0 = 0.001$, and after four epochs, the learning rate gradually increased to $lr_n = lr_{n-1} \cdot e^{-0.1}$. To avoid overfitting the model, an early stopping callback was implemented based on the validation loss, with a patience set to 7. Lastly, Adam was used as an optimizer.

3.8.2 Stump Segmentation Model

Since the Inceptionv3 backbone is trained on standard RGB data, its pre-trained weights cannot be applied to a four-channel RGB-D input immediately. Instead, to still make use of the pre-trained weights, the RGB-D input has to be condensed to a three-dimensional feature space. This was done by applying a single convolutional layer across the input prior to the standard U-Net network, using 3 filters of size 1x1. The three channel output

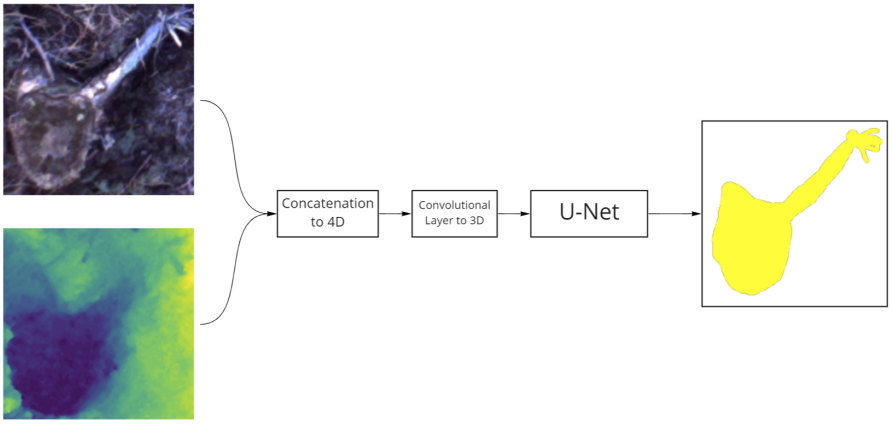


Figure 3.8: *Schematic overview of the stump segmentation network. First, the isolated RGB image is concatenated with its corresponding disparity array. The resulting 4D feature space is then reduced to 3D using a single convolutional layer. Next, the 3D output is used as input for a U-Net model, in hopes of predicting a segmentation mask as illustrated to the right.*

can then be passed on to the U-Net network and make use of the pre-trained backbone. The general structure can be seen in Figure 3.8.

The data was normalized before being applied to the network. The reason for this was to increase the speed of training, as the training process for each feature starts on the same scale [34]. Since all values from the RGB part of the input are within the range of $[0, 255]$, these arrays were simply divided by 255. Regarding the disparity array, the integer values can hold any value between 0 and 2049 as determined by the camera. To better identify local maxima in the disparity, the disparity matrix was normalized independently for each input, first subtracting the minimum value across the entire matrix and then dividing by the maximum value, giving a range of $[0, 1]$ for each input. Because the input area is a cropped stump rather than a full scene, the data set is more balanced; looking at the ground truth segmentation masks from the training set, approximately 45% of the pixels represent the stump class, while 55% of the pixels denote background. As a result, binary cross-entropy loss was used exclusively. The same learning rate schedule and optimizer as used to train the stick segmentation model was also used for this model, and early stoppage was implemented based on the validation loss, with patience set to 5. As a comparative measure, an additional model was also trained using only the RGB input, ignoring the disparity values and removing the first convolutional layer, but this yielded less accurate results. More on this in 5.2.2.

3.9 Main Algorithm

In this section, the main algorithm for selecting planting spots based on the information provided by all individual models will be discussed.

3.9.1 Fusing the Models

With all individual segmentation and object detection models trained, all models were assembled into a complete model to analyze the entire frame and combine the results of all models. To increase the simplicity of the project, all models and their corresponding weights were converted to the Onnx format, an open format built to represent machine learning models [35]. This allowed each model and its weights to be reduced to a single file, as well as the ability to run inference on the different models in a similar way. A schematic overview of the combined model is shown in Figure 3.9. Briefly explained, the combined model performs the following steps:

- Load an image (denoted I) and a corresponding RGB representation of the disparity map (denoted D).
- Copy and resize I to the required input shape of 256x256, send it to the stick segmentation network, and receive a segmentation mask M_{sticks}
- Resize M_{sticks} back to the original image shape and add it to a complete segmentation mask $\mathbf{M}$.

- Copy and resize I to the required input shape of 640x640, send it to the object detection network and receive a list of bounding boxes B
- Loop through B. If the object in the box is a stone or a pool of water, resize the box to match the original image shape and add it to $\mathbf{M}$. If it is a stump:
  - Copy and crop I and D with respect to the predicted bounding box coordinates, so that they isolate the predicted stump
  - Convert the pixels of D to single integers and concatenate the result with I to create a 4D array
  - Resize the 4D array to the required input shape of 256x256, send it to the stump segmentation network, and receive a segmentation mask M_{stump}
  - Resize M_{stump} to match the original image shape and add it to the full segmentation mask $\mathbf{M}$.
- The output of all models is now stored in $\mathbf{M}$. The results can be visualized by overlaying $\mathbf{M}$ on the initial input image I.

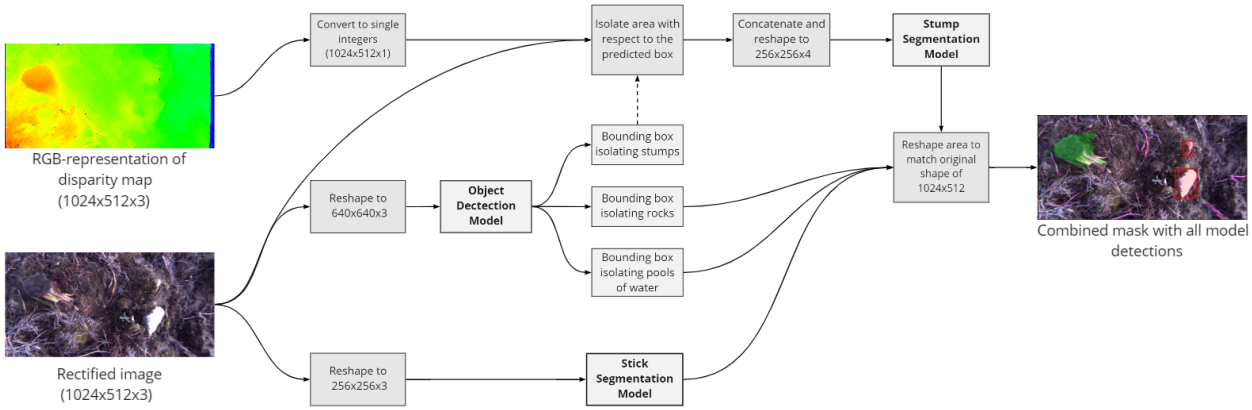


Figure 3.9: *Schematic overview of how the different models are combined to create a single output.*

3.9.2 Pre-processing the Combined Output

After combining all individual models into a single output where all obstacles are identified, the next step was to pre-process the combined output. The pre-processing involved removing obstacles that are small enough to not hinder the excavator, such as small rocks and sticks. While the real-world coordinates of the frame could be derived to estimate the exact size of each object, the pre-processing was simplified to a 2 dimensional feature space to reduce complexity and increase inference time. Since each image was intended to capture an area of approximately 3x3 meters and the image had a shape of 1024x512 pixels, it was estimated that each pixel corresponded to an approximate size of $0.171cm^2$. Although this reduction is less accurate than deriving the exact size, it was deemed sufficient to evaluate the general proof of concept.

For rocks, the threshold area was set to 1000 pixels, or approximately $172cm^2$, roughly corresponding to the size of a fist. Since YOLOv5 returns the width and height of each bounding box, the area could easily be calculated to see if it was large enough to obstruct the location: If a bounding box was smaller than this, it was discarded from the final output.

The size of each stick was estimated by analyzing each cluster of pixels classified as sticks. This threshold value was set to 600 pixels, approximately $102.6cm^2$, which means that all clusters smaller than this were removed from the final output.

