2019•2020
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Synthetic data generation via Blender for training object detection networks: evaluation on apple detection in orchards

PROMOTOR :
Prof. dr. ir. Eric DEMEESTER

PROMOTOR :
dr. ir. Serge REMY

BEGELEIDER :
ing. Bart MOYAERS

Senne Colson

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Gezamenlijke opleiding UHasselt en KU Leuven

2019 • 2020

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Synthetic data generation via Blender for training object detection networks: evaluation on apple detection in orchards

PROMOTOR :

Prof. dr. ir. Eric DEMEESTER

PROMOTOR :

dr. ir. Serge REMY

BEGELEIDER :

ing. Bart MOYAERS

Senne Colson

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

UHASSELT KU LEUVEN

Deze masterproef werd geschreven tijdens de COVID-19 crisis in 2020. Deze wereldwijde gezondheidscrisis heeft mogelijk een impact gehad op de opdracht, de onderzoekshandelingen en de onderzoeksresultaten.

Table of Contents

List of Tables

List of Figures

Nomenclature

CNN	Convolutional Neural Network, page 11
COCO	Common Objects in Context, page 11
Mask R-CNN	Mask Region with Convolutional Neural Network, page 11
RoI	Regions of interest, page 18
MCT	Minimum Confidence Threshold, page 20
TPs	True Positives, page 21
FPs	False Positives, page 21
FNs	False Negatives, page 21
(m)AR	(Mean) Average Recall, page 21
(m)AP	(Mean) Average Precision, page 21

Abstract

Around 75% of the pears, produced in Belgium, are currently exported. Most export countries demand that these pears originate from land plots that are free of fire blight. As a first step towards automatic detection of fire blight, this thesis explores the potential of state-of-the-art machine learning techniques to detect apples, leaves and branches in RGB images of an orchard. These techniques require a large, labelled image dataset, which is typically very time consuming to obtain. A solution to this problem is created in this thesis.

In this work, a Python program was written that offers the ability to generate a large annotated dataset consisting of synthetic images in a common object segmentation image format (COCO) via a 3D creation software (Blender). Furthermore, the researcher has control over every aspect in every scene, such as brightness, light exposure, camera position, etc. This dataset can then be used to train several neural networks. Here, this software was used to create a dataset consisting of 2750 synthetic train images of different kinds of apple trees carrying fruits. These images and annotations were used to train a Mask Region based Convolutional Neural Network (Mask R-CNN) to detect and annotate apples, leaves and branches in an orchard.

The results based on apples only are promising, with a F1-score of 0.85. With respect to the Mask R-CNN model trained with the standard COCO dataset, this is an improvement of 0.78, evaluated on orchards. Finally, a method is proposed to remove the false positives.

Abstract (Dutch)

In België wordt 75% van de geproduceerde peren geëxporteerd. De meeste exportlanden eisen dat deze peren afkomstig zijn van percelen die vrij zijn van bacterievuur. In een eerste stap wordt er gekeken naar de mogelijkheid om appels, bladeren en takken te detecteren in RGB-afbeeldingen van boomgaarden via machine learning-technieken. Deze technieken vereisen een grote, gelabelde dataset, wat erg tijdrovend is om te verkrijgen. Echter wordt hiervoor ook een oplossing gegeven.

In deze masterthesis werd een programma geschreven in Python waarmee een grote dataset van geannoteerde afbeeldingen gegenereerd kan worden in *common object segmentation image* formaat (COCO) via een 3D modeleringssoftware (Blender). Daarnaast heeft de onderzoeker controle over elk aspect in elke scene, zoals helderheid, lichtinval, camerapositie… Deze dataset kan vervolgens gebruikt worden om verschillende neurale netwerken te trainen. In deze thesis werd het programma gebruikt om een COCO-dataset bestaande uit synthetische afbeeldingen van verschillende appelbomen te genereren. Aansluitend worden deze afbeeldingen en annotaties gebruikt om een Mask Region based Convolutional Neural Network (Mask R-CNN) te trainen om zo appels, takken en bladeren in een boomgaard te herkennen.

De resultaten zijn alvast veelbelovend met een F1-score van 0.85, gebaseerd op enkel appels. Dit is een vooruitgang van 0.78 ten opzichte van het Mask R-CNN model getraind met een COCO-dataset. Tot slot wordt nog een methode voorgesteld om de huidige valse positieven te verwijderen.

Chapter 1:

Introduction

Context

In Belgium, pear cultivation is actually a 'Conference' pear cultivation: more than 90% of the pear cultivation area in Belgium consists of Conference pears. This percentage kept increasing in recent years up to more than 10.000 ha. More than 75% of the total pear production is exported, mainly within the EU, but also to China, USA, Brazil, Canada, Vietnam and Israel. In a number of export countries, the pears must come from fire blight *(Erwinia amylovora)* free plots. However, detection of this bacterial disease is still very time-consuming and, hence, costly. Currently, it is done by visually monitoring each row of trees and carefully cutting out infected flower clusters, twigs, branches and fruitlets. Disinfection of used tools and cutting wounds as well as follow-up monitoring are crucial. In case of severe infection, the entire orchard has to be eliminated, i.e. all trees need to be removed and burned, including the rootstock. Whenever this is the case, the economic loss will amount to up to tens of thousands of Euros or more, since the grower not only loses the infected trees, but he will also have to wait several years until the new planting reaches full production.

pcfruit, a research centre specialized in fruit search, is searching for a solution to detect fire blight symptoms automatically and, if possible, early. One of the possible approaches was introduced as a master's thesis for the students of the Faculty of Engineering Technology from the joint program of KULeuven and UHasselt.

Problem definition and goals

The initial aim of this master's thesis was to establish a method to detect fire blight symptoms via drone images using AI techniques, such as Deep Learning or other predictive modelling techniques. However, after a meeting with VITO, who conducted previous research with pcfruit on spectral detection of fire blight symptoms in orchards using drone images, it was clear that this is currently not possible because of the lack of ground truth images to train the Mask R-CNN. In agreement with VITO, pcfruit and ACRO, we decided to shift the aim of this master's thesis. The new aim is to solve this problem of sparse ground truth by generating synthetic images for which ground truth is perfectly known. Furthermore, the approach will be tested and evaluated by verifying how well the trained Convolutional Neural Network (CNN) model is able to detect apples in the first place, and later on also leaves and branches in real images of an orchard. This solution may be used in the future to detect fire blight symptoms in earlier stages.

Tools

During this master's thesis, three major tools were used. First, a 3D creation software (Blender) was used to create 3D models of every object in the scene. In this project, only apples, leaves and branches were created. The other tool that was used, was Python. Blender comes with a built-in Python plug-in. Using this plug-in, everything in every scene can be controlled, while every process in Blender can be automized. The last tool was a Mask R-CNN framework from matterport. More information about this framework can be found in 2.1.1.

Overview

This thesis has the following structure. First, chapter 2 handles the literature study that was written to acquire the information that was necessary to choose and use the training-frameworks. Apart from a

convolution neural network (CNN) in general, two specific CNN's are discussed as well. In addition, some important terms and different approaches to gather ground truth are explained here. Chapter 3 explains the materials and methods. First, it is explained how the synthetic images and the corresponding ground truth images are made using both Blender and Python. It dives into detail on how each function in the written Python-script works and how it is used during this master's thesis. Furthermore, it explains how the ground truth images are converted to actual ground truth in COCO-format and gives the format in which these files are saved. Lastly, chapter 3 will also give an explanation on how to both train and evaluate a model using the synthetic dataset. In chapter 4, the results of the different approaches are discussed, while some extensions are given that can be implemented in the future in order to improve the current model and results. These extensions are based on some weaknesses of the current method which will also be explained in this fourth chapter. In chapter 5, a conclusion is formed based on these results and suggestions for future work are given.

Chapter 2

Literature study

First, this chapter will discuss some related work about fruit detection in orchards and image segmentation based on synthetic ground truth images. Furthermore, this chapter will also discuss some concepts and definitions, which will be used later in this master's thesis. These concepts and definitions are important to have a better understanding on how a CNN can be trained using only synthetic data. First, the operating principle of an artificial neural network in general is explained, followed by two more specific CNN's; Mask R-CNN and YOLO. Next, previous research on synthetic ground truth and machine learning is summarised. Finally, the adopted evaluation criteria for the trained model are explained.

Related work

2.1.1 Fruit detection in orchards

An earlier approach to detect apples in orchards using CNN's, was conducted by the University of Minnesota in 2018. [1] 13 000 images were labelled manually by human labelled. After this process, they were able to achieve an accuracy of 94%. However, this method is only used and evaluated to count apples. During this master's thesis, the trained model will be evaluated based on the instance segmentation of every apple in the evaluate images, which is more difficult to achieve.

2.1.2 Image segmentation based on synthetic ground truth images

A research conducted by the University of Chicago in 2018 proved that training a neural network is possible with only synthetic images. The idea to combine both synthetic images and domain randomization originated from this research. The researchers generated 100 000 synthetic images to transfer learn (from the COCO weights) a CNN. This resulted in an average precision of 83.7%. [2]

Neural network

Every neural network consists of multiple layers. Each layer has its own purpose, working principle and function. Together, all these layers will be able to make decisions based on the input and previous experiences. In this master's thesis, the working principle of two different neural networks will be discussed, namely Mask R-CNN and YOLO. Purely based on theory, one of those two networks will be chosen and used during this project to detect apples in an orchard.

Basically, there are multiple detection possibilities. The first one is called 'image classification' and, as illustrated in [3, Fig. 1a], will only tell us which objects are present in the image. The second possibility, called 'image detection', will draw square bounding boxes around every detected object and will name these bounding boxes accordingly. This is visualized in [3, Fig. 1b]. The third way to label objects in an image is called 'image segmentation'. In this method, the network will label every object pixel-based, however, it will not make any distinction between every individual object in a group of the same objects, as shown in [3, Fig. 1c]. The last method, called instance segmentation, is able to make this distinction as illustrated in [3, Fig. 1d]. This literature study will mainly focus on the latter, since objects during this thesis should be detected separately. [3]

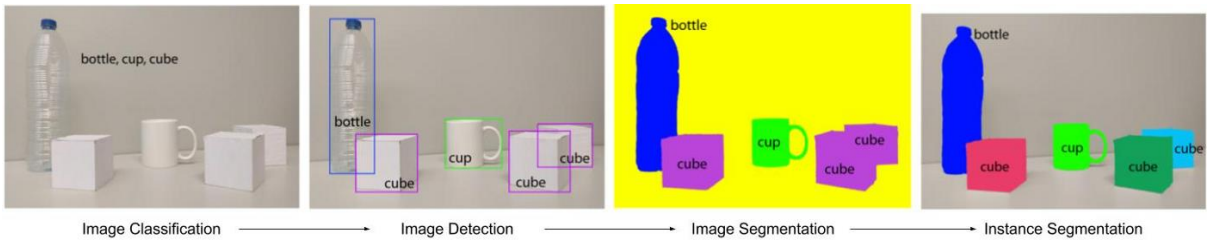

Figure 1: Different image detection possibilities [3].