As for re-processing the stumps, pixels that belonged to the actual stump, received from the segmentation model, remained as invalid planting areas. However, the bounding boxes that enclose the general region of a stump, but whose pixels were not classified by the segmentation model, were classified as good areas as they

are located near the stump; see Figure 3.10. Lastly, the pools of water remained, regardless of their size, since digging a hole next to a pool of water could also cause the planting hole to fill up with water.

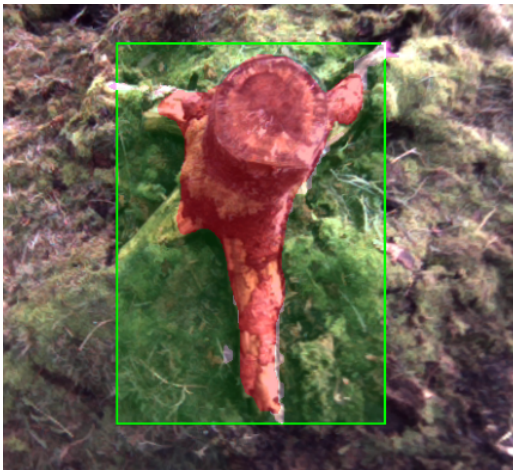


Figure 3.10: *Illustration of how stumps are treated when selecting optimal planting spots. Red areas, denoting the output of the stump segmentation model, are ruled out as they hinder the excavator. Green areas, denoting the output of the object detection model, are good as they are close to the stump while still avoiding its exact location.*

3.9.3 Selecting Optimal Planting Spots

Once all small objects had been discarded and the output consisted only of obstacles of substantial size, a list of valid planting spots was created. This was done by projecting the size of the excavator bucket onto the ground and analyzing the pixels it occupies. The part of the excavator bucket that is applied to the ground has a size of 0.80x0.45 meters. For this part, the pixels were converted to actual meters to better estimate the size. First, the disparity integers were converted to meters using

$$Z = \frac{4 \cdot B \cdot f}{D} \tag{3.3}$$

where D denotes the integers of disparity, f is the focal length of the camera in pixels, and B is the baseline of the camera in meters. Each pixel (u, v) was then converted to (x, y, z) using the following equations:

$$\begin{aligned} X &= (u - u_0) \cdot Z/f \\ Y &= (v - v_0) \cdot Z/f \end{aligned} \tag{3.4}$$

where u_0 and v_0 are the optical center positions on the depth map in pixels. The valid planting points were then identified by analyzing all areas where the excavator bucket could fit without interfering with any obstacles.

For each identified planting area that did not interfere with any obstacles, two ranking metrics were saved: The first and primary ranking metric was the number of pixels within the excavator bucket area that were classified as *"close to a stump"*, that is, the green pixels in Figure 3.10. The secondary metric was the average height of the pixels covered by the excavator bucket. These metrics allowed for a comparison of the valid planting spots with respect to the steps mentioned in 3.1.

Ultimately, two planting spots were selected: the first selected spot was simply the one with the highest ranking score. The second selected spot was the one with the second best ranking score while still being at least 1 meters away from the first planting spot, so as not to plant the saplings too close to each other.

4 Final Data Set

The long-term goal is for the planting spot selection model to work in all clear-felled areas owned by Södra under all the weather conditions that will occur during the planting period, which is during the warmer half of the year. Since the main obstacles in these locations are naturally generated, they can differ substantially in appearance, as illustrated in Figure 4.1. This makes the task of detecting all the objects difficult, and detecting everything, including the most irregular looking objects, is not a realistic goal. Instead, this project has focused on developing a proof-of-concept to detect recurring obstacles and, in turn, favorable planting spots. Therefore, when collecting the data, the main focus has been to capture objects that have a more common appearance.

Figure 4.1: *Example images of different stumps from the same clear-felled area. As seen, they differ substantially in both shape and size.*

In addition to the proof of concept, this project also aims to explore and compare different methods that could solve the task of identifying a good planting spot. The first step was to collect enough data to be to compare different approaches. By collecting data at a single location on one day, a limited data set could be created. By splitting the data and isolating certain objects so that they appear only in the test set, it could be used to evaluate a model. Evaluating the models performance on the test set would still show whether the model is able to generalize the obstacles at all, since the objects in the test set have never been encountered by the model. But, by having data from at least two different days and clear-felled areas, the possibility of seeing some generalization between different weather conditions also exists.

The data set consists of image data, where the images were collected on two different days from two different clear-felled areas, both owned by Södra Skogsägarna. The conditions for capturing images with the Omega Camera differed substantially between the two days. As can be seen in Figure 4.2, the images captured on the first day were more consistent in quality from an exposure perspective, compared to the second day, which can be seen in Figure 4.3. Since these two subsets differ in appearance, they provide a good setting to evaluate how well a model generalizes obstacles. By training a model on one subset and evaluating it on the other, it can be concluded whether the training data is sufficient to detect obstacles in other environments.

4.1 Subset 1

The images for this part of the data set were collected on February 7th. The weather was cloudy, resulting in consistent lighting conditions in the scene. As can be seen in Figure 4.2, the brightness is relatively consistent throughout the images, both between the images and also in each image. This led to well-captured images without any poorly exposed areas.

4.1.1 Object Detection Statistics

The following list shows the most important statistics of the object detection data for subset 1:

- 2015 images
- 6528 bounding box annotations in total
- 4387 rock bounding boxes annotated
- 1492 stump bounding boxes annotated
- 649 water bounding boxes annotated

Figure 4.2: *Three example images from subset 1. As can be seen, even though the images are taken at different scenes and the colors of the objects differ, the brightness of the images are very consistent. Since the weather was cloudy, the same exposure time was suitable for all images.*

- 113 test images

In the first subset, rocks are the most common obstacle, since the clear-felled area where the images were collected was very hilly with a lot of rocks. It was a relatively humid day in February, with recent rainy days, which meant that there were multiple pools of water in the area. Most of the pools of water were small, which was good, as the smaller pools are more relevant within the scope of this thesis: The larger pools will, ideally, be avoided directly by the E-beaver when navigating forward, since it is intended to autonomously avoid large obstacles where it can otherwise get stuck.

4.1.2 Segmentation Statistics

The following list shows the most important statistics of the semantic segmentation data for subset 1:

- 151 total images for stumps.
- 44 test images for stumps.
- 78 images for sticks.
- 27 test images for sticks.

Figure 4.3: *Three example images from subset 2. As seen, the lighting conditions are very inconsistent compared to the images in subset 1. The varying lighting conditions led to many images being partially over- or underexposed, resulting in a loss of detail.*

4.2 Subset 2

The images in this part of the data set were collected on April 22nd. On this day, the weather was sunny, which resulted in objects casting more shadows compared to the previous day that data was collected. As a result, many images were overexposed in some parts, especially if the image consisted of a shadow and a bright part at the same time. This usually led to the image being either overexposed in the bright part or underexposed in the dark part.

4.2.1 Object Detection Statistics

The following list shows the most important statistics of the object detection data for subset 2:

- 512 images
- 1404 bounding box annotations in total
- 861 rock bounding boxes annotated
- 512 stump bounding boxes annotated
- 31 water bounding boxes annotated
- 29 test images

On this day, it had been sunny for a few days. As a result, there was less water in the second clear-felled area, compared to the first area visited in February. Although the small pools of water had evaporated, there was still some water in the larger holes. Therefore, the pools of water look very different in this subset compared to subset 1, and as seen, only 31 images contained pools of water. In addition, there was a layer of green algae on the surface of the water in almost all the pools of water in this subset, which further distinguishes the appearance of the water between the subsets.

4.2.2 Segmentation Statistics

The following list shows the most important statistics of the semantic segmentation data for subset 2:

- 74 total images for stumps.
- 23 test images for stumps.
- 47 total images for sticks.
- 22 test images for sticks

5 Results and Discussion

In this chapter, the training and evaluation results of the deep learning models and the final plant spot location algorithm will be discussed.

5.1 Training Results of YOLOv5

As mentioned in 3.6, the YOLOv5 architecture was trained on a total of three different data sets. First, it trained exclusively on subset 1, then exclusively on subset 2, and lastly, the union of the two subsets, subset 1+2. Each model was trained for a maximum of 30 epochs, where the final model restored the best weights achieved throughout these epochs.

Training on Subset 1

All the training results of the model trained on subset 1 can be seen in Figure 5.1. Naturally, all the different loss metrics evaluated on the training data decreased as the model trained for more epochs. However, the validation loss stagnated after around 10 epochs for the box loss and the class loss, while the objectness loss increased almost linearly after around 10 epochs. Precision and recall metrics, both desirable to be as large as possible, also stagnated after around 10 epochs with no significant improvement. The same applies to the mAP values. Since all validation metrics stopped improving after 10 epochs, it was assumed that a maximum of 30 epochs was more than enough to train the model. Again, the weights from the epoch that produced the best validation metrics were restored and saved.

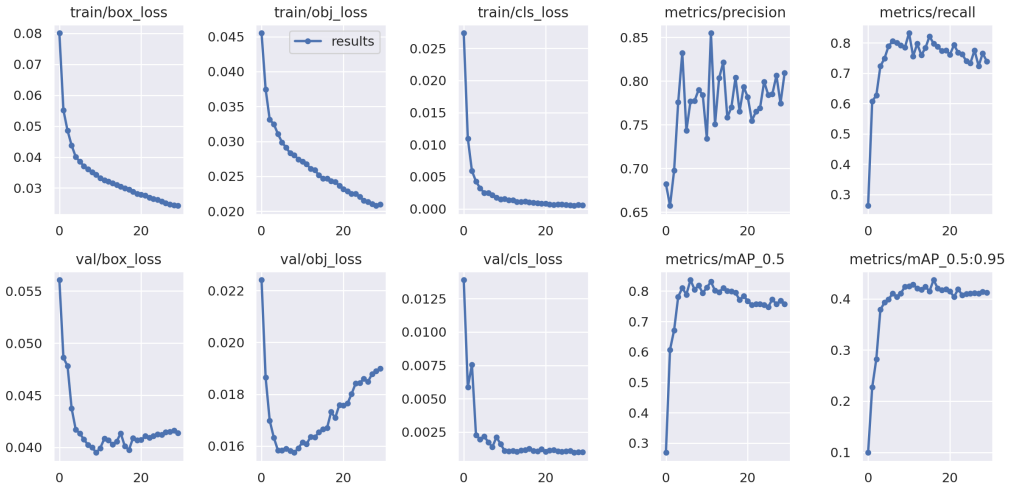


Figure 5.1: *Training results from the model trained on data from subset 1. Both the box loss and the object loss stopped decreasing after around 5-10 epochs, and eventually started to increase. The class loss was almost constant after this point, with a slight increase.*

Training on Subset 2

The training results from training the model exclusively on subset 2 can be seen in Figure 5.2. Again, the training loss steadily decreased, while the validation loss started to stagnate after around 10 epochs. The only loss that continued to decrease for longer was the class loss, which seemed to stagnate after around 20 epochs. The precision and recall metrics were much more volatile during this training session, compared to the one in which data from subset 1 was used. However, their values did not improve significantly after 15 epochs, although they continued to fluctuate. The mAP values increased rapidly throughout the first 10 training epochs, before eventually reaching somewhat constant values. Looking at the different plots, one could argue that this model could have benefited from training for a few additional epochs. Because of this, an extended training was

initialized, consisting of an additional 10 epochs, but without further improvements of the validation metrics. Therefore, it was assumed that these 30 epochs were sufficient in the search for the best weights.

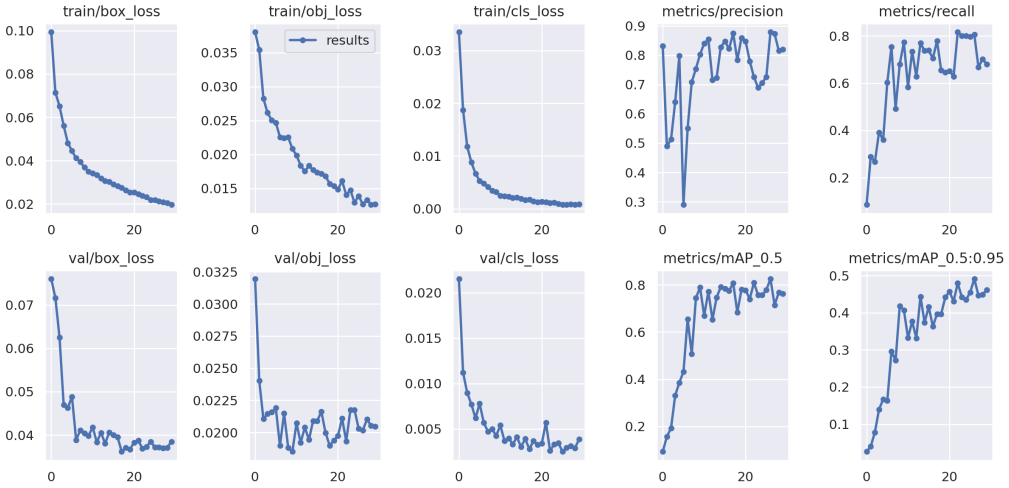


Figure 5.2: *Training results from the model trained on data from subset 2. The validation loss started to stagnate for all three loss metrics, and the mAP did not increase much either for the last couple of epochs.*

Training on Subset 1+2

The results after training the model on subset 1+2 can be seen in Figure 5.3. The results of this training session were similar to the one in which the model was trained exclusively on data from subset 1. The losses had the same behavior, where the only difference was that the objectness loss started to increase even faster than in the first training session. Both mAP curves clearly stagnated after approximately 10 epochs. At first glance, the precision curve seems to fluctuate a lot. However, the scale of the y-axis is zoomed in very close at 0.8, which implies that the metric in fact stayed relatively constant for the last few epochs. The recall curve clearly stagnated as well after around 10 epochs.

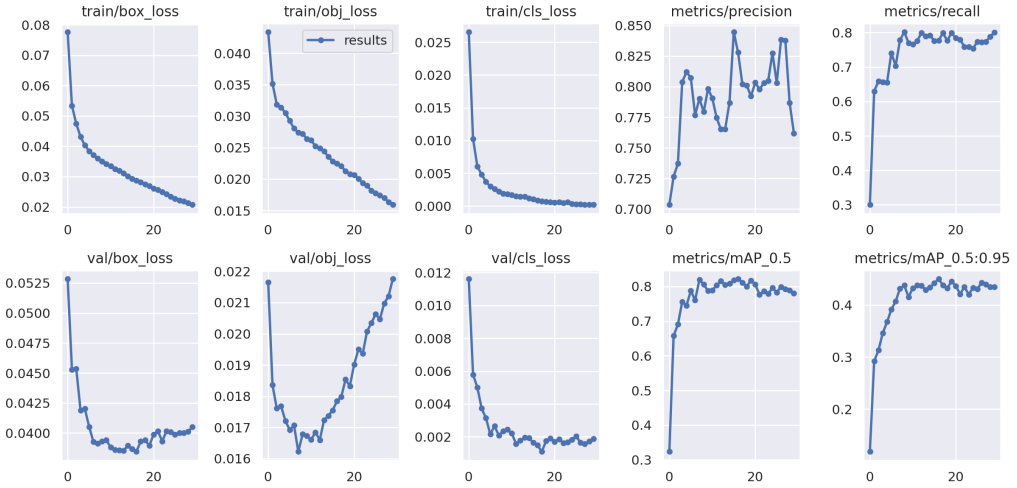


Figure 5.3: *Training results from the model trained on the full data set.*

It is quite clear that 30 epochs were more than enough to train each model, since all the validation losses stopped improving after around 10 epochs for every run. To evaluate and compare the different models, three different test sets were constructed: one test set consisting only of images from day 1, one consisting only of

images from day 2 and the last test set that contained images from both days. These images were not present in any of the training or validation sets. Furthermore, none of the objects included in any of the test set images was present in any of the images in the training or validation set. Instead, the test sets consisted of unique images, with unique stumps, rocks, and pools of water.

5.1.1 Evaluating Object Detection Models

The object detection models were evaluated and compared numerically, based on their recall and precision values across each test set. Before presenting the results, a brief discussion will be given on how these metrics can be interpreted and why they were chosen. As explained in Section 3.6.2, recall and precision are two natural metrics to use when evaluating an object detection model. Together, these metrics show how many of the objects in the images the model is able to find and how many of the proposed bounding boxes are correct.