2.1.1 Mask R-CNN

This deep neural network results from Faster R-CNN and aims to solve instance segmentation problems in both computer vision and machine learning. [4] Mask R-CNN consists of 4 major parts and networks:

1. The first layer consists of a ResNet 101. This is a CNN with 101 layers. Its purpose is to extract feature maps from a given image, so the output will be several feature maps. A feature map of a CNN is the result of applying a filter in a certain layer to the input image. A method to understand which features the CNN is able to detect, is to visualize these feature maps. [5]
2. The following layer is a region proposal network (RPN) with these feature maps as input. The RPN is responsible for proposing different areas in order to detect objects. This is done by dividing the image in about 200.000 regions of different sizes, called anchors. These anchors must overlap to cover as much of the image as possible. Since the RPN is able to scan over the backbone feature map instead of the image itself, it is considered to be a rather fast process. Furthermore, RPN is able to scan the anchors in parallel. Finally, the anchors with the highest probability to contain an object are selected. These anchors are called the regions of interest (RoI) and passed to the next layer. [6]
3. This next stage consists of a RoI classifier and takes the previously mentioned RoIs as input. For each RoI it calculates both the class (foreground or background) and bounding box of the object in the anchor. However, these anchors probably are of different sizes which could cause a problem. To solve this, RoI pooling is used. This means that every anchor will be cropped and resized into a fixed size. [7]
4. These first three layers are similar to a Faster R-CNN framework. Only the fourth layer will distinguish a Mask-RCNN framework from a Faster-RCNN one. This fourth layer is once again a CNN. This network, however, is responsible for constructing the segmentation masks which will outline and locate the detected objects in an image. These masks have a low resolution, nevertheless, they consist of float numbers instead of bits. This means that each pixel contains more information. Using this extra information, this framework is later able to return pixel-based masks for (almost) every object in the image. [7]

Mask R-CNN only annotates an object when the confidence is above a certain threshold. This threshold is called minimum confidence threshold (MCT).

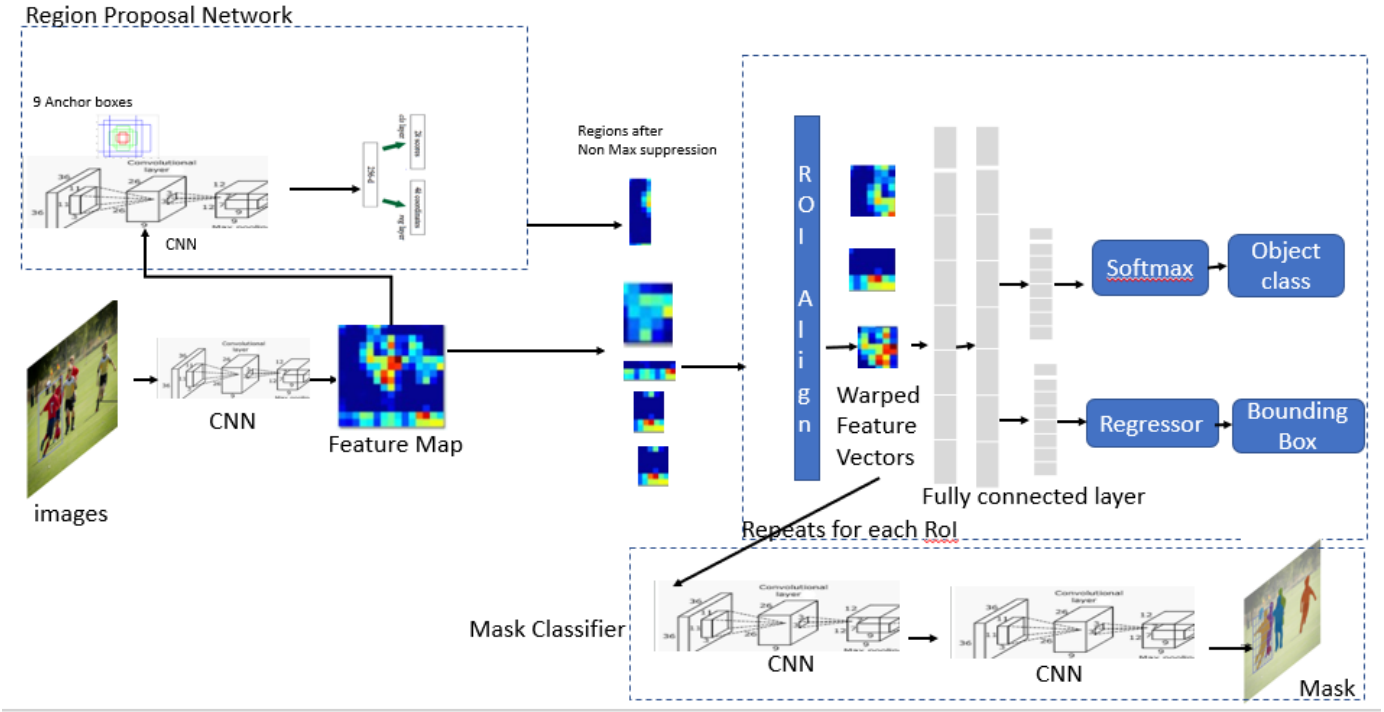

Figure 2: Schematic visualization of Mask R-CNN [4].

2.1.2 You Only Look Once

Another neural network is called You Only Look Once (YOLO). This framework can detect several objects with only one glance at the image. Unlike Mask R-CNN, YOLO consists of only one single neural network. Using this single neural network, YOLO is able to return both the bounding box and the confidence for (almost) every object in an image with only one look at this image. [8]

This first step in YOLO is to resize the image to 448 x 448 pixels. Next, the image is divided into a grid. Finally, this grid is combined with the results of the CNN which results in different bounding boxes with each a certain probability. This framework is rather fast since it only has to look at the image once and only one neural network is needed. However, it is not as accurate as other neural networks, since it could experience problems predicting false positives in the background and because it learns rather general representations of objects. [8]

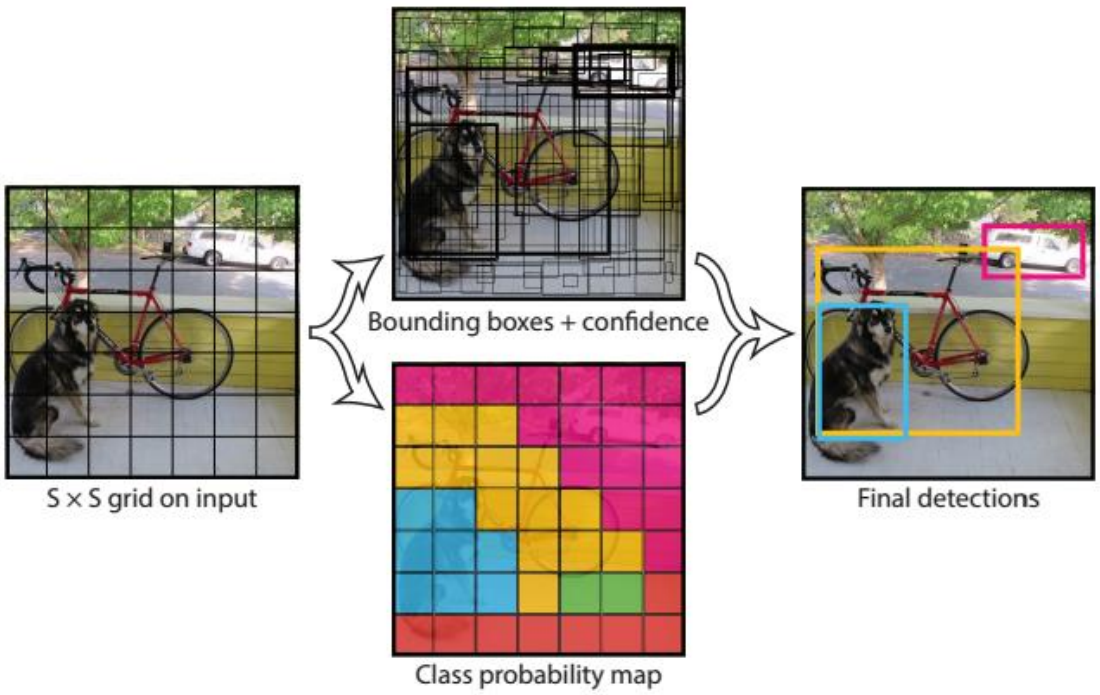

Figure 3: Schematic visualization of YOLO [5].

Since Mask R-CNN is the most accurate framework for pixel-based instance segmentation, it is the better choice for this project. Furthermore, for this project, it is not necessary to detect objects in real-time, like YOLO is capable of.

Machine learning for semantic segmentation with synthetic data

Semantic segmentation refers to pixel-based recognition of objects in images. Each pixel is assigned to either a class (e.g. an apple, a leaf, a tree trunk) or the background. In order to train a model to recognize those objects, gigabytes of ground truth are necessary. One option could be to take hundreds of thousands of pictures and annotate them manually. However, this would be a very time-consuming process in both collecting and annotating. A more efficient alternative is to generate synthetic data via a computer program, like Blender. Nonetheless, this could mean a drop in run-time performance, as well as accuracy, of the trained network. Another approach, called 'domain randomization', tackles these problems by randomizing multiple properties of real images, like brightness, colour distribution, alpha compositing, etc. In this literature study, both approaches will be discussed thoroughly.