The recall values, which will be shown below, indicate that many objects are not detected by the models. However, when looking at the images made from the inference shown in Section 5.1.2, it appears that the model detects most objects. There are two important things to take into consideration here, the first is regarding the rocks. As mentioned in Section 3.5, all rocks are annotated, regardless of their size. However, models often only locate large rocks and fail to locate smaller ones. Although this results in a low recall value, it does not affect the end result of the model, since these small rocks are not immediate obstructions for the excavator.

When looking at the precision and recall metrics with regard to stumps, it is important to consider how a true positive is defined. For a prediction to be considered correct, it needs to predict the correct class and predict a box with a large enough IoU computed against the ground-truth box. When annotating the ground truth boxes, the entire stump was included in the box, including all of its visible roots. This made some of the boxes very large. Sometimes the model predicts a smaller box that only contains the top of the stump, which for stumps with large visible roots sometimes ends up being less than half the size of the complete bounding box. Consequently, this resulted in a very low IoU score, even though the model detected the stump. It is important that the model captures as much of the roots as possible of the stumps so that the final algorithm can segment these roots and select a planting spot close to the stump. However, it is still interesting to see whether the model detects the stumps at all, even if it just detects the top of the stumps. Therefore, the performance of each model was evaluated on a higher IoU threshold and a lower IoU threshold, changing the threshold for a prediction to count as a true positive.

When comparing the different models, both prediction and recall will be considered. Both of these metrics should, ideally, be as high as possible. While increasing the IoU threshold generally results in larger recall and precision values, it does so by allowing less accurate bounding boxes to be considered correct. In the end, the model with the best trade-off between an IoU-threshold as large as possible, and recall/precision as large as possible, is best.

Model Trained on Subset 1

In tables 5.1 and 5.2, the recall and precision results of the evaluation of the trained model on the different test sets can be found.

Recall on the test sets for the model trained on subset 1				
Test Data	IoU-thres	Rock	Stump	Water
Subset 1	0.45	61%	49%	100%
Subset 2	0.45	29%	19%	0%
Subset 1+2	0.45	59%	40%	30%
Subset 1	0.20	67%	77%	100%
Subset 2	0.20	29%	23%	0%
Subset 1+2	0.20	65%	60%	30%

Table 5.1: Recall values for the object detection model trained exclusively on subset 1.

When using an IoU threshold of 0.45, the model is only able to identify 49% of the stumps from the same day and about a fifth of the stumps from the other day. However, by lowering the IoU threshold to 0.2, the number of stumps detected on the same day increases to 77%. As previously described, this indicates that the model probably finds most stumps, but that it is unable to generate well-fitted bounding boxes around them. As an example such a bounding box can be seen in Figure 5.4a. The rocks do not appear to be affected as much by the IoU threshold, with only a slight increase in recall and precision. The model finds all the pools of water in the test set from subset 1, but fails to detect any pools of water from subset 2.

Precision on the test sets for the model trained on subset 1				
Test Data	IoU-thres	Rock	Stump	Water
Subset 1	0.45	53%	30%	100%
Subset 2	0.45	38%	86%	0%
Subset 1+2	0.45	52%	43%	0%
Subset 1	0.20	62%	50%	100%
Subset 2	0.20	45%	84%	0%
Subset 1+2	0.20	62%	62%	0%

Table 5.2: Precision values for the object detection model trained exclusively on subset 1.

The model manages to find all the pools of water from the same day but does not detect every pool from the second day. In general, the model performs best overall when evaluated on images from the same day, with the major exception being the precision of stumps from subset 2. However, since the recall is very low on that subset, a possible conclusion could be that the model struggles to recognize stumps from this data set, but when it does make a prediction, it is usually correct.

Model Trained on Subset 2

In tables 5.3 and 5.4, the recall and precision results of the evaluation of the model that were trained on subset 2 can be seen.

Recall on the test sets for the model trained on subset 2				
Test Data	IoU-thres	Rock	Stump	Water
Subset 1	0.45	42%	43%	0%
Subset 2	0.45	50%	90%	0%
Subset 1+2	0.45	42%	57%	0%
Subset 1	0.20	48%	74%	0%
Subset 2	0.20	50%	90%	0%
Subset 1+2	0.20	48%	79%	0%

Table 5.3: Recall values for the object detection model trained exclusively on subset 2.

This model manages to find most of the stumps in the test set from subset 2, with good fitting bounding boxes. The 74% recall on stumps with an IoU threshold of 0.2 indicates that it also finds many stumps from Subset 1, but the lower recall of 43% on stumps when the IoU threshold is set to 0.45 indicates that the bounding boxes are not well-fitted. The model misses a lot of rocks for both IoU thresholds, and it cannot find any pools of water from the test sets at all.

Model Trained on Subset 1+2

In tables 5.5 and 5.6, the precision and recall results from validating on the test sets can be seen for the model trained on both subsets.

Precision on the test sets for the model trained on subset 2				
Test Data	IoU-thres	Rock	Stump	Water
Subset 1	0.45	70%	50%	0%
Subset 2	0.45	47%	87%	0%
Subset 1+2	0.45	70%	55%	0%
Subset 1	0.20	75%	83%	0%
Subset 2	0.20	45%	100%	0%
Subset 1+2	0.20	74%	86%	0%

Table 5.4: Precision values for the object detection model trained exclusively on subset 2.

Recall on the test sets for the model trained on both subsets				
Test Data	IoU-thres	Rock	Stump	Water
Subset 1	0.45	64%	73%	100%
Subset 2	0.45	58%	81%	0%
Subset 1+2	0.45	64%	75%	30%
Subset 1	0.20	70%	84%	100%
Subset 2	0.20	62%	84%	0%
Subset 1+2	0.20	70%	84%	30%

Table 5.5: Recall values for the object detection model trained on subset 1+2.

This model has a high recall on stumps in all test sets. The recall is not significantly smaller when using an IoU threshold of 0.45 compared to an IoU threshold of 0.2, which indicates that this model is better than the other two models at finding well-fit bounding boxes. This is also the model with the highest recall value on rocks across all test sets. The recall values for water are the same as for the model trained on subset 1.

Precision on the test sets for the model trained on both subsets				
Test data	IoU-thres	Rock	Stump	Water
Subset 1	0.45	68%	59%	81%
Subset 2	0.45	45%	93%	0%
Subset 1+2	0.45	66%	70%	61%
Subset 1	0.20	78%	70%	80%
Subset 2	0.20	48%	99%	0%
Subset 1+2	0.20	75%	79%	61%

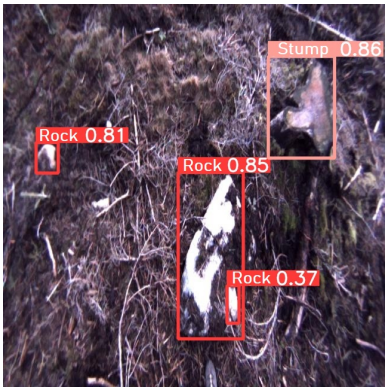
Table 5.6: Precision alues for the object detection model trained on subset 1+2.

An interesting thing to analyze is whether the model trained on subset 1+2 outperforms models trained exclusively on data from one day, when evaluated on the test set for that day. In other words, if the training images from both sets are similar enough to provide additional information for the overall detection system. For rocks, the model trained on subset 1+2 outperformed both other models. This shows that by adding more images to the training set, even if the images are taken in other locations and under different lighting conditions, the model's ability to detect rocks will increase. The same argument can be applied for most stumps: When evaluated on test data from day 1, the model trained on subset 1+2 outperforms the model that was trained exclusively on images from day 1. Again, this shows that adding more data generally increases performance. The only exception is stumps from day 2: The model trained exclusively on data from that day performs better than the one trained on data from both days. Lastly, when it comes to water, the model trained on both subsets performs the same as the model trained on subset 1. This indicates that the number of pools of water in subset 2 is insufficient to contribute to increased learning. In summary, despite minor discrepancies, additional training images seem to generally increase performance, regardless of where or when the images were collected.

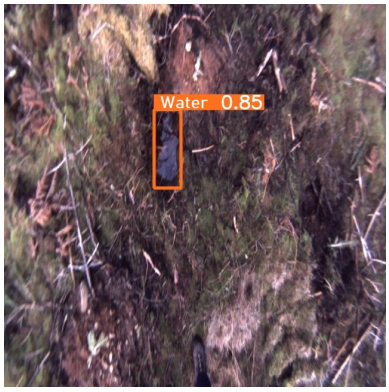
Another thing to note after looking at these tables is that stumps are more affected by lowering the IoU threshold than any other obstacle. By lowering the IoU threshold, rocks have almost the same recall and precision scores across all models, while both metrics increase for stumps. This implies that stumps, compared to other obstacles, are more difficult to completely enclose with the predicted bounding boxes. Instead, for many predictions, only the stump stem is identified, while the root legs are undetected.

5.1.2 Detection Samples from YOLOv5

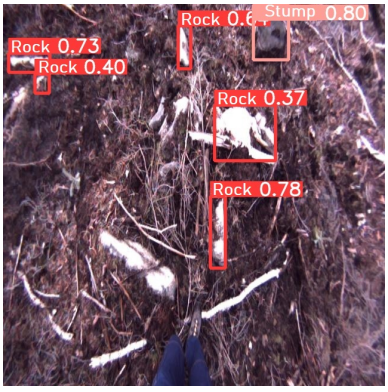
The results in the previous section point to the fact that the model increased its ability to detect objects from both subsets when the model was trained on images from both days. Many quantitative measurements of the performance of the models were presented. However, it can be difficult to interpret how these metrics actually relate to the end product. In this chapter, some detection samples are presented to visualize how the model performs. In Figure 5.4, the resulting bounding boxes proposed by the best object detection model, i.e. the model that was trained on subset 1+2, are presented for four different test images. As mentioned above, it



(a) *Inference on an image with a stump and some small rocks. The model detect the stump and all substantial rocks.*



(b) *An image of a small pool of water and a very small rock. The model detects the water but fails to detect the negligibly small rock.*



(c) *An image of rocks and a stump. The model fails to detect a large rock in the lower left part of the image.*

(d) *Inference on an image with a stump and a rock. The model successfully detect both objects.*

Figure 5.4: *Inference from the model that has been trained on both subsets.*

is also important to consider the actual output of the models and not only to look blindly at the evaluation metrics. Looking at Figure 5.4a, there are two large visible rocks that the model successfully detects. However, there are also at least two smaller stones in the same image that can be more difficult to see and that are not detected by the model. However, these rocks are probably small enough to not obstruct the excavator. Therefore, although the recall value for rocks is not 100% in this image, all stones that are important to avoid are still detected. Another thing to note is that in the same image the model predicts part of the largest rock

to be a separate stone. This does not affect the end result, since the entire area where the rock is located will be marked as an invalid planting spot in the next step of the algorithm. However, since the proposed bounding box does not match any of the annotated bounding boxes, it will be considered as a false positive in the evaluation and therefore decrease the precision score. This is a common occurrence for rock detection, as stones sometimes have patches of moss on them that can make parts of a large rock look like a small rock. Sometimes, the model completely fails to detect large rocks, as in Figure 5.4c. The stone located slightly to the left and below the center of the image would be important to detect, as the excavator should not attempt to dig there.

Looking at Figure 5.4a again, there is another important thing to note, namely the detection of the stump. Although the model detects the stump, it does not provide a well-fitting bounding box. The bounding box only captures the top of the stump, but the stump has a large root that continues downward. To be able to plant close to the stumps, the roots must be inside the bounding box so that they are included in the image sent to the stump segmentation model.

Interpreting the results for the water is not as difficult as for the other two objects. Most predictions end up looking like the one in Figure 5.4b, with a single well-fitting bounding box. As can be seen in Figure 5.4d, the model is also capable of detecting objects in images with varying lighting conditions.

5.2 Semantic Segmentation

In this section, the results of the two segmentation tasks will be presented.

5.2.1 Stick Segmentation Model

In the same manner as for object detection, the stick segmentation network was trained and evaluated on all possible combinations of the two subsets. The results of the evaluation on the test sets can be seen in Table 5.7.

Accuracy of the stick segmentation model across different data sets			
Training Data	Subset 1	Subset 2	Subset 1+2
Subset 1	**96.72%**	94.66%	95.80%
Subset 2	92.24%	94.96%	93.46%
Subset 1+2	96.12%	**95.93%**	**96.04%**

Table 5.7: Accuracy of the stick segmentation model, where the different rows represent different training data, and the different columns represent different test data. The highest accuracy for each test set is highlighted in bold text.

The model that is trained on both subsets performs best on the test set from the second day and on the combined test set, while the model that was trained solely on subset 1 has the highest precision in the test set of subset 1. This can probably be explained by the fact that there are more large fallen trees in Subset 2, while Subset 1 contains more small sticks. As mentioned in chapter 3.8.1, 94.6% of the pixels in the ground truth labels correspond to the background, and if a model predicted that all pixels are background, the model would still have an accuracy of 94.6%. Although the results presented in Table 5.7 do not exceed this number much, it is important to note that it is difficult to increase this number. Even a slight increase in accuracy would indicate a good model. To some extent, even a decrease in accuracy could be better than in the base case, since a higher recall is more important than high precision in this project in order to avoid obstacles.

For this reason, it is important to examine the images that the network produces and not only to look at the accuracy, to understand how the network performs. In Figure 5.5, four images are presented together with their corresponding annotated masks and the predicted masks of the network. Looking at the resulting segmentation masks, the network seems to find most of the sticks in the images. Most of the sticks that are not found are of a smaller size. The model finds larger sticks or felled trees, but sometimes the masks do not cover the entire object. Another thing to note is that in the first row, the network predicts that part of the stone is a stick. Although this is wrong, it is not a big problem, since the aim is for these obstacles to be detected and

ruled out by the YOLO network anyways.

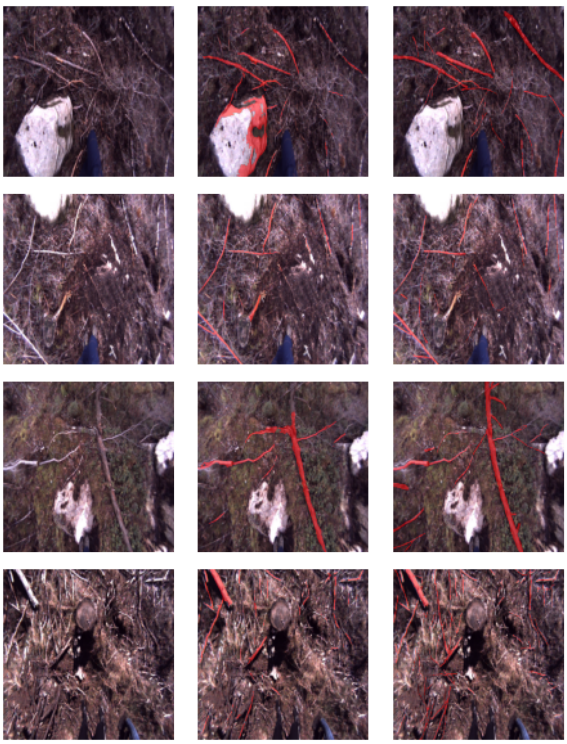


Figure 5.5: *Sample predictions of the selected stick segmentation model.*

5.2.2 Stump Segmentation Model

As for the other networks, the stump segmentation network was trained and evaluated on all combinations of the two subsets. The results are displayed in Table 5.8

Accuracy of the stump segmentation model across different data sets			
Training Data	Subset 1	Subset 2	Subset 1+2
Subset 1	87.49%	86.02%	86.98%
Subset 2	84.06%	87.66%	85.30%
Subset 1+2	**89.08%**	**90.45%**	**89.55%**

Table 5.8: Accuracy of the stump segmentation model in different training and test sets. Each row denotes the training set used, and each column represents the test set on which the trained model was evaluated. The highest accuracy for each test set is highlighted in bold text.