2.2.1 Synthetic data via 3D graphics software

In the first approach, a researcher will use 3D graphics software, such as Blender or Unity, to generate a large quantity of synthetic images with their corresponding bit masks. Next, these bit masks will be converted to annotations in a dataset. This image dataset can be in different formats: COCO, PASCAL VOC, ILSVRC, SUN… This literature study will focus on the COCO dataset, since this format will be important and was used later on in this thesis. The COCO dataset was originally introduced by Microsoft but became quickly a standard format for image detection and segmentation. [6] It consists of a JSON-file with 5 different objects. The first object is called 'info' and only stores some basic information like

description, URL, version, year, etc. The next object, ‘licenses’, contains all the licenses for the images in a list, if they are necessary. Every license consists of the URL, an incremental id and a name for the license. The third object is called ‘images’ and stores all the necessary information of every image in a list. Every single image possesses information about its unique id, dimension, license if necessary, file name, etc. For the fourth and fifth parts of the JSON-file, ‘categories’ and ‘annotations’, there are several options. Since in this master’s thesis only instance segmentation was used, this literature study will only explain this option. The former, ‘categories’, is a list of object classes that could be detected in the image. This object class contains in its turn a supercategory, a unique id and a name. The latter, ‘annotations’, is a list of annotations in the image that makes up most of the dataset. For every recognized object it stores the coordinates of every single pixel in the contour of this object. It also stores the surface area, the id of the image, the coordinates of a bounding box around this object, the id of the category that was recognized, whether or not this object is part of a bigger crowd of objects from the same class, and a unique id for this annotation. [9]

Using the method of generating synthetic images, for which the software is written during this master’s thesis, the programmer has total control of every aspect of the image. It is possible to change the light exposure in every way by changing the position of the sun, the texture, volume, shape and position of the objects, etc. All of this can be done via the Python script in Blender. In addition, the programmer can easily render an unlimited number of images with their corresponding bit masks. However, this process can take a long time if the program is running on a CPU instead of a GPU. This approach is very useful for a programmer when he wants full control of every single synthetic image. However, using this method, non-realistic images could be generated without the programmer’s control. This can be prevented by setting very strict limits when randomizing some properties but can also be useful to prevent overfitting in some way. Lastly a framework for this approach could be made where different users can generate their synthetic ground truth with less effort. [10]

2.2.2 Synthetic data via domain randomization

The other approach, called ‘domain randomization’, solves the problem of too sparse ground truth by randomizing different properties of existing images. When the programmer would like to generate and annotate an image with two different objects (with each a different class), they would take an appropriate background along with an image for every single object. After pasting each object onto the background in a different position, a basic image is generated. Once again, the programmer has control over both the position and the size of the objects in the image. This way, the bit mask can be generated as well. Once this image is generated, the programmer can apply domain randomization on the image. This means that they will change different properties like size, contrast, rotation, lighting… Hence, the programmer can, once again, generate a somewhat large number of images with its corresponding ground truth. However, this approach is not as efficient as the first approach when the amount of ground truth necessary becomes too large. [11]

Evaluation criteria

To determine how good a model is, multiple formulas can be used. The most well-known one is the F1-score. It considers both the average precision (AP) and average recall (AR). To calculate these numbers, some factors should be determined first. These are (i) the true positives (TPs), which is represented by the number of objects that are correctly predicted, (ii) the false positives (FPs), which shows how many objects were wrongly predicted to be part of a certain class, and, (iii) the false negatives (FNs) , which is equal to the number of objects that were wrongly predicted not to be part of a certain class. [12]

2.3.1 Precision

The precision of a model indicates how many of the retrieved instances were actually relevant in one image, and thus part of a valid class. Whenever the precision is low, the number of false positives will be too high. [9, eq. (2.1)] is used in order to calculate the average precision of a model in all images used to evaluate. [12]

$$AP = \frac{TPs}{TPs + FPs} \tag{2.1}$$

2.3.2 Recall

The recall of a model indicates how many relevant instances were actually retrieved and selected in one image. Whenever the recall is low, the number of false negatives will be too high. [9, eq. (2.2)] is able to calculate the average recall of a model in all images used to evaluate.

$$AR = \frac{TPs}{TPs + FNs} \tag{2.2}$$

2.3.3 F1-score

Since both recall and precision are important in order to indicate the correctness of a model, the F1 score is used. This score considers both the recall and the precision of the evaluation test. It is not possible to just multiply both recall and precision, since one could be rather low while the other can be very high. This means the F1 score will be rather low as well, since the model is not good at all. That is why the F1 score is calculated using [9, eq. (2.3)]. It aims to be equal to one. [12]

$$F1 = 2 * \frac{AP * AR}{AR + AP} \tag{2.3}$$

Chapter 3

Materials & methods

The work executed broadly involves four major steps. The first step is to generate synthetic images, while the second step generates corresponding annotations of every single image, which will define the ground truth. The third step consists of converting these images and annotations to a dataset in COCO format. The fourth step is to train a Mask R-CNN model using this dataset and, additionally, evaluate this model on orchard images.

Synthetic images in Blender

In order to generate synthetic images, multiple steps need to be executed. The first step consists of modelling and painting the model of a realistic apple in Blender. Additionally, this procedure is also used to create realistic models of branches and leaves. The second step is to randomize these models via Python. This needs to happen in such way that every model stays as realistic as possible. Every function in this Python-script will thus be explained thoroughly.

3.1.1 Synthetic modelling in Blender

Using Blender, a 3D model of an apple, consisting of 242 vertices, is made. The number of vertices, which is 242, was chosen via trial and error. When the number is too low, the resolution will be too low as well. However, when the number of vertices is too high, the modelling will be too complicated and, thus, take a long time. In order to model such an apple, a UV sphere is added to the scene in Blender as a starting point for the apples. This UV-sphere is a basically a ball in Blender. The radius is set to the radius of a realistic and average apple, which is approximately 4 cm, while the number of horizontal rings that shapes the apple is decreased to 12. This can be seen as temporarily lowering the smoothness and resolution of the apple. This way the modelling process is much simpler, since the model is simpler as well. However, in the end, the resolution will be increased.

When going to the ‘edit-mode’ in Blender, the shape of the apple can be changed by moving or resizing every single vertex, node or plane. The resolution of the model was increased by adding a Subdivision Surface modifier. This modifier is able to increase the smoothness and resolution by dividing every surface into multiple smaller surfaces. The resolution for modelling purposes can be increased by increasing the viewport subdivisions. However, since the resolution should be higher in the renderer than in the viewport, the render subdivisions must always be higher than or equal to the viewport subdivisions. In this master’s thesis, both were set to two. When the modelling was done, the apple was smoothened by right clicking the model and selecting ‘Shade Smooth’.

After modelling the apple and getting a resulting 3D model as illustrated in Figure 4a, multiple materials are generated by painting the model with images of different existing apples, as shown in Figure 4c. In order to paint these images onto the objects, the 3D models must be unfolded using the UV editor in Blender. A result is visualized in Figure 4b. Every 3D model will be cut precisely so its plane will fit on a 2D surface. In order to project the 2D images from Figure 4c onto the 3D object, which is converted to 2D planes shown in Figure 4b, texture paint in Blender is used. Using a paintbrush, the colours of the image can be drawn onto the 2D planes, as shown in Figure 4d.

First, 10 pictures of red apples are used. Afterwards, the red- and green-values of every pixel are swapped in order to generate 10 pictures of green apples. This way, an apple can be applied with a total of 20 different textures, of which 10 are red and the other 10 are green.

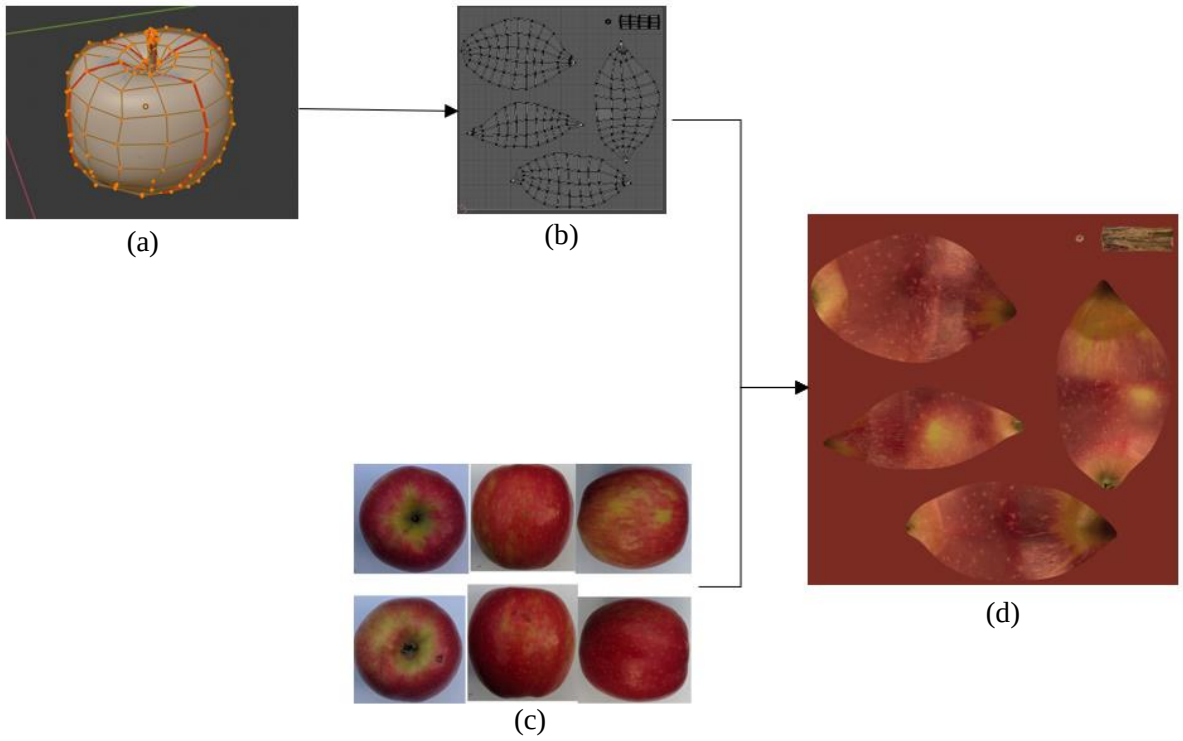

Figure 4: Schematic visualization of texture painting

3.1.2 Domain randomization of models using Python

Blender also comes with an embedded Python interpreter when it is installed on your PC. It provides a typical Python environment and carries its own modules, such as mathutils and bpy. Using these, a program in Python is written with multiple functions for this project, which is explained below.

The **first function** (select_obj) in this program deals with selecting given 3D objects. It takes a list of objects as input, next it will deselect every single object in the scene, while, lastly, it selects the given objects. Besides a list with multiple objects, this function can also work with a list containing only one element. This function will later on be used multiple times.

A **second function** (save_*_vertices_as_default) will save the coordinates of all the 242 vertices in three-dimensional space of an apple to a csv-file. So, when an apple should be saved as default, it is possible to save its vertices to a csv-file using this function. An additional **third function** (set_default_*) is able to load these coordinates in Blender and apply it to an object. This way, every object can be set to a certain default apple. This function can also be used when the user would like to duplicate an object.