As can be seen, the model trained on both subsets exceeds the other models on all metrics. For example, when training on both subsets and only evaluating on stumps from subset 2, the model yields an increased accuracy of more than 5 percentage points compared to the model trained and evaluated exclusively on subset 2, although the latter is trained solely on stumps from the same location and under similar lighting and weather conditions. This suggests that the model can generalize the appearance of stumps, despite the differences between the data sets. If this is the case, acquiring more data on different days and locations could allow the overall accuracy to increase further. As a comparative measure, a standard RGB segmentation model was also trained and evaluated on subset 1+2 to examine the contribution of the disparity. This was done simply by ignoring the disparity arrays and removing the first convolutional layer that condenses the 4 dimensional input into a 3 dimensional feature space. While doing this allows the pre-trained backbone weights to be used for its intended purpose, i.e. standard RGB input instead of a 4 dimensional reduction, the resulting accuracy on the

test set was reduced to 88.10% compared to 89.55% when RGB-D input was used. This indicates that the disparity features, in fact, increase accuracy - albeit slightly.

Some sample predictions using the best stump segmentation network can be seen in Figure 5.6. The model is able to predict the correct segmentation mask for most stumps, despite the difference in lighting and weather conditions. This is especially apparent in the third row of Figure 5.6, where some parts of the stump are heavily shaded while others are bright. Still, the model manages to predict an almost identical segmentation mask as the ground truth. The model seems to struggle the most with (1) stumps that are comparatively small and (2) stumps located in the distant periphery of the cameras view. The former is illustrated in the bottom row of Figure 5.6, where the predicted mask for the small stump differs substantially from the ground truth mask. Possible reasons for these difficulties are explained below.

(1) The model has a required input shape of 256x256, which means that a cropped area larger than this will be scaled down simply by removing excessive pixel information. However, if the isolated area is smaller, as is the case for small stumps, the input must be scaled to the required shape. This upsampling is performed using some interpolation method, where additional data is fabricated to fill the desired shape. In practice, this means that the model receives less authentic pixel information, which might contribute to it struggling to correctly segment smaller stumps.

(2) Since the camera has a wide field of view, which is a consequence of the camera's fisheye lens, stumps located near the periphery of an image can sometimes appear to be slightly tilted. Stumps from this tilted angle are infrequent in the data set compared to images of stumps from above. As a result, the model may not be able to generalize the stumps at an angle, again indicating that additional data could improve the result. However, since stumps located far away are out-of-reach for the excavator anyway, it does not affect the final algorithm.

,

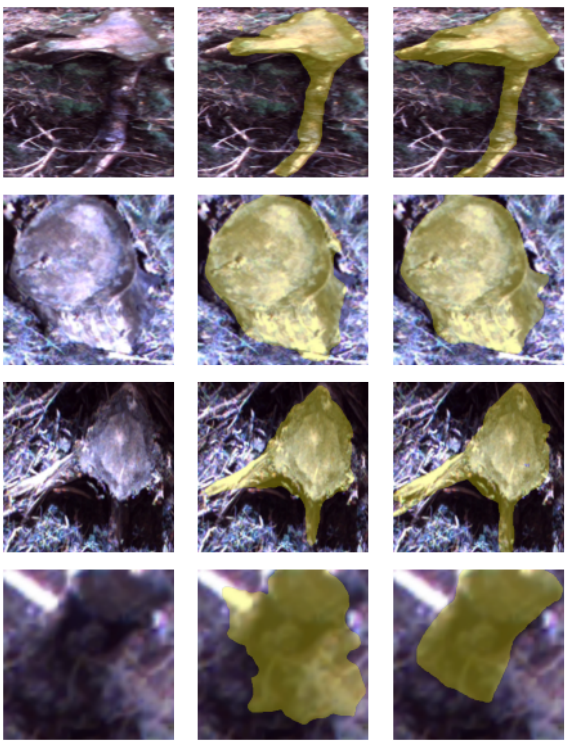


Figure 5.6: *Sample predictions of the selected stump segmentation model across different test images from both subsets. The leftmost image of each row illustrates the RGB input for the model, the middle column illustrates the predicted mask and the right image denotes the ground truth segmentation mask. Note that a disparity array is included in the input in addition to the RGB image, but is not shown here for illustrative purposes.*

5.3 Combining the Models

As mentioned in Section 3.9.1, all different models were combined into a single algorithm that, given an RGB image and a corresponding RGB disparity map, fuses the output of all models into a single mask where all obstacles are visualized. Besides combining the models into a single output, some additional pre-processing was also performed with regard to some of the shortcomings of the models discussed earlier. For example, since the object detection model often fails to detect the entire stump and only encloses the stump stem, each predicted bounding box was expanded by 20 pixels in each direction before sending the isolated area to the stump segmentation network. This decreased the risk of missing the stump roots while still providing a sufficiently concise area for the model. Some example predictions from the different test sets can be seen in Figure 5.7.

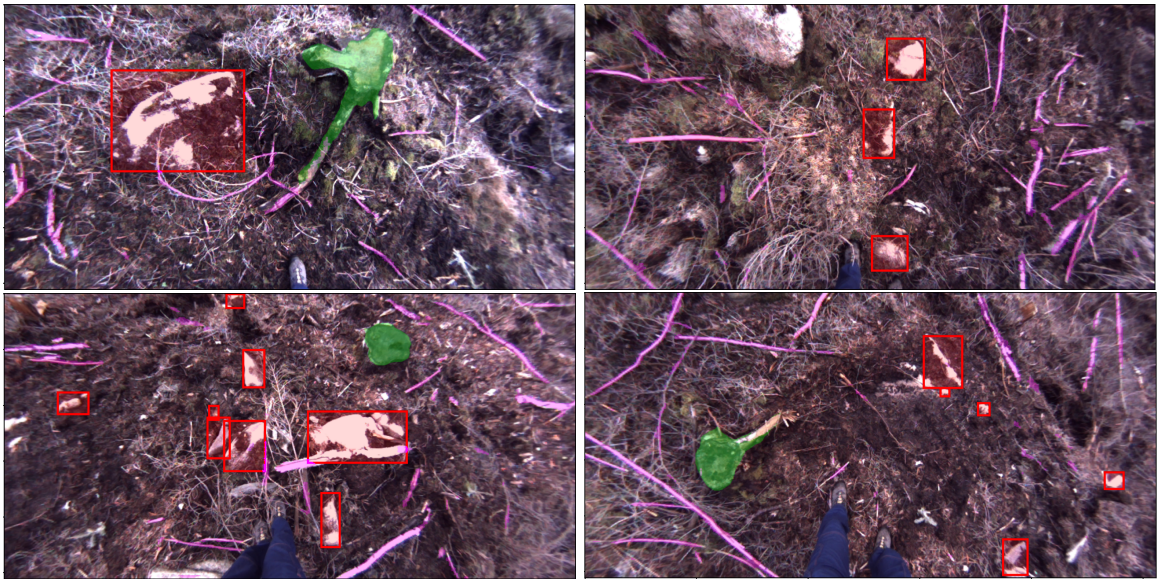


Figure 5.7: *Output samples from the main algorithm. Each pixel is either classified as a rock, stump, stick or felled tree, or plantable area. Red boxes denote rocks, green areas denote stumps, and magenta areas denote large sticks and felled trees. These images are then be used to find a suitable planting spot, as explained in Section 3.9.3.*

Looking at the sample images, the model seems to find most of the obstacles. In the upper left image, the model does not detect a large rock, but the model does find all other important obstacles. Some small sticks are not segmented, but they are small enough to not hinder the excavator.

5.4 Choosing the Planting Spots

After identifying all substantial obstacles in a scene, the final part of the algorithm is to select two planting spots that yield favorable growing conditions for the saplings.

The final planting spot model was tested on the full test set, some examples of the output can be seen in Figure 5.8. Human evaluation was used to analyze the full system by inspecting the output images and estimating if the proposed planting spots were correct.

By presenting all test images to the algorithm, a total of 286 planting spots are proposed. When manually inspecting the results, it could be concluded that 232 of them, or roughly 81%, are valid. In addition to being valid, they are often favorable spots; generally high up and when stumps are identified, it selects a spot close to it. 45 of the spots, or 16%, are invalid and a visible obstacle obstructs the location. 9 proposed spots, or 3%, are ambiguous. In these cases, it is difficult to decide whether the spot is valid or not, and sometimes only a small fraction of an obstacle is within the proposed area.