The program also contains a **fourth function,** which consists of four smaller functions, that can randomize every apple a bit. Hence, not every apple will look the same and there will be some variation. This function will rotate, translate and reshape every apple sufficiently. The first smaller function (rotate) takes the object to rotate and the minimum and maximum rotation angle as input. The given object is selected and rotated with three different and random angles (one angle in every dimension). In the second smaller function (translate) the given object is selected and translated in a similar way. The given object is again selected and three different numbers in a given range are generated. Finally, the object is translated to the three given points; one for every dimension. The ranges are set in such way

that the objects will fit in the rendered images. Via the third smaller function an object can be resized, however, this works in a different manner. The object will not be physically resized, but, since the ranges in the x dimension are larger than in the other dimensions, the objects will be nearer or farther away from the camera. Hence, they will seem smaller or bigger. The fourth smaller function (random_*) is used to randomize the shape of an object a bit. It takes as input (i) the object, which should be randomized, and (ii) a range, which is proportional to how much the object should be randomized, and thus reshaped. This randomizing is done by relatively translating every node of the object. The rotation- and translation-ranges are predetermined. After careful consideration and multiple tests, ranges in Table 1 gave the best results. The ranges for the reshape-function depend on which object should be randomized and are described later in Table 2.

	X	Y	Z
Rotation	[0, 2π[	[0, 2π[	[0, 2π[
Translation	[-1.5, 1.5[	[-0.8, 0.8[	[-0.8, 0.8[

Table 1: Ranges for rotation and translation in every dimension

Next, a **fifth function** (apply_texture) that is responsible for assigning textures to objects was written. This function needs the object, material name of the texture and two booleans as input. The object describes which object should be changed to the specific texture, which is defined by the material name. However, when the first boolean is true, a random material will be chosen from material name, which is now a list. The second boolean refers to whether or not the object actually is a list of multiple objects. When this is true, every object in this list will get a texture assigned independently of the other objects. To apply the material to the object, the function follows three steps. First, it receives the actual material object via the given material name. Second, it determines the index of the given object, which can be seen as an ID. And third, it changes this material slot of the object to the actual material.

At a later stage in this project, leaves and branches were also added to the program. Every function mentioned above, can also be applied to these two different objects. Once again, multiple textures are made for every object. In total, there are 22 textures for leaves and 9 for the branches. Additionally, the textures of every object -apples, leaves and branches- will be selected from a list on a random basis. The previously mentioned reshape-ranges of these objects are shown in Table 2. The plus-sign determines how much every vertex can be changed at most in the positive x-, y- and z-direction, while minus limits the change in the negative directions. These values were determined by trial-and-error, such that the shape of objects remained realistic.

	Apple	Branch (small)	Branch (big)	Leaf
Minus	0.01	0	0.02	0.008
Plus	0.01	0.008	0.02	0.008

Table 2: Ranges for reshaping for every class

The above steps ensure that a synthetic 3D-scene can be generated with random objects and random textures. To receive different images from these scenes, the camera will rotate around the scene. In 200 steps, the camera will rotate around the scene while zooming in and out and rendering images every 20 frames. This is done by generating an empty with three axes in the middle of the scene. This empty will rotate around its own centre, which is also the centre of the scene. The rotation happens in the way it is told to, while the camera will only move on one axis of the empty in both directions, which provides a zooming effect. After the empty is done rotating around its own centre, 11 different images will be rendered.

To conclude, the first step of generating synthetic images is able to randomise a scene with different apples, branches and leaves. The number of apples, leaves and branches in every scene can easily be changed. However, in this project, they were set to 8 leaves, 4 apples and 5 branches. This seemed to be an appropriate number of objects, however, as will be proved later in this thesis, the number of leaves was too low. The five branches consist of four smaller branches and one bigger branch. However, both will be labelled as a tree. The above steps will be repeated several times. During this project, 11 images were rendered per scene and 250 scenes were made. Hence, 2750 different synthetic images were generated. A list of the changeable parameters and explanation is given in Appendix A.

Ground truth via Blender and Python

Besides the synthetic images, the software, written during this master's thesis, is also able to create the corresponding ground truth for every individual synthetic image. First, ground truth images are generated in Blender by changing the texture for every object. Next, these ground truth images are converted to annotations in COCO-format, since these can be saved into a JSON-file.

3.2.1 Pixel-wise label colouring of each object in Blender

After one scene is rendered and multiple (in this project 11) synthetic images are generated, the program written in Python and Blender will change the texture of every object in the scene to a unique colour, which can be seen as a label. This colour is defined via nodes. When a new object is generated, it will get its own unique object index, which will be used as an ID.

Using the nodes visualized in [Fig. 5], every object will get its own unique colour, which can be seen as a label. Via the first node, the object index is retrieved. Next, 1 is subtracted from this number, since the object index for the first object is equal to 1. Then, the number is divided by the number of objects in the scene, which is 17 in this project (8 leaves + 4 apples + 5 branches). This is done so the number will fit on a scale from 0 to 1. The next node will convert this number to specific RGB-values via a ColorRamp-node in Blender. Lastly, the colour is applied to the material via an emission-node. To change the texture of every object, the function to change textures, which is explained in the previous paragraph 'Synthetic Images in Blender', was used. However, in this case the list of materials will only contain one material which is the label. This label-texture will set a unique color to every object with regard to the object index via the nodes.

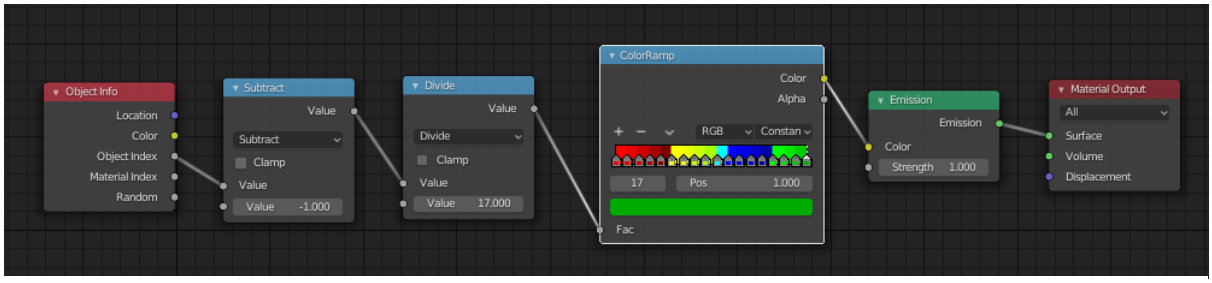

Figure 5: Blender nodes to generate ground truth images

Apart from the textures of every object, the sun should also be hidden in the render image. Furthermore, the 'Filter Size' in Blender should be set as small as possible, which is 0.01. This way, the transition between the object and the background is direct and not smooth. The background will be transparent. Thus, no shadow will be seen in the ground truth images, the contours of every object will be as sharp as possible, and the segmentation map will not be disrupted. Furthermore, the empty axis will rotate in the same way as previously mentioned, as will the camera. This will ensure that the same images will be rendered, while only the textures have been changed in label colours. The method described above generates ground truth images. However, a dataset in COCO-format only supports ground truth annotations, not images.

3.2.2 Converting colour labels into COCO-annotations

In the first approach during this project, every single bit mask, which corresponds to an annotation, was individually saved in a file. However, this takes too much space and can cause performance-issues. Furthermore, it slows the process down significantly. A better approach is to save the segmentations immediately to a Python dictionary, which will later be saved to a JSON-file. As mentioned previously, a dataset in COCO-format is a JSON-file consisting of multiple objects.

The first three objects, 'info', 'licences' and 'categories', can be hardcoded in Python. The next object, 'images', contains a list with information about every image that is generated in the previous section. This information holds a unique id, the filename and the width and height of the image. The fifth part, 'annotations', contains several subobjects. An annotation should be made for every single object in an image. So, in total, there should be 17 annotations per image in this project, unless an object is hidden behind another, larger object. The first subobject, 'image_id', is the id of the image which contains the object where this annotation corresponds to. The second sub-object, 'category_id', represents the id of the class to which this object belongs. Since there are three different classes during this project, there will be three different id's: 1 for an apple, 2 for a leaf and 3 for a branch. The third subobject in 'annotations', 'id', is just a unique id for the annotation. The fourth subobject, 'segmentation', contains every coordinate in the biggest contour of the object in the following format: [x1, y1, x2, y2, x3, y3, …, xn, yn]. In order to find these coordinates, various functions are used. The first function comes from scikit-image and will calculate the contour of the object in the bitmask. This bitmask only contains the object which should be annotated, the other pixels are left blank. Next, the contour is converted to another format and a polygon is made from these coordinates. Furthermore, this polygon will be simplified. Otherwise, the annotations would be too long, and the dataset would be too big. The polygon is one last time converted to the right format, which is mentioned previously. From this annotation, the fifth and sixth sub-objects, 'area' and 'bbox', respectively, are calculated. The former refers to the area of the contour, while the latter contains the maximum coordinates of the contour in order to hold the bounding box in following format: [xmin, ymin, width, height]. The seventh and last subobject,

‘iscrowd’, refers to whether multiple objects from the same class are annotated as one, as in a crowd. However, this should always be set to zero when creating a training dataset.

Using the methods above, a test and a smaller validate dataset will be made as well. The structure of the total dataset will look as follows:

appels

|-train

|-annotations

|-instances_minival2020.json

|-instances_train2020.json

|-val2020

|-train2020

The two json-files, instances_minival2020.json and instances_train2020.json, will contain the annotations of the real images to evaluate and the annotations of synthetic images to train respectively. The two other folders, val2020 and train2020, will contain these images as jpg-files.

Training and evaluation of Mask R-CNN

In this paragraph, first, it is explained how it is possible to use the previously generated dataset (consisting of synthetic images and ground truth) to train a Mask R-CNN on a server. Furthermore, this trained model is evaluated with a hand-made dataset, consisting of real images.

3.3.1 Training of mask R-CNN

During this project, the Mask R-CNN implementation of Matterport was used. This implementation uses Python3, Keras and TensorFlow, is built on a Feature Pyramid Network (FPN) and a ResNet 101 backbone and generates both segmentation masks and bounding boxes for each recognized object in the image. [4]

After cloning this repository from GitHub, two classes need to be extended: the config- and the dataset-class. Via the former, different parameters for training can be set. These parameters are the name, the images per GPU, the number of GPU’s, the number of classes that need to be trained and the minimum confidence for an object to be detected. In this project, the name was kept as ‘COCO’ and the number of images per GPU set at 1. However, the number of GPU’s was set to one, since only one GPU was used, the HP NVIDIA Quadro K5200. The method to install the right software and to train on a GPU can be found in Appendix B. Furthermore, the number of classes, including the background, was changed to 4 (apple, leaf, branch and background) instead of 81, while the minimum detection confidence was initially set to 0.7, which is the standard value in this Mask R-CNN implementation. When extending the dataset-class, the correct dataset can be implemented. Once again, the correct parameters need to be set. Thus, the correct class needs to be enumerated and the images and annotations will be added to this class. Lastly, the number of epochs (or times the algorithm will loop over the whole dataset) and the learning rate can also be changed per training step. The number of epochs cannot be too high, nor can they be too low. When they are too high, the model will overfit on the training dataset. If done correctly, both classes will be filled correctly when following command is ran:

```
python coco.py train --dataset=*PATH_TO_DATASET* --model=coco
```

For this project, training was divided into three stages. In the first stage only the network heads, which is the ResNet-101, were trained, and takes 60 epochs. The second stage takes 30 epochs and is supposed to finetune every layer from ResNet and up. The last stage will finetune every layer one more time and takes 30 more epochs.