The most common reason for proposing an invalid planting spot is if either the YOLO-model failed to

Figure 5.8: *Output from the final algorithm. All pixels where obstacles have been found are colored red, the stumps are colored green and the proposed plantings pots are colored blue. The proposed planting area of* $0.45x0.80m$ *is projected onto the 3D world coordinates as to better estimate the affected area.*

detect an object, or if the segmentation model failed to detect a large stick. If, for example, a large stump or rock remains undetected, these objects are usually one of the highest points. As a result, the algorithm tends to choose to plant on top of the obstacles as the optimal planting spot, as they are given a high ranking score based on their altitude. An example of this can be seen in the lower left image of Figure 5.8. However, the opposite tends to happen for undetected pools of water; since they often are the lowest points, they still receive a low ranking score and are therefore still avoided.

6 Conclusions and Future Work

Overall, the proof-of-concept is promising. The algorithm successfully identifies favorable planting spots in most cases and mainly struggles when obstacles are left undetected. However, after analyzing the results, it can be concluded that additional data seem to improve all individual model performances. Therefore, if this proof of concept was to be continued into a final product, the overall robustness of the algorithm should increase if more data is added.

In the continuation of this chapter, the conclusions that have been drawn from the project will be presented together with some propositions for future work. The main topics will be sensors, deep learning models, and the overall method.

6.1 Sensors

Since the entire project is based on computer vision, the choice of camera was an important decision. The camera is not only used to create the entire data set for all deep learning models, but the same camera is also intended to be used for inference on the BraSatt project to ensure that the training data match the actual data. The Arcure Omega camera used for the project has some features that are specifically suited for the project; it is water resistant and durable, meaning that it can be used at the intended clear-felled locations without much caution. Additionally, the fact that it is a stereo camera enables the use of stereo matching, which in turn allows for a conversion to 3D world coordinates. If a single camera were used instead, a LiDAR or some other external sensor would be needed to correctly transform the image to the real world. Another possibility could be to train a machine learning model for monocular depth estimation, i.e., estimate the distance given only a single image [36]. Although there are examples of unsupervised models that attempt to estimate monocular depth, most models are supervised [37]. This introduces new problems, such as acquiring labeled depth data that are suitable for the task in order to train such a model. In conclusion, using a stereo camera is both simple and efficient for the purpose of this project.

However, there were also many drawbacks to using the Arcure Omega camera. First, the camera uses fisheye lenses, which sometimes captures an excessively large area with respect to the excavators' reach. Stereo matching with a fisheye lens is also a more complicated task, as most state-of-the-art disparity map algorithms are made for linearly projected images. Using fisheye lenses often involves additional steps, such as rectification of the image before computing the disparity [38]. A camera with a normal lens would, therefore, be more suitable for the task than the fisheye lenses used on the Arcure Omega camera. Another drawback with the selected camera is the resolution: The rectified images produced have a width and height of 1024 and 512 pixels, respectively, corresponding to a total resolution of 0.52 megapixels. The object detection algorithm used in this project requires an input of 640x640 pixels, which means that the image has to be resized by shrinking the width and expanding the height. The expansive part is performed through some interpolation method, where additional data is fabricated to fill the desired shape, yielding less authentic pixel information. Ideally, the camera used should produce images with at least 640 pixels in each direction.

A final drawback is the ability of the Arcure Omega camera to handle different lighting conditions. When a camera takes a photo, the exposure time determines the length of time at which the sensor in the camera is exposed to light. The Arcure Omega camera takes a single image, using a single value for the exposure time. When there is too much of a lighting difference within a frame, parts of the image can then become over- or underexposed, which in turn can lead to a loss of information. To combat this, a High Dynamic Range (HDR) camera could be used instead. An HDR camera takes several images in rapid succession, using multiple different shutter speeds, and then merges them into a single image, which can be done in several ways [39]. Using this method, it is possible to capture an image with vivid details in different lighting conditions.

All of these changes could improve the quality of the data and thus improve the performance of the neural networks trained on it. Additionally, a major part of creating the data set was to annotate the images. Due to the drawbacks discussed above, it can be difficult to correctly identify all relevant objects in the images. If a human does not notice an object and, thus, does not annotate it, the neural network trained on the data set will also not be taught to identify the object. In summary, a better camera could not only increase the quality

of the data but also improve the correctness of the ground-truth annotations. Both of these improvements could, in turn, result in more accurate neural network models.

The scope of this project was to use computer vision to solve the problem of finding a good planting spot using a stereo camera that provides RGB images and a disparity map. Future work could include investigating additional sensors, such as an infrared (IR) camera or a reflection sensor. Using these sensors, it can be possible to collect more information on the material of an object, and this information could then be used to help identify the obstacle [40].

6.2 Object Detection

After evaluating the YOLOv5 algorithm on both subsets, it was shown that the model was able to find most of the substantial objects, but not all. One thing to keep in mind is that if the model has a 90% recall on stumps, it finds 90% of the annotated bounding boxes. However, since each stump is included in more than one image, it is still possible that the model detects each individual stump at least once, although not every time each stump appears in an image. By using the detection model on a few images that are taken in succession while the E-Beaver drives up to each scene, there are more images in which to detect the obstacles. If an object is found in either of the images, this could then be fused to the final output, which in the end could increase the recall score for the objects.

From looking at the results, it can be concluded that the use of YOLOv5 for the object detection task will suffice to meet the requirements set in this thesis. The model is able to detect most of the important objects, and the results show that the model performance increases as more data is added, even if the weather conditions differ in the new data. Although it was decided to use YOLOv5, it is important to keep in mind that the field of computer vision and deep learning changes very rapidly. New methods are proposed frequently, and what is considered to be state-of-the-art changes every year. New networks and more sophisticated methods using RGB-D for object detection are expected to be published soon. Therefore, using YOLOv5 for object detection may not be the best option in the future. However, the results of this thesis show that object detection is a sufficient approach to find obstacles in a forest environment.

6.3 Semantic Segmentation

For the segmentation task, two models were created that were both based on a U-Net architecture. Since the two different segmentation models solved two different tasks and had some slight differences, they were analyzed separately. However, what can be said in general is that using the U-Net architecture with a pre-trained backbone was a good choice for solving both tasks.

6.3.1 Stumps

The segmentation of the stumps is a crucial step in finding a planting spot if planting close to the stumps is considered important. Looking at the results, the model performs well and is able to perfectly segment most stumps. The most difficult part is to properly segment the root stems, especially the thinner ones. Using the depth data improved the results by 3%. Although a modified version of the U-Net architecture is used, as explained in Section 3.8.2, the backbone and architecture were still created with the intention of using an RGB image. Therefore, it could be worth looking into using novel networks, customized for RGB-D data. This could, for example, be a network that uses depth-aware convolutions and depth-aware average pooling. These methods have been shown to improve the accuracy of models trained on RGB-D data [41].

6.3.2 Sticks

The stick segmentation model handles all varieties of sticks and trees. However, a small stick and a large tree differ greatly in size, so it is not obvious that these two items should be classified into the same group. Splitting them into two different classes or using two different models might make more sense, since the geometry of

the objects in each class will be more consistent. The smaller sticks are usually not visible in the disparity maps, indicating that they are too small for the stereo camera to detect any height changes when they appear. However, the larger sticks and felled trees are usually clearly visible in the disparity maps. Therefore, by splitting the category, an advantage is that depth information could be used to segment larger trees and, hopefully, improve the performance to detect them. Note that these are also more important to detect than the smaller sticks, since the large ones hinder the excavator, whereas smaller sticks can usually be ignored.

6.4 Main Algorithm

One recurring problem is that the algorithm tends to plant on top of obstacles that the YOLO-model fails to detect, since the tops of the objects are usually the highest point. A solution to this problem could be to analyze the point cloud near a chosen planting spot, stumps and rocks usually have a large z-gradient at the edge. As an extra filter to avoid obstacles, a function could be added that determines how probable it is that a planting spot is on top of an object, based on the 3D shape. In addition, the scoring algorithm used to rank the different valid planting spots can be improved. In its current state, the algorithm ranks the proposed spots based primarily on their proximity to a stump and secondarily on its average altitude. Instead, a function could be used to evaluate the combination of these attributes. This can also allow additional features to be taken into account.