3.3.2 Evaluation of model on apple orchard

In order to evaluate the previously trained model, a validation dataset was necessary. A small part of this dataset consists of images from Google. These images were manually annotated using makesence.ai [13]. However, these annotations were not in the right COCO-format, so a Python script was written in order to convert these annotations to the right format. The bigger part of this dataset, however, consists of real images from The University of Sydney in Australia. [14] These images were gathered by the Australian Centre for Field Robotics (ACFR) and consists of images of apples, mangos and almonds in an orchard. Only the former is needed during this project. Every apple in a picture is individually annotated via a circle. This circle has a centre (x- and y-coordinate) and a radius. These coordinates and distance are saved in a csv-file for every single image. Every row in this csv-file corresponds to a new annotation. In total, there are 5,765 apples in 1,120 images. For this master's thesis, a Python-script was written in order to convert the annotations from ACFR in the csv-files to COCO-annotations. First, the Python-script loops through every single annotation file (which corresponds to an individual image) and extracts the centre and radius of every annotation in the file. Because every point of this circle will be saved in the COCO-dataset, the circle will first be converted into a simplified polygon. Thus, only 6 points will be saved for every annotation, and consequently, the annotation will rather be in the form of a hexagon, while the annotation of the model proposed in this thesis will have the shape of the recognized apple itself. However, this should not cause any problems during the evaluation, since most of the areas will still overlap with each other.

Chapter 4

Results

Synthetic images in Blender

The goal of the first part of this master's thesis is to generate as many synthetic images of apple orchards as possible. Ideally, these images would look like the following real examples in [Fig. 6].

Figure 6: Idealistic examples for synthetic images

However, the objects in the background are less important than the objects in the foreground in most applications, like an apple-picking robot. Therefore, there is no need to render these objects. This means that they do not need to be recognized, but it would not be wrong when they are recognized by the model in this project. Two resulting synthetic images are shown below in [Fig. 7].

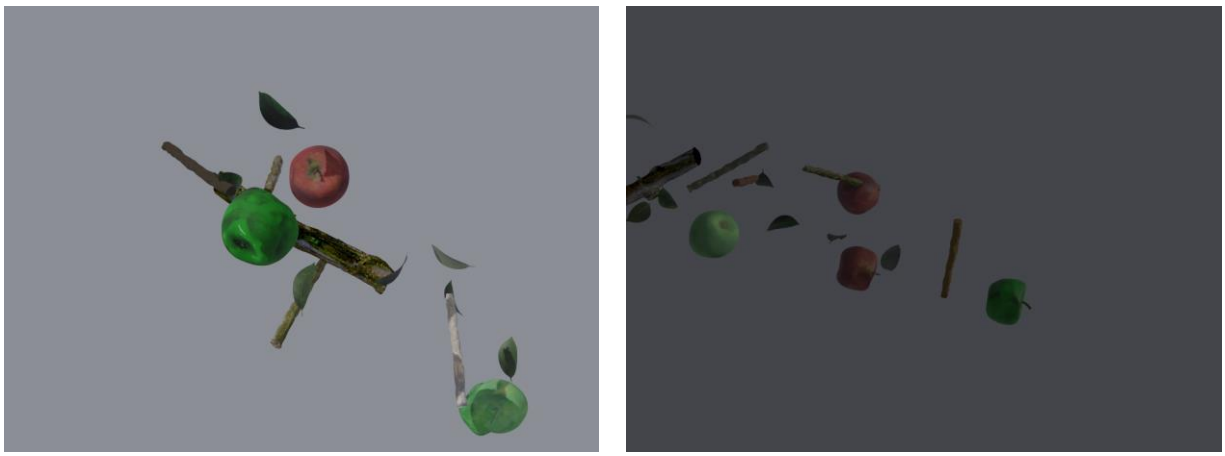

Figure 7: Results of synthetic images

Every apple has a slightly different shape, texture, rotation and translation, even in the same scene. However, the brightness is the same for every apple in one scene. Furthermore, as indicated above, some can be found in the foreground, while others are more in the background. However, since every apple is restricted to lie between certain borders in every dimension, they will all be converted into COCO-annotations later. The same is valid for the twigs and leaves in every scene.

There is a lot of variance for every apple. This will advance the learning process of the model. However, there is not so much variance for the twigs nor for the leaves: the models are not very different and there are not a lot of different textures. The models are also not very realistic. This will benefit the learning process, but the results will not be as good.

In total, the scenes are not very realistic. However, this should not cause any problems during this project since every object that should be detected is realistic most of the time, especially the apples.

Ground truth via Blender and Python

In this section, the resulting ground truth images, which were obtained via Blender and Python, are shown and discussed. Afterwards, the results of the conversion from images to COCO-annotations are shown as well.

4.2.1 Ground truth images

Whenever a scene is rendered, the texture of every object is changed to the label-texture. Thus, every object will have an individual colour, according to their unique id. The two ground truth images of the two synthetic images above can be seen below in [Fig. 8].

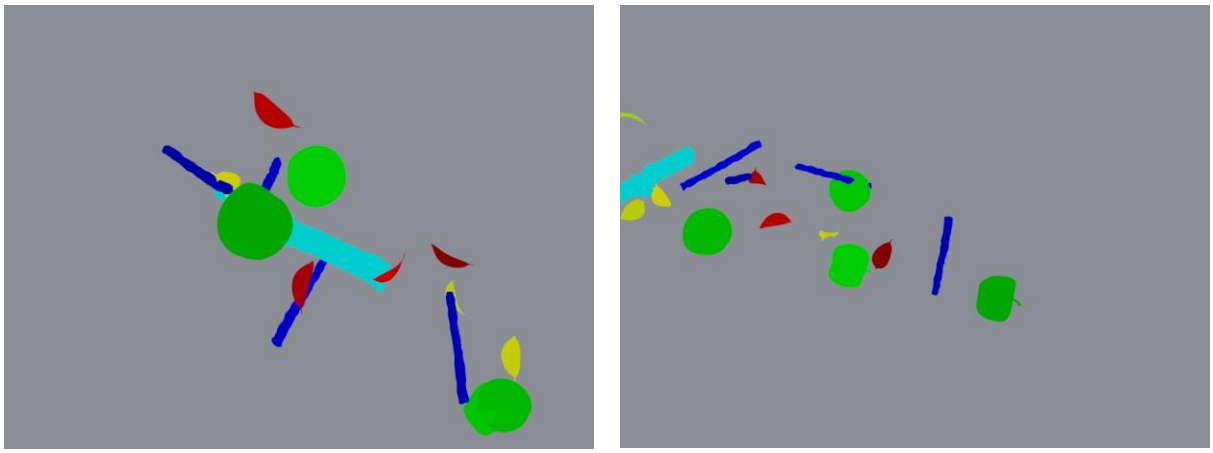

Figure 8: Results of ground truth images in Blender

Every object has an individual colour. For apples, these are different kinds of green, while branches and twigs have different kinds of blue and leaves are either yellow or red.

The run-time performance of the Python script for generating synthetic images depends on the resolution of these synthetic images and is shown in Table 3. These images are rendered on the specs shown in Table 4 below.

Resolution (px)	Time (sec)
80x60	34.29
160x120	37.24
240x180	44.84
320x240	53.70
400x300	65.15
480x360	81.38
560x420	100.33
640x480	123.06
720x540	144.58
800x600	168.40
880x660	195.22
960x720	231.61

Table 3: Performance of Python script in function of resolution of synthetic images

	HP Elitebook G5 850
OS	Ubuntu
CPU	Intel i5-7200U (2.50GHz)
GPU	Intel HD Graphics 620 (Kaby Lake GT2)
RAM	16 GB

Table 4: Specs of PC used to generate synthetic images

Based on the graph in [Fig. 9] below, the best resolution lays between 560x420 and 640x480 px. This is the point where the performance of the program starts to descend a bit more, while the resolution does not rise as quickly. For this master's thesis, a resolution of 640x480 px was chosen.

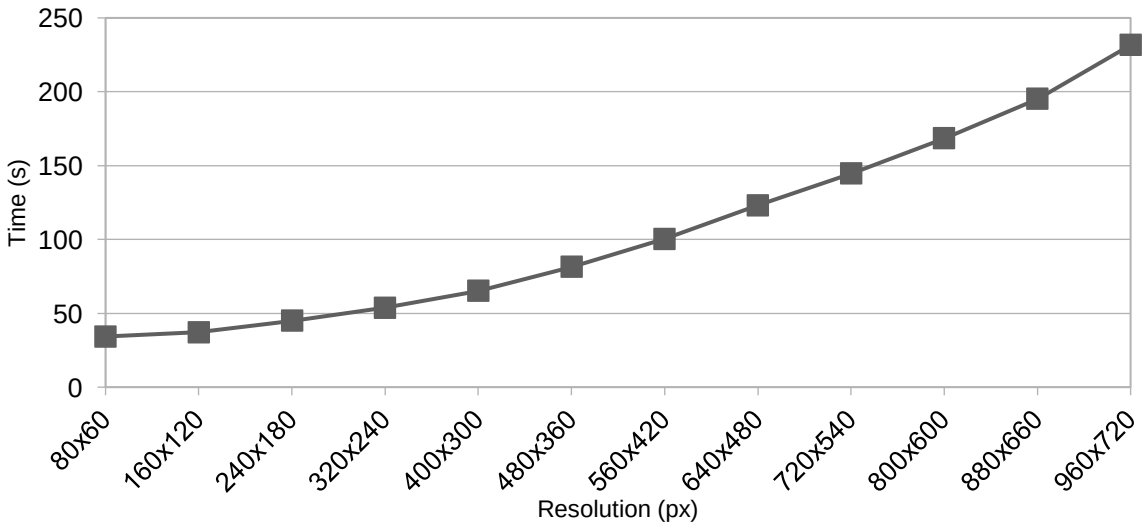

Figure 9: Run-time performance of Python script in function of image resolution

4.2.2 Annotations for COCO-dataset

The above images are converted to actual annotations in a json-file, as explained in paragraph 3.2.2. For example; the annotations in the 2750 synthetic images are converted to a JSON-file of size 12.0 MB. A snippet of this file can be found in Appendix D. To check these annotations, they are reconverted and pasted on a corresponding random image. Some random results are shown below in [Fig. 10]. This is basically the input data, from which the model will try to learn, visualized.

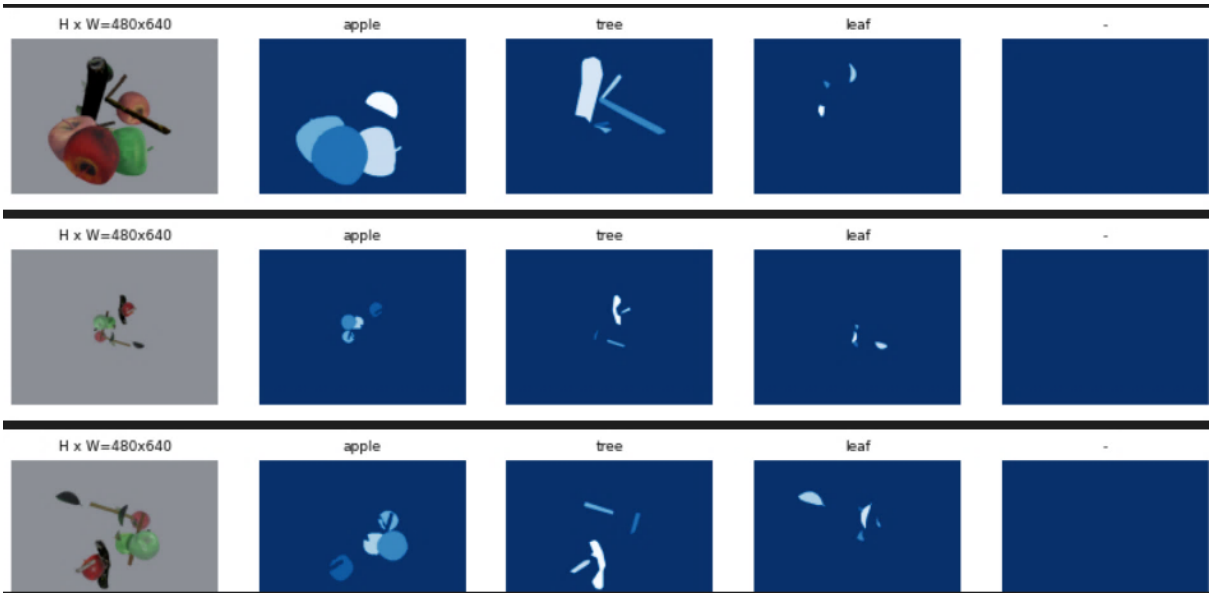

Figure 10: Validation of synthetic COCO-annotations

Training and evaluation of Mask R-CNN model for apple detection

In this paragraph, the results after training the Mask R-CNN model, as explained in paragraph 3.3.1, are shown and discussed. To evaluate the accuracy of the MCT values are used.

4.3.1 Evaluation (MCT = 0.95) after training with apples and obstacles

The first results came after training with only apples. No leaves, nor branches were present in the synthetic images. The results were very bad, with an F1-score approximating 0, whereas the goal is to reach a F1-score towards 1. Two examples are shown below in [Fig. 11]. A bigger version of these images can be found in Appendix E.

Figure 11: Results after training with only apples (MCT=0.95)

Leaves were almost always annotated as an apple, since the model has no reference to make a distinction between an apple and a leaf. This demonstrates the importance of adding leaves to the synthetic images. This way, the model will learn to make a distinction between apples and leaves. Not only will it be able to learn how to recognize leaves, more importantly in this project, it will also know not to annotate leaves as apples. Results from this implementation with the input images from [Fig. 6] are shown below in [Fig. 12], while a bigger version can be found in Appendix F.

Figure 12: Results after training with apples and random objects (leaves and branches) (MCT=0.95)

4.3.2 Evaluation (MCT = 0.7) after training with apples and obstacles

Now that the importance of obstacles (leaves or branches) is proven, the model can be evaluated with multiple MCTs. These obstacles can be leaves or branches, as explained above, for example. The lowest MCT that is discussed in this master's thesis is 0.7. The results are shown in [Fig. 13] and a bigger version can be found in Appendix G.

Figure 13: Results after training with apples and random objects (leaves and branches) (MCT=0.7)

Since the MCT is rather low, the model will be rather flexible and not strict. Thus, almost every apple will be correctly annotated: there are only 4 FNs, while there are 568 TPs. However, with 938 FPs, there are too many FPs. Since the number of FNs is very low, the AR is very high with 99.3%. On the other hand, since the number of FPs is too high, the AP is very low with only 37.7%. This results in a very bad F1-score of 0.55.

4.3.3 Evaluation (MCT = 0.99) after training with apples and obstacles

Conversely, the Mask R-CNN model can also be made very strict. This way, the model must be very sure before annotating an object. The resulting images are shown in [Fig. 14], while a bigger version of these images can be found in Appendix H.

Figure 14: Results after training with apples and random objects (leaves and branches) (MCT=0.99)

This approach induces a sharp decrease in total number of FPs to only 85 over the whole dataset, which brings the AP to 81.7%. However, since the model is not as flexible anymore, the number of FNs increases to 51, which causes the AR to decrease towards 88.2%, which is still rather good. Lastly, the F1-score rose to 0.85. This already gave a much better result than the lower MCT of 0.7.

4.3.4 Evaluation with multiple MCT-values

In order to find how strict, or flexible, the Mask R-CNN model should annotate different objects, the evaluation is done with multiple MCT-values. As mentioned in paragraph 4.3.2, the lowest MCT that will be discussed in this master's thesis is 0.7. The highest MCT, however, is the absolute maximum at 0.99. A plot of the AR, AP and F1-score in function of the MCT over 50 different validate images is given in [Fig. 15].

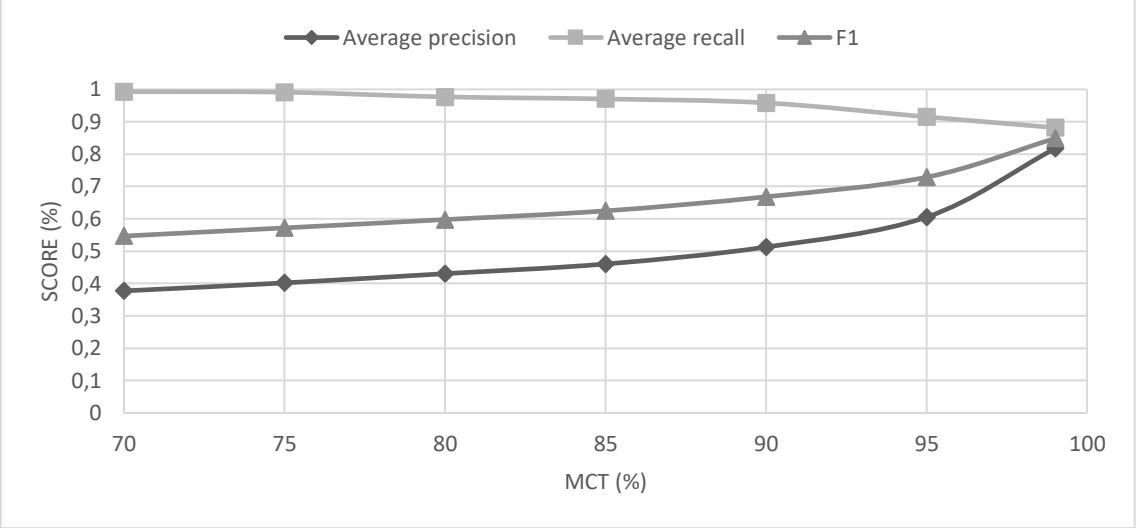

Figure 15: AR, AP and F1 in function of MCT for own model

While the AR stays rather the same, the AP starts increasing from the beginning and increases sharply after an MCT of 0.95. Because the AR is more or less constant at 1 until the MCT reaches 0.90, the AR will not induce any change in the F1-score. Thus, the F1-score will increase parallel with the AP until the MCT reaches 0.90. Afterwards, the AR starts decreasing slightly, which causes the F1-score to increase less sharply. When the MCT is equal to 0.99, from which the results were discussed previously, the F1-score reaches its highest point. This means that the Mask R-CNN model should be initialized as a rather strict model. However, this depends on the purpose of the model. But, in general, a higher F1-score is better.

The flexibility or strictness of the model can also be explained by [Fig. 16]. This histogram plots the frequency with which every true positive is annotated.

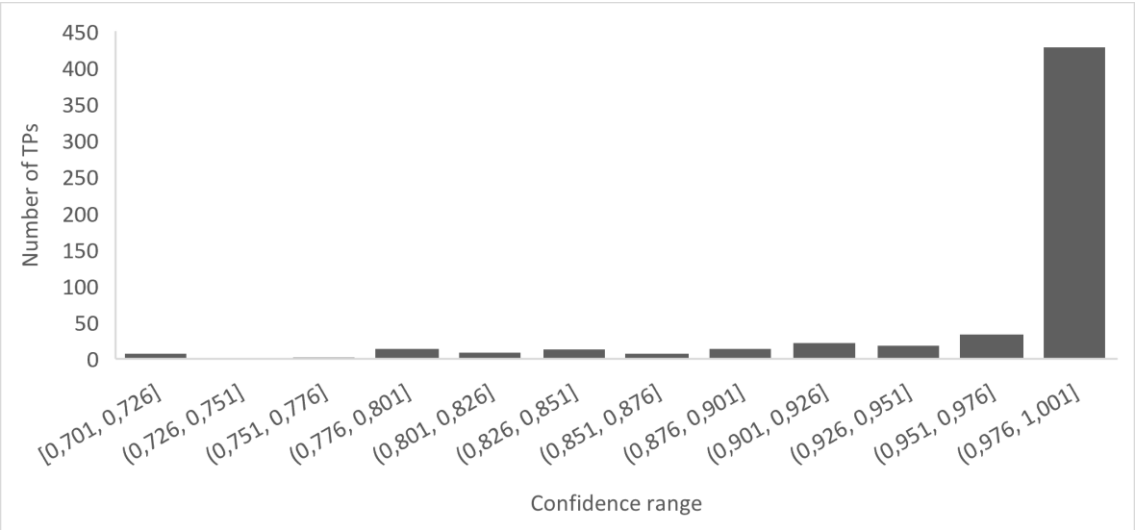

Figure 16: Number of TPs in function of confidence range for own model

From this plot it can also be derived that the AR, which is proportional to the number of TPs and FNs, will stay rather the same. It will also decrease slightly after the MCT reaches 0.95. Furthermore, it is important to note that the model is almost always 100% sure that an object belongs to the apple class, whenever it annotates this object as an apple.

4.3.5 Comparison with COCO dataset trained Mask R-CNN

Since transfer learning, which means that the learning process was started from an already existing model, was used to train the model proposed in this master's thesis, it is important to make a comparison with this first model. The model that was used to perform transfer learning is a Mask R-CNN previously trained with the standard COCO dataset. This dataset consists of 80 classes, including an apple class. However, the model is actually more trained to detect apples in a store or on a kitchen table, not in environments such as an orchard. The comparison is visualized in [Fig. 17].

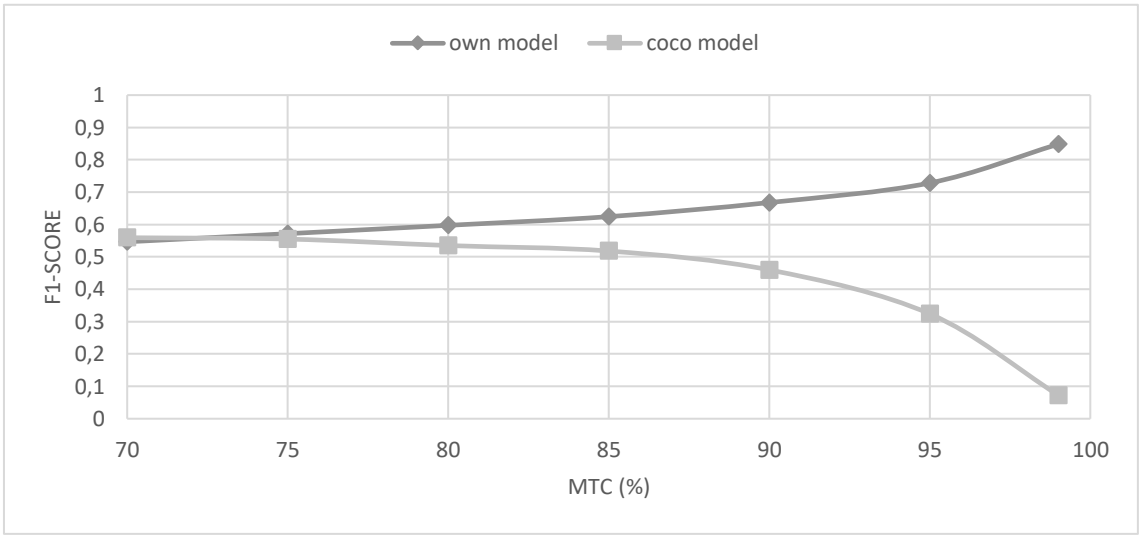

Figure 17: F1-score in function of MTC for model trained in this paper and COCO model

Both models start off with an F1-score of about 0.55 at an MTC of 70%. The F1-score of the own trained model increases rather quickly after it reaches 0.85. However, while the own trained model experiences an increase, the F1-score of the COCO model decreases somewhat exponentially until it reaches an F1-score of 0.07 at MTC equal to 0.99, while it still had a F1-score of 0.32 when the MTC is 0.95. The transfer learning clearly benefitted the Mask R-CNN model in the evaluation-circumstances. These circumstances are environments, like a farm or an orchard, in which the model proposed in this paper was trained. The AR, AP and F1 for the Mask R-CNN model trained with the standard COCO dataset in function of an increasing MCT, like [Fig. 15], is visualized in [Fig. 18]. Once again, this model was evaluated by images of the necessary environments during this master's thesis.

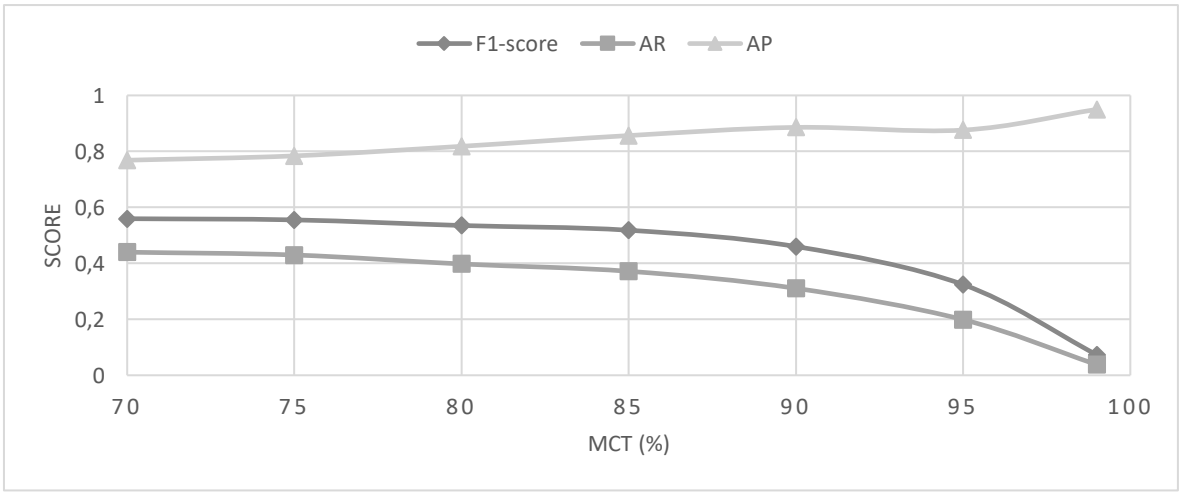

Figure 18: AR, AP and F1 in function of MCT for COCO model

In contrast to the own trained model, the AR decreases rather fast with an increasing MCT, while the AP increases just slightly. Thus, the number of FNs increases a lot with an increasing MCT, while the

number of FPs stays rather low. This means that the model, trained with the COCO standard dataset, needs to get less strict guidelines (lower MCT) to give the most accurate annotations.

Training and evaluation of Mask R-CNN for apples-, leaves- and branches-detection (MCT = 0.85)

In order to prove the functionality of the Blender-program, the Mask R-CNN model was also taught to annotate leaves and branches more or less correctly. In order to do this, only two models were made in Blender: one for a leave and one for a branch. The program randomized these models, as explained in 0, and applied a provided texture. However, these models and textures were not sufficient, nor were they random enough. Two examples from [Fig. 6] are shown below in [Fig. 19].

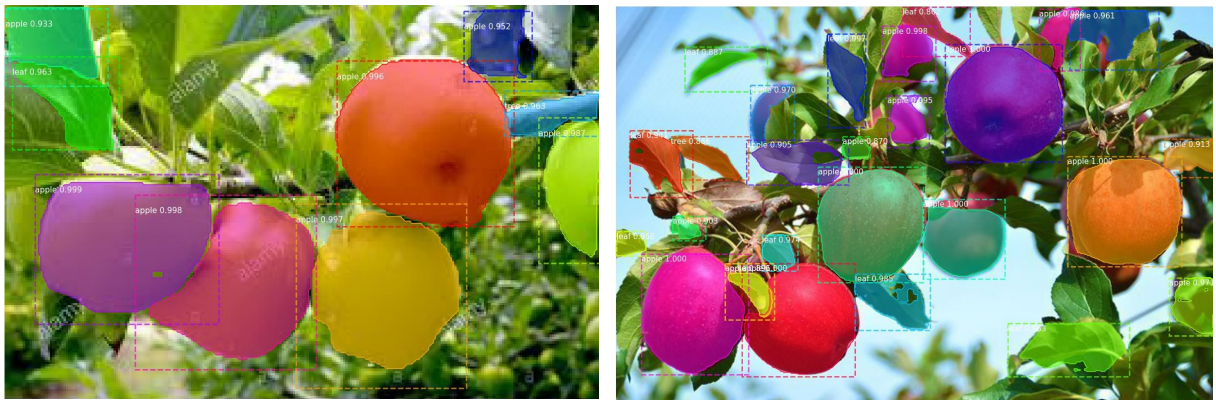

Figure 19: Results after training and detection of three classes (apples, leaves and branches)

It is obvious that the leaves and branches are not as well recognized as the apples. This could be explained by the fact that not much effort was put in the creation of the model and the texture of both the leaves and the branches; there was not a lot of variety in the textures and the models were too basic and unrealistic. Moreover, an insufficient number of leaves and branches were put in the scenes. However, the addition of the models was only to prove the flexibility of the Blender-program and how easy it is to add extra objects.

Chapter 5

Conclusion & Future work

During this master's thesis, a program has been written which offers the user the opportunity to generate synthetic images with the corresponding ground truth in COCO-format. These generated images are composed of multiple objects and textures given from the user. The program will first randomize these objects and apply a random texture. Furthermore, it will save all the necessary information along with the annotation and ground truth to a json-file.

After creating the synthetic images and corresponding ground truth, this approach was tested by training a Mask-RCNN to detect mostly apples, but also leaves and branches, in an orchard. It was clear that the model of the objects needs to be as accurate as possible. Furthermore, multiple textures were needed for every object. During this project, 20 apple-textures were used, which gave an acceptable result. However, more textures can only give better results. Finally, the importance of multiple random objects in every scene was proven. This allows the model to distinguish objects from the background. The program in Blender written in Python is also able to add these random objects whenever a model for these objects is given by the user.

The evaluation of the trained Mask R-CNN model gave an F1-score of 0.73 when MCT was set at 0.95, and a F1-score of 0.85 when MCT was set at 0.99, based on the detection of apples only. The number of false positives was higher than the number of false negatives, which approaches zero. However, these number of false positives can be lowered when improving the models of leaves. In comparison with the pretrained Mask R-CNN model with the standard COCO-dataset, this is an improvement. This pretrained model had an F1-score of 0.56 when MCT was set at 0.7 and an F1-score of 0.07 when MCT was set at 0.99.

Overall, it can be concluded that a Mask R-CNN model is successfully trained using synthetic images. These synthetic images were generated using a very flexible Python script, which was written during this master's thesis, in Blender. Furthermore, using transfer learning, an existing model is also improved to detect apples in an orchard.

The main goal is to further improve the detection of apples in orchard images. Several possible improvements of the current approach are proposed. First, the model of the apple can be improved; there can be more variety in the shapes and textures. This way, the number of false negatives will decrease, which will cause a higher AR. Second, the AP can also be increased. This can be achieved by adding more random objects, such as pears, poles… This way, the model will learn that these random objects are no apples, by being able to make a distinction between those objects and an apple. Another, third, possibility to improve is to improve the models of the leaves and branches. As mentioned before, not much effort was put into these objects. However, when these models are improved, both the AR and the AP will increase, which will cause a higher overall F1-score. The AR, when using the model that only detects apples, will increase since more leaves and branches will be detected. Simultaneously, the AP will also increase, because fewer leaves will be annotated as apples, which means fewer false positives. Finally, more leaves should also be added to every scene, since there are a lot of leaves in the real images. For now, only 8 leaves are generated in each scene, which is not enough. In the future, a model for fire blight can also be invented and created. Using the Python-framework proposed in this thesis, thousands of synthetic images of fire blight in orchard can be generated. This dataset can then be used to train a model to detect fire blight, as was the initial goal of this thesis.

Bibliography

[1] P. R. V. I. Nicolai Häni, “Apple Counting using Convolutional Neural Networks,” in *International Conference on Intelligent Robots and Systems*, Madrid, 2018.

[2] A. P. D. A. M. B. V. J. C. A. T. T. E. C. S. B. S. B. Jonathan Tremblay, “Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization,” Toronto, 2018.

[3] B. Sahu, “The Evolution of Deeplab for Semantic Segmentation,” Medium, 12 07 2019. [Online]. Available: https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571. [Accessed 07 02 2020].

[4] matterport, “Mask R-CNN for Object Detection and Segmentation,” 01 04 2019. [Online]. Available: https://github.com/matterport/Mask_RCNN. [Accessed 12 11 2020].

[5] S.-H. Tsang, “Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection),” Medium, 15 09 2018. [Online]. Available: https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8. [Accessed 13 02 2020].

[6] T. Karmarkar, “Region Proposal Network (RPN) — Backbone of Faster R-CNN,” Medium, 19 08 2018. [Online]. Available: https://medium.com/egen/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9. [Accessed 13 02 2020].

[7] R. Khandelwal, “Computer Vision: Instance Segmentation with Mask R-CNN,” Medium, 31 07 2019. [Online]. Available: https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1. [Accessed 13 02 2020].

[8] A. F. Joseph Redmon, “YOLO: Real-Time Object Detection,” arXiv, 2018. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed 13 02 2020].

[9] M. M. S. B. L. B. R. G. J. H. P. P. D. R. C. L. Z. P. D. Tsung-Yi Lin, “Microsoft COCO: Common Objects in Context,” p. 15, 2015.

[10] T. Sarkar, “Synthetic data generation — a must-have skill for new data scientists,” Medium, 19 12 2018. [Online]. Available: https://towardsdatascience.com/synthetic-data-generation-a-must-have-skill-for-new-data-scientists-915896c0c1ae. [Accessed 16 02 2020].

[11] A. P. D. A. M. B. V. J. C. A. T. T. E. C. S. B. S. B. Jonathan Tremblay, “Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization,” p. 9, 2018.

[12] K. P. Shung, “Accuracy, Precision, Recall or F1?,” Medium, 15 03 2018. [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. [Accessed 22 02 2020].

[13] P. Skalski, “MakeSense,” 2019. [Online]. Available: https://github.com/SkalskiP/make-sense/. [Accessed 12 04 2020].

[14] “Deep Fruit Detection in Orchards,” 2016. [Online]. Available: http://data.acfr.usyd.edu.au/ag/treecrops/2016-multifruit/. [Accessed 02 12 2020].

[15] The SciPy Community, “NumPy v1.19 Manual,” NumPy, 29 06 2020. [Online]. Available: https://numpy.org/doc/stable/. [Accessed 05 08 2020].

[16] SciPy developers, “Scientific computing tools for Python¶,” SciPy, 2020. [Online]. Available: https://www.scipy.org/about.html. [Accessed 05 08 2020].

[17] A. Clark, “Pillow,” 2020. [Online]. Available: https://github.com/python-pillow/Pillow/blob/2bd74943fb9f320def6c066e732b701d1c15f677/docs/index.rst. [Accessed 05 08 2020].

[18] R. B. L. D. M. F. V. M. D. S. S. Stefan Behnel, “Python C-Extensions for Python,” Cython, 07 05 2020. [Online]. Available: https://cython.org/. [Accessed 05 08 2020].

[19] The Matplotlib development team, “Matplotlib: Visualization with Python,” Matplotlib, 17 07 2020. [Online]. Available: https://matplotlib.org/. [Accessed 05 08 2020].

[20] J. L. S. J. N.-I. F. B. J. D. W. N. Y. E. G. T. Y. Stéfan van der Walt, “scikit-image,” 2014. [Online]. Available: https://scikit-image.org/. [Accessed 05 08 2020].

[21] TensorFlow, “An end-to-end open source machine learning platform,” 2020. [Online]. Available: https://www.tensorflow.org/. [Accessed 05 08 2020].

[22] Keras, “Keras,” 2020. [Online]. Available: https://keras.io/. [Accessed 05 08 2020].

[23] OpenCV team, “OpenCV,” 2020. [Online]. Available: https://opencv.org/. [Accessed 05 08 2020].

[24] A. Collette, “HDF5 for Python,” 07 09 2019. [Online]. Available: https://www.h5py.org/. [Accessed 05 08 2020].

[25] A. Jung, “imgaug,” 2020. [Online]. Available: https://imgaug.readthedocs.io/en/latest/. [Accessed 05 08 2020].

[26] The IPython Development Team, “IPython Documentation,” 26 06 2020. [Online]. Available: https://ipython.readthedocs.io/en/stable/. [Accessed 05 08 2020].

Appendixes

Appendix A: parameters for domain randomization in Blender via Python

- Position of object
  The position of every object will be set at random, however, the borders within these objects must be placed can be chosen. These translation ranges can be found in [Tab. 1].

- Rotation of object
  The rotation of every object will be set at random, however, the borders within these objects can rotate can be chosen. These rotation ranges can be found in [Tab. 1].

- Shape of object
  The shape of every object will be randomized by translating every vertex a little bit. The ranges in which every vertex can be moved depend on the type of class to which the object belongs and are shown in [Tab. 2].

- Size of object
  The size of every object will also be changed randomly. However, this is done by zooming in and out with the camera, not by changing the objects physically.

- Texture of object
  The texture of the object will be randomly chosen from a list. This list contains multiple self-made textures. The list of textures should correspond to the type of class. This way, a leave-texture cannot be applied to an apple, for example.

- Amount of objects
  In every scene, the number of objects for every class can also be changed. During this master's thesis. The number of apples were set to 4, the number of branches to 5 and the number of leaves to 8.

- Camera position
  It is possible to determine the path the camera should follow. During this master's thesis, this was done by adding an extra empty. This way, the camera will rotate around the whole scène while zooming in and out. This means that an image will be taken from multiple different angels and the objects will seem smaller or bigger.

- Focus point of camera
  Apart from the position, the focus point of the camera will also change multiple times. This way, the model is able to detect objects that are not in focus in a real image. However, once again, ranges were set.

- Resolution of synthetic image
  The resolution of every synthetic will also be changed randomly. However, this is not as useful in this implementation, since Mask R-CNN automatically resizes every image to a fixed size of 1024x1024 px.

- Light exposure

The light exposure in every scene can also be set to a fixed number. However, in this project, the exposure changes randomly every scene. This is achieved by making the sun in Blender follow a random trajectory. This method will also cause a random angle of light incidence in every scene.

- Brightness
  Apart from the exposure, the brightness will also be random when using the previous approach of making the sun follow a random trajectory and random angle of light incidence.

- Amount of images
  The number of images can also be set as wished. First, it is possible to choose the number of scenes that should be created. Additionally, the user can also choose how many images the camera should capture in every scene. The multiplication of both of these numbers will determine the number of total images. During this project, the camera will capture 11 images in 250 different scenes, which results to 2750 different synthetic images.

Appendix B: Setting up server to train Mask R-CNN

git clone https://github.com/matterport/Mask_RCNN
virtualenv venv
source ./venv/bin/activate
pip3 install -r requirements.txt; these requirements can be found with short explanation in Appendix C
python setup.py install –user
wget https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5
cd ..
git clone https://github.com/cocodataset/cocoapi.git
cd PythonAPI
make
python3 setup.py install –user
pip3 uninstall tensorflow
pip3 install tensorflow-gpu==1.14.0
to train on GPU, follow https://ruthwik.github.io/machinelearning/2019-08-12-tensorflow_gpu/
pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI

Appendix C: requirements necessary to train Mask R-CNN

numpy – package for scientific computing with Python [15]

scipy – scientific computing tools for Python [16]

Pillow – Python image library [17]

Cython – optimizing static compiler for Python [18]

Matplotlib – a comprehensive library for creating static, animated, and interactive visualizations in Python [19]

scikit-image – image processing in Python [20]

tensorflow>=1.3.0 – end-to-end open source platform for machine learning [21]

keras>=2.0.8 – a Python deep learning API [22]

opencv-python – a computer vision library [23]

h5py – Pythonic interface to the HDF5 binary data format [24]

imgaug – Python library for image augmentation in machine learning experiments [25]

IPython[all] – framework to use Python interactively with a notebook [26]

Appendix D: Snippet of COCO-annotations in JSON-file


```
{
"info":
        {"description": "Synthetic COCO Dataset for fireblight",
        "url": "",
        "version": "1.0",
        "year": 2020,
        "contributor": "Senne Colson",
        "date_created": "2020/03/12"},
"licenses": [],
"images":  [{"id": 1, "license": 0, "width": 640, "height": 480, "file_name": "image_0_frame_0_texture.jpg",
"date_captured": ""},
                        {"id": 2, "license": 0, "width": 640, "height": 480, "file_name":
"image_0_frame_100_texture.jpg", "date_captured": ""},
                        {"id": 3, "license": 0, "width": 640, "height": 480, "file_name":
"image_0_frame_200_texture.jpg", "date_captured": ""}],
"categories": [ {"id": 1, "name": "apple", "supercategory": "fruit"},
                                {"id": 2, "name": "leaf", "supercategory": "tree"},
                                {"id": 3, "name": "tree", "supercategory": "tree"}],
"annotations":
        [{"segmentation": [[410.0, 165.5, 402.5, 167.0, 385.0, 231.5, 399.5, 223.0, 416.5, 169.0, 410.0, 165.5]],
        "iscrowd": 0,
        "image_id": 1,
        "category_id": 3,
        "id": 0,
        "bbox": [385.0, 165.5, 31.5, 66.0],
        "area": 806.25},
        {"segmentation": [[247.0, 132.5, 241.5, 143.0, 246.0, 146.5, 331.5, 171.0, 333.0, 160.5, 247.0, 132.5]],
        "iscrowd": 0,
        "image_id": 1,
        "category_id": 3,
        "id": 1,
        "bbox": [241.5, 132.5, 91.5, 38.5],
        "area": 1116.5}]
}
```

Appendix E: Results after training with only apples (MCT=0.95)

Appendix F: Results after training with apples and obstacles (leaves and branches) (MCT = 0.95)

Appendix G: Results after training with apples and obstacles (leaves and branches) (MCT = 0 .7)

Appendix H: Results after training with apples and obstacles (leaves and branches) (MCT = 0 .99)

Appendix I: Results after training with apples, leaves and branches for three-class detection (MCT=0.85)