6.5 Methodology

Since the objective of this project was to solve a problem and not to try to improve an existing methodology, there are no other projects with which to compare the final results. Therefore, it is also difficult to evaluate whether the method proposed in this thesis is good. However, individual models have still been robustly evaluated and compared.

At the beginning of the project, some other approaches were tested, which were not based on machine learning. Although the main focus has been on deep learning methods, some other methods were evaluated, and the early results clearly pointed to the fact that deep learning methods were superior. The deep learning-based methods have been studied more thoroughly and therefore it is not fair to compare the non-machine learning approaches, which received less attention. However, it can still be concluded that deep learning-based methods showed initial results that were more promising, and more traditional computer vision methods would probably not yield as good results.

In the end, the model was able to find a good planting spot in 116 of the 144 test images. Every time the model proposes an invalid planting spot, the proposed planting spot is on top of an obstacle. Therefore, to increase the performance of the model, either the object detection model must improve by training on more data, or, as discussed earlier, some sort of 3D-shape analysis of the proposed planting spots must be introduced, before accepting the planting spots.

6.6 Suggestions for BraSatt

After looking at the results, it can be concluded that the method proposed in this report will probably suffice to meet the requirements established to find good planting spots. As mentioned above, the performance of the main algorithm strongly depends on the object detection and segmentation models. Therefore, to achieve even better results, the next step of the process could be to find a new camera that meets the listed requirements and then start collecting higher quality data. The following steps could be performed:

- Find a camera that meets the requirements mentioned in 6.1.
- Decide how to mount the camera on the final product.
- Start collecting images from the position where the camera will be used.

6.7 Final Remarks

While there are similar projects that explore the task of identifying planting spots, most notably *Stereo Vision Based Tree Planting Spot Detection*, the aim of that project is to identify mounds after already preparing the soil. In contrast, the goal of BraSatt involves additional steps, namely autonomously selecting a valid planting spot, preparing the specific patch, and finally placing a sapling in the prepared area. The main difference is that a prepared planting spot will be *created* in this project, while it was detected in that project [8]. To the best of our knowledge, there are no similar projects investigating this problem. Therefore, this thesis contributes a great step towards finding planting spots using Deep Learning.

Throughout this project, multiple methods have been explored to solve the task. In the final created method, 81% of the proposed planting spots among the test images are good, indicating that the created method is promising. That being said, as mentioned above, there are multiple areas of improvement, most notably the addition of more data. However, the collected data set is available and can be expanded or investigated further. Using the proposed method as a framework and implementing the suggested improvements, even better models can be built to achieve greater results if more resources are invested in the project.

References

[1] *Skogsmarken dominerar Sverige*. https://www.scb.se/hitta-statistik/statistik-efter-amne/miljo/markanvandning/markanvandningen-i-sverige/pong/statistiknyhet/markanvandningen-i-sverige2/. Accessed: 2022-02-10.

[2] Annika Nordlund and Kerstin Westin. Forest Values and Forest Management Attitudes among Private Forest Owners in Sweden. *Forests* **2**.1 (2011), 30–50. ISSN: 1999-4907. DOI: 10.3390/f2010030. URL: https://www.mdpi.com/1999-4907/2/1/30.

[3] *Långsiktig tillväxt startar i skogen*. https://www.sodra.com/sv/se/om-sodra/. Accessed: 2022-02-10.

[4] *Södra köper skog i Lettland*. https://www.sodra.com/sv/se/skog-medlem/aktuellt/sodrakontakt/nyhetsartiklar/2018/nummer-4/sodra-koper-skog-i-lettland/. Accessed: 2022-02-14.

[5] Alice Falk and Maja Östlund. Effekter av slutavverkning och markberedning på marklavar i svenska skogar (2020).

[6] *Examensarbete inom Autonom skogsplantering, 30 hp*. https://bitaddict.se/examensarbete2/. Accessed: 2022-02-14.

[7] *Kontrakt med samarbetspartners för projektet BraSatt är nu klara*. https://www.sodra.com/sv/se/skog-medlem/aktuellt/nyheter/2021/for-alla-vo/kontrakt-med-samarbetspartners-for-projektet-brasatt-ar-nu-klara/. Acessed: 2022-02-14.

[8] Teemu Kemppainen and Arto Visala. "Stereo vision based tree planting spot detection". *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 739–745.

[9] Songyu Li and Håkan Lideskog. Implementation of a System for Real-Time Detection and Localization of Terrain Objects on Harvested Forest Land. *Forests* **12**.9 (2021), 1142.

[10] Ahmad Ostovar. "Object Detection and Recognition in Unstructured Outdoor Environments". PhD thesis. Umeå University, 2019.

[11] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[12] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[13] Niall O'Mahony et al. "Deep learning vs. traditional computer vision". *Science and information conference*. Springer. 2019, pp. 128–144.

[14] Cristian Vancea and Sergiu Nedevschi. "LUT-based image rectification module implemented in FPGA". *2007 IEEE International Conference on Intelligent Computer Communication and Processing*. IEEE. 2007, pp. 147–154.

[15] BERNHARD Mehlig. Machine learning with neural networks. *arXiv preprint arXiv:1901.05639* (2019).

[16] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA, 2017.

[17] Daniel Graupe. *Principles of artificial neural networks*. Vol. 7. World Scientific, 2013.

[18] Joseph Redmon et al. "You only look once: Unified, real-time object detection". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[19] *G. Jocher. yolov5*. https://github.com/ultralytics/yolov5. Accessed: 2022-03-14.

[20] Shibani Santurkar et al. How does batch normalization help optimization? *Advances in neural information processing systems* **31** (2018).

[21] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks* **107** (2018), 3–11.

[22] Chien-Yao Wang et al. "CSPNet: A new backbone that can enhance learning capability of CNN". *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 390–391.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[24] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[25] Kaiming He et al. "Deep residual learning for image recognition". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[27] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
[28] *Arcure Group. Omega Camera*. https://arcure.net/omega-stereo-camera-for-indoor-outdoor-applications. Accessed: 2022-04-04.
[29] *J. Nelson. roboflow*. https://roboflow.com/. Accessed: 2022-03-25.
[30] *Precision and Recall*. https://en.wikipedia.org/wiki/Precision_and_recall. Accessed: 2022-06-10.
[31] *I. Hunt-Isaak. AC295*. https://github.com/ianhi/AC295-final-project-JWI. Accessed: 2022-03-25.
[32] Pavel Yakubovskiy. *Segmentation Models*. https://github.com/qubvel/segmentation_models. 2019.
[33] Shruti Jadon. "A survey of loss functions for semantic segmentation". *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. 2020, pp. 1–7. DOI: 10.1109/CIBCB48159.2020.9277638.
[34] Kevin L Priddy and Paul E Keller. *Artificial neural networks: an introduction*. Vol. 68. SPIE press, 2005.
[35] Junjie Bai, Fang Lu, Ke Zhang, et al. *ONNX: Open Neural Network Exchange*. https://github.com/onnx/onnx. Accessed: 2022-05-23.
[36] Alican Mertan, Damien Jade Duff, and Gozde Unal. Single Image Depth Estimation: An Overview. *CoRR* **abs/2104.06456** (2021). arXiv: 2104.06456. URL: https://arxiv.org/abs/2104.06456.
[37] Ravi Garg, Vijay Kumar B. G, and Ian D. Reid. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. *CoRR* **abs/1603.04992** (2016). arXiv: 1603.04992. URL: http://arxiv.org/abs/1603.04992.
[38] Johannes Schneider, Cyrill Stachniss, and Wolfgang Förstner. On the accuracy of dense fisheye stereo. *IEEE Robotics and Automation Letters* **1**.1 (2016), 227–234.
[39] Shree K Nayar and Tomoo Mitsunaga. "High dynamic range imaging: Spatially varying pixel exposures". *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*. Vol. 1. IEEE. 2000, pp. 472–479.
[40] Håkan Lideskog et al. Determining boreal clearcut object properties and characteristics for identification purposes (2014).
[41] Weiyue Wang and Ulrich Neumann. "Depth-aware cnn for rgb-d segmentation". *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 135–150.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY