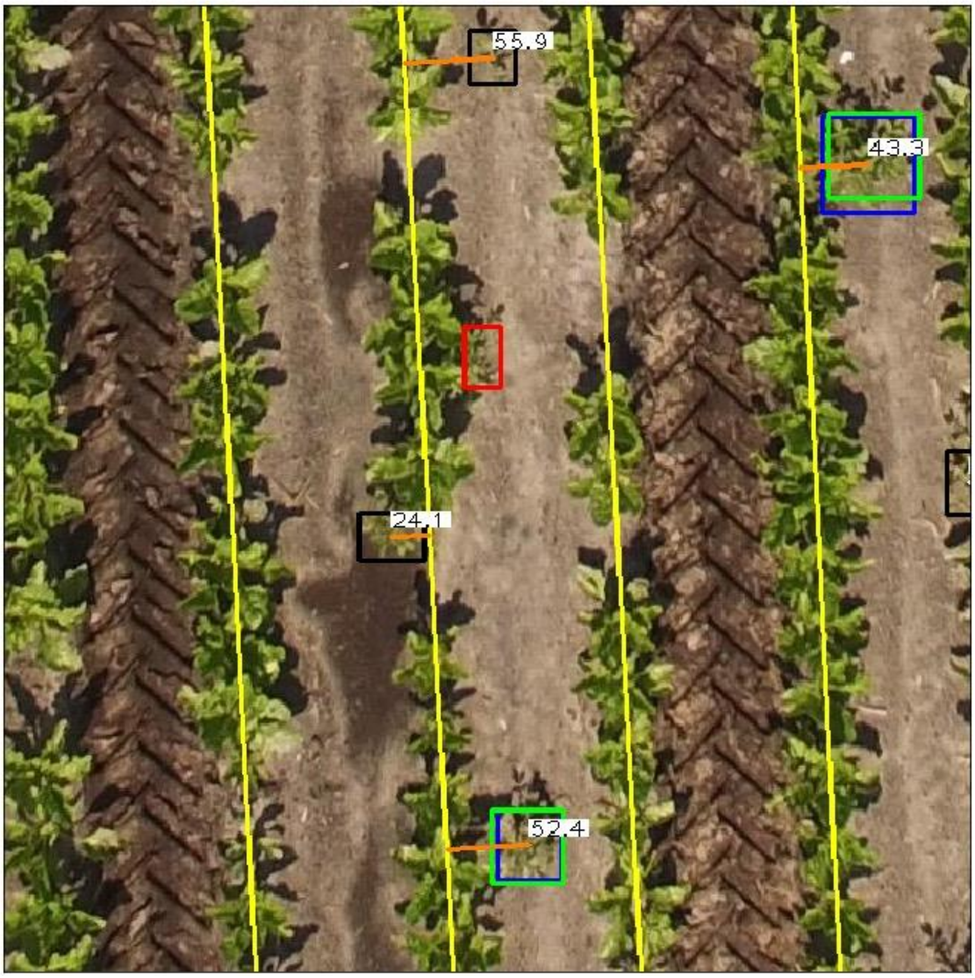
Wageningen University & Research

Thesis Report GIRS-2019-18

Comparing UAV-based image resolution to deep-learning weed-detection performance

Sebastian Paolini van Helfteren



21-05-2019

Comparing UAV-based image resolution to deep-learning weed-detection performance

Sebastian Paolini van Helfteren

Registration number 95 11 20 641 120

Supervisors:

dr. ir. Lammert Kooistra
dr. Gert Kootstra

A thesis submitted in partial fulfilment of the degree of Master of Science
at Wageningen University and Research Centre,
The Netherlands.

21-05-2019
Wageningen, The Netherlands

Thesis code number: GRS-80436
Thesis Report: GIRS-2019-18
Wageningen University and Research Centre
Laboratory of Geo-Information Science and Remote Sensing

ACKNOWLEDGEMENTS

This thesis is the conclusion of my master's degree in Geo-Information Science at Wageningen University. At least, that's what I hope it is. This past year I have been working on this document and although it's the longest document that I have written, when I look back on it, I can't help but wonder how much time I could have saved if I had *just known the answer*. Of course, this is a naïve thought. Part of the beauty of science is slowly putting the pieces together; getting those *a-ha!* moments; and staring angrily at a computer for extended periods of time. In the end, I can say that I learned a lot about many things I knew absolutely nothing about, improved my understanding about many things I knew little about, and realised that there are many more things which I still know nothing about. Just as important, I realised that I actually quite enjoyed the experience and would still like to keep on researching the many questions that this research has provoked.

Logically, I couldn't have written this thesis without the support of friends and family, background knowledge given to me by the various lecturers at Wageningen University, and all those who wrote the literature used in this research. On a more specific note, I would like to thank a couple of people:

Lammert Kooistra, who helped me stay grounded and look at things from a more intuitive perspective. Not only did he give me advice to make my research more scientifically sound, he also taught me that it's sometimes just as good to use my *boerenverstand*.

Gert Kootstra, who helped me stay realistic and set my priorities straight throughout my research. His critical comments often corrected my research trajectory towards one that was more logical, meaningful, or both. Also, special thanks for helping me with the problems I had with my Python script. Analysing the script that you helped me with was one of the most educative elements of this thesis and helped me gain confidence in writing my own scripts.

Thijs Ruigrok, for being of immense help in setting up YOLOv3 and helping me start my thesis in a relatively smooth way. When I started my thesis, he was the one I went to for advice on practical matters – which he always gave enthusiastically.

Martina Vittorietti, for giving me great advice and helping me in all statistical matters. Her enthusiasm for data was contagious and helped me look at data in a new way.

Andrew Ng, for teaching me all the basics of deep learning and object-detection models through his YouTube videos. I decided upon this research subject as I was interested in machine learning, but did not know anything about it. His excellent videos gave me the background knowledge needed to at least understand the papers I needed.

On a personal note, I would like to thank my family and all those whom I hold close for listening (or pretending to listen) to whatever issue was on my mind that day, and for simply being part of my life.

DECLARATION

I, Sebastian Paolini van Helfteren, hereby declare that this thesis is my work. The best possible care was taken to ensure the scientific integrity of this research. External sources were referenced where applicable, taking into account the validity of these sources. To my knowledge, this thesis does not violate any copyright laws or partake on plagiarism.

Sebastian Paolini van Helfteren
21-05-2019

ABSTRACT

Environmental degradation due to conventional chemical weed-management, is a widespread issue in most agricultural land. Precision agriculture aims at sustainable agricultural production; reducing inputs by applying these precisely in space and time. Deep-learning-based object-detection systems, such as YOLOv3, can prove valuable in future agricultural systems to detect weeds and can contribute to sustainable agriculture by applying inputs precisely where they are needed. Such a detection system can be based on an Unmanned Aerial Vehicle (UAV), which is a fast and mobile possibility for weed detection and control. However, these aerial weed-detection systems are in their infancy: the practical feasibility, and relationship between image resolution and system performance is unknown. This research uses YOLOv3 to detect weeds on images taken from a UAV at different resolutions. Multiple datasets are created using k-fold cross validation. This research shows that, theoretically, such a system would economically benefit farmers and that a higher resolution is significantly tied to a better performance. However, this was not the case for all runs of the cross validation, indicating a more complex relationship between field characteristics and performance. Compared to other, more optimised weed-detection systems, results were in line with related work. This research explains how the practical aspect of altitude affects model performance (i.e. relating image resolution to model performance), and further delves into other aspects which influence performance; namely, object viewing angle and the underlying cause of false negative errors. Also, it’s explored how performance differs for orthomosaic images; bringing this research closer to the discipline of remote sensing. The results found demonstrate a possibility for future UAV-based weed management, with an average F1-score, precision, recall and mAP of 0.83, 0.95, 0.77 and 0.75 respectively for the original image resolution. This research can be used as a stepping stone for future research which aims to bring theory to practice. To bring a deep-learning-based UAV weed-detection system to practice, lightweight GPUs which can run such models in real-time are currently the limiting factor.

Keywords: UAV, deep-learning, YOLOv3, weed detection, image resolution, viewing angle, orthomosaic, performance, precision agriculture

LIST OF FIGURES

FIGURE 1: PROPOSED REGIONS. MODIFIED FROM UIJLINGS ET AL. (2013). 6
FIGURE 2: VISUALISATION OF COMPARISON BETWEEN ROIS AND POSITION-SENSITIVE SCORE MAPS USED IN THE R-FCN. IF A POSITION-SENSITIVE SCORE MAP IDENTIFIES PATTERNS ASSOCIATED TO A STORED OBJECT, IT RETURNS A YES (LIGHTER COLOUR ON THE RIGHTMOST GRID). TAKEN FROM DAI ET AL. (2016). 7
FIGURE 3: YOLOV3 NETWORK ARCHITECTURE. TAKEN FROM KATHURIA (2018). 9
FIGURE 4: VGG-16 NETWORK ARCHITECTURE. TAKEN FROM POUDYAL (2018). 10
FIGURE 5: SSD NETWORK ARCHITECTURE. TAKEN FROM LIU ET AL. (2016). 10
FIGURE 6: SCHEMATIC OVERVIEW OF THE METHODOLOGY PER RESOLUTION. 14
FIGURE 7: TEST PLOTS OVERVIEW. 14
FIGURE 8: OVERLAPPING IMAGES AND UAV FLIGHTPATH. ALTHOUGH SOME AREAS HAVE A SMALL NUMBER OF OVERLAPPING IMAGES, THE STUDY AREA IS FOUND IN THE CENTRE OF THE FIELD. 15
FIGURE 9: EXAMPLE OF LABELLED WEEDS. WEEDS COME IN VARIOUS SHAPES AND SIZES. 15
FIGURE 10: TWO SCHEMATIC OVERVIEWS OF THE FIELD TO ILLUSTRATE THE DIFFERENCE IN DATA SPLITTING FOR TWO DIFFERENT RUNS OF THE MODEL. A: 84% OF THE DATA (RED) WILL BE USED FOR TRAINING, 3% (YELLOW) AS BUFFER AND 14% (BLUE) FOR TESTING. B: 81% OF THE DATA (RED) WILL BE USED FOR TRAINING, 5% (YELLOW) AS BUFFER AND 14% (BLUE) FOR TESTING 16
FIGURE 11: SUMMARY OF METHOD TO LOWER IMAGE RESOLUTIONS. TWO RESIZED DATASETS ARE CREATED USING THIS METHOD. EACH DATASET CONTAINS ALL OF THE ORIGINAL IMAGES RESIZED. 17
FIGURE 12: THE ORIGINAL IMAGE (LEFT) WAS RESIZED BY A FACTOR OF 0.5 (MIDDLE) AND 0.31 (RIGHT). A LOSS OF DETAIL IS EVIDENT AFTER RESIZING. 17
FIGURE 13: 608X608 PX TILES CORRESPONDING TO AN ORIGINAL IMAGES (A) AND AN IMAGE RESIZED BY FACTOR 0.5 (B) AND 0.31 (C). 18
FIGURE 14: THE ELEMENTS NEEDED TO CALCULATE IOU. 20
FIGURE 15: PRECISION-RECALL CURVE FOR A CLASS. TAKEN FROM VOORHOEVE (2018). 22
FIGURE 16: CENTRE PIXEL SHIFT DUE TO IMAGE CROPPING. THE CENTRE PIXEL OF THE CROPPED IMAGES CANNOT BE USED AS NADIR. 26
FIGURE 17: VIEWING ANGLE DISTRIBUTION OF THE NUMBER OF ANNOTATIONS PER ANGLE CLASS FOR THE ORIGINAL RESOLUTION IMAGES (4 MM). 27
FIGURE 18: DIFFERENCE IN DIRECTION OF OBJECTS. LEFT: ORIGINAL IMAGE, CROP ROWS ARE VERTICAL. RIGHT: ORTHORECTIFIED IMAGE, CROP ROWS ARE DIAGONAL. 28
FIGURE 19: F1-SCORE FOR ALL FOLDS IN ALL RESOLUTIONS. EVERY DATAPOINT CORRESPONDS TO A FOLD. THE CROSS SYMBOL REPRESENTS THE AVERAGE; THE DIAMONDS REPRESENT OUTLIERS. 29
FIGURE 20: F1-SCORES FOR ALL IMAGES PER FOLD, FOR ALL THREE TESTED RESOLUTIONS. EVERY DATAPOINT CORRESPONDS TO ONE TILE. PER FOLD, 20 IMAGES WERE USED FOR TILE CREATION. 30
FIGURE 21: PRECISION (A) AND RECALL (B) FOR ALL RESOLUTIONS. 32
FIGURE 22: CORRECTLY PREDICTED WEEDS PER ERROR TYPE FOR ALL RESOLUTIONS. DATAPOINTS CORRESPOND TO INDIVIDUAL FOLDS. 32
FIGURE 23: NUMBER OF GROUND-TRUTH LABELS PER RESOLUTION. DATAPOINTS CORRESPOND TO INDIVIDUAL FOLDS. ERROR TYPES FOLLOW THE SAME ORDER PER RESOLUTION AS FIGURE 23. 33
FIGURE 24: MAP FOR ALL TESTED RESOLUTIONS. 33
FIGURE 25: AVERAGE INTERSECTION OVER UNION (IOU) FOR ALL TESTED RESOLUTIONS. 34
FIGURE 26: (A) RELATIONSHIP BETWEEN F1-SCORE AND VIEWING ANGLE FROM NADIR, AND (B) NORMAL PROBABILITY PLOT FOR ALL THREE RESOLUTIONS. 36
FIGURE 27: F1-SCORE FOR THE ORTHOMOSAIC AND ORIGINAL TILES. ORTHORECTIFICATION HAS A CLEAR NEGATIVE EFFECT ON THE F1-SCORE. DATAPOINTS CORRESPOND TO TILES. 37
FIGURE 28: PRECISION (A) AND RECALL (B) FOR THE ORTHOMOSAIC AND ORIGINAL TILES. 37
FIGURE 29: EFFECT OF A DISTORTION IN THE LENS ON THE TOP RIGHT CORNER OF THE IMAGE. 38
FIGURE 30: THREE TRACTOR TIRE TRACKS DELINEATE THE STUDY AREA. THESE TRACKS CAN INFLUENCE THE MODEL'S RESULTS. 39
FIGURE 31: EXAMPLES OF DIFFERENCE IN LEAF SHADOWS BETWEEN SUGAR BEET AND WEED. (A) IMAGE WITH LABELS INDICATING THE DIFFERENT TYPES OF SHADOWS; (B) THE CLEAN IMAGE FOR EASY VIEWING. 39
FIGURE 32: CHALLENGES IN IOU INTERPRETATIONS. BLUE BOXES REPRESENT LABELLED GROUND TRUTH OBJECTS, WHILE RED BOXES REPRESENT OBJECTS PREDICTED BY THE MODEL. IMAGE A DEPICTS THE

MODEL DEMARKING A LARGER AREA THAN THAT WHICH WAS LABELLED. IMAGE B ILLUSTRATES THE MODEL PREDICTING ONE WEED INSTEAD OF THE TWO THAT WERE LABELLED. IMAGE C DEPICTS THE OPPOSITE OF B, WHEREIN THE MODEL PREDICTS MORE WEEDS THAN WHAT WAS LABELLED. 43
FIGURE 33: ILLUSTRATION OF THE BIN PROBLEM: A WEED WHICH IS CORRECTLY IDENTIFIED, IS INSTEAD COUNTED AS A FP (RED BOUNDING BOX) AND A FN (GREEN). 44
FIGURE 34: DIFFERENCE IN A PART OF THE STUDY AREA THAT WAS WATERLOGGED AND IN THE TEST SET OF FOLD 1 (A), A MORE CONVENTIONAL PART OF THE FIELD WHICH WAS IN THE TRAINING SET OF FOLD 1 (B). 46
FIGURE 35: EXAMPLES OF WEEDS BEING SPLIT AMONG TWO TILES, MAKING THEM HARDER TO DETECT AND RESULTING IN A FN (A: BLACK BOX, MIDDLE RIGHT) AND A FP (B: RED BOX, UPPER RIGHT); SIMILAR TO THE PHENOMENON SEEN IN FIGURE 33. GREEN BOXES CORRESPOND TO TP PREDICTIONS, BLUE BOXES ARE CORRECTLY IDENTIFIED GROUND TRUTH LABELS, WHITE LINES SHOW THAT THE WEED IS IN THE CROP ROW, AND ORANGE LINES SHOW THAT THE WEED IS OUTSIDE OF THE CROP ROW. 46
FIGURE 36: EXAMPLE OF THE DATA USED BY ZHONG ET AL. (2018). ALTHOUGH IT'S DIFFERENT TO THE WEEDS DATA, THE STUDY SHOWS AN INTERESTING IMPROVEMENT OF ACCURACY WHEN COMBINING YOLO WITH SVM. 49
FIGURE 37: MOST FP PREDICTIONS (RED BOX) ARE OTHER OBJECTS THAN CROPS. A, WEEDS WHICH WERE NOT LABELLED DETECTED OUTSIDE OF THE STUDY AREA. B, WEED THAT WAS SPLIT DURING TILE CREATION IS COUNTED AS A FP. C, CROP MISTAKENLY PREDICTED AS A WEED. IT'S ASSUMED THAT THE FP IN C IS A CROP ALTHOUGH IT COULD ALSO BE A WEED THAT WASN'T LABELLED. GREEN BOXES CORRESPOND TO TP PREDICTIONS, BLUE BOXES ARE CORRECTLY IDENTIFIED GROUND TRUTH LABELS, WHITE LINES SHOW THAT THE WEED IS IN THE CROP ROW, AND ORANGE LINES SHOW THAT THE WEED IS OUTSIDE OF THE CROP ROW. 51
FIGURE 38: COMPARISON OF AN ORTHOMOSAIC TILE (A) AND AN ORIGINAL RESOLUTION TILE (B). THE MODEL PREDICTS MORE FP (RED BOX) FOR THE ORTHOMOSAIC TILE. GREEN BOXES CORRESPOND TO TP PREDICTIONS, BLUE BOXES ARE CORRECTLY IDENTIFIED GROUND TRUTH LABELS, BLACK BOXES ARE FN, BLUE BOXES ARE CORRECTLY IDENTIFIED GROUND TRUTH LABELS, WHITE LINES SHOW THAT THE WEED IS IN THE CROP ROW, AND ORANGE LINES SHOW THAT THE WEED IS OUTSIDE OF THE CROP ROW. 54
FIGURE 39: COMPARISON OF AN ORTHOMOSAIC TILE (A) AND AN ORIGINAL RESOLUTION TILE. THIS COMPARISON SHOWS AN INCREASE IN FN (BLACK BOX) PREDICTION FOR THE ORTHOMOSAIC TILE. 55
FIGURE 40: STUDY AREA AT VALTHERMOND, THE NETHERLANDS. A DARKER SOIL CAN BE SEEN IN PLACES WHICH WERE WATERLOGGED IN THE PAST. 63
FIGURE 41: EXAMPLE OF AN ORIGINAL IMAGE. 64
FIGURE 42: PERCENTAGE OF ERROR TYPE RELATIVE TO TOTAL NUMBER OF LABELLED WEEDS PER RESOLUTION. 87
FIGURE 43: ANGLE RESIDUAL PLOT FOR THE ORIGINAL RESOLUTION TILES. 88
FIGURE 44: ANGLE RESIDUAL PLOT FOR THE 0.5 DATASET TILES. 88
FIGURE 45: ANGLE RESIDUAL PLOT FOR THE 0.31 DATASET TILES. 89

LIST OF TABLES

TABLE 1: CURRENT COSTS FOR WEED REMOVAL PER HECTARE FOR A CONVENTIONAL SUGAR BEET FARM. 11
TABLE 2: HYPERPARAMETER SETTINGS USED FOR MODEL TRAINING. 20
TABLE 3: CONFUSION MATRIX FOR TYPES OF FN ERRORS. 23
TABLE 4: SUMMARY OF PERFORMANCE VALUES PER RESOLUTION. 29
TABLE 5: F1-SCORE STANDARD DEVIATION PER RESOLUTION. 29
TABLE 6: F1-SCORE STANDARD DEVIATION PER FOLD PER RESOLUTION. 30
TABLE 7: ANOVA. TESTS OF BETWEEN-SUBJECTS EFFECTS FOR THE PERFORMANCE PER RESOLUTION TEST. 31
TABLE 8: PERFORMANCE PER RESOLUTION TEST LINEAR MODEL RESULTS. 31
TABLE 9: AVERAGE NUMBER OF GROUND-TRUTH LABELS PER RESOLUTION. SMALLER WEEDS WERE LABELLED MUCH LESS FREQUENTLY. 33
TABLE 10: PERFORMANCE PER RESOLUTION TEST MAP STANDARD DEVIATION PER RESOLUTION. 34
TABLE 11: SUMMARY OF PROCESSING TIME PER RESOLUTION TYPE. 35
TABLE 12: ESTIMATED COSTS PER HECTARE PER RESOLUTION. COSTS ARE CALCULATED WITHOUT TAKING INTO ACCOUNT OPERATIONAL COSTS. 35
TABLE 13: PRECISION, RECALL AND F1-SCORE DIFFERENCE BETWEEN THE PERFORMANCE PER RESOLUTION TEST AND THE VIEWING ANGLE TEST FOR ALL RESOLUTIONS. 44
TABLE 14: MEAN AVERAGE PRECISION (MAP) VALUES FOR DIFFERENT LITERATURE SOURCES. 47
TABLE 15: PERFORMANCE VALUES FOR COMPARABLE WEED-DETECTION LITERATURE SOURCES. 48
TABLE 16: F1-SCORE, PRECISION AND RECALL FOR DIFFERENT WEEDS AND RESOLUTIONS. MODIFIED FROM HUNG ET AL. (2014). 49
TABLE 17: COMPARISON OF RESULTS FOUND FOR DIFFERENT RESOLUTION AND A YOLOV3-BASED UGV WEED-DETECTION SYSTEM. 52

LIST OF ABBREVIATIONS

ANOVA	Analysis of Variance
CNN	Convolutional Neural Network
COCO	Common Objects in Context
ExG	Excess Green Vegetation Index
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Networks
GPU	Graphics Processing Unit
HNM	Hard Negative Mining
IoU	Intersection over Union
LiDAR	Light Detection And Ranging
mAP	mean Average Precision
ML	Machine Learning
NDVI	Normalised Difference Vegetation Index
NIR	Near Infra-Red
PASCAL VOC	PASCAL Visual Object Classes
R-CNN	Region-based Convolutional Neural Network
R-FNC	Region-based Fully Convolutional Network
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
ROI	Region Of Interest
RPN	Region Proposal Networks
SD	Standard Deviation
SSD	Single Shot MultiBox Detector
SVM	Support-vector Machine
TP	True Positive
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
YOLOv3	You-Only-Look-Once version 3

TABLE OF CONTENTS

Acknowledgements iv
Declaration v
Abstract vi
List of figures viii
List of tables x
List of abbreviations xi
1 Introduction 1
  1.1 Current situation 1
  1.2 Desired situation 1
  1.3 Problem definition 2
  1.4 Research objective 3
  1.5 Research questions 3
  1.6 Hypotheses 3
  1.7 Research scope 4
2 Literature review 5
  2.1 Background 5
    2.1.1 Machine Learning vs. Deep Learning 5
    2.1.2 Convolutional neural networks 5
    2.1.3 Object-detection models 6
  2.2 Economic comparison 11
    2.2.1 Conventional weed management 11
    2.2.2 Scenario 1: Scout UAV combined with intervening UGV 12
    2.2.3 Scenario 2: Hopper UAV 12
  2.3 Literature review conclusion 13
3 Material and methods 14
  3.1 Overview 14
  3.2 Study area 14
  3.3 Data acquisition and pre-processing 15
    3.3.1 Data acquisition 15
    3.3.2 Data annotation 15
    3.3.3 Data splitting 16
  3.4 Image resolution change (Step 1) 16
  3.5 Create 608x608 px tiles from images (Step 2) 17
  3.6 Deep-learning models (Step 3) 19
  3.7 Performance evaluation (Step 4) 20
    3.7.1 Metrics 20

3.7.2 Performance per resolution test 24
3.7.3 Viewing angle test 26
3.7.4 Orthomosaic test 27
4 Results 29
4.1 Performance per resolution test 29
4.1.1 F1-score 29
4.1.2 Precision and recall 32
4.1.3 FN classification 32
4.1.4 Mean average precision (mAP) 33
4.1.5 Average intersection over union (IoU) 34
4.1.6 Run time 35
4.1.7 Economic analysis 35
4.2 Viewing angle test 36
4.2.1 F1-score, precision and recall 36
4.3 Orthomosaic test 37
4.3.1 F1-score, precision and recall 37
5 Discussion 38
5.1 Datasets and methodology 38
5.1.1 Datasets 38
5.1.2 Weed labelling method 39
5.1.3 Image resizing method 41
5.1.4 Tile creation method 41
5.1.5 YOLOv3 42
5.1.6 Performance test method 42
5.2 Discussion of test results 45
5.2.1 Performance per resolution test 45
5.2.2 Economic feasibility 50
5.2.3 Comparison to UGV 52
5.2.4 Viewing angle test 52
5.2.5 Orthomosaic test 53
6 Conclusion and recommendations 56
7 Bibliography 58
8 Annexes 63
8.1 Annex I: Orthoimage of the study area 63
8.2 Annex II: Image properties 64
8.3 Annex III: Script to resize label coordinates and adapt labels to tiles (0.5 dataset) 65
8.3.1 Pseudocode 65
8.3.2 Python code 65

8.4 Annex IV: Script to test performance (original resolution) 70
8.5 Annex V: Script to calculate bounding box angle (original resolution) 77
8.6 Annex VI: Script to test angle dependent performance... 79
8.7 Annex VII: Error type relative to total number of ground truth labels 87
8.8 Annex VIII: Angle residual plot for all tested resolutions 88
8.9 Data and results overview 90

1 INTRODUCTION

1.1 Current situation

Weeds are commonly regarded as detrimental for crop production as they compete for resources with crops (Hartzler, 2009; Kropff, 1988). Therefore, farmers are keen on removing weeds mechanically or chemically. Currently, mechanical removal of weeds is not always favourable as crop plant losses always occur and propagation by cuttings and roots can further increase the spread of weeds (Davies et al., 2011; Dedousis and Bartzanas, 2010; Vanhala et al., 2004). Alternatively, without herbicide resistant crops, chemical removal of weeds is costly as labourers must spray individual weeds to reduce the risk of herbicides damaging the crops. Also, inefficient use of agrochemicals can have adverse effects on human health and the biotic and abiotic environment (Igbedioh, 1991).

An attempt to counter these problems is being developed in the realm of precision agriculture systems. Precision agriculture can give farmers an edge regarding the management decisions on their plots, which may lead to an increase in overall yields whilst decreasing dependency of on herbicides (Khosla et al., 2002; Slaughter et al., 2008). Precise imaging systems that aim at increasing the efficient use of agrochemicals by accurately locating weeds are in continuous development (Bawden et al., 2017; Davis and Pradolin, 2016; Lottes et al., 2018; Lottes and Stachniss, 2017; Milioto et al., 2017a). As the growth location of weeds cannot be predicted without a thorough study of the soil and immediate environment, weeds can only be spotted after they have sprouted. This makes imaging systems ideal for the detection of weeds. Among these systems are those that use deep learning to extract information that is not immediately evident visually or numerically from images, and use this information to detect objects. These deep-learning based object-detection models can be applied in a wide range of disciplines. Current development in the discipline of weed detection is still in its relative infancy. Among others; questions regarding the type of deep-learning model to be used, method of data augmentation, needed image resolution, robustness of the models, real-time capabilities of data processing, applicability on orthomosaic images and automation of training data, are being researched to achieve the best possible results for the specified use case. As precision agriculture becomes more accessible with cheaper and more sophisticated technologies, research attempting to answer the abovementioned questions are starting to pick up pace.

To effectively apply herbicides, weeds must be accurately located (Steward and Tian, 1998). Locating and removing weeds can be done on the field, which costs labour; in images and in orthomosaic images. In this research's study area, the main weeds are voluntary potatoes. Deep-learning based object-detection models are promising to aid in this task. Object detection is an ever-improving area in computer vision and is widely used for the detection of weeds (Chu and Cai, 2018; Lottes et al., 2018; Lottes and Stachniss, 2017). Due to the rapidly changing nature of this academic field, it is easy to work with outdated systems. For this reason, it is important to keep in mind developments in this field.

1.2 Desired situation

Ideally, weed detection should be accurate, quick and robust. Accuracy, or object-detection performance, is needed to correctly identify the location of weeds. For example, in the case of automatic herbicide application, this accuracy is needed to differentiate between crop and weed as to not apply herbicides on crops, and to reduce material costs. A system that results in lower weed-removal costs compared to current costs is deemed accurate enough to be functional in practice. Besides costs, introducing robotic weeding technologies coupled with sustainable management practices can potentially reduce labour demands by 85% for sugar beet production (Sørensen et al., 2005).

Furthermore, weed detection systems should preferably be quick. How quick the system should be, depends on the application: e.g. automatic weed removal systems should have processing times of a second or less, while weed growth monitoring systems can take minutes or hours depending on the user requirements. Real-time weed detection is desirable in the realm of robotic agriculture, where systems are being developed to detect and remove weeds automatically without human intervention or predefined locations (Bawden et al., 2017; Sujaritha et al., 2017). For such systems a relatively high number of frames per second are desirable as higher frames per second result in information closer to real-time. Milioto et al. (2017b) set the upper bound for real-time weed classification speed at 1 image per second (1 Hz).

Robustness of the systems comes into play when training data are not varied enough. As weeds have a large variety in appearance, it would be ideal for a weed-detecting model to detect most types of weeds in different environments. The detection of weeds should not be influenced by the crop type on the field or soil type.

Besides the abovementioned system requirements, it is desired that the system runs on a lightweight *unmanned aerial vehicle* (UAV). UAVs are gaining popularity in precision farming applications as they can map relatively large areas in a shot amount of time (Milioto et al., 2017a). In the realm of remote sensing, UAVs are often used for the creation of high-resolution orthomosaic images. Weed detection on orthorectified images would open new possibilities as coordinates would be tied to the detected objects.

1.3 Problem definition

Detecting weeds using deep-learning based object-detection models from aerial platforms is still in its infancy. Many deep-learning models have been developed which have their positive and negative aspects. For example, a preliminary literature research has shown that the *Single Shot MultiBox Detector* (SSD) and the *You Only Look Once version 3* (YOLOv3) models are adept to be used in the described application and interesting to compare. The choice of model will depend on the characteristics most suited for the detection of weeds in real-time. It is not yet known which combination of object-detection model and image resolution is the most suitable for accurate real-time weed detection from a UAV.

To achieve high accuracies and a good measure of robustness using deep-learning models, well-crafted training data should be available to train the models. However, such data are not easily created. The training data should consist of many samples of weeds in various growing stages, soil types, proximity to other plants, weather and other such factors which influence the appearance of the weed but are not always available in the datasets used. Varied training data are needed to make the system accurate (it can pinpoint the location of a weed) and robust (it can correctly identify weeds in various contexts). The needed accuracy depends on the crops being herbicide resistant or not (i.e. in fields with herbicide resistant crops accuracy can be lower as it is not as negative for production to spray these crops compared to non-resistant crops) and the value of the crop compared to the growth-diminishing factors of damaging the crop with herbicides or letting weeds grow close to the crop. Although the effect of weeds and herbicides on crops is well known, the accuracy that an object-detection model should have to optimise the costs of removing weeds against the costs of accidentally removing a crop or letting weeds grow, is not known. On the other hand, model robustness can be increased by implementing data-augmentation techniques, which create more training data. However, data-augmentation techniques applied by most object-detection models are not specific for agricultural applications.

Weed-detection systems can be both ground-based or aerial. This research will focus on aerial systems. Aerial systems have the advantage of being able to acquire data over larger areas with increased mobility. However, with UAVs this mobility still comes at the cost of the ability to carry more powerful computers and batteries needed to power these devices. Thus, reducing processing power needed to run the object-detection models in real-time is needed to increase the feasibility of these systems. Furthermore, the total area viewed is related to flying altitude which in turn establishes image resolution. Lower image resolutions reduce run times but also contain less details. It is not known which flying altitude provides a high enough resolution to detect weeds while covering the largest possible area. Additionally, such aerial systems can create orthomosaic images which have the added benefit of tying objects to their geographical coordinate. It's not clear how orthorectification influences detection performance.

Faster models that are robust and accurate enough to be used practically, are more processing intensive as more complex calculations are needed. If the system is not quick enough, new weeds can appear and existent weeds can further make use of valuable resources needed for crop growth. However, timespans needed to effectively counter such problems are large enough to not be a problem for most object-detection models. Model speed becomes more important when assessing the possibility of real-time object detection. This is needed to rapidly assess and act upon the information gained through the model.

1.4 Research objective

This research will test the real-time performance of the *You-Only-Look-Once version 3* deep-learning based object-detection model to detect voluntary potatoes (*Solanum tuberosum*), a weed, in a sugar beet (*Beta vulgaris ssp. vulgaris*) field using various image resolutions obtained from an UAV. As image resolution is related to flying altitude, understanding how resolution affects performance leads to a better understanding of which flying altitude is preferable when detecting weeds. This research will also analyse the relationship between object viewing-angle from nadir and performance, to better understand how the location relative to the sensor affects performance. Thus, gaining a deeper understanding of another factor which influences performance. Lastly, the effect of orthorectification on performance will be researched to bring this research closer to the remote sensing discipline.

1.5 Research questions

1. Which requirements should the real-time UAV weed-detection system meet to be an improvement over conventional chemical weeding measures?
    a. Looking at precision, recall and cost per hectare, how well should the model perform to be economically practical?
2. How does image resolution affect performance and processing speed of the model?
3. How does viewing angle from nadir affect performance of the model?
4. How does orthorectification affect the performance of the model?

1.6 Hypotheses

1. Requirements that should be met are dependent on the scenario. For most scenarios however, both precision and recall should be as high as possible. High processing speeds are mostly important in scenarios where detection is immediately followed by removal. All scenarios would benefit from a robust system that can detect different weeds and does not mistake crops for weeds.
2. Higher image resolution is expected to yield more accurate object detection in exchange for a decrease in frames per second for both models. YOLOv3 is expected to work well in detecting smaller objects at higher resolutions while maintaining accuracy at higher frames per second using higher resolution images.
3. Weeds closer to nadir, relative to the sensor, are expected to be easier to detect.
4. Orthorectification is expected to negatively influence the detection of weeds.

1.7 Research scope

Although there are many types of deep-learning models, this research will focus on the *You Only Look Once version 3* (YOLOv3) deep-learning architecture based on the preliminary literature research found in the *Literature review*. The objects to be detected are volunteer potatoes, which are defined as weeds in this research. Furthermore, the platform on which the model is expected to be run in the future is on a UAV. Thus, real-time data processing is preferred compared to slower, albeit possibly more accurate, processing. A balance between speed and object-detection performance should be obtained for this research.

2 LITERATURE REVIEW

2.1 Background

2.1.1 Machine Learning vs. Deep Learning

Machine-learning (ML) algorithms parse data, learn from this data and then make predictions based on what was learned. An algorithm can learn from an experience E which is related to some task T and performance measure P. If performance at task T, measured by P, improves with experience E, the algorithm is said to learn (Mitchell, 1997). For object-detection tasks T, P is usually a quantitative measure for accuracy and E is the dataset fed to the algorithm. What is contained in the dataset depends on the type of machine learning algorithm used. There are three types of ML algorithms: *supervised*, *unsupervised*, and *reinforcement* learning. In the case of *supervised* learning, the dataset contains training samples of correctly labelled objects which have been selected by a data-labeller (Li, 2017; Schmidhuber, 2015). These labelled data are used to teach the algorithm to identify similar objects in never before seen test data. *Unsupervised* learning does not make use of labelled data. Instead, it aims at structuring the data based on similarities between data points making use of e.g. clustering (Li, 2017). *Reinforcement* learning does not make use of a fixed dataset. Instead, it tries to maximise rewards (i.e. accurate predictions) by entering a feedback loop which trains the system by coupling performance to experiences (Li, 2017). This type of learning relies on trial and error of an agent interacting with the environment that can reward or punish depending on the actions taken by the agent (Li, 2017).

Generally, machine learning algorithms are composed of (1) a dataset, (2) a cost/loss function, (3) an optimization procedure and (4) a model (Goodfellow et al., 2016). (1) Datasets are often separated into training, validation and testing data. In the case of weed detection; training data are used to teach the model how weeds look, validation data are used to improve the model parameters to better fit the data, and testing data are used to test model performance. (2) The cost/loss function is necessary to measure model performance. The loss function is needed to define how good individual predictions are compared to the corresponding true label, while the cost function measures how well the model is doing on the entire training set. (3) A common optimization approach is the use of stochastic gradient descent (Goodfellow et al., 2016). This method aims at tweaking parameter values to minimise the cost. (4) The model's end goal is to reduce the gap between testing and training error (Li, 2017). However, according the *no free lunch* theorem, there is no perfect model although some models can outperform others if they are specifically designed for the task (Wolpert and Macready, 1997).

In "shallow" machine learning, an input layer passes data to an output layer that contains a function which changes the input data. In deep learning, there are one or more *hidden layers* between the input and output layers. Except for the input layer, the weighted sum of units from previous layers is computed per unit. To decide to what degree the layer will apply the input of previous layer units, a nonlinear transformation or activation function such as *Rectified Linear Unit* (ReLU) is used. Weights are assigned to links between units of different consequent layers which further change the result of transformations between units. Computation always starts from input to output with generally randomly selected parameters such as weight (forwards propagation). However, after forwards propagation, error derivatives can be computed starting from the output layer to update weights to optimise the loss function (backwards propagation) (Li, 2017).

2.1.2 Convolutional neural networks

Deep learning enables the computer to build complex concepts (e.g. identifying a person) out of simpler concepts (e.g. identifying the shape of a body). However, not all types of deep-learning algorithms can be used for object detection. A type of deep learning network that is often used with multiple-array data such as colour images, is the *convolutional neural network* (CNN) (LeCun et al., 2015). The blocks that build these networks are the (1) convolutional, (2) pooling, (3) fully-connected and (4) ReLU layers as hidden layers:

1. *convolutional layer* is a filter (window) that convolves with the image and extracts features from the image while preserving the spatial relationship between pixels (Goodfellow et al., 2016). The filter has a certain operation tied to it (e.g. edge detection, sharpen) which combined with other filters can help identify valuable objects in images.
2. *A pooling layer* reduces dimensionality of feature maps but keeps important information (LeCun et al., 2015). Pooling layers are also convolving sliding windows that have an operation tied to them (e.g. average, sum, max) (Zhou and Chellappa, 1988). For example, Max Pooling with a 2x2 window takes the largest pixel value within that window. Pooling layers also makes the network more robust to small changes in objects (LeCun et al., 2015).
3. A *fully-connected layer* computes the probability of an object belonging to a predetermined class (Krizhevsky et al., 2012). This layer commonly uses a *softmax* activation function at the end of the network of which the sum of outputs equals 1, thus denoting a probability. Every unit in the previous layer is connected to every unit in the next layer. The combination of features that results from this interconnectedness, often creates relationships between features that help improve the classification task. Furthermore, to reduce overfitting of the model a technique called *dropout* is used. Dropout sets the output of hidden units with values under a certain threshold to zero (Hinton et al., 2012). Computation is hereby reduced as these units are not included during backwards propagation.
4. A *ReLU activation layer* is applied after every convolutional and fully-connected layer (LeCun et al., 2015). The ReLU function replaces all negative pixels by zero which reduces computation times and introduces non-linearity (Krizhevsky et al., 2012). Non-linearity is needed since most real-world data are non-linear.

2.1.3Object-detection models

There are various deep-learning models currently in use for object detection. This section will elucidate the positive and negative aspects of the most state-of-the-art models in relation to the real-time object detection. All these models make use of the abovementioned concepts which lay at the foundation of deep-learning based object-detection models.

2.1.3.1 R-CNN

The *Region-based Convolutional Neural Network* (R-CNN) algorithm developed by Girshick et al. (2014), proposes regions based on image structure as seen in Uijlings et al. (2013) (Figure 1). Each region is then resized to be fed into the CNN and the feature vector output is then classified.

No literature was found in which R-CNN was used for agricultural applications. However, for the *PASCAL Visual Object Classes* (PASCAL VOC) (Everingham et al., 2010), an often used dataset in the realm of object-detection models, the best R-CNN models achieved a *mean Average Precision* (mAP) of 53.3% (Girshick et al., 2013). As region proposal is processing intensive, this model is not suitable for real-time data processing.

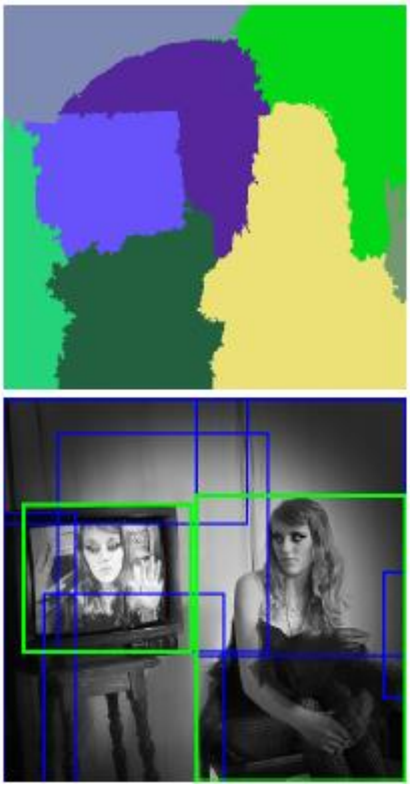


Figure 1: Proposed regions. Modified from Uijlings et al. (2013).

2.1.3.2 Fast R-CNN

Fast R-CNN developed by Girshick (2015), was developed to reduce the time needed to analyse all region proposals. Instead of dividing the whole image into different proposed regions that have to be analysed, Fast R-CNN assigns *Regions of Interest* (ROIs). These ROIs are then analysed thoroughly as they are fed to a max pooling layer to reduce the feature map size, and then fed to fully-connected layers. The fully connected layers create feature vectors which are used to predict the object.

No literature was found in which Fast R-CNN was used for agricultural applications. However, for the PASCAL VOC test dataset, a mAP of 68.4% was achieved (Girshick, 2015). Although this model is quicker than R-CNN, it is not quick enough for real-time data processing.

2.1.3.3 Faster R-CNN

To counter the computational expensive region proposals, Ren et al. (2015) introduced *Region Proposal Networks* (RPN) to propose regions, predict bounding boxes and detect objects at once. RPN works as follows: The model first applies a CNN over the whole image which results in feature vectors linked to two fully-connected layers, which predict region proposals. These regions are fitted to anchor boxes which are selected by looking at the relevance of the shape of the anchor box in relation to the object being detected. These anchor boxes and the feature maps computed by the CNN are fed to the Fast R-CNN. Thus, faster R-CNN is a combination of RPN and Fast R-CNN.

Faster R-CNN has been used for agricultural applications. Among other research; Sagar et al. (2018), propose a method to estimate tomato yield. Sa et al. (2016) presents an adapted version of the Faster R-CNN model to detect fruits in real-time combining RGB with *Near Infra-Red* (NIR) imagery. Zhao et al. (2017) propose a method to predict melon yield using small UAVs which result in relatively high accuracies. However, the processing of this data was not in real-time. Lastly, Xia et al. (2018) have developed a method to detect weeds based on an altered version of the Faster R-CNN model. However, the images used are not comparable to images taken from a UAV. Besides agricultural applications, for the PASCAL VOC test dataset, a mAP of 75.9% was achieved (Ren et al., 2015). Although this model is more accurate and 34 times faster in some cases compared to Fast R-CNN, the real-time capabilities are limited to 5 frames per second for the original model (Ren et al., 2015).

2.1.3.4 R-FCN

The *Region-based Fully Convolutional Network* (R-FCN) developed by Dai et al. (2016) combines region detecting proposals and object recognition into a single model which consists of only convolutional layers and a fully-connected layer. This allows backpropagation for training and inference throughout the whole model, and consideration for both object detection and location simultaneously. In principle, the model passes the input image through a CNN and creates position-sensitive score maps

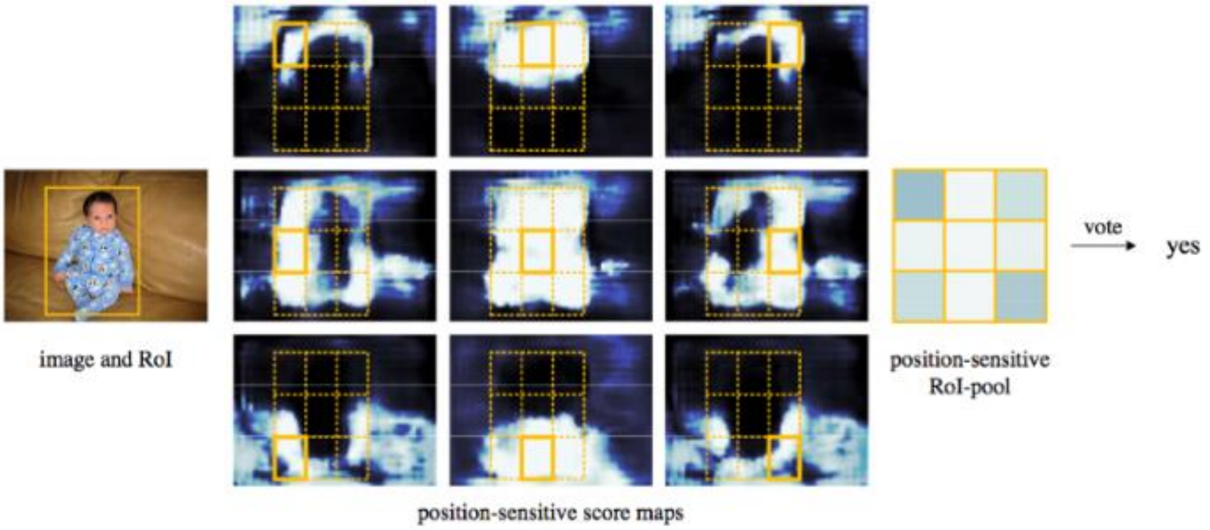


Figure 2: Visualisation of comparison between ROIs and position-sensitive score maps used in the R-FCN. If a position-sensitive score map identifies patterns associated to a stored object, it returns a yes (lighter colour on the rightmost grid). Taken from Dai et al. (2016).

that are specialised in detecting a category at a certain location. These score maps are stored and used to recognise an object or part of an object. At the same time, a RPN generates ROIs which are compared to the feature maps (Figure 2).

R-FCN has achieved a mAP of 85.98% to detect plant diseases and pests (Fuentes et al., 2017). Huang et al. (2018) also use a FCN to map weeds using a UAV claiming to achieve weed recognition accuracies of 88% in capable of being done in real-time. Besides agricultural applications, a mAP of 82% was achieved for the PASCAL VOC test dataset while being 2.5-20 times faster than Faster R-CNN in some cases (Dai et al., 2016). This speed brings the R-FCN closer to real-time possibilities.

2.1.3.5 YOLO

The *You Only Look Once* model (YOLO) developed by Redmon et al. (2015) predicts bounding boxes and class probabilities in a single image evaluation (single shot). In principle, the model divides the image into a predefined grid. Each grid can predict the location of an object based on bounding boxes with a confidence score. The YOLO network predicts a high number of bounding boxes which do not always correspond to an object. To prevent multiple classification of the same object, *Non-Maximum Suppression* (NMS) is applied. This merges overlapping bounding boxes that correspond to the same object. The CNN used to analyse the image in search of objects makes use of inception modules. These modules allow the system itself to choose which combination of filter/pooling layers it will use, which increase the detection performance and computation costs (Szegedy et al., 2015). YOLO is highly generalizable, which makes it more adept when applied to new fields or unexpected input. Being more general, could make the system less accurate in the case of weed detection as plants can look rather similar.

Not much literature can be found regarding the application of YOLO on agriculture. Zhong et al. (2018), have developed a vision-based flying insect recognition system using YOLO modified with a support vector machine. Average classification accuracies of 90.2% were achieved on a Raspberry Pi in real-time, while a YOLO only model achieved accuracies of 60.8%. Although the YOLO model functions in real-time, a mAP of 57.9% was achieved for the PASCAL VOC test dataset (Redmon et al., 2015).

2.1.3.6 YOLOv3

The latest version of the YOLO model is YOLOv3 (Redmon and Farhadi, 2018). This method keeps positive aspects of YOLO and improves on the negatives. The YOLOv3 architecture consists of 106-layer fully-convolutional network. To increase speed of such a relatively large network, this architecture makes use of residual skip connections. These skip connections can be found in residual blocks. Residual blocks can be seen as “shortcuts” which help traverse information deeper into a neural network and increase model accuracy (He et al., 2015).

Furthermore, YOLOv3 makes use of upsampling which is a technique that increases image resolution. These upsampled layers can be combined with lower-resolution layers to help preserve details when detecting smaller objects. Another factor that helps detect smaller objects is *Feature Pyramid Networks* (FPN): Instead of detecting objects at the end of the network, YOLOv3 detects at three different scales. To realise this, a 1x1 detection filter passes through feature maps of three different sizes in three places in the network using different strides (Figure 3). Although upsampling and FPN aid in detecting smaller objects, YOLOv3 performs worse on medium and large objects compared to YOLO. (Redmon and Farhadi, 2018)

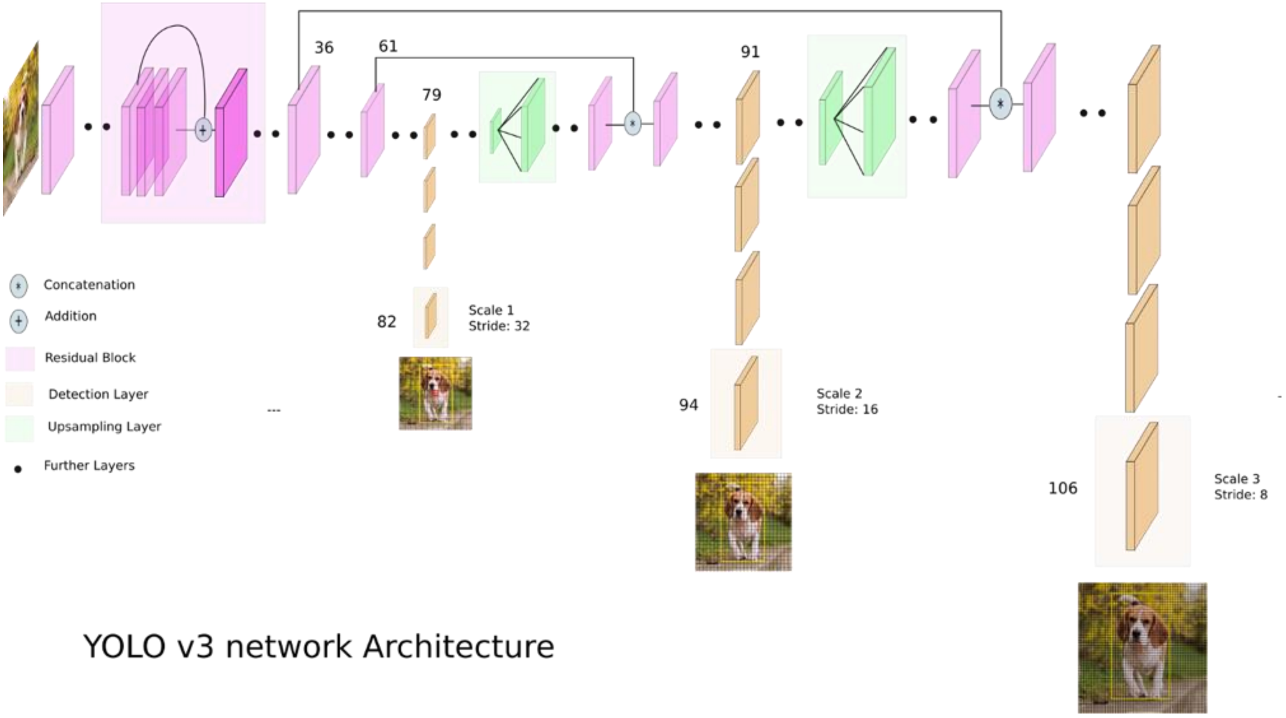


Figure 3: YOLOv3 network architecture. Taken from Kathuria (2018).

Lastly, contrary to most CNN, YOLOv3 does not make use of softmax to classify. Instead the model performs multilabel classification. This method does not assume that an object can only correspond to one class, which is the case with softmax. The class with the highest score is assigned to the predicted box. (Redmon and Farhadi, 2018)

In the agricultural domain, Underwood et al. (2018) compare methods to estimate fruit load in mango orchards. Among the methods compared, YOLOv3 results in an $R^2 = 0.97$ between human count and YOLOv3. For the *Common Objects in Context* (COCO) IoU 0.5 test dataset, an object-detection dataset developed by Lin et al. (2014) which is often used, YOLOv3 achieved a mAP of 57.9% while running 30 fps on a high-end GPU (Titan X) (Redmon and Farhadi, 2018). No literature was found on results on the PASCAL VOC test dataset.

2.1.3.7 SSD

The *Single Shot MultiBox Detector* (SSD) developed by Liu et al. (2016) predicts bounding boxes and class probability at once, similar to YOLOv3. SSD is based on the VGG-16 architecture (Figure 4). VGG-16 is a CNN consisting of 16 layers, characterised by its simplicity and relative accuracy (Simonyan and Zisserman, 2014). SSD adds six convolutional layers to the end of the VGG-16 architecture to be able to classify features at multiple scales (Figure 5). Furthermore, it is relative simple as it eliminates proposal generation and subsequent pixel or feature resampling stages and is all done in one network. SSD feeds the input image to a CNN which creates feature maps used to predict bounding boxes. The shapes of these bounding boxes are pre-selected manually as not all boxes fit all objects. The box that matches best with the object is selected. To further increase accuracy, NMS is used at the end of the SSD model to choose relevant boxes and *Hard Negative Mining* (HNM) is used as many erroneous boxes are predicted. HNM selects boxes which have the highest confidence. (Liu et al., 2016)

SSD has been used to detect plant diseases and pests in real-time with an mAP of 82.53% (Fuentes et al., 2017). Besides agricultural applications, an mAP of 82.2% and 50.4% was obtained for the PASCAL VOC and COCO IoU 0.5 test dataset respectively (Liu et al., 2016).

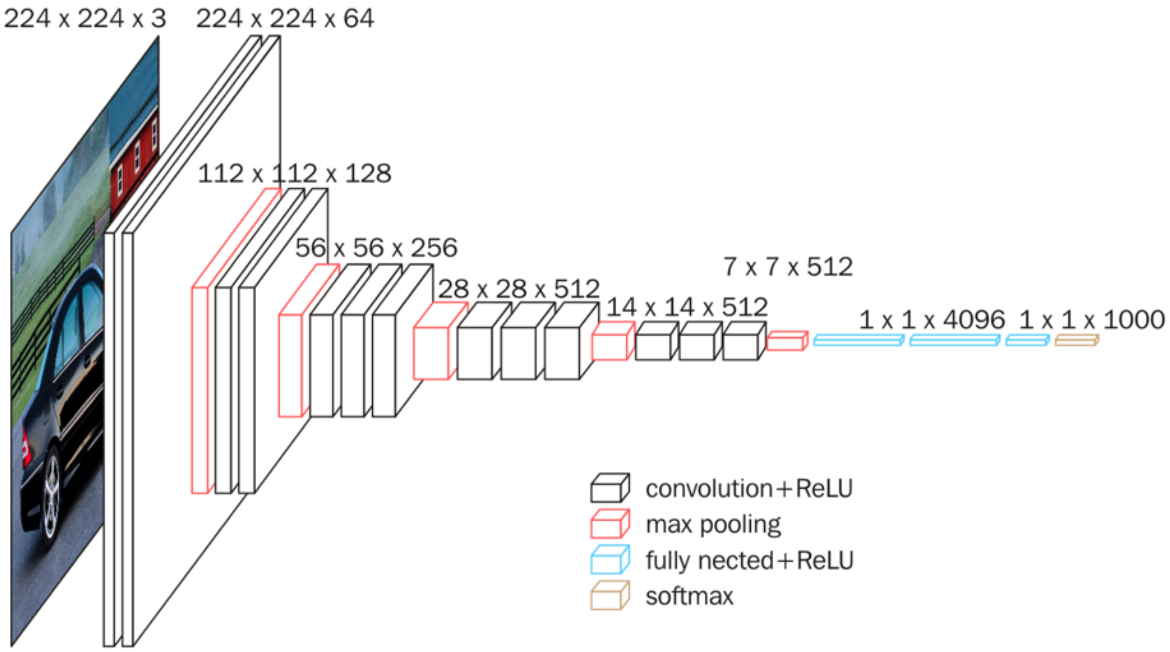


Figure 4: VGG-16 network architecture. Taken from Poudyal (2018).

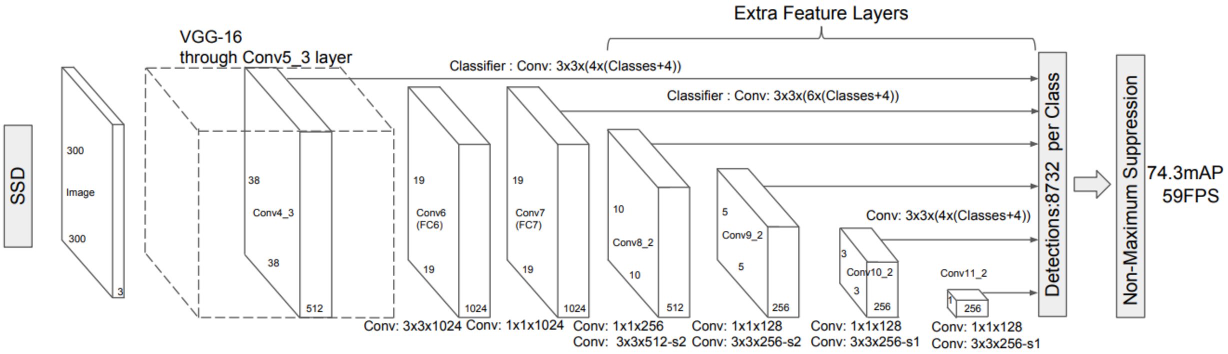


Figure 5: SSD network architecture. Taken from Liu et al. (2016).

2.1.3.8 Specialised weed detection models

The abovementioned algorithms are not designed specifically for the detection of weeds. Although a wide variety of specialised weed detection algorithms have been developed, these do not focus on real-time processing or make use of data that are not available for this research such as NIR images. However, much can be learned by understanding these systems and adapting their methodologies to this research.

Pérez-Ortiz et al. (2015) developed a weed detection system based on image patches. These patches are formed selecting by multispectral pixel intensities and geometric information of the crop rows. The patches are then used to create classification features. Using this model, a maximum classification accuracy of 75% was achieved flying at an altitude of 30m (14 mm resolution). In later work, Perez-Ortiz et al. (2016) apply a support vector machine classifier in combination with the exploitation of pixel intensities, geometrical information, textures and shape to differentiate between crops and weeds using RGB images and the *Excess Green Vegetation Index* (ExG). This led to a weed-detection accuracy of 79% in a maize field.

Guerrero et al. (2012) also make use of support vector machines to detect weeds in maize fields. This method also takes into account the difference in visual appearance evident due to rainfall, herbicide treatment or dry spells.

Lottes et al. (2017) exploit the *Normalised Difference Vegetation Index* (NDVI) and ExG to segment vegetation from soil. Then a random forest classifier is applied on the segmented vegetation to distinguish crops from weeds. Expanding on this method, Milioto et al. (2017b) achieve mean weed-detection accuracies 97% on hardware that can be embedded in a UAV. This is done by combining blob-wise vegetation segmentation and a custom neural network architecture. This runs quickly as the neural network does not need to evaluate all pixels, only the segmented vegetation pixels.

weedNet, developed by (Sa et al., 2018) is a weed and crop classification model which makes use of SegNet. SegNet is a CNN for semantic pixel-wise segmentation which also makes use of the VGG-16 network architecture (Badrinarayanan et al., 2017). *weedNet* retains SegNet's original architecture but modifies it as follows: (1) errors for classes which appear less frequently are penalised more than other classes, and (2) an input/output layer was added to allow feeding the model any number of input images.

2.2 Economic comparison

There are several scenarios that might make use of object detection based on RGB images acquired from a UAV. A selection of scenarios will be discussed and compared economically to conventional weed management practices, as well as the requirements that these scenarios have for a realistic application of object-detection systems. Requirements will be based on object-detection performance, model speed, and the relation of these with the costs attributed to the system.

As no figures are available to calculate the costs of operating the UGV and UAV, these costs are not considered in the economic feasibility analysis. However, we expect that these operational costs are not more expensive compared to current spraying equipment. Costs related to the new systems (e.g. cameras and processing units) are assumed to be a small factor in the long-term investment.

Costs for the conventional weed management system are expressed per hectare. This is not the case for the two scenarios which are dependent on the number of weeds detected, or missed, per hectare. Thus, this research's results will yield proper values to compare all three scenarios. This section demonstrates the different scenarios and the approximate costs related to these scenarios.

2.2.1 Conventional weed management

Conventional volunteer potato removal is done manually. This costs 20-30 hours of manual labour per hectare according to Paauw and Molendijk (2000), this seems a relatively high estimate. Lower estimates could not be found. Precision agriculture aims at eliminating these costs which are estimated at €350 (Spruijt and Van der Voort, 2015). A conventional Dutch sugar beet farm spends approximately €166 on herbicides per ha (Spruijt and Van der Voort, 2015). The herbicide is sprayed on the whole field instead of on the individual weeds.

Table 1: Current costs for weed removal per hectare for a conventional sugar beet farm.

	Costs per ha.
Herbicides	€166
Manual labour (crop removal)	€350
Total	€516

In principle, the object-detection model is deemed economically feasible to use whenever the costs attached are lower than the total current costs (Table 1).

2.2.2 Scenario 1: Scout UAV combined with intervening UGV

As seen in the Flourish Project (2018), such a scenario relies on two platforms working in unison. First, a UAV is sent so survey the field in question to locate weeds. The location of these weeds is then transmitted to an *Unmanned Ground Vehicle* (UGV). Hereafter, the UGV drives to the indicated locations and removes the weeds. The UGV makes use of a more detailed camera for on-site detection of weeds. The object detection performance and speed of this camera will be based on the findings of the MSc thesis research done by Voorhoeve (2018).

2.2.2.1 Object-detection performance

When only looking for weeds, the UAV object-detector should have a high recall as weeds which are predicted incorrectly can be revised by the UGV. The UAV should give the UGV the approximate location of weeds as the UGV can assess the location of weeds more accurately. Thus, as long as there is some overlap between labelled and predicted boxes, predictions are deemed correct. I.e. *True Positives* (TP).

False Positives (FP), crops that were detected as weeds, bring extra operational costs as the UGV must drive to the location. It is assumed that the UGV would not remove the crop even if it was marked as weed by the UAV, as precisions of 95% to 99% have been found in literature (Milioto et al., 2017a; Voorhoeve, 2018). Although such a relatively precise system would still incorrectly apply herbicides on up to 5% of the sugar beets, these costs are expected to be acceptable when compared to the costs of conventional weed management. Furthermore, future weed-detection systems are anticipated to achieve higher precisions.

False Negatives (FN), weeds that were not detected by the UAV, have a negative effect on surrounding crops. These weeds cannot be removed by the UGV as there is no motivation for the UGV to drive to the weed's location. Calculating a cost price for missed weeds is difficult to quantify as the effect of a weed on a crop is complex and dependent on many variables. Furthermore, besides the yield diminishing effect of a weed on a crop, other costs which are even more difficult to estimate can also be attributed to weeds. E.g. volunteer potatoes collected during the collection of sugar beet can cause obstructions in the production line as the product in not homogenous. For this study, an estimated cost of €0.02 per weed is assumed. This value is symbolises the financial loss due to reduced crop growth.

2.2.2.2 Speed

As weed-growth is not a matter of minutes or hours, the speed needed to process the UAV images should not necessarily be in "real-time". Weed locations must first be transmitted to the UGV before removal of weeds starts. In principle, image processing could be done after the acquisition of images on a more powerful computer. However, it is preferable that the system does not limit weeding speed. For the Flourish Project (2018), the speed at which the UGV can cover a hectare is comparable to manual weeding (0.25 ha per hour for manual weeding). Thus, 0.25 ha per hour is the minimum processing speed at which the UAV deep-learning model should function. Quick removal of weeds is preferred to minimise the effect on the crop and to save the farmer time.

2.2.3 Scenario 2: Hopper UAV

This scenario relies on a UAV that both locates and treats weeds. As the UAV is flying across the field, it first detects weeds, then treats the weeds using herbicides, and then "hops" to other weeds. Although such a system would need to be trained on images wherein the size of the object varies greatly (i.e. the weed becomes bigger in the image when the UAV nears it), this aspect is not taken into account in this research. However, by extrapolating between the different resolutions found for this research and the resolution which was tested by Voorhoeve (2018), an estimation of performance can be made regarding the precision of the hopper UAV closer to the object.

2.2.3.1 Object-detection performance

Compared to *Scenario 1*, this system does not rely on a more accurate system to spray. Precision and recall should be high enough to minimise costs attached to operating the system and applying the herbicide. Each type of prediction has its own costs attached:

(1) TP predictions have operational costs (i.e. flying the drone) and the cost of herbicide dosage per weed. Applying herbicide on a weed growing near crops, might these crops. Although additional costs are expected in such a case, it is assumed that the application of the herbicide is precise enough to not affect the crops.
(2) FN predictions have costs related to the effect of weeds on surrounding crops. These costs are estimated to be €0.02 per weed; equal to scenario 1.
(3) FP predictions have operational costs, herbicide costs, and the cost tied to applying herbicide on a crop, killing it. The cost of losing a crop is estimated to be €0.07, which is a known price of a single sugar beet (Spruijt and van der Voort, 2015). Produce prices are dynamic which makes this cost an estimate with the aim to give a general idea.

2.2.3.2 Speed

As weed location should be updated while the UAV is flying, real-time processing is preferred. Following Milioto et al. (2017b), real-time is defined as having a refreshing rate of 1 image per second.

2.3 Literature review conclusion

The literature review shows that there is a plethora of deep-learning object-detection models which have been proven to generally perform well. However, the use case is the factor which decides which deep-learning model is most useful for the specific case. For this research, the use case is the detection of weeds (volunteer potatoes) in a sugar beet field. Two scenarios were developed in which it is needed to detect weeds from UAV imagery: (1) a UAV which scouts the area to identify the general location of weeds. This information is relayed to a UGV which drives to the weeds automatically and eliminates them. The second scenario consists of (2) a UAV which automatically detects weeds and sprays herbicides on the weeds; effectively flying from weed to weed.

Which object-detection model is used is thus dependent on the scenario. The first scenario does not require real-time (1 Hz) image processing as processing can occur after the flight, while the second scenario does require real-time processing as the drone should dynamically fly towards the weed, spray it, and locate another weed to repeat the process. Both scenarios benefit from an object-detection performance that is as high as possible. To satisfy both scenarios, the YOLOv3 deep-learning object-detection model developed by Redmon et al. (2015) and Redmon and Farhadi (2018) was deemed adequate for the use case and both scenarios as it's capable of real-time image processing and it performs relatively well compared to the other models.

This research also economically compares both scenarios to the conventional method of weed treatment. Although the practical economic feasibility of the scenarios can only be decided after the this research's results, it is clear that both scenarios should cost less per hectare compared to the conventional measure. Elements that influence the scenarios' costs are: the number of FP and FN. FP are assumed to bring larger cost as these could be crops which are killed by herbicides. The exact costs tied to these predictions depend on the market price per crop and one how neighbouring weeds affect crop growth. These values are variable and hard to determine. Thus, an estimation was made based on known sugar beet prices.

3 MATERIAL AND METHODS

3.1 Overview

The processing required for this research consists of (step 1) changing the resolution of UAV RGB images to have three different resolutions, (step 2) cropping, creating tiles and translating the bounding box annotations from the original images, (step 3) training and testing the YOLOv3 object-detection model five times per different resolution, and (step 4 & 5) comparing model performance for all image resolutions through three different tests. A schematic overview of the methodology used for this research is illustrated in Figure 6. The following section will explain different steps more in depth.

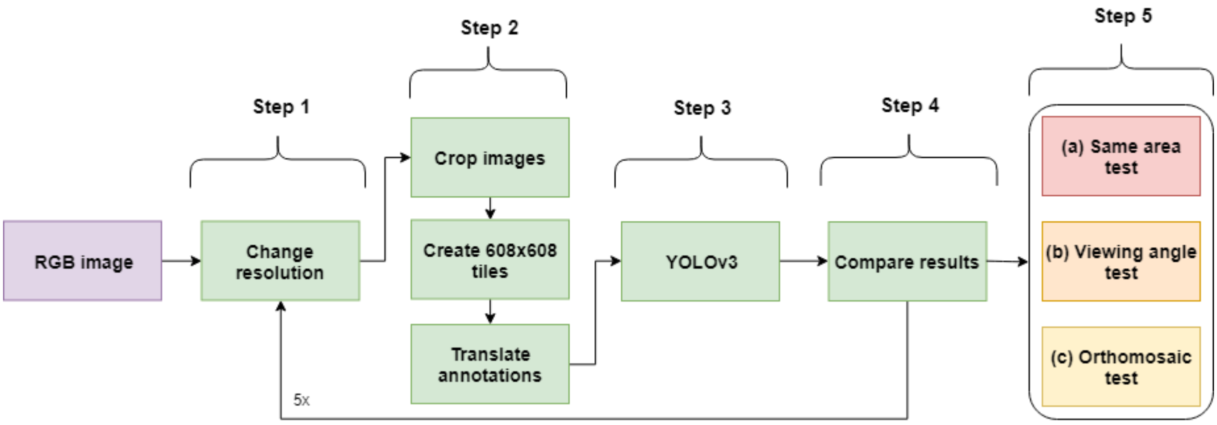


Figure 6: Schematic overview of the methodology per resolution.

3.2 Study area

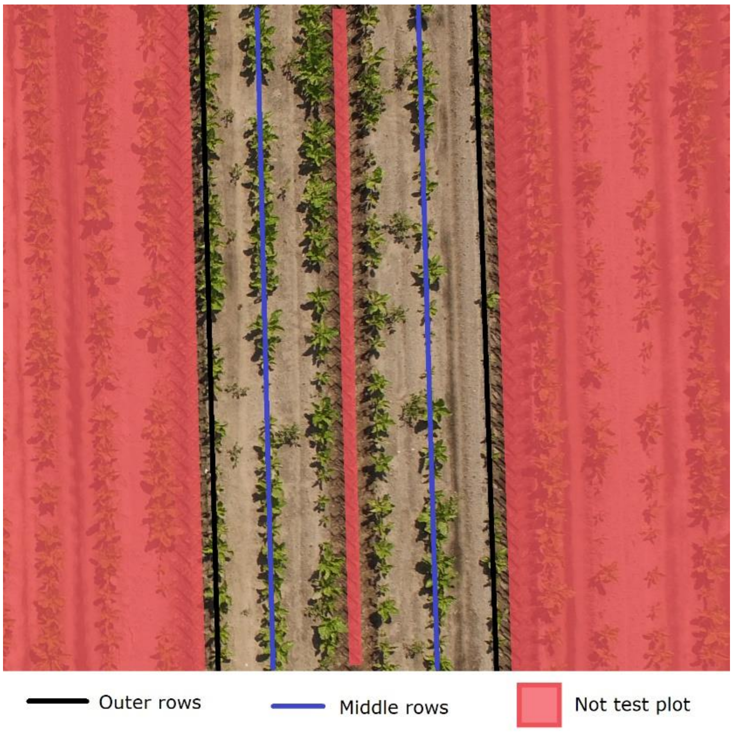


Figure 7: Test plots overview.

Data was gathered on the 6th of June 2018 around midday in Valthermond, The Netherlands. This research focussed on a sugar beet field planted for the SMARAGD project on the 21st of April 2018. Past crops include: barley (2016), and potatoes (2017). For every four sugar beet plants, one potato was planted randomly throughout the test plots to simulate voluntary potatoes. Potatoes were planted both between rows and in the rows. The dimensions of the test plot with voluntary potatoes are: two subplots of approximately 250 m by 1 m, which accumulates to 500 m^2. These subplots are separated by a tractor tire track where no plants are found. Similar tire tracks were also found on the outer sides of the subplots. The row distance is 0.5 m, while the planting distance between sugar beets in a row is 0.2 m. This results in a total of 6 rows with approximately 1250 sugar beets per row. For every subplot, the sugar beets in the middle row were taken into account to decide upon the number of potatoes to be planted (Figure 7). Thus, for the 1250 sugar beets per middle row, 313 potatoes were planted. However, according to Beukema and Van der Zaag (1990), more than 90% of potato sprouts with visible root growing points developed into stems.

This results in approximately 282 sprouted potatoes. The voluntary potatoes are deemed weeds and are the focus of the weed detection system. An actual count of the number of potatoes planted and sprouted is not available.

The above mentioned crop and weed count is assumed for potato crop grown under ideal conditions. However, field conditions did not allow for the crops and weeds to grow homogenously. The main reason for heterogeneity in the field is due to two regions which displayed signs of being waterlogged in the past. This phenomenon is identifiable by the dark and moist soil in the orthomosaic image (ANNEX I).

3.3 Data acquisition and pre-processing

Before being able to process the data, data had to be acquired and pre-processed.

3.3.1 Data acquisition

Individual RGB images taken by UAV were the primary data of this research. A summary of the image properties can be found in ANNEX II. 148 images were taken at an altitude of 10 m with a *Phantom 3 Standard* UAV which results in a pixel size of 4 mm. Image resolution is 4000x3000 px. The forward and lateral overlap between neighbouring images was approximately 60% and 75% respectively. This resulted in a large part of all images overlapping. Double-counts cause by overlapping images are irrelevant for this research as an actual ground-truth count of weeds is unknown and weed labelling is done per image. The images were captured along four flight paths (Figure 8). Each path views the weeds from a different range of angles. The available viewing angles relative to the centre point of the image (nadir) corresponding to the four flights paths are: 21.9° to 39.5° from nadir (Flightpath 1), 10.9° to 39.5° from nadir (Flightpath 2), 0° to 24.5° from nadir (Flightpath 3), and 0° to 31° from nadir (Flightpath 4).

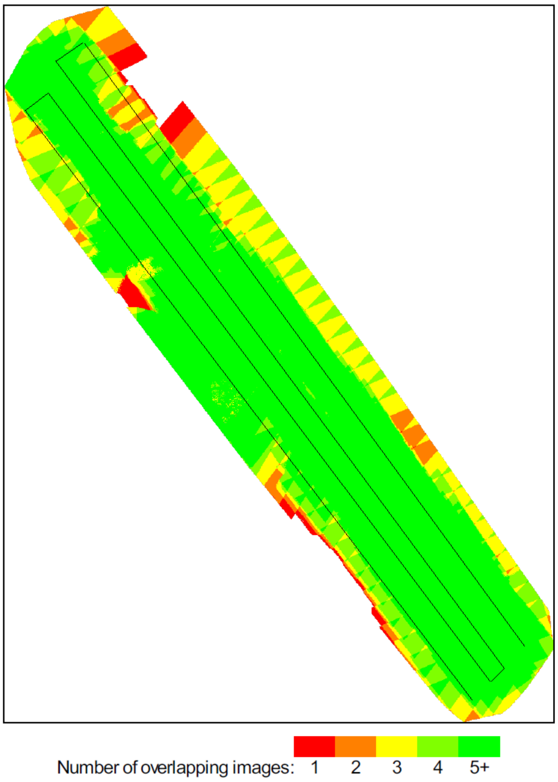


Figure 8: Overlapping images and UAV flightpath. Although some areas have a small number of overlapping images, the study area is found in the centre of the field.

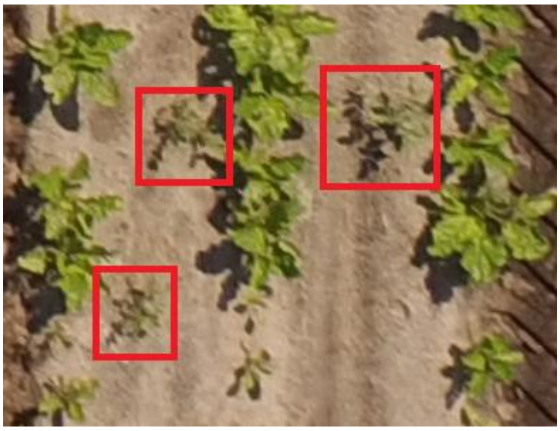


Figure 9: Example of labelled weeds. Weeds come in various shapes and sizes.

3.3.2 Data annotation

Training samples were labelled using the LabelImg software. Labelling refers to visually identifying and manually demarking the location of a weed in an image. A label is a box surrounding the weed (Figure 9). Using this tool, only weeds were labelled in the original images. However, these labels will be automatically translated to the lower resolution images as shown in *3.5 Create 608x608 px tiles from images (Step 2)*. Although there is no clear definition of a correct number of training data, previous studies have used training datasets containing 3.000 to 65.000 weed training samples (Lottes and Stachniss, 2017; Milioto et al., 2017a). This large range led to the decision of labelling as much weeds as possible to have an idea of the number of weeds on the field and to have sufficient training data. This resulted in a total number of labelled weeds for training of approximately 1400 to 1700 for the two best resolutions tested and 1000 to 1400 for the worst resolution tested; depending on the dataset tested (see *3.3.3 Data splitting*). Information regarding the tested resolutions can be found in *3.4 Image resolution change (Step 1)*.

3.3.3 Data splitting

Due to overlap in images, most weeds were found in more than one image. Although the repeated weeds are seen from different angles in different images, this can result in unfairly accurate predictions if training data are selected randomly as the model will be tested on weeds which were used for model training. To prevent testing on weeds used to train the model, the data were split into test and training data. To prevent overlap between these sets, a buffer was created between them (Figure 10). The test set consists of 20 images and was used to test the performance of the trained model. This accounted to roughly 14% of the total data. The buffer set consists of 4 or 8 images, depending on the spatial location of the test set. I.e. if the test set is at a longitudinal border of the field, it only needs a buffer on one side (Figure 10-A), otherwise it needs borders on both sides (Figure 10-B). The buffer accounted to approximately 3% or 5% of the total number of images. The training set consisted of 120 or 124 images, depending on the size of the buffer. This accounted to roughly 84% or 81% of the total number of images. Using this method of data splitting, weeds in the training set were assured not to be in the test set.

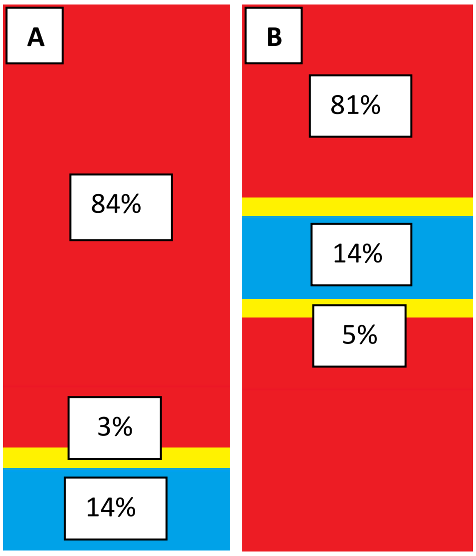


Figure 10: Two schematic overviews of the field to illustrate the difference in data splitting for two different runs of the model. A: 84% of the data (red) will be used for training, 3% (yellow) as buffer and 14% (blue) for testing. B: 81% of the data (red) will be used for training, 5% (yellow) as buffer and 14% (blue) for testing

An average of results was created by using k-folds cross validation with k = 5. k-folds cross validation divides the data in a k number of folds. Every fold consists of different set of training, test and buffer images, which depict a different area of the field. For each fold, the deep-learning model is run, resulting in k different performance results which are then averaged. This method maximises the number of usable data items (i.e. images and the objects found in those images). Only the first fold has a single buffer (Figure 10-A). As deep-learning results depend on training, an average of every run of the cross validation is needed to take into account outliers and give a better estimation of system performance.

The k-fold cross validation will be done per tested resolution. As the original images were resized three times, a total of 15 models were trained and run. More information regarding resizing the original images can be found in the next chapter: *3.4 Image resolution change (Step 1)*.

3.4 Image resolution change (Step 1)

RGB images were fed into the object-detection model to train, validate and test. However, to attain datasets with lower image resolutions and lower pixel size, a Python script was developed. Resizing was done using *pixel area relation* as interpolation method. This method calculates an average value for a certain window which is defined by a resize factor for the x and y axis (e.g. 1 to keep the image identical, 0.5 to make the image two times smaller) and the original image size. The window passes through the image and averages pixel values in the window. It’s recommended to use *pixel area relation* when lowering image resolution (Tanbakuchi, 2018). For this research, the OpenCV library module named *resize* was used as it is freely available (Figure 11). This module takes an input image and resizes it according to an x- and y-axis scale factor using an interpolation method of choice.

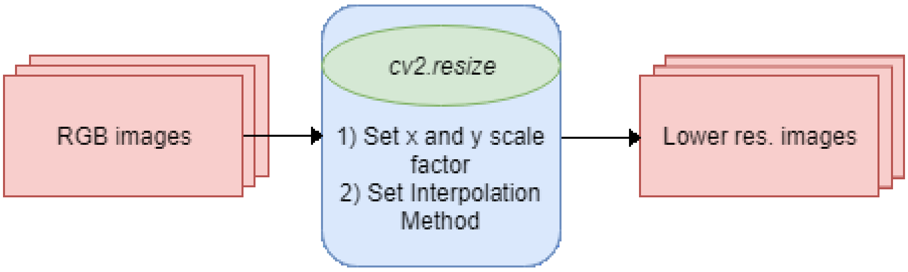


Figure 11: Summary of method to lower image resolutions. Two resized datasets are created using this method. Each dataset contains all of the original images resized.

Figure 12 shows the change in resolution using the *resize* module. The original image was downscaled with a factor of 0.5 for both the x- and the y-axis using the proposed interpolation method. The method proposed was repeated two times with all images to emulate two different flying altitudes. Scale factors of 0.5 and 0.31 were used; henceforth called the “0.5 dataset” and “0.31 dataset” respectively. Factor 0.31 is the lowest factor possible that does not result in the loss of a large part of the data in Step 2. The original images were also fed to the model, resulting in a total of three virtual flying altitudes.

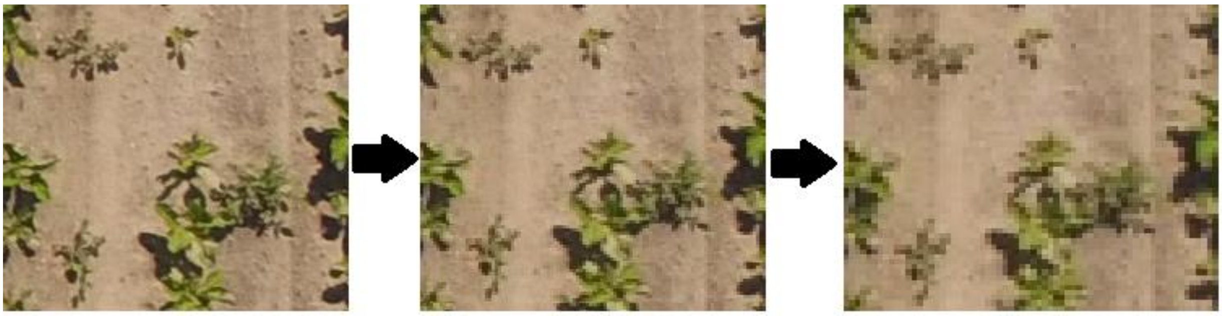


Figure 12: The original image (left) was resized by a factor of 0.5 (middle) and 0.31 (right). A loss of detail is evident after resizing.

3.5 Create 608x608 px tiles from images (Step 2)

The YOLOv3 deep-learning model automatically resizes images to a resolution of 608x608 px. As the data collected consists of larger images, resizing will remove detail, which is unwanted for this research. Thus, to maintain the original detail, the original images were split up into tiles of 608x608 px. These tiles split the image into non-overlapping sections that were used to train and test the model.

As the original images were 4000x3000 px, both dimensions not dividable by 608, the images were first cropped to a resolution of 3648x2432 px. Although roughly 1/5 of the area was cropped out, the large overlap between images reduces the effect of loss of data. The images resized by factor 0.5 and 0.31 were cropped to a resolution of 1824x1216 px and 1216x608 px respectively. The area per image corresponding to these different datasets are 0.014195 (141.95 m^2) ha for the original and 0.5 dataset, and 0.012303 ha (123.03 m^2) for the 0.31 dataset. Hereafter, the cropped images were split into tiles whilst maintaining the ground-truth annotation labels.

As the original annotation labels cannot be used for the resized images, a python script was developed to automatically adapt the ground-truth annotations to the tiles (ANNEX III). This script transforms the ground-truth bounding boxes to the tiles of the resized images. These steps were repeated with the images with changed resolution (*3.4 Image resolution change (Step 1)*). The resized images have different resolutions depending on the resize factor. This resulted in having to slightly adapt the script mentioned above to crop the images according to their corresponding resolutions. Figure 13 below illustrates the difference in 608x608 px tiles on different resolution images. As seen in Figure 12 and 13, detail was lost when resizing. However, the area covered is much larger in the resized tile; making the coverable area larger per unit of time.

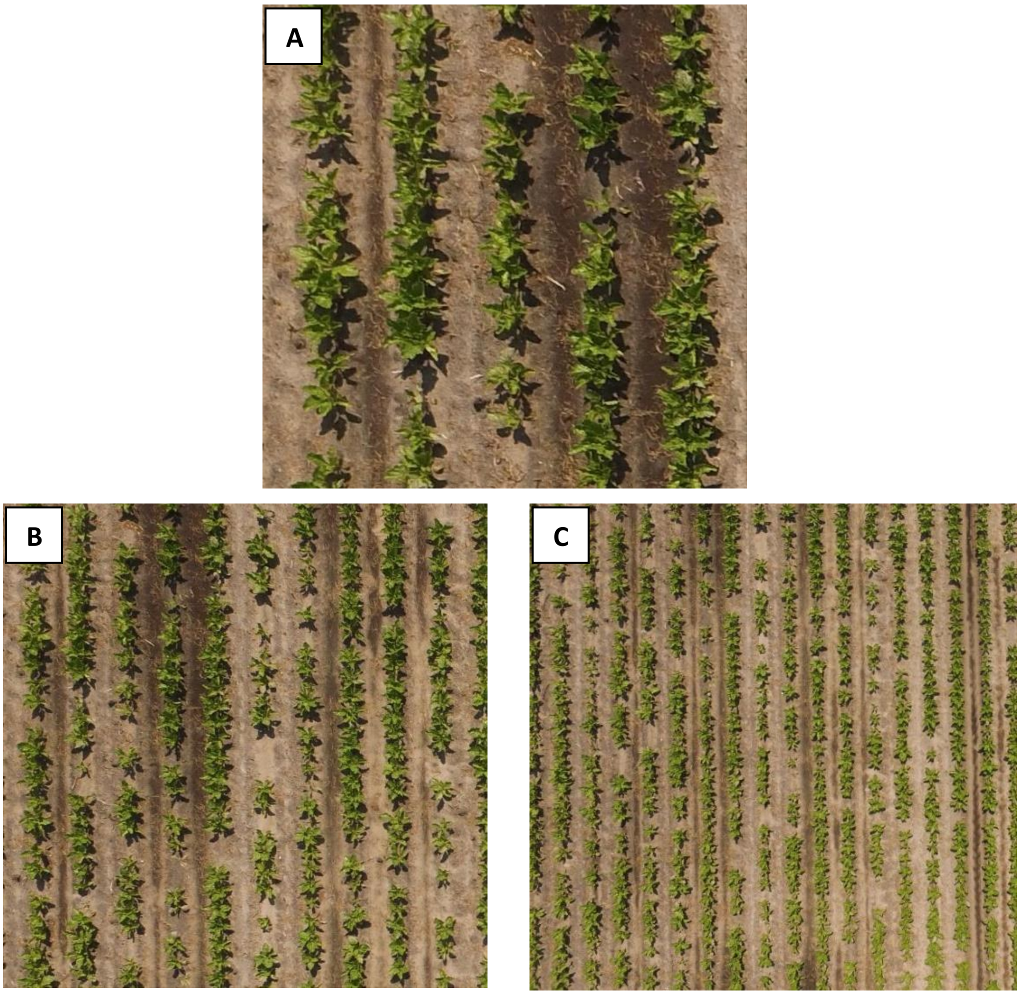


Figure 13: 608x608 px tiles corresponding to an original images (A) and an image resized by factor 0.5 (B) and 0.31 (C).

3.6 Deep-learning models (Step 3)

The basic functioning of the YOLOv3 deep-learning object-detector model is explained in *2.1.3 Object-detection models*. This section will explain YOLOv3 in further detail and clarify the effect of hyperparameters.

This model was first trained and tested for the different runs of the k-folds cross validation of all three resolutions to retrieve prediction bounding-box labels. Although it is possible to change model hyperparameters that might make the YOLOv3 more adept at detecting weeds, this research does not focus on improving the YOLOv3 model. Thus, model hyperparameters identical to those selected by Voorhoeve (2018) were chosen for this research. The chosen hyperparameters can be found in Table 2. To better understand YOLOv3 these hyperparameters will be explained:

Firstly, *momentum* and *decay* are hyperparameters which determine how weights are updated. YOLOv3 uses stochastic gradient descent to optimise the model weights (see *2.1.1 Machine learning vs. Deep* learning for more information on gradient descent). The hyperparameter *momentum* helps stabilise gradients in the right direction by penalising large weight changes between iterations, leading faster to the optimum. *Decay* controls the penalising term for large weight values which can lead to model overfitting.

Secondly, *angle, saturation, exposure* and *hue* control the data augmentation steps in YOLOv3. Data augmentation is applied on input data during the training phase of the object-detection models. This is done to make models more robust to various input (Howard, 2013; Liu et al., 2016). As raw training data is limited in the semantic data that can be extracted from it, this data is augmented to represent changes in the training object and thus result in more training data. The most common augmentation techniques change the original shape (e.g. rotation, scale, shear) and pixel values (e.g. changes in colour space uniformly for the whole image) of the objects (Howard, 2013; Liu et al., 2016; Redmon et al., 2015; Wirges et al., 2018). The YOLOv3 model randomly scales the original training images and translates up to 20% of the original size. Later, exposure and saturation of the sampled training objects are randomly adjusted up to a factor of 1.5. Focussing on the specific hyperparameters; *angle, saturation, exposure* and *hue* determine the range in which the training image is rotated, saturated, brightened and coloured during data augmentation.

Thirdly, as the model is not trained with all data at the same time but with batches of the total training data, *learning rate* controls how heavily the model should learn based on the current batch of data. At the beginning of the learning process, the *learning rate* should generally be high as the model has no information. However, as more information is added further in the learning process, *learning rate* should decrease progressively so that weights change less aggressively. This change in *learning rate* is defined by the hyperparameters *policy*, *steps* and *scales*. *Policy* specifies that *steps* will define when the *learning rate* will decrease. *Steps* stands for the number of iterations in which the *learning rate* will stay the same. After n *steps*, the *learning rate* is multiplied by *scales*; decreasing the *learning rate*. Although it was stated previously that the *learning rate* should be high at the beginning, empirical tests show that in practice a *burn-in* period should be created at the very beginning of the training process wherein the *learning rate* is lower. *Burn-in* controls the number of batches at the beginning wherein the *learning rate* is low.

Lastly, *max batches* determines how many iterations will run for the training process. Training a model with the original image resolution takes approximately one day – according to preliminary tests done on a NVIDIA 1080Ti *Graphics Processing Unit* (GPU).

Table 2: Hyperparameter settings used for model training.

Hyperparameter	Value
Momentum	0.9
Decay	0.0005
Angle	0
Saturation	1.5
Exposure	1.5
Hue	0.1
Learning rate	0.001
Policy	Steps
Steps	3800
Scales	0.1
Burn-in	1000
Max batches	15000

After training, the model can be tested on the test dataset. This results bounding boxes per tile which were used to evaluate the model performance.

3.7 Performance evaluation (Step 4)

This section first describes all metrics which will be used to assess the object-detection performance. Then, the different tests to evaluate performance will be described. The performance tests in combination with the literature study answer the all research questions.

3.7.1 Metrics

Different metrics were used to assess the performance of the deep-learning model. Some metrics were chosen as these are intuitive indicators of performance for the case at hand. Other metrics were chosen for comparative purposes as object-detection literature uses different metrics.

3.7.1.1 Average Intersection over Union

Intersection over Union (IoU) is a commonly used measure for accuracy of object detectors (Bottger et al., 2017). IoU measures the grade of overlap between labelled bounding boxes and predicted bounding boxes (Figure 14; Equation 1). Thus, a high IoU corresponds to a prediction very similar to the labelled location and size.

$$IoU = \frac{GT \cap P}{GT \cup P} \qquad \text{(Equation 1)}$$

Where: GT = Ground truth bounding box area
P = Prediction bounding box area

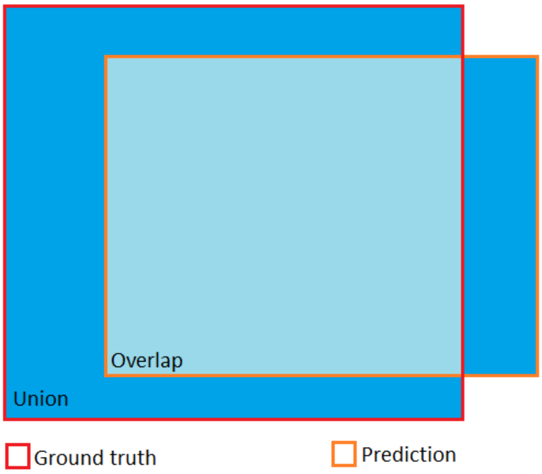


Figure 14: The elements needed to calculate IoU.

The IoU is used to find the TP, FN and FP values. The script used for this can be found in ANNEX IV. This script is based on the following stepwise method per tile:

1. A table was created with all ground truth bounding boxes on one axis and all predicted bounding boxes on the other. The IoU of all possible combinations was calculated and this value was put in its corresponding place in the table;
2. When available, matching ground truth and predictions were found in the table. This was done by selecting the largest IoU value in the table; if the IoU value was equal or larger than the IoU threshold (0.25 in this research), it was deemed a TP. As a ground-truth bounding box corresponds to only one weed, all other ground-truth and predicted bounding boxes with a IoU > 0 corresponding to the same TP were removed to prevent double counting;
3. FNs were calculated as the difference between the total number of ground truth bounding boxes and the TP. Similarly, FPs were calculated as the difference between the total number of predicted bounding boxes and the TP.

An IoU threshold of 0.25 was chosen for the assessment of a correct prediction. Meaning, that a prediction is deemed a true positive if the predicted bounding box overlaps for more than 25% with the annotated bounding box. This threshold was chosen as it was mostly deemed important that the system knows the approximate location of the weeds. In practice, a system could be used which has a higher threshold (for precise spraying) as it get in spraying distance.

3.7.1.2 Precision

This is the fraction of true positive predictions among all model predictions (Equation 2). Thus, it measures the percentage of correct positive predictions. A true positive observation in weed detection is a weed that was marked as weed during labelling, and is also detected by the model. For this research, a prediction with an IoU ≥ 25% was deemed a true positive. A false positive prediction would be a model prediction that does not correspond to an actual weed plant, thus a prediction with an IoU < 25%.

$$precision = \frac{TP}{TP+FP} \qquad \text{(Equation 2)}$$

Where: TP = True positive observations
FP = False positive observations

3.7.1.3 Recall

This is the fraction of TP predictions among all ground truth weed plants (Equation 3). A false negative observation in this case is a labelled weed that is not detected by the system, thus a missing true positive prediction when compared to the actual count of weeds.

$$recall = \frac{TP}{TP+FN} \qquad \text{(Equation 3)}$$

Where: FN = False negative observations

3.7.1.4 F1-Score

The F1-Score is the harmonic mean of precision and recall. This metric is deemed most useful to describe the performance of a model as it takes into account both precision and recall and assumes that both are equally important. The best and worst possible F1-scores are 1 and 0 respectively. It was calculated using Equation 4 below.

$$F_1 = 2\,\frac{precision * recall}{precision + recall} \qquad \text{(Equation 4)}$$

3.7.1.5 Mean Average Precision (mAP)

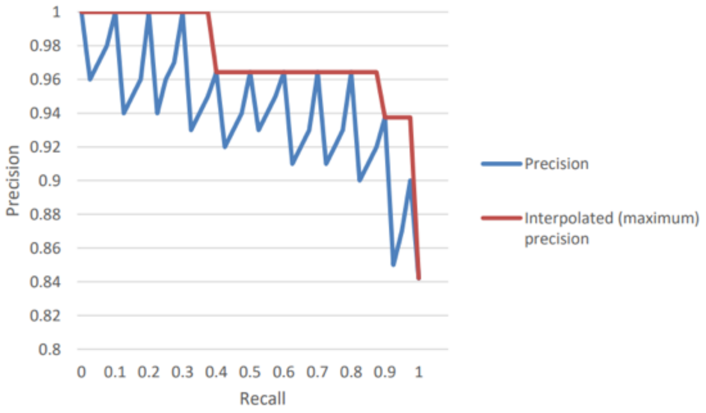


Figure 15: Precision-recall curve for a class. Taken from Voorhoeve (2018).

To calculate the mAP one must first understand average precision (AP). AP consists of the average maximum precision at different recall values (0, 0.1, 0.2 … 1) without taking into account the IoU threshold mentioned above. Thus, AP looks at the overall performance of the detector instead of focussing on precision for higher recall values (Oksuz et al., 2018). AP is calculated by sorting predictions by decreasing confidence score and assigning these model predictions to ground truth objects of the same class which have an IoU higher than the set threshold. A match is deemed a TP if the ground truth weed has not been paired to another model prediction before. These criteria are used to calculate the precision/recall curve (Figure 15). The precision/recall curve is used to compute the interpolated (maximum) precision line which is constituted of the maximum precision obtained for any recall. AP can be defined as the area under the interpolated (maximum) precision curve (Figure 15, red line). mAP is the average AP over the total number of classes (Equation 5).

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q} \qquad \text{(Equation 5)}$$

Where: Q = number of classes, 1 for this research

3.7.1.6 Classification of errors

Analysing images visually is prone to subjectivity by the analyser. Hence, objective measures to classify predictions are needed. After a preliminary visual analysis of the data, two different factors were found which influence the ability of the model to return FN weed predictions: weed size and occlusion by the crop row. Thus, FNs were divided into two types:

1. Small weeds: These are weeds that were thought not be predicted by the model due to their small size. Although size is relative and can be visually difficult to assess from images, it was based on the size of labelled bounding box. Following a preliminary study, a weed was deemed small for the original resolution when it has a pixel area of 500 px^2 which corresponds to 8000 mm^2 This value changes quadratically to 125 px^2 and 48.05 px^2 for the 0.5 and 0.31 datasets respectively. If a weed was not small it was deemed large. This area was determined by evaluating the images and measuring the average size of several weeds that were deemed to be small.
2. In-row weeds: These are weeds that were thought to be missed by the model due to them being in the sugar beet row. Occlusion by larger plants and a lack of clear boundaries between plants was expected to cause problems for the model when detecting these. A weed was counted as being in the sugar beet row, when the centre point of the bounding box was found in the row-space. The row-space was defined by a line in the middle of the crop row; by calculating the shortest distance from the centre line to centre point of a ground truth bounding box, one can classify the weed as being in or out of the crop row. For the original resolution, a weed was deemed to be in the crop row when this distance was smaller of equal to 20 px (80 mm). This results in a distance of 10 px and 6.2 px for the 0.5 and 0.31 datasets respectively. This distance was determined by evaluating the original images and measuring the distance between the centre of the row and the outer edges of the crop row.

These two types of FN errors ultimately resulted in four classes which are the result all possible combination of both types (Table 3). FP predictions were not analysed using this method due to time constraints.

Table 3: Confusion matrix for types of FN errors.

	Small	**Large**
In row	In and small	In and large
Out row	Out and small	Out and large

3.7.1.7 Time to run the model

The time it takes to run the models per dataset gives an indication on the real-time capabilities of the system. A minimal number of frames per second was expected to be retrieved from this measure. To make comparison fair, an NVIDIA GeForce GTX 1080Ti GPU was used to run all models. It is important to note that this GPU is not suitable for on-board processing on a lightweight UAV as it is relatively heavy and requires much energy. Thus, this measure did give a definitive answer on the capabilities for the models to run on a UAV but an indication of run times.

3.7.1.8 Model cost

In principle, there are three measured types of prediction: TP, FN and FP. These prediction types have different costs attached to them, the sum of which decide if using the model is practical. First, a TP yields the operational costs: herbicide and UAV/UGV run-time. Herbicide was estimated to cost €0.00033 per weed based on Ruigrok et al. (2018). Second, a cost of €0.02 related to the effect of not removing a weed and letting the weed reduce the yield of nearby plants was assumed for every FN. Third, a FP yields UAV/UGV run-time costs as well as the effect of herbicide on a crop, which was assumed to be €0.07. These costs are taken from *2.2 Economic comparison*.

An exception can occur when a weed is found close to a crop and herbicide might also be applied on the crop. In such a case, a TP yields both operational costs and the effect of the herbicide on the crop. This last case was not taken into account when calculating the model costs as it is assumed that the sprayer is accurate. Furthermore, from these abovementioned costs, the run time costs of both UAV and UGV were not included in the model cost calculation as these were assumed to be relatively low compared to conventional machine costs.

The practical feasibility of the model was defined by the ratio of TP, FP and FN and costs assigned to these. As long as the theoretical cost of using the system proposed in this research is lower than conventional methods, the model is deemed usable in reality.

The metrics described above will be used to evaluate model performance for the following three experiments described in further detail in the following sections:

a. A *performance per resolution test*, wherein the general performance of YOLOv3 was tested on a geographically defined area for different resolutions. Every fold, as described in *3.3 Data acquisition and pre-processing* above, tested a different area, resulting in five different areas which were tested.
b. A *viewing angle test*, wherein performance dependent on viewing angle from nadir was tested for different resolutions. For every fold, all labelled and predicted annotations were separated into different bins corresponding to different angles.
c. An orthomosaic test, wherein tiles from an orthomosaic image of the study area were tested. These tiles were selected and tested taking into account the fold to which they belong geographically.

3.7.2Performance per resolution test

The *performance per resolution test* attempts to show the effect of resolution on the different performance metrics for a geographically defined area. First, TP, FN and FP were found. These three values are the basis for precision, recall and consequently the F1-score and mAP. Then, the types of FN were retrieved:

1. For every individual FN bounding box, the shortest distance to the closest line was calculated. As a single tile can contain several crop rows, it is important that the distance to the closest crop row is calculated, as this is the row that logically has the highest probability of influencing the prediction;
2. If the calculated distance was equal or smaller than half of the defined crop row width, the FN prediction was counted as in the row. Otherwise it was counted as out of the row;
3. Lastly, the size of the bounding box was calculated to evaluate if the weed is small or large.

The method described in pseudocode above was run per tiles or all the different folds, for all tested resolutions. The script used to realise this performance test on the original resolution (4 mm) can be found in ANNEX IV. The method used to classify the types of FN errors was changed slightly to analyse the type of ground truth annotations. These changes include only looking at the ground truth annotations instead of first selecting TP, FP and FN as explained in *3.7.1.1 Average Intersection over Union*, and running the pseudocode described above. The scripts for the other resolutions are identical, but use other input values and parameters such as *distance to row*, and *small weed area*.

To better understand the relationship between resolution, fold and F1-score, an *Analysis of Variance* (ANOVA) will be performed. Understanding the influence of the fold on the F1-score can explain if the model works equally on all areas of the field, or if there is a significant effect of the area on the F1-score. To gain more insight on the direction of these relationships, a linear model will be fitted through the data. These statistical methods will answer the first research question.

An effect of an independent variable on a response variable is deemed significant for p-value < 0.05 (i.e. conclusions are valid at 95% probability).

This research follows the following statistcal terminology:

A_i = Fold, i = 1,2,3,4,5
B_J = Resolution, j = 1,2,3
Y_{ijk} = F_1 Score, k = 1,2,…,300
(300 is the sum of number of images for all folds and all resolutions, so the total number of observations)
Independent variables: Resolution and Fold
Response variable: F1-score

The model used for the ANOVA test is:

$$Y_{ijk} = \mu_{ij} + \varepsilon_{ijk}$$

with $\varepsilon_{ijk} \sim N(0, \sigma^2)$ and $\mu_{ij} = \mu + \alpha_i + \beta_j + \gamma_{ij}$

Where: μ = the general mean
α_i = the effect of the Fold
β_j = the effect of the Resolution
γ_{ij} = the joint effect of Fold and Resolution.

We are interested in testing the difference of the mean F_1 score according to the different Folds:
H_0: $\alpha_i = 0, i = 1, \dots ,4$
The null hypothesis: there is no significant effect of fold on the F1-score.
H_1: $\alpha_i \neq 0, i = 1, \dots ,4$
The alternative hypothesis: there is a significant effect of fold on the F1-score.

We are interested in testing the difference of the mean F_1 score according to the different Resolution:
H_0: $\beta_j = 0, j = 1,2$
The null hypothesis: there is no significant effect of resolution on the F1-score.
H_1: $\beta_j \neq 0, j = 1,2$
The alternative hypothesis: there is a significant effect of resolution on the F1-score.

We are interested in testing the difference of the mean F_1 score according to the different Fold and the different Resolution together:
H_0: $\gamma_{ij} = 0, i = 1, \dots ,4, j = 1,2$
The null hypothesis: there is no significant effect of fold and resolution combined on the F1-score.
H_1: $\gamma_{ij} \neq 0, i = 1, \dots ,4, j = 1,2$
The alternative hypothesis: there is a significant effect of fold and resolution combined on the F1-score.

3.7.3 Viewing angle test

To assess the relationship between viewing angle from nadir and performance, the viewing angle from nadir for every bounding box was calculated first. This was done by first calculating the ground distance from the centre point of the image (directly at nadir) to the centre point of every bounding box. Knowing this distance and combining it with a known flying altitude of 10 metres, allowed for the calculation of the viewing angle from nadir.

As the whole images were cropped to allow the creation of tiles, the centre point of the cropped images is not directly at nadir relative to the sensor (Figure 16). Thus, when calculating the angles, this was taken into account by adding the difference in pixel centre x and y position to the absolute coordinates of the bounding boxes – making all values relative to the original image dimensions.

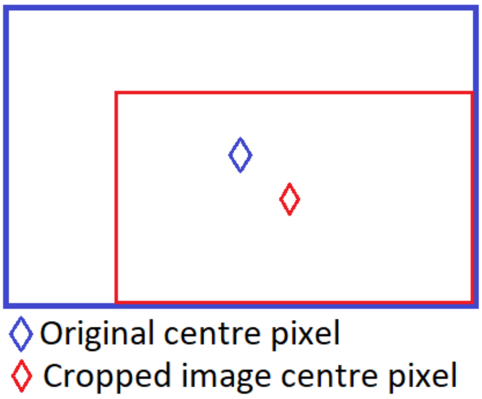


Figure 16: Centre pixel shift due to image cropping. The centre pixel of the cropped images cannot be used as nadir.

In pseudocode, the calculation of object viewing angle from nadir for a single weed was as follows. The script used to calculate the angles can be found in ANNEX V.

1. The x and y centre coordinates of the bounding box relative to the original image are calculated. This is done with the following equations:

$$x_{abs} = (x * tile_{width}) + (col_{number} * 608) + x_{difference} \qquad \text{(Equation 6)}$$
$$y_{abs} = \left(y * tile_{height}\right) + (row_{number} * 608) + y_{difference} \qquad \text{(Equation 7)}$$

Where: x/y_{abs} = the absolute x and y bounding box centre-point coordinate for the original image
y_{abs} = the absolute y-coordinate of the bounding box for the whole image
x = the x-coordinate relative to the cropped image
y = the y-coordinate relative to the cropped image
$tile_{width/height}$ = the width and height of the tile
col/row_{number} = the row and column number of the respective tile
$x/y_{difference}$ = the difference in x and y dimension of between the original and cropped image

2. The absolute x and y coordinates derived from Equation 6 and 7 were then used to calculate the distance from nadir to the centre of the bounding box using Pythagoras' theorem ($A^2 + B^2 = C^2$). A^2 and B^2 were calculated using Equation 8 and 9:

$$A^2 = (abs(x_{abs} - x_{nadir}) * pixelSize)^2 \qquad \text{(Equation 8)}$$
$$B^2 = (abs(y_{abs} - y_{nadir}) * pixelSize)^2 \qquad \text{(Equation 9)}$$

Where: x/y_{nadir} = the centre-point coordinate of the whole image
pixelSize = original pixel size of the image

3. Using the C^2 the viewing angle from nadir can be calculated using Equation 10:

$$angle = \operatorname{atan}\left(\frac{C}{FlyingAltitude}\right) \qquad \text{(Equation 10)}$$

Where: c = distance from object centre-point to nadir
FlyingAltitude = UAV flying altitude

After the angles were calculated for all bounding boxes, the spread of these angles were studied for all resolutions. An example of the histogram for the original resolution is depicted in Figure 17. Such histograms were used to select adequate bins for the *viewing angle test*. A distance of 3 degrees was preferred between all bins. However, the first and last bin do have a larger difference as these would otherwise be separated into bins which would contain less than 30 annotations, a sample size that was deemed too small. The same bin sizes were used for all resolutions to enhance comparability.

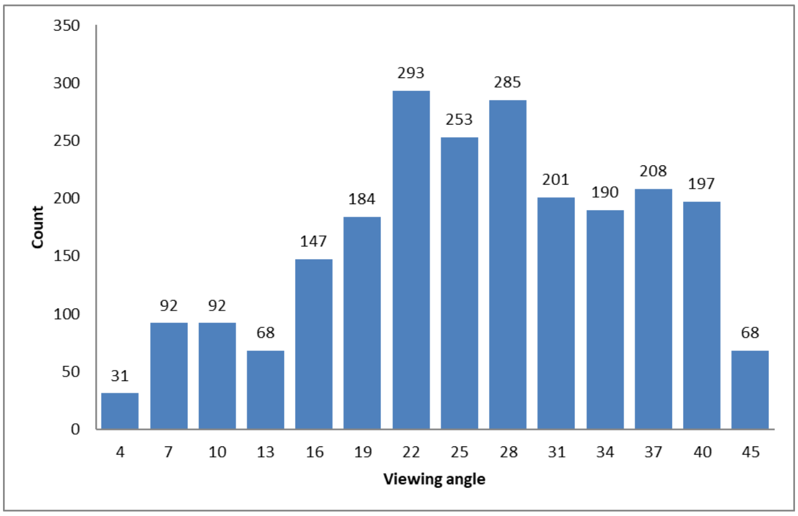


Figure 17: Viewing angle distribution of the number of annotations per angle class for the original resolution images (4 mm).

To assess performance based on viewing angle, the method to find TP, FN and FP discussed in *3.7.2 Performance per resolution test* was used. However, for this test, both ground truth and predicted bounding boxes are first classified based on their corresponding viewing-angle bin and then the TP, FN and FP predictions are found for all predictions per bin. The script used for this method can be found in ANNEX VI.

To assess the statistical significance of the relationship between viewing angle and performance, a linear regression model was fitted to the data for all resolutions and the correlation coefficient (R) was calculated per resolution. For this test, the results from all folds are combined based on their resolution.

3.7.4 Orthomosaic test

Orthomosaic images are often used in the realm of remote sensing, but research on object detection using orthomosaic images is limited. Such images are a geometrically corrected composite of different overlapping images, making it possible to attach geographical coordinates to any place in the image. The *orthomosaic test* aims at testing the deep-learning system trained for this research on an orthomosaic image. This test was relatively small scale (i.e. not much data was collected) and done to get a general ideal of how YOLOv3 performs on orthomosaic images to provoke thought and ideas for future research.

As orthomosaic images are large composites of several images, it was not possible to feed a whole orthomosaic image to the model. Such an image is orthorectified, which changes the visual direction of objects in the image (Figure 18). Thus, image cropping could not be done in the same way as before. Instead, the orthomosaic image was compared visually to the original tiles. Once the same area was identified on both images, a tile corresponding to that area was cropped from the orthomosaic image using QGIS (QGIS Development Team, 2016). Extraction of tiles from the orthomosaic was done by first creating a shapefile with the approximate dimensions of a tile, and then using the *Clipper* tool in QGIS whilst using the shapefile as a mask. Due to orthorectification, identical tiles were not possible to create. However, orthomosaic tiles were clipped in a way so as many identical objects were found in both tiles (Figure 18). As the orthomosaic image was of the GTIFF, which also depicts transparency and is not processable by YOLOv3; the cropped tiles were reformatted to the JPG format. These orthomosaic tiles were subsequently cropped to be exactly 608x608 px using a simple Python script, labelled identically to the original tiles using LabelImg and comparing the labels with the labels of the original image. Lastly, crop lines were drawn again using the line tool used for the original dataset.

To test performance, the same script was used as for the *Performance per resolution test*. However, results were not compared between resolutions; instead only the original resolution will be compared to the orthomosaic image. This test was done for 22 tiles corresponding to the test set of the fifth fold (original resolution). Thus, the model trained for the fifth fold was used to predict weeds in the orthomosaic tiles.

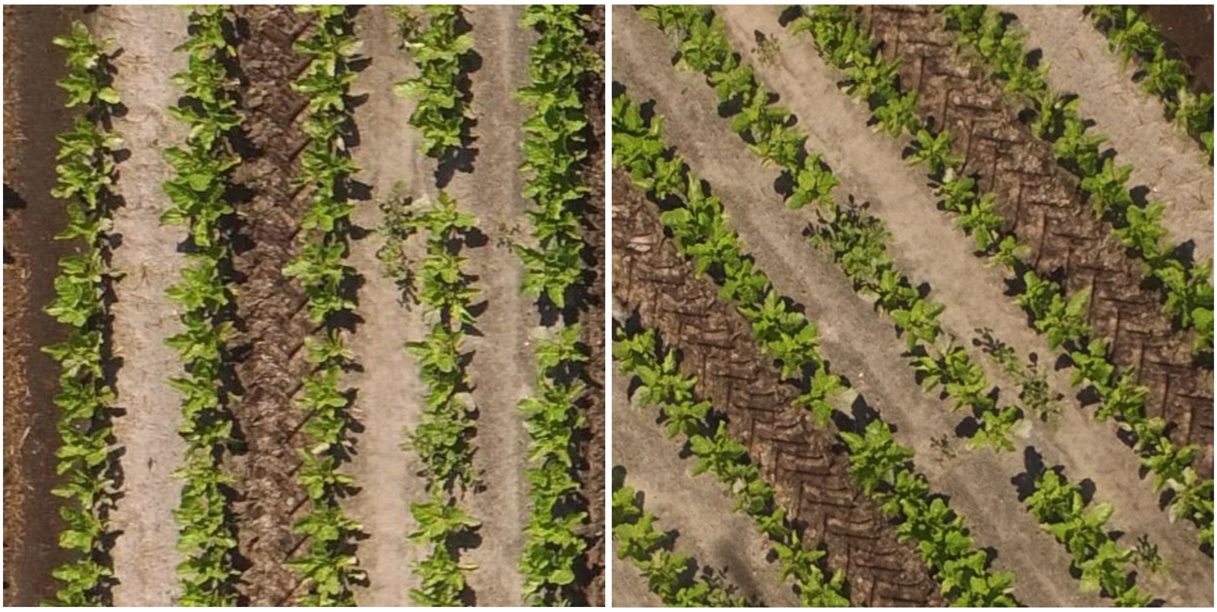


Figure 18: Difference in direction of objects. Left: Original image, crop rows are vertical. Right: Orthorectified image, crop rows are diagonal.

4 RESULTS

Running YOLOv3 for the three different resolutions results in a set of results for every resolution. Most metrics described in section *3.7.1 Metrics* were used to measure performance of all tests. However, mAP could only be calculated for the performance per resolution test. Furthermore, average IoU, time to run the model and model cost are metrics which are only relevant per resolution. These three metrics will be discussed separately from the three tests mentioned in the methodology.

4.1 Performance per resolution test

After training and testing five folds for every one of the three resolutions, the following results were acquired from the object-detection model. A summary of these results is shown in Table 4.

Table 4: Summary of performance values per resolution.

Resolution	F1-score	Precision	Recall	mAP	Run time
Original (4 mm)	0.83	0.95	0.77	75%	49 sec/ha
0.5 (8 mm)	0.82	0.94	0.75	72%	14 sec/ha
0.31 (12.9 mm)	0.78	0.95	0.71	62%	6 sec/ha

4.1.1F1-score

Focussing on the F1-score for five folds per resolution; a trend is found when looking at the average (Figure 19). The better the resolution, the higher the average F1-score. The original, 0.5 and 0.31 dataset have an approximate average F1-score of 0.83, 0.82 and 0.78 respectively. The highest F1-score was achieved by the 0.5 dataset (0.89), while the lowest was scored by the 0.31 dataset (0.48).

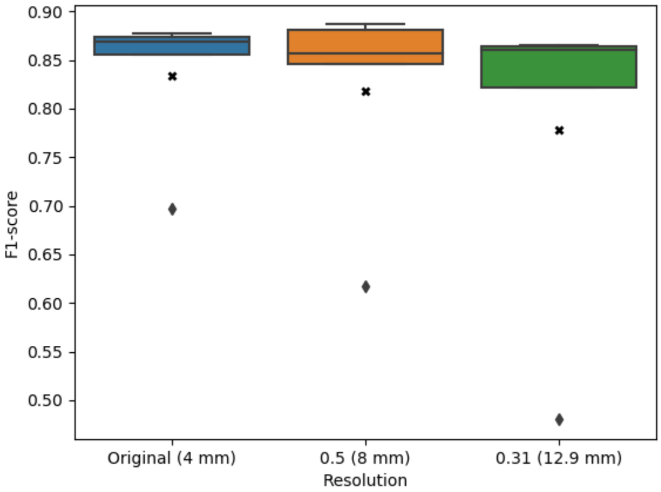


Figure 19: F1-score for all folds in all resolutions. Every datapoint corresponds to a fold. The cross symbol represents the average; the diamonds represent outliers.

Another trend that can be found in the F1-scores has to do with the standard deviation. The standard deviation of these different datasets are 0.07, 0.10 and 0.15 for the original, 0.5 and 0.31 datasets respectively (Table 5). Thus, the higher the resolution, the lower the dispersion.

The averages are brought down in part by the results of the first fold. Removing the first fold returns average F1-scores for the original, 0.5 and 0.31 dataset of 0.8684, 0.8676 and 0.8527 respectively. Although the same trend can be seen, the difference in F1-scores between resolutions is minimal when removing the first fold. The standard deviation in the case without outliers is 0.01, 0.02 and 0.02 for the original, 0.5 and 0.31 dataset correspondingly (Table 5).

Table 5: F1-score standard deviation per resolution.

	SD	SD No Outliers
Original	0.069	0.008
0.5	0.101	0.016
0.31	0.150	0.018

Further zooming into individual resolutions, a spread in F1-scores is evident (Figure 20). Concentrating on the original resolution first, fold 2 to 5 depict relatively stable average F1-scores (between 0.86 and 0.88). Conversely, fold 1 has a lower average F1-score of roughly 0.7. This same trend is seen in the other two resolutions. However, for the 0.5 and 0.31 dataset, fold 1 has an average of 0.62 and 0.48 respectively. Fold 2 and 5 on the other hand, average between 0.85 and 0.89 for the 0.5 dataset, and 0.82 and 0.86 for the 0.31 dataset.

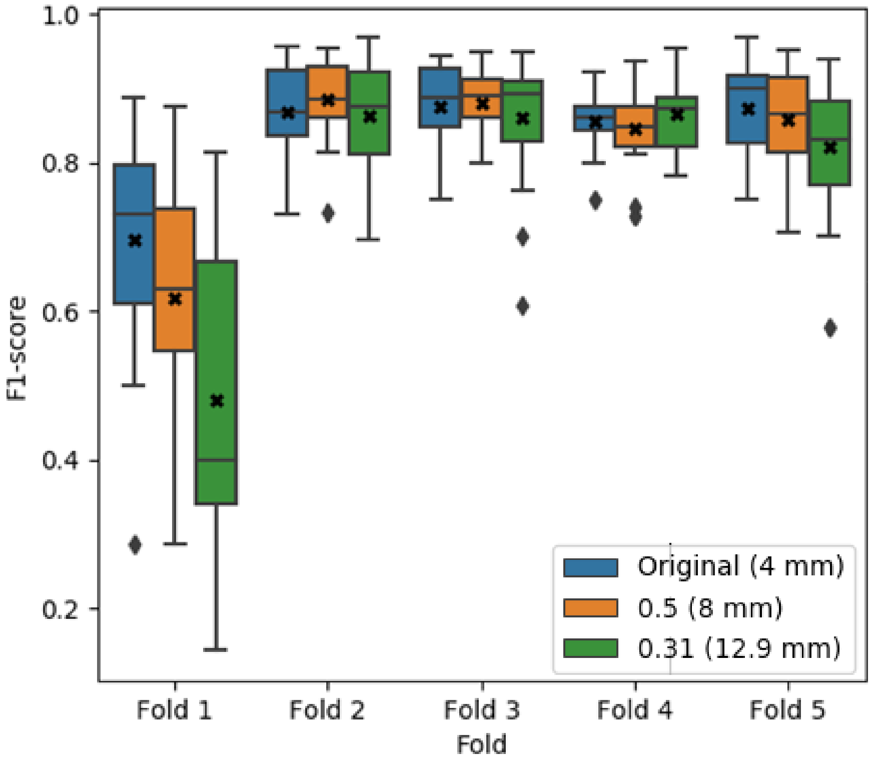


Figure 20: F1-scores for all images per fold, for all three tested resolutions. Every datapoint corresponds to one tile. Per fold, 20 images were used for tile creation.

Looking at the spread of F1-scores in every fold, the largest standard deviation always corresponds to the first fold (Table 6). For the original, 0.5 and 0.31 datasets, the F1-score standard deviation for the first fold is 0.15, 0.16 and 0.21 respectively. The lowest standard deviation corresponds to fold 4 for the original and 0.31 dataset, 0.04 and 0.05 respectively. The lowest standard deviation for the 0.5 dataset was found for fold 3 at 0.04.

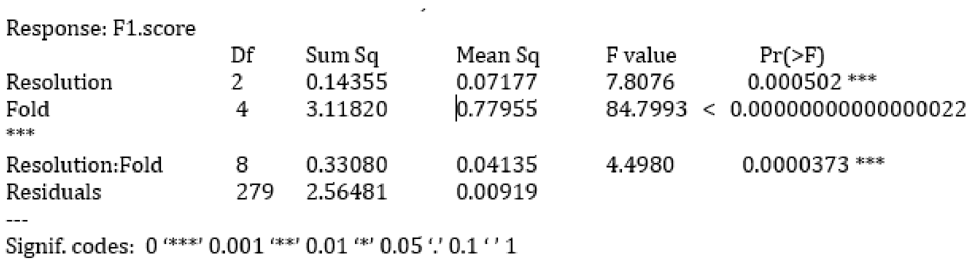
Table 6: F1-score standard deviation per fold per resolution.

	Original	**0.5**	**0.31**
Fold 1	0.15	0.16	0.21
Fold 2	0.06	0.05	0.07
Fold 3	0.05	0.04	0.09
Fold 4	0.04	0.05	0.05
Fold 5	0.06	0.07	0.08

Based on the averages per resolution, the trend that resolution shows is different dependent on the fold. The first and fifth fold show the same trend depicted in Figure 20; the second and third fold resulted in the 0.5 dataset performing better in average whilst the 0.31 dataset performed worst; for the fourth fold, the 0.31 dataset resulted in the best performance while the 0.5 dataset performed worst.

An ANOVA was performed on the F1-scores depicted in Figure 20 (Table 7). This analysis shows that: For resolution, fold and the combination of both, the F value is significantly (p-value < 0.05) larger than 0. Thus, the mean F1-score is significantly different according to the different levels of the resolution, of the fold and of the resolution and the fold together.

Table 7: ANOVA. Tests of between-subjects effects for the performance per resolution test.

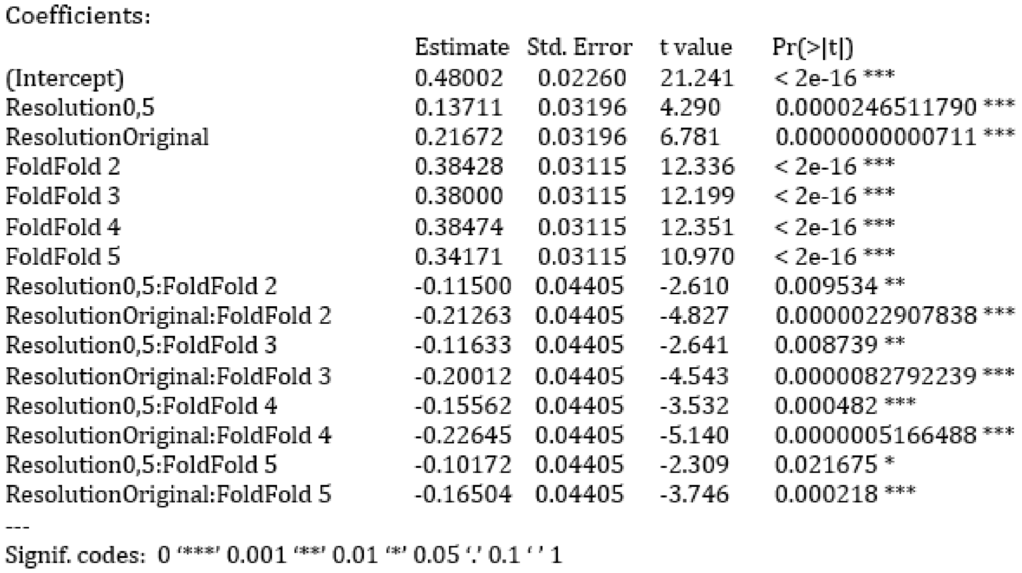


```
Response: F1.score
                    Df      Sum Sq      Mean Sq     F value       Pr(>F)
Resolution          2       0.14355     0.07177     7.8076        0.000502 ***
Fold                 4      3.11820     |0.77955    84.7993  <  0.00000000000000022
***
Resolution:Fold      8      0.33080     0.04135     4.4980       0.0000373 ***
Residuals          279      2.56481     0.00919
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

To better understand the direction of the relationship between resolution, fold and F1-score, a linear model was fitted to the data (Table 8). Following the ANOVA results, the linear model shows that:

1. A higher resolution has a significant positive effect on the mean F1-score. As the estimate value for both Resolution0.5 and ResolutionOriginal > 0, the baseline (0.31 dataset) performs significantly worse compared to the other resolutions.
2. The mean F1-score is significantly dependent of the fold being tested. As all FoldFold *N* estimate values > 0, the baseline (Fold 1) performs significantly worse compared to the other folds.
3. Combining both the resolution and fold, significantly improves the explanation of F1-score given that the joint effect of resolution and fold on the F1-score is statistically significant. The effect of resolution on the F1-score is dependent on the fold as all combinations of resolution and fold with respect to the chosen baseline shows a p-value lower than 0.05.

Table 8: Performance per resolution test linear model results.



```
Coefficients:
                                Estimate  Std. Error   t value    Pr(>|t|)
(Intercept)                     0.48002    0.02260     21.241     < 2e-16 ***
Resolution0,5                   0.13711    0.03196     4.290      0.0000246511790 ***
ResolutionOriginal              0.21672    0.03196     6.781      0.0000000000711 ***
FoldFold 2                      0.38428    0.03115     12.336     < 2e-16 ***
FoldFold 3                      0.38000    0.03115     12.199     < 2e-16 ***
FoldFold 4                      0.38474    0.03115     12.351     < 2e-16 ***
FoldFold 5                      0.34171    0.03115     10.970     < 2e-16 ***
Resolution0,5:FoldFold 2        -0.11500   0.04405     -2.610     0.009534 **
ResolutionOriginal:FoldFold 2   -0.21263   0.04405     -4.827     0.0000022907838 ***
Resolution0,5:FoldFold 3        -0.11633   0.04405     -2.641     0.008739 **
ResolutionOriginal:FoldFold 3   -0.20012   0.04405     -4.543     0.0000082792239 ***
Resolution0,5:FoldFold 4        -0.15562   0.04405     -3.532     0.000482 ***
ResolutionOriginal:FoldFold 4   -0.22645   0.04405     -5.140     0.0000005166488 ***
Resolution0,5:FoldFold 5        -0.10172   0.04405     -2.309     0.021675 *
ResolutionOriginal:FoldFold 5   -0.16504   0.04405     -3.746     0.000218 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4.1.2 Precision and recall

To better understand the F1-score, it's constituents should be explored. Focussing on precision first, a lower resolution does not necessarily lead to a lower average precision (Figure 21-A). The original, 0.5 and 0.31 dataset have an average precision of 0.945, 0.938 and 0.946 respectively. Thus, precision is deemed relatively high for all tested resolutions. The spread per dataset shows that in some cases the worst resolution returns higher precision values compared to the higher resolutions. However, the differences between datasets are in most cases minimal. E.g. all maxima are between 0.95 and 0.96. Minima for all datasets lie between 0.90 and 0.93. These minima correspond to fold 4 for the original dataset, and fold 1 for the 0.5 and 0.31 datasets. The maxima however, correspond to fold 2 for the original and 0.5 dataset, and fold 3 for the 0.31 dataset.

Similar to F1-score, average recall also diminishes as resolution worsens (Figure 21-B). The original, 0.5 and 0.31 dataset have an average recall of 0.77, 0.75 and 0.71 respectively. The outlier minima in recall values are found for the first fold, while the maxima correspond to the fold 3 for the original and 0.5 dataset, and fold 4 for the 0.31 dataset.

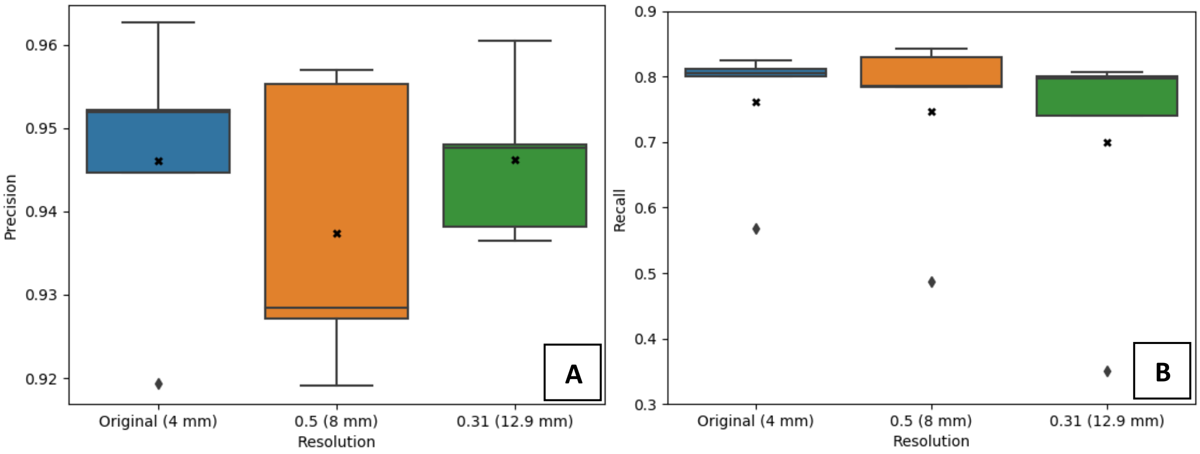


Figure 21: Precision (A) and recall (B) for all resolutions.

4.1.3 FN classification

Relative to the total number of ground truth annotations per resolution; the higher the resolution, the lower the percentage of FN (ANNEX VII). The corresponding percentage of FN for the original, 0.5 and 0.31 dataset are 22%, 23% and 27% respectively. These total percentages of FN are constituted differently per resolution.

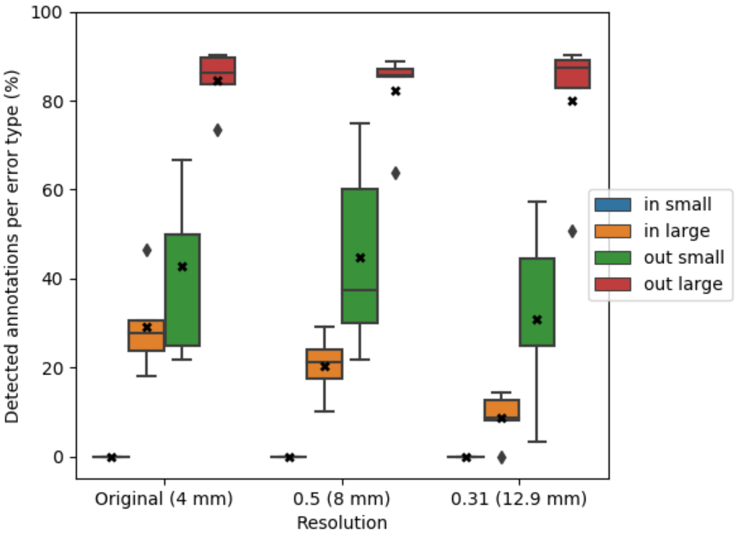


Figure 22: Correctly predicted weeds per error type for all resolutions. Datapoints correspond to individual folds.

Figure 22 depicts the percentage of correctly predicted annotations per type of error. Weeds in the class "in small" are most difficult to detect as none of these are detected for any of the resolutions. However, this class is also the less common class (Figure 23; Table 9). Following these weeds from worst detection performance to best, the "in large" weeds are detected by YOLOv3 an average 9 to 29% relative to the total number of "in large" annotated weeds – The higher the resolution, the better the performance. For the “out small” class, the 0.5 dataset resulted in the highest average (45%), while the 0.31 dataset resulted in the lowest average (31%). Thus, the trend seen before does not seem to be valid for “out small” weeds. Lastly, similar to the “in large” class, the “out large” class shows the same trend. The highest resolution has the highest percentage (85%), while the lowest resolution results in the lowest (80%).

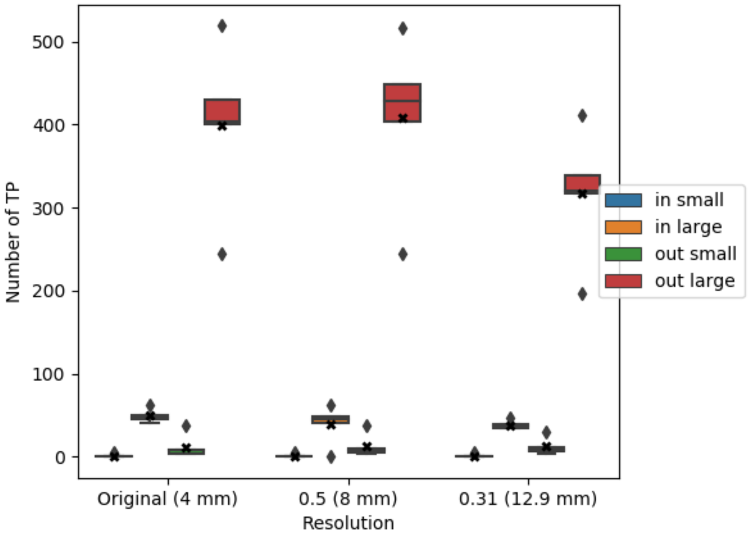


Figure 23: Number of ground-truth labels per resolution. Datapoints correspond to individual folds. Error types follow the same order per resolution as Figure 23.

Looking at the spread of ground truth labels throughout the field, shows that most weeds are large and out of the crop row (Figure 23; Table 9). Table 9 shows the absolute average number of ground truth annotations predictions per resolution (i.e. the averages in Figure 23). Figure 23 and Table 9, show that in average there are almost no weeds in the “in small” class; making it more difficult for the model to achieve high precision and recall values for this class. Although it is possible that there were less smaller weeds and weeds in the crop row, the lower values found for these classes are thought to be caused by these classes being more difficult to label.

Table 9: Average number of ground-truth labels per resolution. Smaller weeds were labelled much less frequently.

Resolution	**in small**	**in large**	**out small**	**out large**
Original (4 mm)	1	49.4	12	399.4
0.5 (8 mm)	1.2	39.6	12.8	408.2
0.31 (12.9 mm)	1.2	38.4	12.4	316.8

4.1.4Mean average precision (mAP)

Looking at the mAP, the same trend found for the F1-score can also be seen in Figure 24. The higher the resolution the higher the mAP. The highest mAP averaged over the five folds was found for the original dataset (75.04%). The lowest average mAP corresponds to the 0.31 dataset (61.98%). An average mAP of 71.6% was found for the 0.5 dataset. However, as was the case for the F1-score, these averages are brought down significantly by the lower outliers. Removing these outliers result in an average mAP of 78.53%, 76.62% and 68.80% for the original, 0.5 and 0.31 dataset respectively.

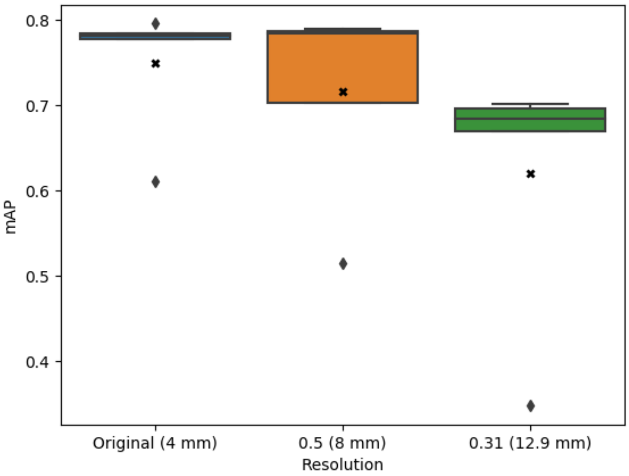


Figure 24: mAP for all tested resolutions.

The standard deviation corresponding to the original, 0.5 and 0.31 datasets are 0.07, 0.11 and 0.14 respectively. Again, following the trend seen for the F1-score, the higher the resolution the lower the standard deviation. Removing the lower outlier logically results in lower standard deviations for all resolutions. However, the 0.31 dataset does return a lower standard deviation compared to the 0.5 dataset. A summary of the standard deviations for the mAP results are depicted in Table 10.

Table 10: Performance per resolution test mAP standard deviation per resolution.

	SD	SD No Outliers
Original	0.069	0.007
0.5	0.106	0.037
0.31	0.137	0.012

4.1.5Average intersection over union (IoU)

A lower resolution results in a lower average IoU (Figure 25). The average IoU corresponding to the original, 0.5 and 0.31 datasets are 65%, 63% and 60%. The first fold does not underperform for all resolutions. For the original dataset, the lowest IoU was found for the fourth fold; the 0.5 and 0.31 datasets do get their lowest IoU from the first fold.

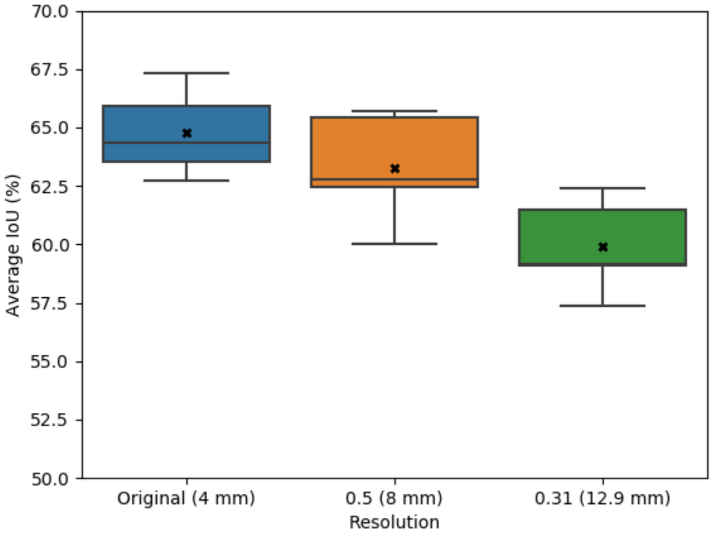


Figure 25: Average Intersection over Union (IoU) for all tested resolutions.

4.1.6 Run time

On average the processing time for the original resolution test set, consisting of 20 images which are separated into 480 tiles, is roughly 14.6 seconds. This results in approximately 0.7 seconds needed to process an image. As every image with the original resolution covers an area of 141.95 m^2 (0.0142 ha), roughly 49 seconds are needed to process a hectare.

Using the same method as for the original resolution, processing times for the other resolutions have been calculated (Table 11).

Table 11: Summary of processing time per resolution type.

Resolution	# Images	P.t. per test set*	P.t. per image	Area per image	P.t per ha
Original (4 mm)	20	14.6 sec	0.7 sec	0.0142 ha	49 sec/ha
0.5 (8 mm)	20	3.4 sec	0.2 sec	0.0142 ha	14 sec/ha
0.31 (12.9 mm)	20	1.4 sec	0.07 sec	0.0123 ha	6 sec/ha

*P.t. stands for processing time

4.1.7 Economic analysis

Looking at the FP count from an economic point of view and assuming that a FP equals to a crop that has been killed using herbicides, the larger the number of FP the worse that the system performs economically (Table 12). The estimated cost for a sugar beet crop is €0.07. A weed that is not detected by the system (FN) is estimated to affect surrounding crops; adding an estimated cost of €0.02 per weed. Costs were taken from Spruijt and Van der Voort (2015). The total area covered by the original and 0.5 test dataset is 0.2839 ha (0.014195 * 20 images in the test set). The 0.31 dataset covers an area of 0.246064 (0.0123032 * 20). For the economic analysis a relatively homogenous spread of weeds is assumed throughout the field.

Table 12: Estimated costs per hectare per resolution. Costs are calculated without taking into account operational costs.

	Original	*0.5*	*0.31*
*Average FP cost/ha (€)**	5.42	5.92	4.55
*Average FN cost/ha (€)***	7.12	7.54	8.70
*Pesticide cost/ha (€) ****	0.05	0.05	0.05
Average total cost (€)	**12.59**	**13.51**	**13.30**

* The average FP cost was calculated by taking the average number of FP of all folds and multiplying by €0.07; then dividing by the total test area. The corresponding average values are: 22, 24 and 16 for the original, 0.5 and 0.31 dataset respectively.

** The average FN cost was calculated by taking the average number of FN of all folds and multiplying by €0.02; then dividing by the total test area. The corresponding average values are: 101, 107 and 101 for the original, 0.5 and 0.31 dataset respectively.

*** Following the values in Ruigrok et al. (2018), an estimated cost of €0.00033 herbicide per weed was calculated. Multiplying this value by the number of FP and then dividing by the total test area, gives an estimation of pesticide cost per hectare.

4.2 Viewing angle test

After testing the model to assess the effect of viewing angle on performance, the following results were gathered.

4.2.1 F1-score, precision and recall

As seen in Figure 26-A, the closer the angle ranges are to nadir, the better the model generally performs – As defined by the F1-score.

The correlation coefficient (R) correspondent to the different resolutions are: 0.32, 0.25 and 0.12 for the original, 0.5 and 0.31 datasets respectively (Figure 26-A). R tells how strong the linear relationship is. These R are deemed relatively low, indicating that a relationship between angle and performance exists, but is not strong. The p-values correspondent to the original, 0.5 and 0.31 datasets are 0.01, 0.04 and 0.33 respectively. Assuming a commonly used α-value of 0.05, shows that the effect of angle on F1-score is statistically significant for the original and 0.5 dataset as p-value < 0.05. The relationship and statistical significance is stronger as resolution increases. Furthermore, an assumption of normality is discouraged as the lines seen in Figure 26-B are not the bisector of the plot.

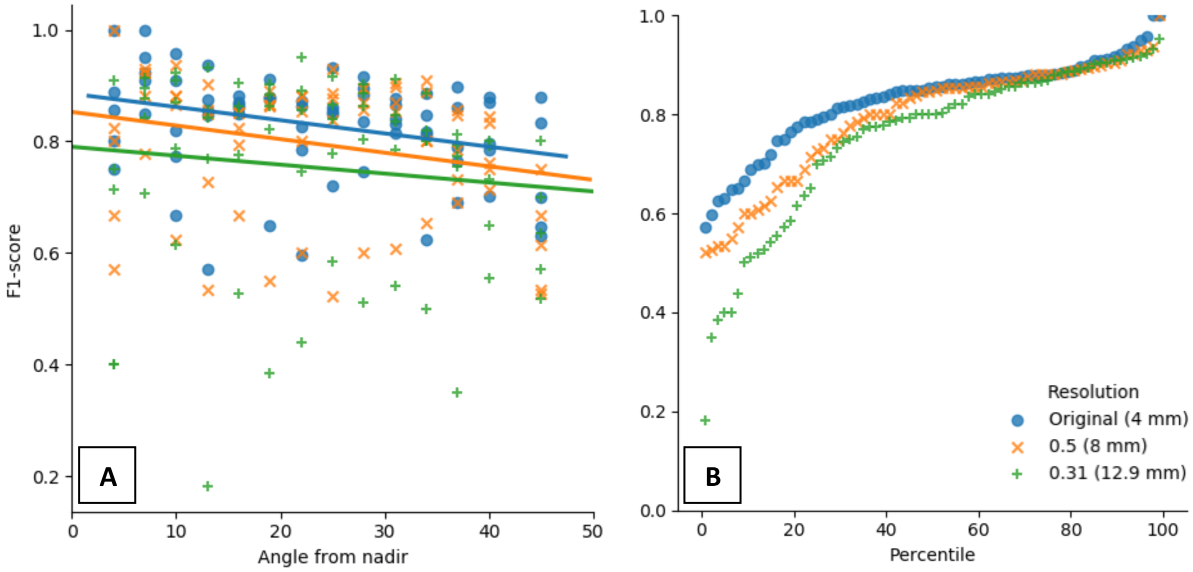


Figure 26: (A) Relationship between F1-score and viewing angle from nadir, and (B) normal probability plot for all three resolutions.

4.3 Orthomosaic test

The orthomosaic test was done as an additional test, to bring the results of this research closer to the remote sensing domain. Orthomosaics are derived from overlapping aerial photo's using the Structure-from-Motion methodology (Westoby et al., 2012).

4.3.1F1-score, precision and recall

A clear distinction in performance was found between the original and orthomosaic tiles (Figure 27). Orthomosaic tiles perform significantly worse compared to the original images for the tested dataset. The average F1-scores correspondent to the original and orthomosaic images are 0.90 and 0.65 respectively. The standard deviation found for these data are 0.12 and 0.20 for the original and orthomosaic data respectively; indicating a larger deviation in predictions for the orthomosaic images.

Furthermore, looking at precision and recall, the orthomosaic images perform worse for both cases (Figure 28). The average precision and recall values found for the original tiles are 0.96 and 0.86 respectively; for the orthomosaic tiles, these values are 0.76 and 0.61 respectively.

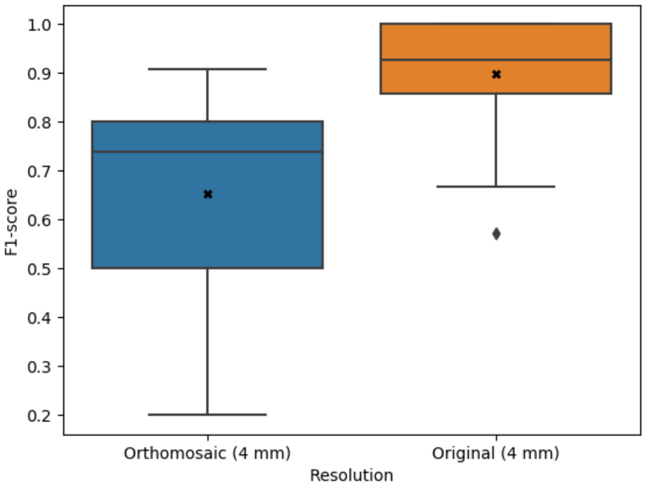


Figure 27: F1-score for the orthomosaic and original tiles. Orthorectification has a clear negative effect on the F1-score. Datapoints correspond to tiles.

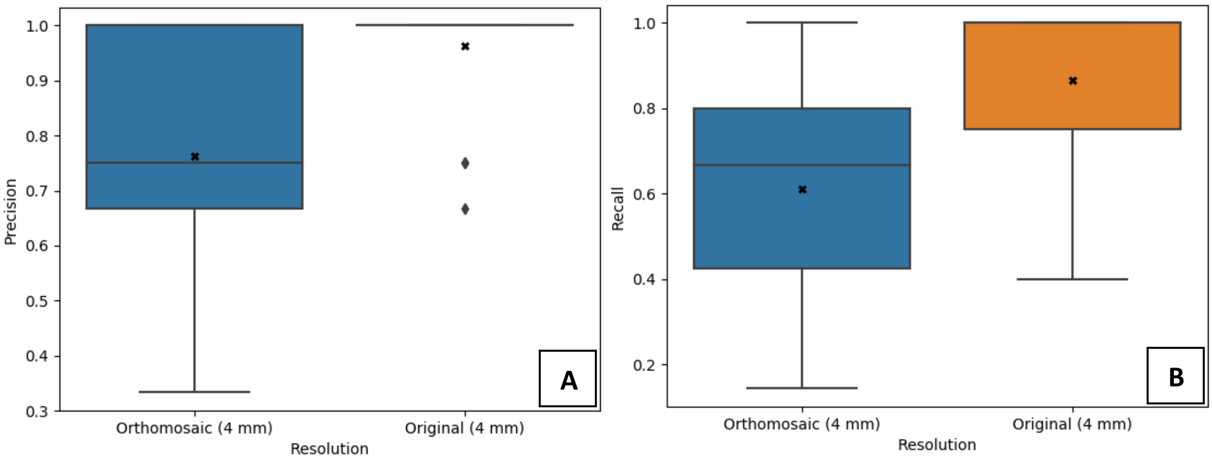


Figure 28: Precision (A) and recall (B) for the orthomosaic and original tiles.

5 DISCUSSION

The discussion is divided in two sections, starting with general points which put the results in context in *5.1 Datasets and methodology* and narrowing towards the results and the practical application of the weed-detection system in *5.2 Discussion of test results*.

5.1 Datasets and methodology

Starting with the source of all results, the datasets and the methodology, a variety of points can be discussed.

5.1.1 Datasets

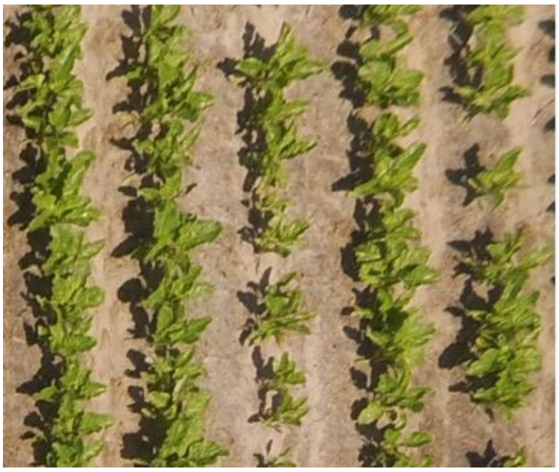


Figure 29: Effect of a distortion in the lens on the top right corner of the image.

Firstly, regarding the quality of the images, the overall resolution of the original dataset could be better by using better sensors. A higher resolution would result in more detail, but less area per tile as YOLOv3 has a limit of 608x608 px per image. Extrapolating the findings gained in this research, result in an expectation that higher resolutions would result in more accurate predictions. This expectation is also supported by other research as shown in section *5.2.1 Performance per resolution test* (Cao et al., 2016; Haris et al., 2018; Pérez-Ortiz et al., 2015; Redmon et al., 2015; Tayara and Chong, 2018). Such sensors exist, but were not available during data gathering. However, the resolution used in this research yielded results deemed sufficiently high for the scenarios. Furthermore, the images gathered were automatically compressed to a JPG file format which contain artefacts and are overall of lower quality, but easier to share. Images in RAW file format do not have these issues. Another issue with image quality has to do with blurry corners (Figure 29). This issue is thought to be caused by a distortion in the lens, but no sources were found which verify this hypothesis. Although the affected area is relatively small (approximately 1 m^2 per image), it does affect the quality of the images at the top and bottom right corners. The only images that contain weeds that are blurred by this effect are those of the first flightpath, making this a minor problem.

Secondly, the effect of cropping the images for the 0.31 dataset, results in this dataset containing less training data. Cropping images to dimensions processable by a deep-learning model is common practice as smaller images reduce the processing strain on the GPU (Pérez-Ortiz et al., 2015). However, no research found claimed to crop images to maintain detail, which was the case for this research. The original and 0.5 datasets have approximately 2800 ± 100 training objects, while the 0.31 dataset has 2200 ± 100. In general, it is recommended to use as much training data of high quality as are available, which in the case of the 0.31 dataset is limited due to the method used. Comparing the original and 0.5 dataset with the 0.31 dataset, an area of 18.92 m^2 is lost due to cropping; this is approximately 13% of area lost per image. Increasing the amount of training data for the 0.31 dataset results might yield better results and should be looked into. Improved hardware, which can process tiles with dimensions larger than 608x608 px could also help with this issue. Due to an unavailability of more data, this was not possible.

Thirdly, the data used only contained weeds in a certain area which was prepared for the experiment. This area is easily identifiable as three tractor tire tracks run parallel of the length of the study area; one in the middle and two to the sides of the study area (Figure 30). These tire tracks can mislead the

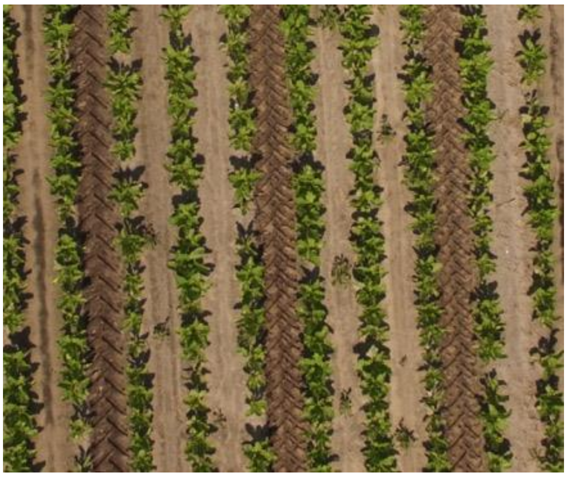


Figure 30: Three tractor tire tracks delineate the study area. These tracks can influence the model's results.

deep-learning algorithm into relating weed location to tire track location. Thus, making the system less robust for images where the weeds are not found between tire tracks. A dataset without tire tracks would need to be tested to investigate the effect of this environmental factor on object detection. However, Milioto et al. (2017a) which also attempt to detect weeds in a sugar beet field do not have data with such clear markings which might unfairly aid the model, but still achieve a better performance compared to this research. Namely, a precision and recall of 97% and 98% respectively. This leads to the hypothesis that the tire marks do not affect the result much. One must keep in mind that Milioto et al. (2017a) do use a different deep-learning model. For a definitive answer on this issue, YOLOv3 should be tested on the same dataset as Milioto et al. (2017a). Thus, it would be interesting to compare model performance on another dataset. Also, using data from another dataset to further train the model is expected to yield better results. Images of weeds and crops in different growth stages would make the model more robust and useful in practice. However, as stated above, external data similar to the data used were not readily available to use.

Lastly, the exact number of weeds planted and their location is unknown. Consequently, it is not known how many annotated weeds there should be, and where they should be. This results in a skewedness regarding the labelling of weeds.

5.1.2 Weed labelling method

Labelling weeds is foremostly a lot of work and very dependent on the labeller's expertise (Hoermann et al., 2018). As labelling is a visual task, it is naturally skewed towards a more accurate labelling of weeds that are easier to see. Weeds that are veiled by other objects are more prone to being missed by the labeller. This effect is thought to be the main cause of the difference in "easy" and "difficult" weeds in Table 9. In many cases there is uncertainty when choosing weeds. Weeds can be hidden beneath crops, very small, merged with other weeds, or it can simply be unclear from the image if an object is a weed. To reduce uncertainty, it is recommended to look at environmental indicators such as shadows and crop placement. Figure 31 shows the difference in shadows; volunteer potatoes have smaller leaves which result in smaller, more wispy, shadows (Figure 31-A, i). On the other hand, sugar

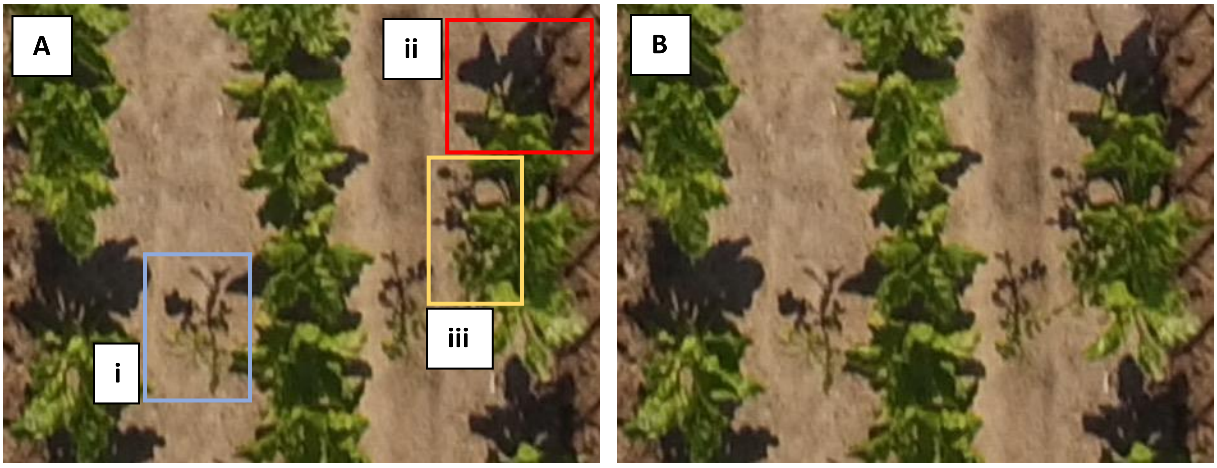


Figure 31: Examples of difference in leaf shadows between sugar beet and weed. (A) image with labels indicating the different types of shadows; (B) the clean image for easy viewing.

beet leaves have larger leaves which cast larger, more rounded, shadows (Figure 31-A, ii). Figure 31-A, iii, is an example of a weed and its shadow found by taking into account shadows. Figure 31-B is displayed for visual clarity for the reader. On the other hand, the predictability of crop placement, in this case every 0.2 m, helps in removing uncertainty when objects are clearly visible in the row.

Once a weed has been located, the following question arises: “How should one label the weed?”. As weeds can grow in an infinite variety of shapes, it is not always clear how to annotate the bounding box. The following issues were met when labelling:

1. Weeds growing in between crops are not visible at all or only visible in part. In the first case, these weeds are logically not labelled as ground truth locations are not available. In the second case, the visible part of the weed is labelled. In some cases, this can be a single lateral stem or grouping of leaves.
2. A single potato plant can have several stems which can appear to be different plants growing close to each other. As it is difficult to identify individual organisms from image data, a grouping of potato plants is labelled if they seem to be part of the same plant. This is thought to be in line with the real world application of such a system; spraying the object recognised as a weed is important, not identifying individual plants when they are grouped together.
3. In some cases, a lateral stem of a weed might grow strongly. Here, the largest part of the weed would be labelled as it is most important for the application. Extending the bounding box so the large lateral stem would also be included, adds unnecessary data to the bounding box. This unnecessary data includes a large portion of the bounding box area being background soil and whatever may lie therein.

Another aspect to be taken into account is the expertise of the labeller. For this research, the labeller had a basic knowledge of the targeted weed. Thus, knowledge was gained while labelling. This results in the first images being labelled differently compared to the later images. To counteract this gradient and to labelled weeds that were possibly missed, all images were revised a second time.

As labelling takes a lot of time and is dependent on the labeller, it’s one of the major downsides in supervised machine learning (Hoermann et al., 2018). Piewak et al. (2019), present a method to automatically label *Light Detection And Ranging* (LiDAR) objects using a combination of LiDAR and RGB images. One or more RGB cameras are used to identify objects using the Cityscapes dataset. These identified objects are then spatially correlated to the LiDAR image and then annotated. Cityscapes is a large-scale dataset which provides labelled images of common objects in cities (Cordts et al., 2016). Although this method proves to greatly reduce labelling times, it still needs a dataset with manually labelled objects. Such datasets exist (e.g. COCO, PASCAL and Cityscapes) which makes the method proposed by Piewak et al. (2019) possible. However, in the absence of such large-scale datasets (e.g. weeds) this method becomes less attractive.

Pordel and Hellström (2015), propose a method to semi-automatically label images using depth information. This method labels objects by performing image segmentation based on object boundary edges. As identifying objects and object boundaries has been proven a difficult task using only RGB (Bayr and Puschmann, 2019; Chevrefils et al., 2009; Du et al., 2006; Pordel and Hellström, 2015), a Microsoft Kinect 3D scanner is used in conjunction with RGB images to delineate object borders. The objects are then segmented using Fuzzy C-Means, to be labelled by a human. This was tested on object types important for forestry robots and was proven to minimise labelling time. However, human interaction with the system is still necessary; although the tool reduces the number of clicks needed.

Methods as those described above could greatly help in the creation of training data for weed-detection systems. Deep-learning can thus be used to create systems which could help in labelling. E.g. a system which starts with a relatively small, manually labelled, dataset and uses this data to predict other weeds. The user would then have to manually check if the model correctly predicted an object; correctly predicted objects would then be added to the training dataset. This would reduce labelling workload as the labeller would only need to accept or decline a proposed image. Furthermore, including depth in conjunction with RGB images as proposed by Hoermann et al. (2018) and (Pordel and Hellström (2015) can minimise labelling time and partially automate it for similar data to the one used for this research. As conventional agricultural fields are relatively predictable (i.e. crops are in a row, crops are generally significantly higher than the soil) using depth information is deemed suitable for labelling. Additionally, LiDAR data can also be gathered from a UAV (RIEGL, 2019), making the combination of RGB and 3D data easier. For the creation of large-scale agricultural datasets, more research should be done regarding the abovementioned combination of 3D and RGB data.

A relatively simple method to label weeds reliably and without depending on visually labelling images is described by Hung et al. (2014). Here, two sets of images are taken; one for labelling and one for evaluating. For labelling, weed experts are sent to the field to mark weeds with red tape. The easily identifiable tape is used to label the location of the weeds easily. These locations are then used on the second set of images, which does not contain red tape, which results in reliable labelling. The problem with this method is that it requires on-field manual labour.

5.1.3Image resizing method

No images taken from heights different than 10 m were available. Thus, the different heights were simulated digitally. Although digital image decimation can result in the moiré effect and aliasing, the chosen algorithm, resampling using pixel area relation, gives moiré-free results and avoids aliasing (Tanbakuchi, 2018). These characteristics make this algorithm the preferred one, as moiré effect and aliasing are image distortions which are not preferable.

Further research should look into weed detection using images that were actually taken at different heights. Such images would not have the handicap of data loss as is the case with the cropping of the 0.31 dataset. Noticeable differences are not expected in theory as the relation between flying distance and pixel area is theoretically sound for this research. This expectation is also confirmed by Pérez-Ortiz et al. (2015). However, Pérez-Ortiz et al. (2015) aim at detecting crops on orthomosaic images.

5.1.4Tile creation method

The creation of tiles was needed to be able to feed data to the YOLOv3 model as it can only process images with dimensions which are a multiple of 32, and the hardware used for processing could not handle tiles larger than 608x608 px. Consequently, cutting the original images into tiles does result in a loss of data. Primarily, due to cropping the images so they are divisible by 608; which completely removes parts of the original images, resulting in less objects to use for training. This effect of this issue was reduced by cropping sides of the images which contained as little weeds as possible. Secondarily, due to weeds that happen to be between two tiles. Such weeds are separated by the tiles, which results in part of the weed appearing in one tile while the other part appears in another tile. The bounding box is not transferred to both tiles; only to the tile which contains the centre of the bounding box. A further improvement of the method used would be to also give the other part of the weed a bounding box, so the model can train with more weeds that are at the edges of the tiles. The effect of this issue is deemed relatively small as the grand majority of weeds are not found between tiles.

Generally, a higher input resolution for training is related to a better YOLOv3 performance (Redmon and Farhadi, 2018). However, Voorhoeve (2018) claims that between resolutions of 320, 416 and 608 px, the 416x416 px resulted in the highest mAP. The difference in mAP between 608 and 416 px however, is only 1.01%. This difference was not deemed sufficient and the statistical relevance of these findings are lacking. Therefore, the instruction of Redmon and Farhadi (2018) was chosen and the highest possible input resolution was chosen. With the GPU processing power as limiting factor, a resolution of 608x608 px was chosen. Preliminary tests were done for higher resolutions, but the available GPU was not sufficiently powerful to process even relatively small batches of images.

Most literature does not clearly discuss the method used to adapt image dimensions so it is possible for the object-detection model to use the data. Lottes et al. (2018), which use a tailor made convolutional neural network, simply state that "*we train all networks from scratch using down sampled images with resolution Width = 512 and Height = 384*" (p. 6), with no further explanation. Using the same model, Milioto et al. (2017b) also does not explain how the resizing is done.

5.1.5 YOLOv3

This research did not focus on possible improvements of the object-detection model; instead, standard settings were used to train and test all datasets. Future research can look at changing hyperparameters which better suit the task of detecting weeds. A common method to optimise model hyperparameters is by calculating the *Receiver Operating Characteristic* (ROC) curve and selecting the hyperparameters which result in the largest area under the curve (Unterthiner et al., 2015).

Hyperparameter values could also be altitude dependent: as flying higher results in relatively smaller objects to be detected, it would be interesting to select hyperparameters which are more appropriate for the detection of smaller objects. For example, Alexey (2019) has compiled a list of possible hyperparameter adaptations which claim improve training for small objects, training for both small and large objects, and overall precision. Thus, in practice, a system which changes model hyperparameters depending on flying altitude might be preferable to a single model. This was not experimented with in this research as it would make comparison between resolutions unfair. However, further exploring these changes might yield interesting results.

Looking into other object-detection models can be of interest to better understand which elements of a model are beneficial for the proposed task. YOLOv3 was chosen as it was state-of-the-art during the time of this research and as it was already in use for other research at Wageningen University. Such research has demonstrated that YOLOv3 delivers promising results when detecting weeds from a UGV, making weed control from a UAV and/or UGV scenario possible in the near future (Voorhoeve, 2018).

Better, faster and more adequate models compared to YOLOv3 are expected to come out in the future as this has been the historical trend. Models which are more specific to the detection of weeds already exist and better performances compared to those found in this research have been claimed (Guerrero et al., 2012; Lottes et al., 2017; Milioto et al., 2017a; Pérez-Ortiz et al., 2016). However, the development of such newer models does not eliminate the importance of this research. In principle, the relationship between image resolution and model performance found, is expected to be relevant for future models as well.

5.1.6 Performance test method

Although five folds per resolution were tested in the *performance per resolution* and *viewing angle tests* to get a better understanding of the spread of performance values, more folds would be preferable. As can be seen in the results, fold 1 underperformed compared to the other folds in most cases (Figure 20). It is not known if fold 1 was a general outlier which does not occur often in reality, or if testing more folds would result in values between fold 1 and the rest. Five folds were chosen due to time constraints.

The *orthomosaic test* was done with a relatively small number of tiles which makes it less statistically sound compared to the other performance tests. This is due to the process of identifying and clipping tiles from the orthomosaic image on QGIS being manual and taking a relatively long time to do well. Similar to cropping of normal images, as discussed in section *5.1.4 Tile creation method*, the cropping method of orthomosaics into tiles is not clearly discussed in literature. Pérez-Ortiz et al. (2015) simply state that “*the mosaicked image is partitioned into multiple subimages of approximately 1000 × 1000 pixels*” (p. 536). Längkvist et al. (2016), state that “*the map is divided into regions of 1000-by-1000 pixels*”. Both not explaining if the method is manual or automated. However, this test was not meant to be the focus of this research. Instead, it brings another facet of the possible practical uses of such a system and gives insight into future research possibilities.

The choice of metric, is in part dependent on (1) the researcher’s personal choice and (2) the case being researched. This research focussed on intuitiveness and practical use of the hypothetical model. Thus, the F1-score, precision, recall and IoU metrics are the focus of this research to assess performance. Although most object-detection literature focusses on mAP as a metric for performance as shown in section *5.2.1 Performance per resolution test*, this metric was also included but is not seen as the most important. Instead, the future real world application of the system is taken into account when discussing the results. I.e. an object-detector that should be able to confidently detect weeds and not mistake crops for weeds, would need to score high in precision. However, if the goal is to find all possible weeds, without it mattering too much if a crop is mistaken for a weed, then a high recall might be more important. IoU becomes important for automatic spraying, as a high IoU is related to a higher spraying precision. mAP will mostly be used to compare the overall performance of the system with values found in literature.

The IoU metric brings some challenges which make the IoU more difficult to interpret. For example, for a weed that grows between two crops, the ground truth box might only surround the leaves corresponding to the weed while the predicted box might cover a larger area (Figure 32-A). Although in this case the IoU is relatively small, it could be high enough to be counted as a TP. If this is the case, the predicted bounding box (red) is too large and in practice the UAV could spray herbicide on the crop. A second challenge arises as individual potatoes can sprout different stems which look as though there are several plants. The model can classify both plants as one (Figure 32-B). The inverse is also possible. Thus, labelling two plants as one and having the model predict two separate plants. In practice, this would lead to the system spraying both weeds correctly. A third challenge can arise when the model predicts two overlapping weeds while only one weed was labelled (Figure 32-C). In such a case, the IoU of individual predictions is low while the combination of both predictions is rather high.

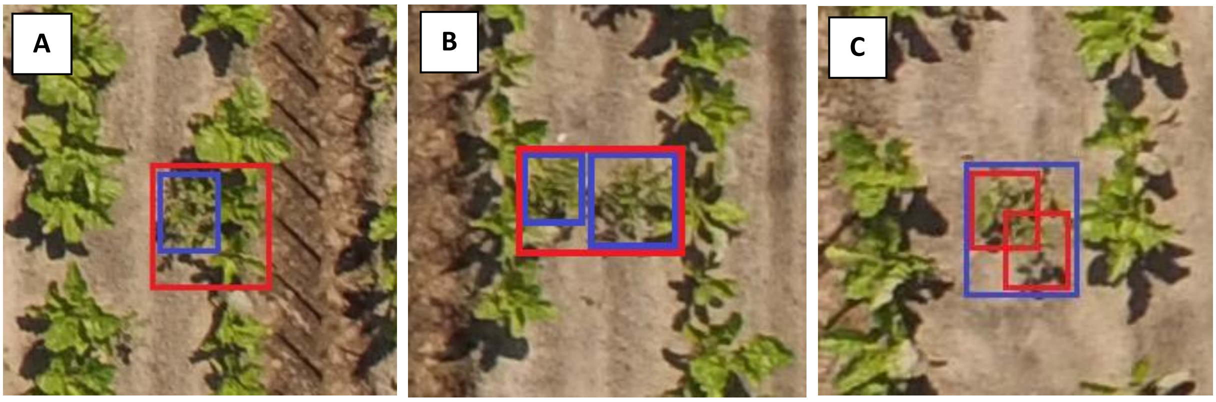


Figure 32: Challenges in IoU interpretations. Blue boxes represent labelled ground truth objects, while red boxes represent objects predicted by the model. Image A depicts the model demarking a larger area than that which was labelled. Image B illustrates the model predicting one weed instead of the two that were labelled. Image C depicts the opposite of B, wherein the model predicts more weeds than what was labelled.

Focussing on the method used for the viewing angle tests, shows that it could be improved. As the bins are defined by viewing angle and the location of a bounding box is defined by its centre point, in some cases the ground truth and predicted bounding boxes for a weed are put into different bins. If for the *performance per resolution test* this weed would be counted as a TP (i.e. IoU of both bounding boxes is equal or more than the set threshold), for the *viewing angle test* this weed would be counted as both a FN and a FP. This problem is illustrated in Figure 33, wherein the predicted bounding box falls in one bin which includes the border between bins, and the ground truth bounding box falls in another.

For the original dataset as an example, this issue accounted for 30 TP less, and 30 FN and FP more, when comparing the total number of TP, FN and FP of all folds. This accounts to 1.7% more TP, 5.9% and 27.8% more FN and FP respectively. The percentage difference is much larger for the FP as less FP were predicted for all datasets. However, this issue does not change the trend found in the results. Although the performance described for the *viewing angle test* in the results is lower than was shown for the *performance per resolution test*, the difference is relatively small when looking at the average of all folds for a certain resolution (Table 13). The difference does become larger for lower resolutions.

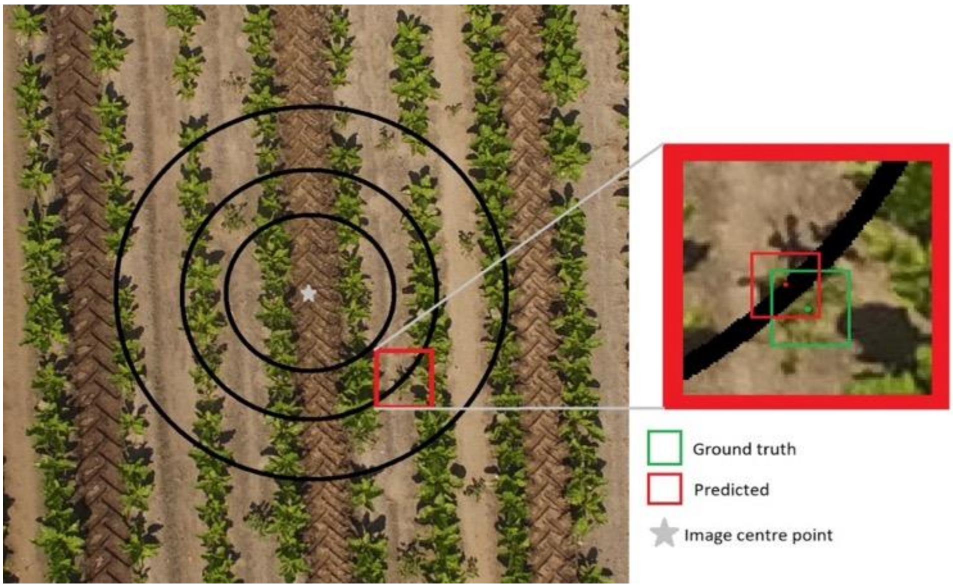


Figure 33: Illustration of the bin problem: a weed which is correctly identified, is instead counted as a FP (red bounding box) and a FN (green).

Table 13: Precision, recall and F1-score difference between the performance per resolution test and the viewing angle test for all resolutions.

	Precision	**Recall**	**F1-score**
Original (4 mm)			
Performance per resolution test	0.95	0.77	0.83
Viewing angle test	0.93	0.76	0.83
0.5 (8 mm)			
Performance per resolution test	0.94	0.75	0.82
Viewing angle test	0.90	0.73	0.80
0.31 (12.9 mm)			
Performance per resolution test	0.95	0.71	0.78
Viewing angle test	0.89	0.67	0.75

Possible solutions to this problem which were not explored in this research due to time constraints are: (1) Pairing bounding boxes which form a TP with an index and assigning the TP to one of the two relevant bins. This would add the possibility of creating more bins without amplifying the problem. (2) Removing TPs which are at the edge of a bin. This would possibly delete valuable data and should therefore only be done if there are many data. No research was found which addressed a method similar to the one used in this research.

5.2 Discussion of test results

The general trend found is that a higher resolution results in better performance; as defined by the precision, recall, F1-score, mAP, average IoU, run-time and the economic analysis (Figure 19, 21, 24, 25; Table 4, 11, 12). This trend was found for both the *performance per resolution* and *viewing angle* tests, and although not tested in this research, it's also expected for the orthomosaic images – based on Pérez-Ortiz et al. (2015). This chapter will discuss the results in depth, taking into account the practical use of the proposed system. One must keep in mind that other factors not studied in this research like discrepancies between weed growth stage in the training and test set, ambient lighting and model hyperparameters, could influence the result.

5.2.1 Performance per resolution test

The ANOVA applied on the *performance per resolution test* data shows that there is a significant interaction between resolution, fold and mean F1-score (Table 7, 8). It has been proven with clear significance that a higher resolution positively affects the mean F1-score. However dependent on the area being tested, the resolution can also have a negative effect on the mean F1-score. I.e. it's preferable to collect images at the original resolution in most cases, but its importance varies according to the fold considered. ANOVA was deemed a fair method to use as it's a general and well-known procedure to isolate sources of variability in a dataset (Girden, 1992).

It is thus not always the case that a higher resolution results in a higher F1-score (Figure 19, 20). In practice this means that dependent on field properties, it might be beneficial for a farmer to fly with a lower than possible resolution. For example, Figure 20 demonstrates that it's preferable to fly at a higher altitude for the area in Fold 4. Further research is necessary to better understand what drives the model to perform better at lower resolutions in specific cases. Characteristics of the field and discrepancies between training and testing data could be interesting to investigate to clarify this matter. In other cases, the difference between two resolutions is relatively small. Pérez-Ortiz et al. (2015), make the point that a higher altitude results in less images needed to cover the area. Consequently, flight time is shortened, mosaicking is less processing intensive due to the reduced number of images, and UAV battery limitations are reduced. It then becomes the user's choice which altitude to choose.

For all resolutions, fold 1 resulted in an exceptional underperformance compared to the other folds (Figure 20). This is thought to be due to the physical nature of the field that is contained in fold 1. Fold 1 has a relatively large number of weeds which are difficult to detect due to (1) them being stunted in growth due to past waterlogging and (2) the background soil being different, due to waterlogging, compared to the soil in the training data (Figure 34).

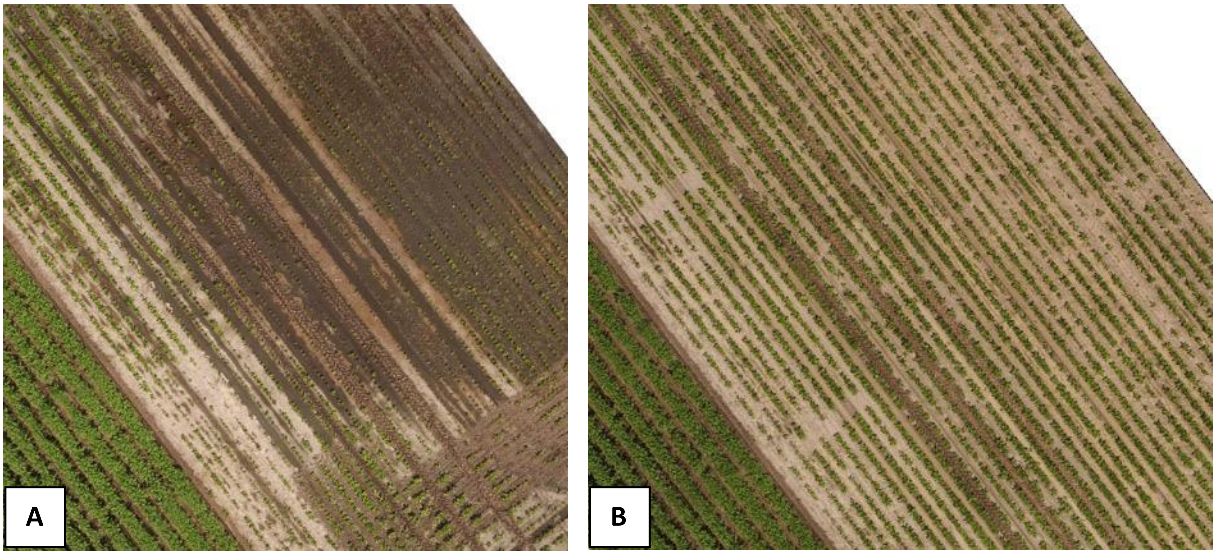


Figure 34: Difference in a part of the study area that was waterlogged and in the test set of Fold 1 (A), a more conventional part of the field which was in the training set of Fold 1 (B).

The smaller variance between the other folds (Figure 19) and between images (Figure 20) can be explained due to (1) the heterogeneity of the field (i.e. not all images have the same number of easy/hard to detect weeds), (2) cropping or tile creation results in weeds being partially cropped and thus making them harder to detect correctly (Figure 35), and (3) differences between test and training datasets which aren't as extreme as the difference found for fold 1.

Logically, removing the outliers caused by fold 1 decreases the standard deviation for all resolutions (Table 5). A lower standard deviation means a system which is more predictable in practice. However, situations such as the outliers caused by fold 1 are important as they can occur in practice. This was still tested to evaluate the effect on the deviation. As the system stands now, it might not be predictable enough for a user. E.g.: For the original resolution, a user using this system could not be guaranteed a performance with an F1-score between 0.85 and 0.90. Instead, if used for an area which is difficult for the system, the user can expect an F1-score of approximately 0.72 (Figure 19). Thus, in a

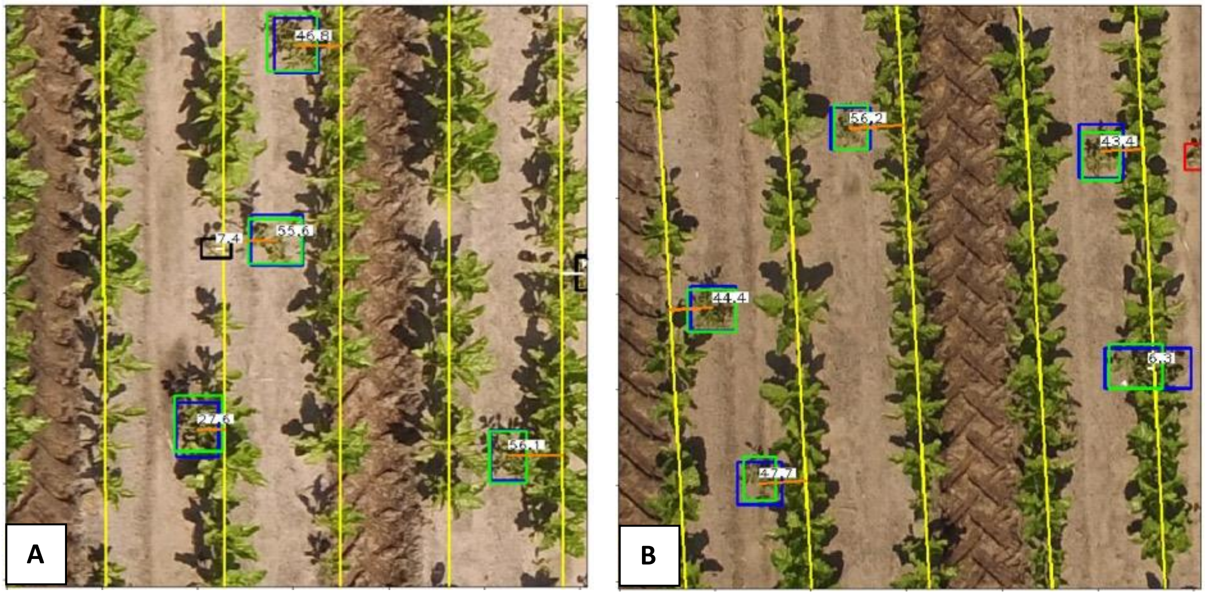


Figure 35: Examples of weeds being split among two tiles, making them harder to detect and resulting in a FN (A: black box, middle right) and a FP (B: red box, upper right); similar to the phenomenon seen in Figure 33. Green boxes correspond to TP predictions, blue boxes are correctly identified ground truth labels, white lines show that the weed is in the crop row, and orange lines show that the weed is outside of the crop row.

practical sense, a system which has a low standard deviation is preferable. Decreasing the standard deviation can be done by training the system better, this would make the system more robust which leads to a performance which is more predictable.

Looking at precision and recall (Figure 21) a peculiarity arises: the 0.5 dataset has the lowest average precision. Intuitively, a higher resolution is paired with a better performance as it increases the number of distinguishable features in an object (Haris et al., 2018). This intuition has also been proven to be correct in different cases (Cao et al., 2016; Haris et al., 2018; Hung et al., 2014; Pérez-Ortiz et al., 2015; Redmon et al., 2015; Tayara and Chong, 2018). Consequently, a lower flying altitude is also related to a better performance as is shown in Pérez-Ortiz et al. (2015). In this research, this was not always the case (Figure 21). Although no definitive proof was found to explain this phenomenon, it is thought to be caused by the 0.5 dataset having a high enough resolution to confuse the system into predicting FP, while the resolution for the 0.31 dataset is too low to even mistake erroneous objects with weeds. The 0.31 dataset generally has a lower recall, which indicates that it predicts less weeds. This could be indicative of a system which in general does less predictions (TP, FP or FN) and thus would also have a lower tendency to predict things wrongly. This could explain why the 0.31 dataset has a higher average precision than the 0.5 dataset. However, more in depth research should be done to better understand the effect of resolution on a per-object basis.

Figure 22 shows a general trend for the different kinds of FN. The results are relatively intuitive: weeds that are outside of the row are easier to detect than those in the row, and larger weeds are easier to detect than smaller weeds. Detecting weeds within the crop row is a known challenge for precision agriculture (Pérez-Ortiz et al., 2015). Moreover, one must keep in mind a possible bias which starts during weed labelling. As labelling must be done manually on the images and no ground-truth weed coordinates were available, a natural bias towards labelling weeds which are relatively easy to detect on an image arises. This results in less training data for in-row weeds as these are more difficult to spot (Figure 23). Although precautions were taken to reduce the effect of this bias by having two rounds of labelling all images, the bias is still thought to be prevalent. A possible solution to this problem, is having all weed coordinates to have a better idea of where to label. Detection of smaller weeds could be improved by gathering data in earlier growth stages. In the case of volunteer potatoes this is relatively easy as a potato field could be used to collect training data. A classification of FP would also be interesting to research further. This was not done due to time constraints.

Table 14: Mean average precision (mAP) values for different literature sources.

Model name	**mAP (%)**	**Dataset**	**Bands**	**Source**
YOLOv3	75.0	Weeds	RGB	Own research
R-CNN	53.3	PASCAL VOC	RGB	(Girshick et al., 2014)
Fast R-CNN	68.4	PASCAL VOC	RGB	(Girshick, 2015)
Faster R-CNN	75.9	PASCAL VOC	RGB	(Ren et al., 2015)
R-FCN	82.0	PASCAL VOC	RGB	(Dai et al., 2016)
YOLO	57.9	PASCAL VOC	RGB	(Redmon et al., 2015)
YOLOv3	57.9	COCO IoU 0.5	RGB	(Redmon and Farhadi, 2018)
SSD	82.2	PASCAL VOC	RGB	(Liu et al., 2016)
SSD	50.4	COCO IoU 0.5	RGB	(Liu et al., 2016)

The mAP follows an intuitive trend: the higher the resolution, the higher the mAP (Figure 24). Comparing the average mAP values found in this research for the highest resolution to other literature, shows that this research's model fairs relatively well (Table 14). It performs better than the YOLOv3 and SSD models for the COCO IoU 0.5 dataset but underperforms compared to R-FCN and SSD models trained and tested using the PASCAL VOC dataset. One must keep in mind that the PASCAL VOC and COCO datasets, described in more detail in the *2.1 Background* section, are not very comparable to the data used in this research. These commonly used datasets include a wide variety of types of common objects in many different contexts (Everingham et al., 2010; Lin et al., 2014). Although the datasets lack comparability with the sugar beet field dataset used for this research, they have been compared to give the reader a general idea of performance in a way that is often seen in object-detection model literature. Literature does not always make clear how the achieved mAP was calculated. In some papers, it is the highest mAP found for all the datasets while in other papers it is not specified. For the latter, it is assumed that the highest found mAP was chosen.

Comparing this research's results with comparable literature shows that this research performs relatively well (Table 15). However, it does underperform compared to other weed-detection literature. To make the comparison more relevant, it is important to understand the subject of each paper. Milioto et al. (2017a) uses a self-developed CNN and both RGB and NIR images to detect weeds in a sugar beet field. For this research, Bonn University was contacted to request the data used by Milioto et al. (2017a) for further comparative purposes. However, it was not possible to get the annotated data. Images were freely available, but these could not be annotated due to time constraints. Hung et al. (2014), aimed at weed detection on three different types of invasive weeds (water hyacinth, tropical soda apple and serrated tussock) at different altitudes (5, 10, 20 and 30 m). The value noted in Table 15 is the highest performance found in Hung et al. (2014), values for other weeds and resolutions are illustrated in Table 16. The study areas in Hung et al. (2014) are not very comparable to a sugar beet field and is deemed a more challenging problem due to it being a natural area with a heterogenous background, no clear lines and tractor tire tracks to help the model contextualise the location of weeds, which makes the result achieved more impressive. Sa et al. (2018), based their weed-detection model on the SegNet deep convolutional segmentation framework developed by Badrinarayanan et al. (2017). The dataset is highly comparable as it consists of images of a sugar beet field with weeds. The addition of NIR images in Milioto et al. (2017a) and Sa et al. (2018) is expected to yield better results as it gives an object-detection model more elements to differentiate between plants.

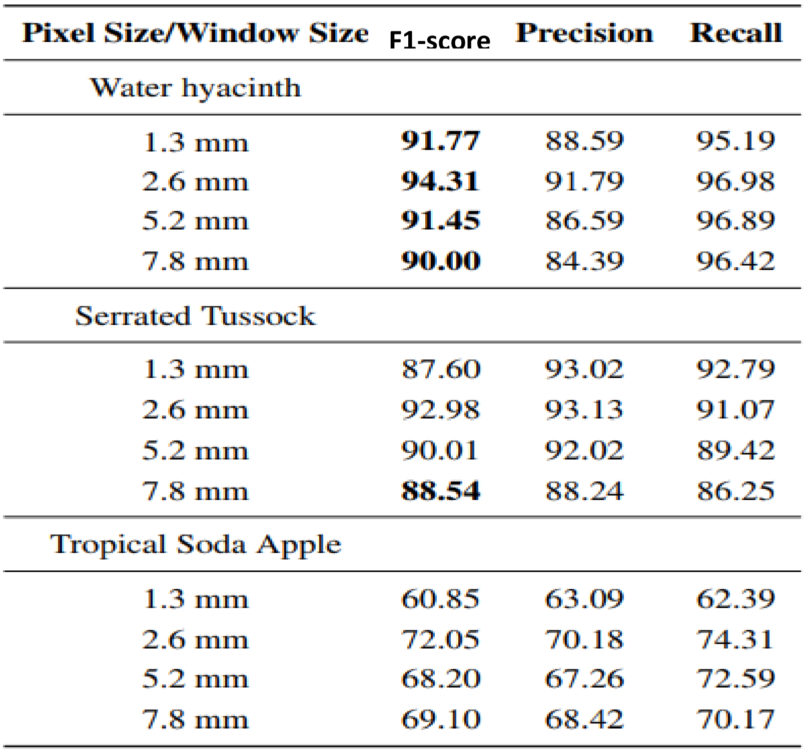
Table 15: Performance values for comparable weed-detection literature sources.

Model	**F1-score**	**Precision**	**Recall**	**Object**	**Resolution**	**Bands**	**Source**
YOLOv3	0.83	0.95	0.77	Weeds	4 mm	RGB	Own research
YOLOv3	0.82	0.94	0.75	Weeds	8 mm	RGB	Own research
YOLOv3	0.78	0.95	0.71	Weeds	13 mm	RGB	Own research
CNN*	-	0.97	0.98	Weeds	4 mm	RGB + NIR	(Milioto et al., 2017a)
Feature Learning*	0.94	0.92	0.97	Weeds	2.6 mm	RGB	(Hung et al., 2014)
SegNet*	0.84	-	-	Weeds	n.a.**	RGB + NIR	(Sa et al., 2018)
SegNet*	0.95	-	-	Crops	n.a.**	RGB + NIR	(Sa et al., 2018)

For more information regarding the methods and data used, see the respective papers.

* Basic object-detection model was modified to better suit the task.
** The source specifies a flying altitude of 2 m, but exact resolution was not specified.

Table 16: F1-score, precision and recall for different weeds and resolutions. Modified from Hung et al. (2014).



Pixel Size/Window Size	F1-score	Precision	Recall
Water hyacinth			
1.3 mm	**91.77**	88.59	95.19
2.6 mm	**94.31**	91.79	96.98
5.2 mm	**91.45**	86.59	96.89
7.8 mm	**90.00**	84.39	96.42
Serrated Tussock			
1.3 mm	87.60	93.02	92.79
2.6 mm	92.98	93.13	91.07
5.2 mm	90.01	92.02	89.42
7.8 mm	**88.54**	88.24	86.25
Tropical Soda Apple			
1.3 mm	60.85	63.09	62.39
2.6 mm	72.05	70.18	74.31
5.2 mm	68.20	67.26	72.59
7.8 mm	69.10	68.42	70.17

Although no uses of the YOLO object-detection model were found in weed-detection literature, Zhong et al. (2018) used YOLO to look at very high detail images of insects (Figure 36). Although this dataset is not comparable due to the relatively constant background colour and different subjects, it is interesting to look at the difference between both of Zhong's results: modifying the object-detection pipeline by combining YOLO and a Support-vector Machine (SVM) greatly increases the accuracy: YOLO resulted in an accuracy of 61%, while the modified pipeline yielded an accuracy of 90%. Accuracy could not be calculated for this research as a True Negative (TN) count was not calculated.

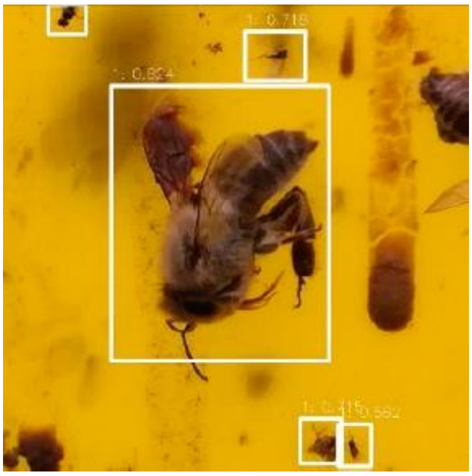


Figure 36: Example of the data used by Zhong et al. (2018). Although it's different to the weeds data, the study shows an interesting improvement of accuracy when combining YOLO with SVM.

It seems that object-detection models which have been specifically designed for weed detection, are usually more accurate than the more general models such as the one used for this research which are commonly tested on datasets such as PASCAL VOC and COCO. Furthermore, the addition of NIR images is thought to be improve performance due to NIR images giving an object-detection model more elements to differentiate between plants. More research should be done regarding task specific object detection models and the addition of NIR images. General object-detection models can be used as a basis for more specific models, as seen in Zhong et al. (2018).

When comparing these results it's important to keep in mind the number of training objects used. For this research, a relatively small dataset was used as no more data was available (approximately 2800 training objects for the original resolution). For example, Milioto et al. (2017a), uses 3987 objects to train and has a much more homogenous field and plant growth (i.e. no waterlogged areas) while Zhong et al. (2018) uses over 10.000 labels. Hung et al. (2014) and Sa et al. (2018) do not describe the number of training objects used. The lack of data with different soils, growth stages and crops make it difficult to assess the robustness of the trained models. Including data from other fields might result in a more robust system. Several problems regarding the quantity of training data used can occur: (1) using too few objects can result in a model which is not robust as it can overfit on the training data, (2) training for too long can result in overfitting which results in the model being too specifically trained on the training data, and (3) too much data with very similar features can result in the model focussing more on those features, thus responding badly if the features are not apparent.

Looking at run time, Milioto et al. (2017b) defined "real-time" as 1 image per second. However, they do not specify how many hectares are contained in one image. According to this definition of real-time, the results found for this research can be attained in real-time, satisfying the needs of both the scout UAV with intervening UGV and the Hopper UAV scenario (Table 11). Although the GPU used (NVIDIA GeForce GTX 1080Ti) could be mounted on a UAV, it is rather heavy and would come at cost of energy. However, future GPU and UAV developments are expected to make this issue relatively irrelevant. Further research could look into testing the same model with less powerful GPUs and running the model on a UAV in real-time.

5.2.2Economic feasibility

In a more practical sense, such a system should be economically feasible for a farmer to use. As long as it remains cheaper and easier to use more pesticides rather than less, farmers will maintain their practices. This research shows that, theoretically, such a system can be much cheaper compared to current practices. The two scenarios described in the *2.2 Economic comparison* section, (1) a scout UAV with intervening UGV and (2) a Hopper UAV, seem realistic as shown by this research. Both scenarios are deemed economically feasible as long as they cost less than conventional weeding measures. (1) The scenario only takes into account the weeds missed by the system (FN) and pesticide costs, as FP are assumed to be skipped by the UGV. For the study area, this leads to an approximate cost of €7.17, €7.59 and €8.75 per hectare for the original, 0.5 and 0.31 resolution respectively (Table 12). (2) The second scenario takes into account FN, FP and pesticide costs. This resulted in an approximate cost of €12.59, €13.51 and €13.30 per hectare for the original, 0.5 and 0.31 resolution respectively (Table 12). Both scenarios cost much less compared to the €516 per hectare of conventional measures. No good estimates of operational costs were found during this research. Ideally such costs should also be added up for both scenarios and the conventional method; however, the operational costs for both scenarios are expected to be less than the conventional method.

Although it is assumed that all FP predictions are misidentified crops and thus rise the cost of the system by €0.07 per weed, visual analysis of the images show that this is only very rarely the case. In a small number of cases the FP were predicted in a region outside of the study area (Figure 37-A). In other cases, the weed was actually correctly identified but is counted as a FP because the ground truth label was assigned to another tile during tile creation (Figure 37-B). Seldomly does the model actually misidentify a crop for a weed (Figure 37-C). No quantitative data was collected to show the type of FP errors. Such information could help clarify why the model results in FP. The case seen in Figure 37-A, is not ideal. Images outside of the study area could be omitted or labelled in future research. This was not done for this research as labelling was done on the understanding that the whole field (also areas outside of the study area) was made free of weeds. These weeds seem to have been missed by the field workers.

In theory such a system would benefit farmers greatly. However, adapting to new strategies and technologies can be difficult for some farmers. Another aspect to keep in mind, is that this research only used data gathered in one day at a certain time. In practice, farmers could choose to fly more frequently in case some weeds were missed. The deep-learning model could react differently due to different lighting conditions and growth stages. This could affect the results both positively (i.e. weeds that were missed before, are detected and taken care of) and negatively (i.e. more FP are predicted, resulting in more crop loss). Further research into testing and training the model for different datasets taken at different times could elucidate these issues.

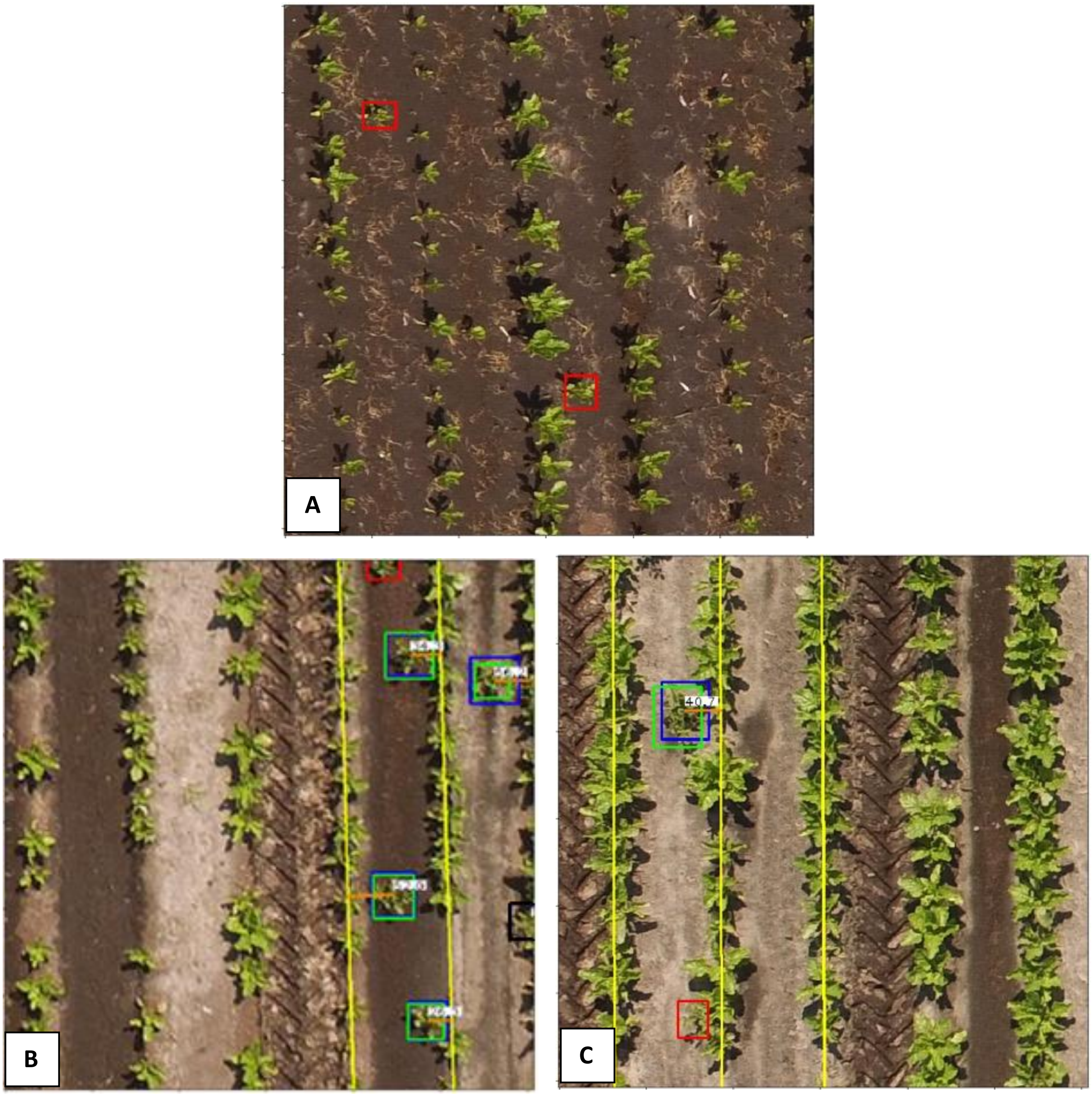


Figure 37: Most FP predictions (red box) are other objects than crops. A, weeds which were not labelled detected outside of the study area. B, weed that was split during tile creation is counted as a FP. C, crop mistakenly predicted as a weed. It's assumed that the FP in C is a crop although it could also be a weed that wasn't labelled. Green boxes correspond to TP predictions, blue boxes are correctly identified ground truth labels, white lines show that the weed is in the crop row, and orange lines show that the weed is outside of the crop row.

5.2.3 Comparison to UGV

Comparing the results found with those of Lottes et al. (2018) and Voorhoeve (2018), the latter a Wageningen University MSc thesis, shows a promising future for weed removal systems (Table 17). Lottes et al. (2018) and Voorhoeve (2018), trained the model to detect both weeds and crops on a sugar beet field using a UGV. Voorhoeve (2018) used the same study area as this research. There is a distinction in results between both classes: the detection of crops resulted in a better performance.

Comparing the UGV weed detection results with this research's UAV results, shows that the overall performance of the UGV is better. This strengthens the hypothesis that higher resolutions result in a better performance. These results also allude to a realistic possibility of a future weed removal system which combine a UAV and a UGV (Scenario 1). However, for this scenario, it would be ideal for the UAV to have the higher recall (to not underestimate the number of weeds) and for the UGV to have the higher precision (to differentiate between crop and weed if the UAV predicted something incorrectly).

The higher IoU found for the UGV is ideal for Scenario 1. Such a ground system would need to spray the weed relatively precisely to reduce the negative impact of pesticides on surrounding crops. For this scenario, the UAV should have an IoU high enough to transmit the general location of weeds to the UGV. For the second scenario however, the UAV would spray and thus would need a higher IoU; making the current findings suboptimal. However, here the idea of a system which changes hyperparameters and/or model depending on the altitude becomes interesting. In practice, the following is proposed: to get a general idea of the location of weeds, the UAV would fly high using a model trained on low resolution images (e.g. 0.31 dataset resolution). Once a weed is spotted the drone decreases its altitude towards the weed, changing the low resolution model to a model which was trained using high resolution images (e.g. UGV images). This allows the UAV to spray with more accuracy.

Table 17: Comparison of results found for different resolution and a YOLOv3-based UGV weed-detection system.

Model	**F1-score**	**Precision**	**Recall**	**mAP (%)**	**IoU (%)**	**Object**	**Source**
YOLOv3	0.83	0.94	0.77	75	65	Weeds	Own research (4 mm)
YOLOv3	0.82	0.93	0.75	72	63	Weeds	Own research (8 mm)
YOLOv3	0.78	0.94	0.71	62	60	Weeds	Own research (12.9 mm)
YOLOv3	0.95	0.95	0.95	88	83	Crops	(Voorhoeve, 2018)
YOLOv3	0.89	0.89	0.88	Idem.	Idem.	Weeds	(Voorhoeve, 2018)
FCN*	0.92	0.89	0.95	-	-	Crops	(Lottes et al., 2018)
FCN*	0.92	0.98	0.88	-	-	Weeds	(Lottes et al., 2018)

For more information regarding the methods and data used, see the respective papers.
* Basic object-detection model was modified to better suit the task.

5.2.4 Viewing angle test

The viewing angle test shows the following trend: for all resolutions, a viewing angle closer to nadir results in a better performance (Figure 26). Although there is a relationship between angle and F1-score, a linear regression analysis performed leads to the belief that this relationship is not linear. The residual plots don't depict a clear linear trend as residuals don't seem to be evenly spread throughout the plot (i.e. there are more positive residuals, and negative residuals reach larger values) (ANNEX VIII). A more thorough statistical investigation is needed to better understand the relationship between viewing angle and F1-score.

Although literature regarding the relationship between viewing angle and object-detector performance is relatively scarce, the research found agrees that viewing angle changes make object detection a more difficult task (Alotaibi and Mahmood, 2017; Masood et al., 2017). Alotaibi and Mahmood (2017), show a dramatic decrease in accuracy with large viewing angles for a gait-recognition model. Proposed methods to improve results for different angles include: (1) a costly multi-camera setup to create a 3D model of the object, and (2) training the system with images depicting the object from multiple views. Masood et al. (2017) also encounter the problem of viewing angle affecting performance of a deep-learning license-plate-detection model. However, the relationship is not studied; instead, the problem is circumvented by training with large quantities of data depicting the objects from multiple views.

For the scenario of weed detection from a UAV, training with more data of weeds at different angles appears to be the most feasible way to improve performance at larger viewing angles. Again, the issue of collecting and labelling larger quantities of data is the limiting factor. Future research could concentrate on assessing the effect of data quantity of objects at different viewing angles on performance.

5.2.5Orthomosaic test

The orthomosaic test gives indications for future improvements of a system which uses deep-learning on orthomosaic images. As shown in this research, the orthomosaic images return a lower performance compared to the original images (Figure 27, 28) . For better results, it's recommended to train a separate model with training objects labelled in the orthomosaic image. This might give a better indication of the actual performance of such a system.

The relationship between resolution and performance is expected to be equal for orthomosaic images as for conventional RGB images. Although only orthomosaic images created by orthorectifying images at the original resolution (4 mm) were tested in this research, Pérez-Ortiz et al. (2015) determine a similar trend for orthomosaic images at 30, 60 and 100 meters.

Visual analysis is useful to understand the shortcomings of the results found. Figure 38 shows the many possible things which can go wrong with orthomosaic images and the method used. Firstly, looking at the quality of the image, it's clear that the orthomosaic image shows much less detail due to a blurring effect created during orthorectification. As objects are sometimes seen from different angles, all angles are superimposed on each other and depicted as such. Also, both tiles do not share exactly the same area which leads to some objects being missed or being added to the image. Secondly, focussing on the FP predictions, this analysis shows that for the orthomosaic tiles, FP are often crops; which is not the case for the other tests. Thirdly, there is an increase in FN for the orthomosaic tile. Two of the three weeds which resulted in a FN in the orthomosaic tile are not visible in the original tile. The last FN corresponds to a different weed in both images, which shows the unintuitive unpredictability of a deep-learning model. Figure 39 depicts a clearer example of the increase in FN for the orthomosaic images.

Due to time constraints and due to the explorative nature of this test, not much data was be labelled for this experiment. Only 22 tiles were labelled, which is deemed an insufficient amount of data for the foundation of conclusive findings. However, this test does give insight into a methodology to work with orthomosaic images and possible problems that one could encounter when working with orthomosaic images. This alludes to future research which could look at training with larger quantities of orthomosaic data. It's unknown if a deep-learning object-detection model generally perform equally or better than a comparable model which uses conventional RGB images; if this is the case, orthomosaic images have the added benefit of being geo-referenced. As locations in the orthomosaic image are bound to a coordinate, labelling could be semi-automated. The first orthomosaic image would need to be labelled manually. After this, the coordinates of the labels can be used in conjunction with orthomosaic images collected later in the growing season. This method has the added benefit of including multi-temporal data to the deep-learning method, possibly making it more robust for different growing stages.

Research which looks at object detection using orthoimagery shows promising results. Längkvist et al. (2016) achieved a classification accuracy of 94.49% for the classification of vegetation, ground, roads, buildings and water in a town using a CNN. Although this dataset is rather different to the sugar beet field, the results do suggest a possible future for well performing orthomosaic-based object-detection models. Dornaika et al. (2016), propose several machine-learning based methods to detect buildings from orthoimagery. The best method achieved a precision, recall, F1-score and accuracy of 92%, 87%, 89% and 96% respectively; all values which are significantly higher compared to those found for this research (Figure 27; Figure 28). Studies for datasets comparable to the sugar beet field could not be found. Thus, future research could attempt to close his knowledge gap.

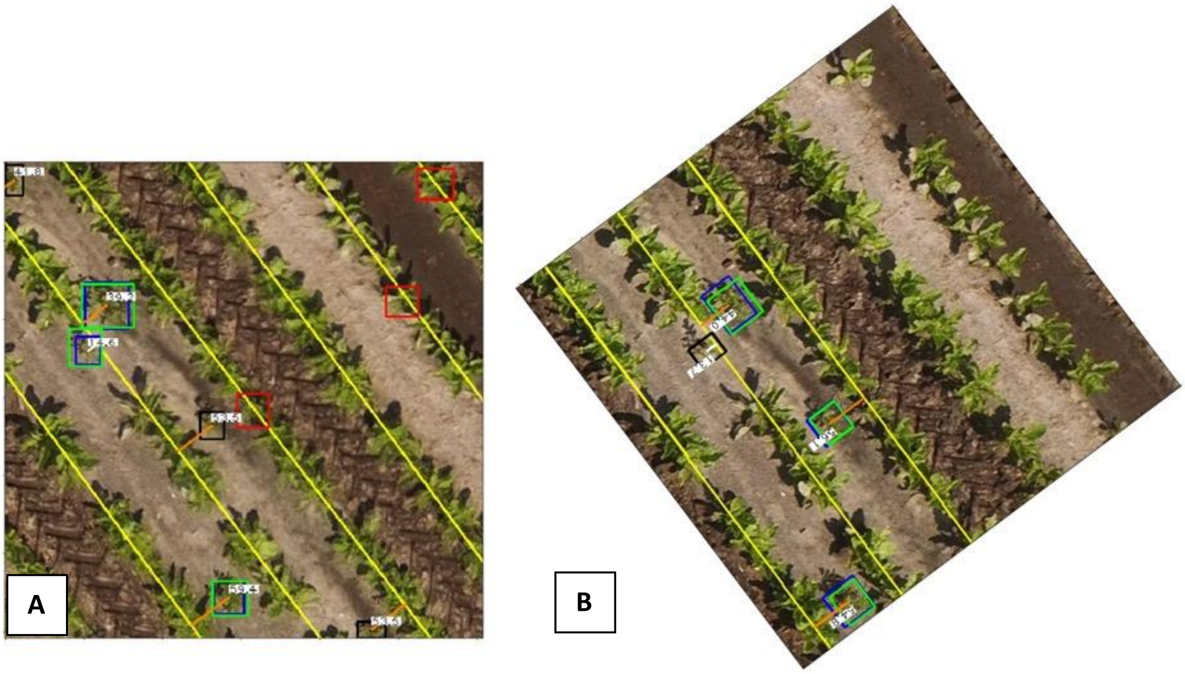


Figure 38: Comparison of an orthomosaic tile (A) and an original resolution tile (B). The model predicts more FP (red box) for the orthomosaic tile. Green boxes correspond to TP predictions, blue boxes are correctly identified ground truth labels, black boxes are FN, blue boxes are correctly identified ground truth labels, white lines show that the weed is in the crop row, and orange lines show that the weed is outside of the crop row.

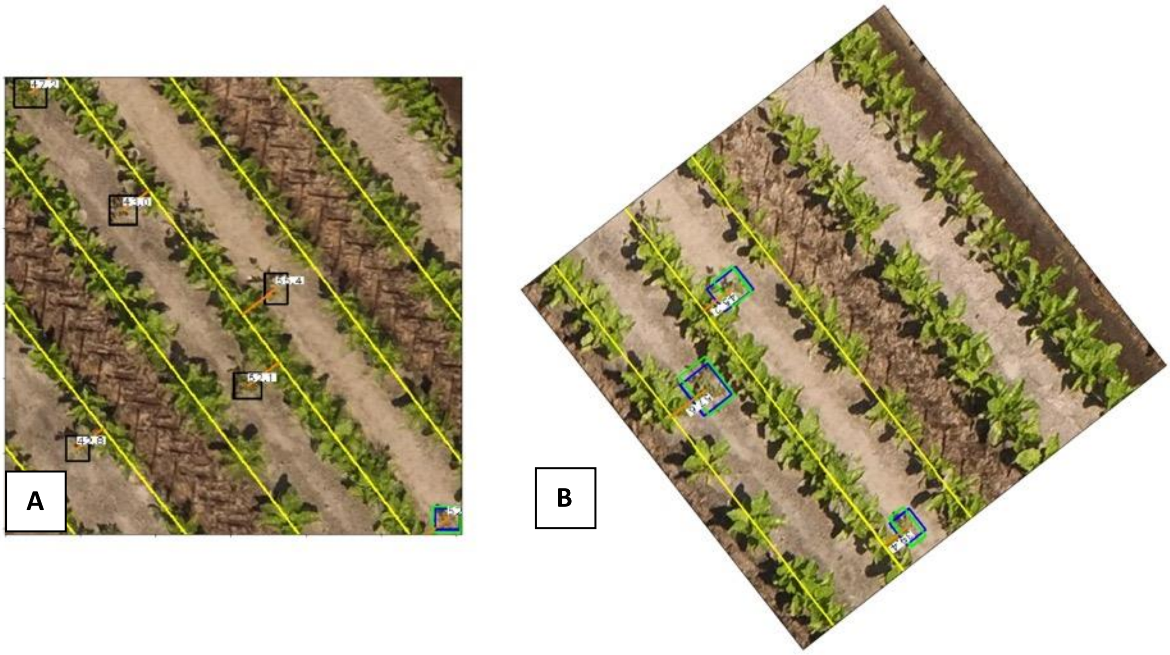


Figure 39: Comparison of an orthomosaic tile (A) and an original resolution tile. This comparison shows an increase in FN (black box) prediction for the orthomosaic tile.

6 CONCLUSION AND RECOMMENDATIONS

All hypotheses as specified in section *1.6 Hypotheses* were proven to be correct. However, in retrospect, specific knowledge gained through this research shows that the original hypotheses were too general. Nevertheless, the following findings can be concluded from this research:

A significant relationship between image resolution and YOLOv3 weed-detection performance exists: the higher the resolution, the better the average performance but the slower the image processing speed. The 4, 8 and 12.9 mm resolution datasets resulted in an average F1-score of 0.83, 0.82 and 0.78 and a processing speed of 49, 14 and 6 sec/ha respectively. Although this is the general trend, further analysis showed that a significant relationship also exists between performance and the area being monitored. For some areas, a lower resolution performed better than higher resolutions; making the lower resolutions worthy to be used. This gives the farmer a choice between different resolutions depending on the field's characteristics.

Further analysis was done to better understand the relationship between object viewing angle from nadir and performance. This resulted in a significant relationship independent of the resolution: weeds closer to nadir (directly under the sensor) yielded a better performance compared to weeds with larger viewing angles.

To bring this research closer to the discipline of remote sensing, wherein orthomosaic images are frequently used, the model trained for the original resolution was tested on orthomosaic images composed of the original images. This test showed that the orthomosaic images performed worse (F1-score = 0.65) compared to the original images (F1-score = 0.90). However, this test used relatively few data, making it statistically less relevant. Nevertheless, the explorative nature of this test gives a taste of what to expect when working with orthomosaic images in combination with a deep-learning object-detection model and insight into possible future research.

This research described two scenarios in which this model could be used in practice. The first scenario consists of a UAV which scouts the field beforehand to identify the location of weeds, and a UGV which goes to the areas appointed by the UAV to remove weeds. For this scenario, it's important for the weed-detection model in the UAV to have a relatively high recall as the UGV is assumed to have a high precision. Real-time processing capacity is deemed unnecessary as the weed removal is done by the UGV. Therefore, processing can be done on the ground on more powerful computers after the images have been captured. The second scenario consists of a UAV (or a swarm of UAVs) which flies from weed to weed, spraying herbicide. This would require a relatively precise system as larger costs are attributed to spraying an incorrect object such as a crop. Real-time processing is important in this scenario. Considering a minimal real-time capacity needed of 1 image per second, all findings point towards YOLOv3 being capable of real-time processing for all tested resolutions using a relatively heavy GPU. Real-time processing on a UAV could not be tested as an adequate GPU was not available. However, developments in UAV and GPU technology lead to the assumption that this will not be an issue in the future. Economically speaking, comparing the two use scenarios with the conventional method of herbicide application, shows that both discussed scenarios are economically highly beneficial per hectare for the farmer.

Further looking at the future of precision agriculture and how such an object-detection system could help in the realisation of a more sustainable future, several recommendations for future research are presented:

Firstly, the different scenarios were theoretically proven to be economically advantageous for the farmer. However, for certainty and to promote the adoption of such systems, these scenarios would need to be tested on the field. UGV technology is already capable as shown, amongst others, by

Voorhoeve (2018) and Lottes et al. (2018). UAV weed-spraying technology on the other hand, is still in its relative infancy. Future research could look at the practical implementation of a UAV based weed-detection system. Interesting research paths have been proposed in this research; e.g. a UAV based weed-detection which dynamically changes model hyperparameters depending on the height to achieve the best possible result.

Secondly, future research could attempt a similar research but with images taken at different altitudes. As images were only available at 10 m, the different heights were digitally simulated. Furthermore, it's recommended to research more altitudes instead of three. As was found, the relationship between resolution and performance was not linear; however, more resolutions were needed to better assess the existing relationship.

Thirdly, the performance of the model was significantly affected by the area being monitored. This resulted in counterintuitive cases in which the lower resolutions yielded better results than the higher resolutions. Research could look into what it is that causes some areas to yield better results for lower resolutions. Looking at field and object characteristics might give an indication on factors that result in this phenomenon. Farmers could then act upon these findings to make their field as complacent for the model as possible.

Fourthly, it was shown in the discussion that deep-learning models developed specifically for weed detection performed substantially better than more general models. The best performing weed-detection models in literature, made use of RGB, hyperspectral images, geometric relationships between crops and usually combined a CNN with other methods such as a SVM. However, the economic benefit of using a RGB-only system makes it worth to research the development deep-learning models optimised to detect weeds using only RGB.

Fifthly, most literature focusses on a maximum of two research fields. To operationalise such systems, the models should be proven to be robust in many different types of fields. Not only can gathered data from other fields improve the model by training, but testing on other fields will give insight into the robustness of the weed-detection system. In theory, the system works and is economically beneficial. Now it's a matter of putting theory into practice.

Penultimately, a special focus should be assigned to automated labelling systems. Labelling is currently manual work which is subject to the expertise of the labeller and which can result in a bias towards easier to detect objects. By automating or semi-automating labelling, not only will the manual work be diminished but much larger quantities of training data could be created. This would partly remove the hurdle that is object labelling.

Ultimately, weed detection on orthomosaic images should be researched further. Orthoimages have the benefit of begin orthorectified which leads to interesting possibilities when combined with a deep-learning model. A label on an orthoimage is bound to a geographical location, making it possible to use a same label at different times. This could be used to gather training data at different growth stages, as well as allowing a UAV to geographically point a UGV towards a weed.

7 BIBLIOGRAPHY

Alexey, 2019. Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection (Tensor Cores are used): AlexeyAB/darknet.

Alotaibi, M., Mahmood, A., 2017. Improved gait recognition based on specialized deep convolutional neural network. Computer Vision and Image Understanding 164, 103–110. https://doi.org/10.1016/j.cviu.2017.10.004

Badrinarayanan, V., Kendall, A., Cipolla, R., 2017. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEEE Trans. Pattern Anal. Mach. Intell. 39, 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615

Bawden, O., Kulk, J., Russell, R., McCool, C., English, A., Dayoub, F., Lehnert, C., Perez, T., 2017. Robot for weed species plant-specific management. Journal of Field Robotics 34, 1179–1199. https://doi.org/10.1002/rob.21727

Bayr, U., Puschmann, O., 2019. Automatic detection of woody vegetation in repeat landscape photographs using a convolutional neural network. Ecological Informatics 50, 220–233. https://doi.org/10.1016/j.ecoinf.2019.01.012

Beukema, H.P., Van der Zaag, D.E., 1990. Introduction to potato production. Pudoc, Wageningen.

Bottger, T., Follmann, P., Fauser, M., 2017. Measuring the Accuracy of Object Detectors and Trackers. arXiv:1704.07293 [cs] 10496. https://doi.org/10.1007/978-3-319-66709-6

Cao, L., Wang, C., Li, J., 2016. Vehicle detection from highway satellite images via transfer learning. Information Sciences 366, 177–187. https://doi.org/10.1016/j.ins.2016.01.004

Chevrefils, C., Cheriet, F., Aubin, C.-E., Grimard, G., 2009. Texture Analysis for Automatic Segmentation of Intervertebral Disks of Scoliotic Spines From MR Images. IEEE Trans. Inform. Technol. Biomed. 13, 608–620. https://doi.org/10.1109/TITB.2009.2018286

Chu, W., Cai, D., 2018. Deep feature based contextual model for object detection. Neurocomputing 275, 1035–1042. https://doi.org/10.1016/j.neucom.2017.09.048

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B., 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, pp. 3213–3223. https://doi.org/10.1109/CVPR.2016.350

Dai, J., Li, Y., He, K., Sun, J., 2016. R-FCN: Object Detection via Region-based Fully Convolutional Networks. arXiv:1605.06409 [cs].

Davies, F.T., Geneve, R.L., Kester, D.E., Hartmann, H.T., 2011. Hartmann and Kester's plant propagation: principles and practice, 8th ed. ed. Prentice Hall, Boston.

Davis, A.M., Pradolin, J., 2016. Precision Herbicide Application Technologies To Decrease Herbicide Losses in Furrow Irrigation Outflows in a Northeastern Australian Cropping System. Journal of Agricultural and Food Chemistry 64, 4021–4028. https://doi.org/10.1021/acs.jafc.5b04987

Dedousis, A.P., Bartzanas, T. (Eds.), 2010. Soil engineering, Soil biology. Springer, New York.

Dornaika, F., Moujahid, A., El Merabet, Y., Ruichek, Y., 2016. Building detection from orthophotos using a machine learning approach: An empirical study on image segmentation and descriptors. Expert Systems with Applications 58, 130–142. https://doi.org/10.1016/j.eswa.2016.03.024

Du, Q., Gunzburger, M., Ju, L., Wang, X., 2006. Centroidal Voronoi Tessellation Algorithms for Image Compression, Segmentation, and Multichannel Restoration. J Math Imaging Vis 24, 177–194. https://doi.org/10.1007/s10851-005-3620-4

Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A., 2010. The Pascal Visual Object Classes (VOC) Challenge. Int J Comput Vis 88, 303–338. https://doi.org/10.1007/s11263-009-0275-4

Flourish Project [WWW Document], 2018. URL http://flourish-project.eu/ (accessed 10.11.18).

Fuentes, A., Yoon, S., Kim, S., Park, D., Fuentes, A., Yoon, S., Kim, S.C., Park, D.S., 2017. A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition. Sensors 17, 2022. https://doi.org/10.3390/s17092022

Girden, E.R., 1992. ANOVA: Repeated Measures. SAGE.

Girshick, R., 2015. Fast R-CNN, in: ArXiv:1504.08083 [Cs]. Presented at the 2015 IEEE International Conference on Computer Vision (ICCV), IEEE, Santiago, Chile. https://doi.org/10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition. Presented at the 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587. https://doi.org/10.1109/CVPR.2014.81

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2013. Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524 [cs].

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. The MIT Press, Cambridge.

Guerrero, J.M., Pajares, G., Montalvo, M., Romeo, J., Guijarro, M., 2012. Support Vector Machines for crop/weeds identification in maize fields. Expert Systems with Applications 39, 11149–11155. https://doi.org/10.1016/j.eswa.2012.03.040

Haris, M., Shakhnarovich, G., Ukita, N., 2018. Task-Driven Super Resolution: Object Detection in Low-resolution Images. arXiv:1803.11316 [cs].

Hartzler, B., 2009. The cost of convenience: The impact of weeds on crop yields, in: Proceedings of the Integrated Crop Management Conference. Presented at the Proceedings of the 21st Annual Integrated Crop Management Conference, Iowa State University, Digital Press. https://doi.org/10.31274/icm-180809-11

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs].

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580 [cs].

Hoermann, S., Bach, M., Dietmayer, K., 2018. Dynamic Occupancy Grid Prediction for Urban Autonomous Driving: A Deep Learning Approach with Fully Automatic Labeling, in: 2018 IEEE International Conference on Robotics and Automation (ICRA). Presented at the 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Brisbane, QLD, pp. 2056–2063. https://doi.org/10.1109/ICRA.2018.8460874

Howard, A.G., 2013. Some Improvements on Deep Convolutional Neural Network Based Image Classification. arXiv:1312.5402 [cs].

Huang, H., Deng, J., Lan, Y., Yang, A., Deng, X., Zhang, L., 2018. A fully convolutional network for weed mapping of unmanned aerial vehicle (UAV) imagery. PLOS ONE 13, e0196302. https://doi.org/10.1371/journal.pone.0196302

Hung, C., Xu, Z., Sukkarieh, S., 2014. Feature Learning Based Approach for Weed Classification Using High Resolution Aerial Images from a Digital Camera Mounted on a UAV. Remote Sensing 6, 12037–12054. https://doi.org/10.3390/rs61212037

Igbedioh, S.O., 1991. Effects of agricultural pesticides on humans, animals, and higher plants in developing countries. Arch. Environ. Health 46, 218–224. https://doi.org/10.1080/00039896.1991.9937452

Kathuria, A., 2018. What’s new in YOLO v3? [WWW Document]. Towards Data Science. URL https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b (accessed 10.10.18).

Khosla, R., Fleming, K., Delgado, J.A., Shaver, T.M., Westfall, D.G., 2002. Use of site-specific management zones to improve nitrogen management for precision agriculture. Journal of Soil and Water Conservation 57, 513–518.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks, in: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105.

Kropff, M.J., 1988. Modelling the effects of weeds on crop production. Weed Research 28, 465–471. https://doi.org/10.1111/j.1365-3180.1988.tb00829.x

Längkvist, M., Kiselev, A., Alirezaie, M., Loutfi, A., 2016. Classification and Segmentation of Satellite Orthoimagery Using Convolutional Neural Networks. Remote Sensing 8, 329. https://doi.org/10.3390/rs8040329

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444. https://doi.org/10.1038/nature14539

Li, Y., 2017. Deep Reinforcement Learning: An Overview. arXiv:1701.07274 [cs].

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P., 2014. Microsoft COCO: Common Objects in Context. arXiv:1405.0312 [cs].

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. SSD: Single Shot MultiBox Detector. arXiv:1512.02325 [cs] 9905, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

Lottes, P., Behley, J., Milioto, A., Stachniss, C., 2018. Fully Convolutional Networks with Sequential Information for Robust Crop and Weed Detection in Precision Farming. IEEE Robotics and Automation Letters 3, 2870–2877. https://doi.org/10.1109/LRA.2018.2846289

Lottes, P., Hörferlin, M., Sander, S., Stachniss, C., 2017. Effective Vision-based Classification for Separating Sugar Beets and Weeds for Precision Farming. Journal of Field Robotics 34, 1160–1178. https://doi.org/10.1002/rob.21675

Lottes, P., Stachniss, C., 2017. Semi-supervised online visual crop and weed classification in precision farming exploiting plant arrangement, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Presented at the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, Vancouver, BC, pp. 5155–5161. https://doi.org/10.1109/IROS.2017.8206403

Masood, S.Z., Shu, G., Dehghan, A., Ortiz, E.G., 2017. License Plate Detection and Recognition Using Deeply Learned Convolutional Neural Networks. arXiv:1703.07330 [cs].

Milioto, A., Lottes, P., Stachniss, C., 2017a. Real-time blob-wise sugar beets vs weeds classification for monitoring fields using convolutional neural networks. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-2/W3, 41–48. https://doi.org/10.5194/isprs-annals-IV-2-W3-41-2017

Milioto, A., Lottes, P., Stachniss, C., 2017b. Real-time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs. arXiv:1709.06764 [cs].

Mitchell, T.M., 1997. Machine Learning, McGraw-Hill series in computer science. McGraw-Hill, New York.

Oksuz, K., Cam, B.C., Akbas, E., Kalkan, S., 2018. Localization Recall Precision (LRP): A New Performance Metric for Object Detection, in: ArXiv:1807.01696 [Cs]. Presented at the Lecture Notes in Computer Science, LNCS.

Paauw, J.G.M., Molendijk, L.P.G., 2000. Aardappelopslag in wintertarwe vermeerdert aardappelcystenaaltjes. Praktijkonderzoek Akkerbouw Vollegrondsgroenteteelt bulletin Akkerbouw.

Pérez-Ortiz, M., Peña, J.M., Gutiérrez, P.A., Torres-Sánchez, J., Hervás-Martínez, C., López-Granados, F., 2016. Selecting patterns and features for between- and within- crop-row weed mapping using UAV-imagery. Expert Systems with Applications 47, 85–94. https://doi.org/10.1016/j.eswa.2015.10.043

Pérez-Ortiz, M., Peña, J.M., Gutiérrez, P.A., Torres-Sánchez, J., Hervás-Martínez, C., López-Granados, F., 2015. A semi-supervised system for weed mapping in sunflower crops using unmanned aerial vehicles and a crop row detection method. Applied Soft Computing 37, 533–544. https://doi.org/10.1016/j.asoc.2015.08.027

Piewak, F., Pinggera, P., Schäfer, M., Peter, D., Schwarz, B., Schneider, N., Enzweiler, M., Pfeiffer, D., Zöllner, M., 2019. Boosting LiDAR-Based Semantic Labeling by Cross-modal Training Data Generation, in: Leal-Taixé, L., Roth, S. (Eds.), Computer Vision – ECCV 2018 Workshops. Springer International Publishing, Cham, pp. 497–513. https://doi.org/10.1007/978-3-030-11024-6_39

Pordel, M., Hellström, T., 2015. Semi-Automatic Image Labelling Using Depth Information. Computers 4, 142–154. https://doi.org/10.3390/computers4020142

Poudyal, R., 2018. SSD(Single Shot Multi-Box Detection) for real time object detection. Medium. URL https://medium.com/@rabinpoudyal1995/ssd-single-shot-multi-box-detection-for-real-time-object-detection-5f2a06e33a4a (accessed 10.10.18).

QGIS Development Team, 2016. QGIS Geographic Information System. Open Source Geospatial Foundation Project.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2015. You Only Look Once: Unified, Real-Time Object Detection, in: ArXiv:1506.02640 [Cs]. Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788. https://doi.org/10.1109/CVPR.2016.91

Redmon, J., Farhadi, A., 2018. YOLOv3: An Incremental Improvement.

Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497 [cs].

RIEGL, 2019. RIEGL - RIEGL RiCOPTER [WWW Document]. URL http://www.riegl.com/products/unmanned-scanning/ricopter/ (accessed 5.7.19).

Ruigrok, T., Unnikrishnan, A., Kootstra, G., 2018. KPI Reduce weed control cost.

Sa, I., Chen, Z., Popovic, M., Khanna, R., Liebisch, F., Nieto, J., Siegwart, R., 2018. weedNet: Dense Semantic Weed Classification Using Multispectral Images and MAV for Smart Farming. IEEE Robot. Autom. Lett. 3, 588–595. https://doi.org/10.1109/LRA.2017.2774979

Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T., McCool, C., Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T., McCool, C., 2016. DeepFruits: A Fruit Detection System Using Deep Neural Networks. Sensors 16, 1222. https://doi.org/10.3390/s16081222

Sagar, V., J Jain, S., B. S., V., 2018. Yield Estimation using faster R-CNN.

Schmidhuber, J., 2015. Deep Learning in Neural Networks: An Overview. Neural Networks 61, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Simonyan, K., Zisserman, A., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs].

Slaughter, D.C., Giles, D.K., Downey, D., 2008. Autonomous robotic weed control systems: A review. Computers and Electronics in Agriculture, Emerging Technologies For Real-time and Integrated Agriculture Decisions 61, 63–78. https://doi.org/10.1016/j.compag.2007.05.008

Sørensen, C.G., Madsen, N.A., Jacobsen, B.H., 2005. Organic Farming Scenarios: Operational Analysis and Costs of implementing Innovative Technologies. Biosystems Engineering 91, 127–137. https://doi.org/10.1016/j.biosystemseng.2005.03.006

Spruijt, J., van der Voort, M., 2015. KWIN AGV 2015. Praktrijkonderzoek Plant & Omgeving, Lelystad.

Steward, B.L., Tian, L.F., 1998. Real-time machine vision weed-sensing 12.

Sujaritha, M., Annadurai, S., Satheeshkumar, J., Kowshik Sharan, S., Mahesh, L., 2017. Weed detecting robot in sugarcane fields using fuzzy real time classifier. Computers and Electronics in Agriculture 134, 160–171. https://doi.org/10.1016/j.compag.2017.01.008

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2015. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567 [cs].

Tanbakuchi, A., 2018. Geometric Image Transformations — OpenCV 2.4.13.7 documentation [WWW Document]. URL https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html (accessed 9.20.18).

Tayara, H., Chong, K.T., 2018. Object Detection in Very High-Resolution Aerial Images Using One-Stage Densely Connected Feature Pyramid Network. Sensors (Basel) 18. https://doi.org/10.3390/s18103341

Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T., Smeulders, A.W.M., 2013. Selective Search for Object Recognition. International Journal of Computer Vision 104, 154–171. https://doi.org/10.1007/s11263-013-0620-5

Underwood, J.P., Rahman, M.M., Robson, A., Walsh, K.B., Koirala, A., Wang, Z., 2018. Fruit load estimation in mango orchards – a method comparison 6.

Unterthiner, T., Mayr, A., Wegner, J.K., 2015. Deep Learning as an Opportunity in Virtual Screening.

Vanhala, P., Kurstjens, D., Ascard, J., Bertram, A., Cloutier, D.C., Mead, A., Raffaelli, M., Rasmussen, J., 2004. Guidelines for physical weed control research: flame weeding, weed harrowing and intra-row cultivation 32.

Voorhoeve, L., 2018. Plant Detection under Natural Illumination Conditions (Msc thesis). Wageningen University, Wageningen.

Westoby, M.J., Brasington, J., Glasser, N.F., Hambrey, M.J., Reynolds, J.M., 2012. 'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications. Geomorphology 179, 300–314. https://doi.org/10.1016/j.geomorph.2012.08.021

Wirges, S., Fischer, T., Frias, J.B., Stiller, C., 2018. Object Detection and Classification in Occupancy Grid Maps using Deep Convolutional Networks. arXiv:1805.08689 [cs].

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1, 67–82. https://doi.org/10.1109/4235.585893

Xia, Y., Yao, H., Sun, X., Zhang, Y., 2018. Shallow and Deep Model Investigation for Distinguishing Corn and Weeds, in: Zeng, B., Huang, Q., El Saddik, A., Li, H., Jiang, S., Fan, X. (Eds.), Advances in Multimedia Information Processing – PCM 2017. Springer International Publishing, Cham, pp. 693–702. https://doi.org/10.1007/978-3-319-77380-3_66

Zhao, T., Wang, Z., Yang, Q., Chen, Y., 2017. Melon yield prediction using small unmanned aerial vehicles, in: Thomasson, J.A., McKee, M., Moorhead, R.J. (Eds.), . Presented at the SPIE Commercial + Scientific Sensing and Imaging, Anaheim, California, United States, p. 1021808. https://doi.org/10.1117/12.2262412

Zhong, Y., Gao, J., Lei, Q., Zhou, Y., Zhong, Y., Gao, J., Lei, Q., Zhou, Y., 2018. A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture. Sensors 18, 1489. https://doi.org/10.3390/s18051489

Zhou, Chellappa, 1988. Computation of optical flow using a neural network 71–78 vol.2. https://doi.org/10.1109/icnn.1988.23914

8 ANNEXES

8.1 Annex I: Orthoimage of the study area

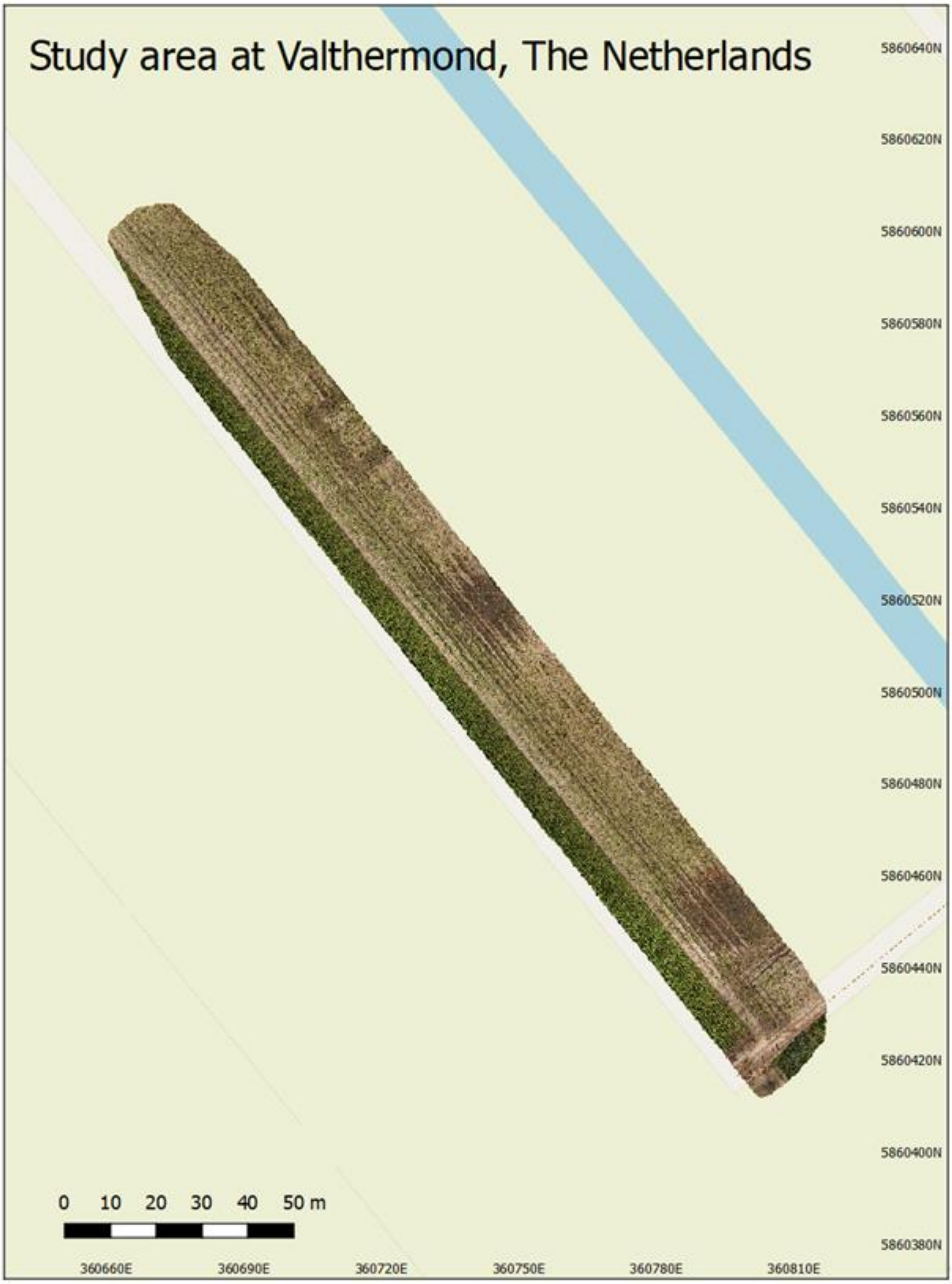


Figure 40: Study area at Valthermond, The Netherlands. A darker soil can be seen in places which were waterlogged in the past.

8.2 Annex II: Image properties

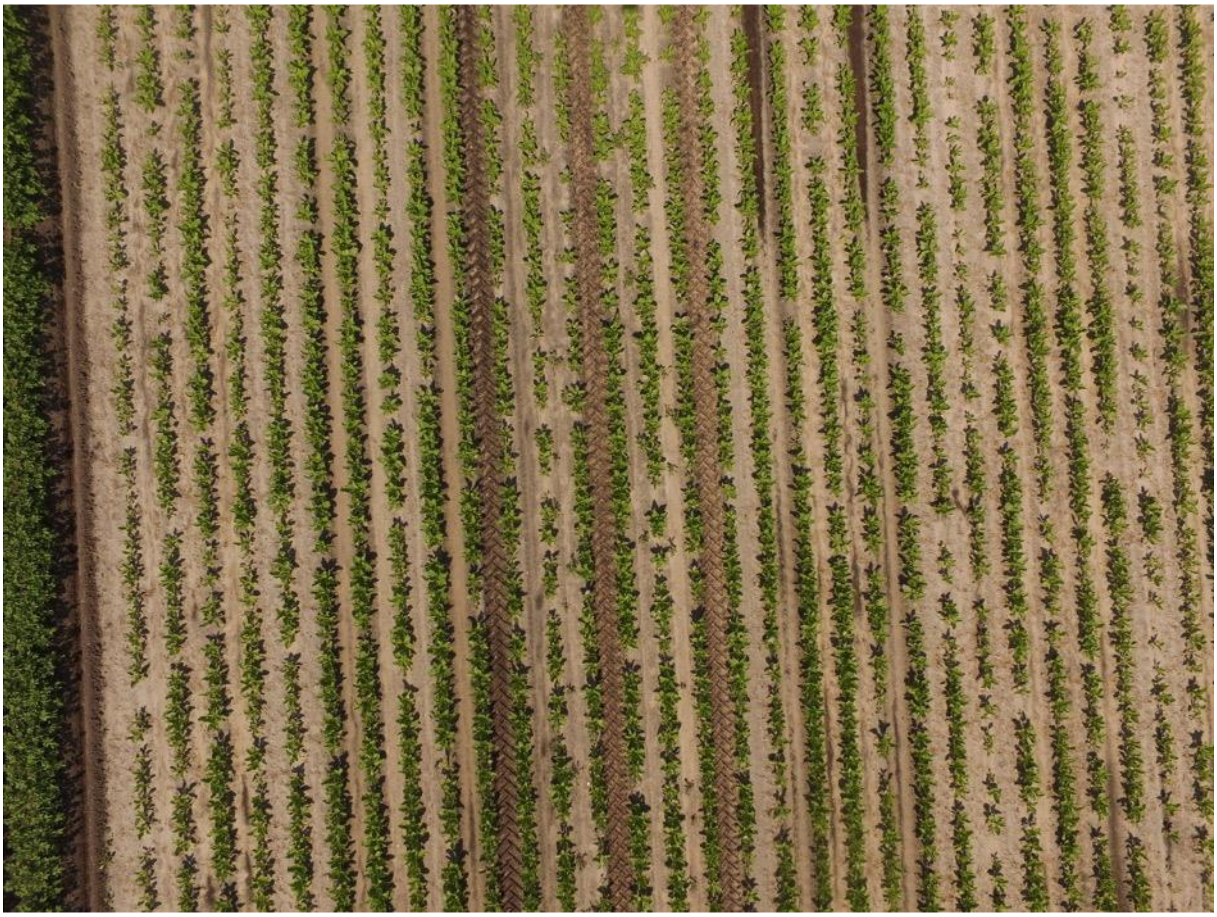


Figure 41: Example of an original image.

Camera: DJI FC300X
Image dimensions: 4000 x 3000 px
Shutter speed: 1/1600 seconds
Focal length: 3.61 mm
Aperture: f/2.8
ISO: 100
Image file format: DJI_*imagenumber* E.g. DJI_0220

8.3 Annex III: Script to resize label coordinates and adapt labels to tiles (0.5 dataset)

8.3.1 Pseudocode

The annotation format used by YOLOv3 consists of four elements per bounding box – It's important to note that all coordinates have their origin in the upper-left corner of the image:

1. x, the absolute x-coordinate value for the centre of the bounding box (abs_x) divided by the image width (width_img);
2. y, the absolute y-coordinate value for the centre of the bounding box (abs_y) divided by the image height (height_img);
3. w, the absolute width of the bounding box (abs_w) divided by the width_img;
4. h, the absolute height of the bounding box (abs_h) divided by the height_img.

To translate the annotations of the original images to the tiles, the following pseudo-code was executed:

1. Multiply the annotation elements by the height and width of the cropped images. This transforms the annotation of each bounding box to: abs_x, abs_y, abs_w, abs_h;
2. Split the image into tiles of 608x608 px and label each row and column (row_id x column_id). For the cropped original images described above, the image would be split into a grid of 6x4. The upper-left and lower-right most tiles would be labelled "1,1" and "6,4" respectively;
3. Divide height_img by the number of row splits (4). Also divide width_img by the number of column splits (6). This results in the resolution of the tiles (608x608 px), which are assigned to the variables h_res and w_res;
4. Move the bounding box annotations to match the corresponding tile by following these formulas:

   ```
   abs_x_new = -(abs_x - (column_id * w_res))
   abs_y_new = -(abs_y - (row_id * h_res))
   ```

   To know which set of elements in the annotation file corresponds to which tile, a logical statement was created which relates abs_x and abs_y to a row_id and column_id. Such a statement can be as follows for the upper-left most tile:

   ```
   If abs_x AND abs_y <= 608:
       row_id and column_id == 1
   ```
5. Normalise the annotations back, to conform the YOLOv3 annotation format. As the bounding box size hasn't changed, abs_w and abs_h are the same as before. However, the new elements for the annotation labels are as follows:
   a. abs_x_new/w_res
   b. abs_y_new/h_res
   c. abs_w/w_res
   d. abs_y/h_res

8.3.2 Python code

```
# Title: Annotation Label Translator Algorithm
# Author: Sebastian Paolini van Helfteren
# Date: 23/10/2018
# For more information, contact the author (sebastianpvh95@gmail.com)

### Libraries ###
import os

### Script ###
```

```
resized_05_annotation_folder = "path to annotation folder"
cropped_05_image_folder = "path to cropped annotation folder"
tile_05_image_folder = “path to new tile annotation folder”

# Set variables needed for the loop below #
width_img_05 = 2000
height_img_05 = 1500
x_crop_05 = 176
y_crop_05 = 284

# Change it all here!
w_cropped_res = width_img_05 - x_crop_05
h_cropped_res = height_img_05 - y_crop_05

# The loop cycles through each annotation in the original annotation folder and saves new
annotations to a new folder #
for annotation in os.listdir(resized_05_annotation_folder):
    crop_txt = open(cropped_05_image_folder + "crop_" + annotation, "w")
    lines = [line.rstrip('\n') for line in open(resized_05_annotation_folder+str(annotation))]
    for box in lines:
        label, x, y, w, h = box.split()
        abs_x = float(x) * width_img_05
        abs_y = float(y) * height_img_05
        abs_w = float(w) * width_img_05
        abs_h = float(h) * height_img_05

        norm_x = (abs_x-x_crop_05)/w_cropped_res
        norm_y = (abs_y-y_crop_05)/h_cropped_res
        norm_w = abs_w/w_cropped_res
        norm_h = abs_h/h_cropped_res

        if abs_y - (abs_h/2) >= y_crop_05:
            crop_txt.write(str(label) + " " + str(norm_x) + " " + str(norm_y) + " " + str(norm_w) + " "
                        + str(norm_h) + "\n")
    crop_txt.close()

# Set variables for the loop below #
w_tile_res = 608
h_tile_res = 608
w_splits = w_cropped_res / w_tile_res
h_splits = h_cropped_res / h_tile_res
column_id = 0
row_id = 0

# For every annotated box, write a new box that fits in the tile
# Not all "tile_N_txt" are needed. This is dependent on the number of tiles that you want/how big
#your original image is
# Comment/delete the ones that you don't need out. Both here and at the end of the script
# In some places you can find "FIRST TILE ROW". This refers to which row of tiles is being calculated
#from that point on
# In every if statement there are some values which can be changed if you don't have tiles of
#608x608
```

```
for cropped_annotation in os.listdir(cropped_05_image_folder):
    tile_lines = [tile_line.rstrip('\n') for tile_line in open(cropped_05_image_folder +
str(cropped_annotation))]

# 0.5 resized tiles! #
    tile_0_txt = open(tile_05_image_folder + cropped_annotation.replace(".txt", "_0") + ".txt", "w")
    tile_1_txt = open(tile_05_image_folder + cropped_annotation.replace(".txt", "_1") + ".txt", "w")
    tile_2_txt = open(tile_05_image_folder + cropped_annotation.replace(".txt", "_2") + ".txt", "w")
    tile_3_txt = open(tile_05_image_folder + cropped_annotation.replace(".txt", "_3") + ".txt", "w")
    tile_4_txt = open(tile_05_image_folder + cropped_annotation.replace(".txt", "_4") + ".txt", "w")
    tile_5_txt = open(tile_05_image_folder + cropped_annotation.replace(".txt", "_5") + ".txt", "w")

    for box in tile_lines:  # box refers to a bounding box
        label_tile, x_tile, y_tile, w_tile, h_tile = box.split()
        abs_x_tile = int(float(x_tile) * w_cropped_res)
        abs_y_tile = int(float(y_tile) * h_cropped_res)
        abs_w_tile = int(float(w_tile) * w_cropped_res)
        abs_h_tile = int(float(h_tile) * h_cropped_res)

        ################# FIRST TILE ROW #################
        # Tile (0,0)
        if (abs_x_tile <= 607) and (abs_y_tile <= 607):
            column_id = 1
            row_id = 1
            abs_x_tile_new = -(abs_x_tile - (column_id * w_tile_res))
            abs_y_tile_new = -(abs_y_tile - (row_id * h_tile_res))

            norm_x_tile = 1 - abs_x_tile_new / w_tile_res
            norm_y_tile = 1 - abs_y_tile_new / h_tile_res
            norm_w_tile = abs_w_tile / w_tile_res
            norm_h_tile = abs_h_tile / h_tile_res

            tile_0_txt.write(str(label_tile) + " " + str(norm_x_tile) + " " + str(norm_y_tile) + " " +
str(norm_w_tile)
                             + " " + str(norm_h_tile) + "\n")

        # Tile (1,0)
        elif (608 <= abs_x_tile <= 1216) and (abs_y_tile <= 607):
            column_id = 2
            row_id = 1
            abs_x_tile_new = -(abs_x_tile - (column_id * w_tile_res))
            abs_y_tile_new = -(abs_y_tile - (row_id * h_tile_res))

            norm_x_tile = 1 - abs_x_tile_new / w_tile_res
            norm_y_tile = 1 - abs_y_tile_new / h_tile_res
            norm_w_tile = abs_w_tile / w_tile_res
            norm_h_tile = abs_h_tile / h_tile_res

            tile_1_txt.write(str(label_tile) + " " + str(norm_x_tile) + " " + str(norm_y_tile) + " " +
str(norm_w_tile)
                             + " " + str(norm_h_tile) + "\n")
        # Tile (2,0)
        elif (1217 <= abs_x_tile <= 1824) and (abs_y_tile <= 607):
            column_id = 3
            row_id = 1
            abs_x_tile_new = -(abs_x_tile - (column_id * w_tile_res))
            abs_y_tile_new = -(abs_y_tile - (row_id * h_tile_res))

            norm_x_tile = 1 - abs_x_tile_new / w_tile_res
            norm_y_tile = 1 - abs_y_tile_new / h_tile_res
            norm_w_tile = abs_w_tile / w_tile_res
            norm_h_tile = abs_h_tile / h_tile_res

            tile_2_txt.write(str(label_tile) + " " + str(norm_x_tile) + " " + str(norm_y_tile) + " " +
str(norm_w_tile)
                             + " " + str(norm_h_tile) + "\n")

        ################# SECOND TILE ROW #################
        # Tile (0,1)
        elif (abs_x_tile <= 607) and (608 <= abs_y_tile <= 1216):
            column_id = 1
            row_id = 2
            abs_x_tile_new = -(abs_x_tile - (column_id * w_tile_res))
            abs_y_tile_new = -(abs_y_tile - (row_id * h_tile_res))

            norm_x_tile = 1 - abs_x_tile_new / w_tile_res
            norm_y_tile = 1 - abs_y_tile_new / h_tile_res
            norm_w_tile = abs_w_tile / w_tile_res
            norm_h_tile = abs_h_tile / h_tile_res

            tile_3_txt.write(str(label_tile) + " " + str(norm_x_tile) + " " + str(norm_y_tile) + " " +
str(norm_w_tile)
                             + " " + str(norm_h_tile) + "\n")


              # Tile (1,1)
        elif (608 <= abs_x_tile <= 1216) and (608 <= abs_y_tile <= 1216):
            column_id = 2
            row_id = 2
            abs_x_tile_new = -(abs_x_tile - (column_id * w_tile_res))
            abs_y_tile_new = -(abs_y_tile - (row_id * h_tile_res))

            norm_x_tile = 1 - abs_x_tile_new / w_tile_res
            norm_y_tile = 1 - abs_y_tile_new / h_tile_res
            norm_w_tile = abs_w_tile / w_tile_res
            norm_h_tile = abs_h_tile / h_tile_res

            tile_4_txt.write(str(label_tile) + " " + str(norm_x_tile) + " " + str(norm_y_tile) + " " +
str(norm_w_tile)
                             + " " + str(norm_h_tile) + "\n")

        # Tile (2,1)
```

```
        elif (1217 <= abs_x_tile <= 1824) and (608 <= abs_y_tile <= 1216):
            column_id = 3
            row_id = 2
            abs_x_tile_new = -(abs_x_tile - (column_id * w_tile_res))
            abs_y_tile_new = -(abs_y_tile - (row_id * h_tile_res))

            norm_x_tile = 1 - abs_x_tile_new / w_tile_res
            norm_y_tile = 1 - abs_y_tile_new / h_tile_res
            norm_w_tile = abs_w_tile / w_tile_res
            norm_h_tile = abs_h_tile / h_tile_res

            tile_5_txt.write(str(label_tile) + " " + str(norm_x_tile) + " " + str(norm_y_tile) + " " +
str(norm_w_tile)
                             + " " + str(norm_h_tile) + "\n")
        else:
            print("Something went wrong. Try again or adapt the script")  # Test it!
# Resized 0.5 Images #
    tile_0_txt.close()
    tile_1_txt.close()
    tile_2_txt.close()
    tile_3_txt.close()
    tile_4_txt.close()
    tile_5_txt.close()
```

8.4 Annex IV: Script to test performance (original resolution)

```
# Title: Performance Tester Algorithm
# Author: Sebastian Paolini van Helfteren & Gert Kootstra
# Date: 17/03/2019
# For more information, contact the author (sebastianpvh95@gmail.com)

### Libraries ###
import cv2
from matplotlib import pyplot as plt
import numpy as np
import math
from IPython.core.pylabtools import figsize
import csv


# Class to represent lines in the standard form (ax + by + c = 0)
# and to perform some calculations with them
class ABCLine:

    def __init__(self):
        self.abc = np.zeros(3)

    def __init__(self, pt_1, pt_2):
        v = pt_2 - pt_1
        v_norm = v / np.linalg.norm(v)
        self.abc = np.zeros(3)
        self.abc[0] = v_norm[1]
        self.abc[1] = -v_norm[0]
        self.abc[2] = -self.abc[0] * pt_1[0] + -self.abc[1] * pt_1[1]

    def setPointAngle(pt, alpha):
        self.abc = np.zeros(3)
        self.abc[0] = np.sin(alpha)
        self.abc[1] = -np.cos(alpha)
        self.abc[2] = -self.abc[0] * pt[0] + -self.abc[1] * pt[1]

    # Predict the y-value based on the x-value
    def predictY(self, x):
        return ((-x * self.abc[0] - self.abc[2]) / self.abc[1]);

    # Predict the x-value based on the y-value
    def predictX(self, y):
        return ((-y * self.abc[1] - self.abc[2]) / self.abc[0]);

        # Calculate the distance from the point pt to the line (shortest distance, perpendicular to the
line)

    def distance(self, pt):
        return (pt[0] * self.abc[0] + pt[1] * self.abc[1] + self.abc[2])

    # The absolute distancen
    def distanceAbs(self, pt):
        return (abs(self.distance(pt)))

    # Project a point to the line
    def projectPoint(self, pt):
        d = self.distance(pt)
        n = self.abc[0:2] * d
        return (pt - n);


# Some functions to read the annotation files:
def getLines(line_file):
    lines = [line.rstrip('\n') for line in open(str(line_file))]
    return (lines)


# Transforms the relateive representation to absolute values with respect to the image
# Output is a tupple with class, top-left point, bottom-right point, center, width and height
def bbToAbsoluteCoordinates(bb, width_tile, height_tile):
    c, x, y, w, h = bb
    x0 = x * width_tile - w * width_tile / 2
    y0 = y * height_tile - h * height_tile / 2
    x1 = x * width_tile + w * width_tile / 2
    y1 = y * height_tile + h * height_tile / 2
    x = x * width_tile
    y = y * height_tile
    w = w * width_tile
    h = h * height_tile
    return ((c, x0, y0, x1, y1, x, y, w, h))


# Get bounding-box info from a text file with space-separated columns.
# 5 columns: 1. class, 2. center_x, 3. center_y, 4. width, 5. height
# The center position and widht and height are relative to the width and height
# of the image. This function uses bbToAbsoluteCoordinates(..) to calculate
# the absolute coordinates and size of the bounding box in pixels
def getBBInfo(bb_file, width_tile, height_tile):
    bb_abs = []
    with open(bb_file, 'rt') as f:
        reader = csv.reader(f, delimiter=' ', quoting=csv.QUOTE_NONNUMERIC)
        for row in reader:
            bb_abs.append(bbToAbsoluteCoordinates(row, width_tile, height_tile))
    return (bb_abs)


# Function to get the column and row number of the tile from the tile_code
def tilecode_to_colrow(tile_code, nr_cols):
    col_num = tile_code % nr_cols
    row_num = tile_code // nr_cols
    return (col_num, row_num)
```

```
# Get line equations ax + by + c = 0 for the given tile
def getLinesForTileABC(lines, tile_code, width_tile, height_tile, dif_x, dif_y):
    col_num, row_num = tilecode_to_colrow(tile_code, 6)

    lines_abc = []
    for line_i, line in enumerate(lines):
        # Get the begin and end point of the line in the original image
        x1, y1, x2, y2 = line.split()
        # Calculate the begin and end point for the modified and tiled image
        minx = dif_x + col_num * width_tile
        miny = dif_y + row_num * height_tile
        x1 = float(x1) - minx
        y1 = float(y1) - miny
        x2 = float(x2) - minx
        y2 = float(y2) - miny

        # Calculate the ax + by + c = 0 standard form of the line
        lines_abc.append(ABCLine(np.array((x1, y1)), np.array((x2, y2))))

    return (lines_abc)

# Functions to evaluate the detection performance:
# Function to calculate the intersection of union of two bounding boxes
def bb_intersection_over_union(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou


# Function to calculate the detection performance:

# First finds the TP, FN and FP. Then inspects FN (in/out row, size)
# Returns the matches, which a list of matching indices of gt and pred: (gt_i, pred_i)
def calcDetectionPerformance(gt_bbs, pred_bbs, lines_abc, iou_th, row_d, small_plant_area):
    # Calculate the IoUs for all combinations of gt_bbs and pred_bbs
    iou_table = np.zeros((len(gt_bbs), len(pred_bbs)))
    for gt_i, gt_bb in enumerate(gt_bbs):
        for pred_i, pred_bb in enumerate(pred_bbs):
            iou_table[gt_i, pred_i] = bb_intersection_over_union(gt_bb[1:5], pred_bb[1:5])

    # Find the matching ground truth and predictions
    matches = []
    while (1):
        # Get the largest value in the iou_table. If above iou_th, log the match and remove gt and pred
from table
        if (len(iou_table) > 0):
            (max_gt_i, max_pred_i) = np.unravel_index(np.argmax(iou_table), iou_table.shape)
            if (iou_table[max_gt_i, max_pred_i] >= iou_th):
                # Log the match
                matches.append((max_gt_i, max_pred_i))
                #a Remove max_gt_i nd max_pred_i from iod_table to prevent double-counting.
                # Set their iou-values with all other instances to 0.0
                iou_table[max_gt_i, :] = 0.0
                iou_table[:, max_pred_i] = 0.0
            else:  # No more matches above the iou_th, so stop
                break
        else:
            break

    # Get the TP, FN and FP
    tp = len(matches)
    fn = len(gt_bbs) - tp
    fp = len(pred_bbs) - tp

    # Analyse the FN to see whether they are in/out-of row and small/large
    fn_in_row_and_small = 0
    fn_in_row_and_large = 0
    fn_out_row_and_small = 0
    fn_out_row_and_large = 0

    detected_gt = [m[0] for m in matches]
    for gt_i, gt_bb in enumerate(gt_bbs):
        if (not (gt_i in detected_gt)):  # If the plant was not detected -> a false negative
            # Get the distance to closest line(row)
            # Sort the lines on distance from the bounding box and get the distance to closest
            sort_lines = sorted(lines_tile_abc, key=lambda obj: obj.distanceAbs(gt_bb[5:7]))
            point_line_d = sort_lines[0].distanceAbs(gt_bb[5:7])
            if (point_line_d <= row_d):
                if (np.prod(gt_bb[7:9]) <= small_plant_area):
                    fn_in_row_and_small += 1
                else:
                    fn_in_row_and_large += 1
            else:
                if (np.prod(gt_bb[7:9]) <= small_plant_area):
                    fn_out_row_and_small += 1
                else:
                    fn_out_row_and_large += 1
    return (
    matches, (tp, fn, fp), (fn_in_row_and_small, fn_in_row_and_large, fn_out_row_and_small,
fn_out_row_and_large))


# Functions to show the results:

# Show an RGB image. Color channels need to be reversed, as OpenCV uses Blue-Green-Red order
instead of RGB
def imshow_rgb(img_bgr):
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    plt.imshow(img_rgb)


def plotLinesABC(img_bgr, lines_abc, col):
    for line_abc in lines_abc:
        if (np.abs(line_abc.abc[1]) > np.abs(line_abc.abc[2])):
            x0 = 0
            x1 = img_bgr.shape[1] - 1
            y0 = int(np.rint(line_abc.predictY(x0)))
            y1 = int(np.rint(line_abc.predictY(x1)))
            cv2.line(img_bgr, (x0, y0), (x1, y1), col, 2)
        else:
            y0 = 0
            y1 = img_bgr.shape[0] - 1
            x0 = int(np.rint(line_abc.predictX(y0)))
            x1 = int(np.rint(line_abc.predictX(y1)))
            cv2.line(img_bgr, (x0, y0), (x1, y1), col, 2)


def plotBB(img_bgr, bbs, col):
    for bb in bbs:
        bb_rnd = [int(np.round(b)) for b in bb]
        cv2.rectangle(img_bgr, (bb_rnd[1], bb_rnd[2]), (bb_rnd[3], bb_rnd[4]), col, 2)


def plotDistanceBBLine(img_bgr, bbs, lines_abc, row_d):
    for bb in bbs:
        # Sort the lines on distance from the bounding box
        sort_lines = sorted(lines_tile_abc, key=lambda obj: obj.distanceAbs(bb[5:7]))
        # For the closest line, get projected point
        proj_pt = sort_lines[0].projectPoint(bb[5:7])
        # And the distance
        point_line_d = sort_lines[0].distanceAbs(bb[5:7])
        if (point_line_d < row_d):  # In row
            col = (255, 255, 255)
        else:
            col = (0, 128, 255)

        # Round the coordinates to integers to plot
        proj_pt_rnd = tuple([int(np.round(x)) for x in proj_pt])
        bb_pt_rnd = tuple([int(np.round(x)) for x in bb[5:7]])

        # Plot the shortest line from bbox to closest line and plot the distance
        cv2.line(img_bgr, bb_pt_rnd, proj_pt_rnd, col, 2)
        txt = ("%1.1f" % point_line_d)
        txt_sz = cv2.getTextSize(txt, 1, 1, 2)
        txt_p1 = (bb_pt_rnd[0], bb_pt_rnd[1] - txt_sz[1])
        txt_p2 = (bb_pt_rnd[0] + txt_sz[0][0], bb_pt_rnd[1] - txt_sz[0][1] - txt_sz[1])
        cv2.rectangle(img_bgr, txt_p1, txt_p2, (255, 255, 255), -1)
        cv2.putText(img_bgr, txt, (bb_pt_rnd[0], bb_pt_rnd[1] - 5), 1, 1, (0, 0, 0), 1)


def plotPerformance(img_bgr, gt_bbs, pred_bbs, lines_tile_abc, performance):
    plotLinesABC(img_bgr, lines_tile_abc, (0, 255, 255))
    # Plot all true plant bboxes and all predicted bboxes
    plotBB(img_bgr, gt_bbs, (0, 0, 0))  # After TP plotting, the remaining FN are black
    plotBB(img_bgr, pred_bbs, (0, 0, 255))  # After TP plotting, the remaining FP are red
    # Plot the true positives (predictions that are correct)
    matches, confusion_mat, fn_analysis = performance
    tp_gt_bbs = []
    tp_pred_bbs = []
    for match in matches:
        tp_gt_bbs.append(gt_bbs[match[1]])
        tp_pred_bbs.append(pred_bbs[match[1]])
    plotBB(img_bgr, tp_gt_bbs, (255, 0, 0))  # Plant that has been detected as blue
    plotBB(img_bgr, tp_pred_bbs, (0, 255, 0))  # Correct detection plotted as green
    plotDistanceBBLine(img_bgr, gt_bbs, lines_tile_abc, row_d)


# Main code:
############################################
######### PARAMETERS AND CONSTANTS #########
############################################
# Three parameters for the evaluation of the detectino algorithm
row_d = 20          # Half-width of a row in pixels
iou_th = 0.25         # Minimum threshold on the IoU score to become a match
small_plant_area = 500 # Threshold for small/large plants

# Parameters about the dimensions of the original and cropped image
original_image_y_dimension = 3000
cropped_image_y_dimension = 2432
dif_y = original_image_y_dimension - cropped_image_y_dimension
original_image_x_dimension = 4000
cropped_image_x_dimension = 3648
dif_x = original_image_x_dimension - cropped_image_x_dimension

# The width and height of the tile images in pixels
width_tile = 608
height_tile = 608

#######################################
######### READING IN THE DATA #########
```

```
########################################
# Path to the data should contain subdirectories images, lines, ground_truth and predicted
path = "path to data folder”
image_nr = 217         # Image to process
tile_id = 11              # Tile to process

# Open the image
img_bgr = cv2.imread(path + ("/images/crop_DJI_%04d_%d.jpg" % (image_nr,tile_id)))

# Get crop rows
line_file = path + ("/lines/DJI_%04d_lines_1.txt" % image_nr)
lines = getLines(line_file)
lines_tile_abc = getLinesForTileABC(lines, tile_id, width_tile, height_tile, dif_x, dif_y)

# Get ground truth annotations
gt_file = path + ("/ground_truth/crop_DJI_%04d_%d.txt" % (image_nr,tile_id))
gt_bbs = getBBInfo(gt_file, width_tile, height_tile)

# Get predicted plants
pred_file = path + ("/predicted/crop_DJI_%04d_%d.txt" % (image_nr,tile_id))
pred_bbs = getBBInfo(pred_file, width_tile, height_tile)

#######################################################
######### EVALUATION OF THE DETECTION METHODS #########
#######################################################
# Calculate prediction performance
performance = calcDetectionPerformance(gt_bbs, pred_bbs, lines_tile_abc, iou_th, row_d,
small_plant_area)
matches, confusion_mat, fn_analysis = performance
tp, fn, fp = confusion_mat
fn_in_row_and_small, fn_in_row_and_large, fn_out_row_and_small, fn_out_row_and_large =
fn_analysis

print("TP: %d, FN: %d, FP: %d" %(tp, fn, fp))

print("Analysis of FNs:")
print("  - in row and small : %d" % fn_in_row_and_small)
print("  - in row and large : %d" % fn_in_row_and_large)
print("  - out row and small: %d" % fn_out_row_and_small)
print("  - out row and large: %d" % fn_out_row_and_large)

####################################
######### SHOW THE RESULTS #########
####################################
# Plot detections, lines, etc
plt.figure()
plotPerformance(img_bgr, gt_bbs, pred_bbs, lines_tile_abc, performance)
figsize(12,12)
imshow_rgb(img_bgr)
```

8.5 Annex V: Script to calculate bounding box angle (original resolution)

```
# Title: Angle Calculator Algorithm
# Author: Sebastian Paolini van Helfteren
# Date: 28/03/2019
# For more information, contact the author (sebastianpvh95@gmail.com)

### Libraries ###
import os
import re
from math import *

### Main script ###
def tilecode_to_colrow(tile_code, nr_cols):
    col_num = tile_code % nr_cols
    row_num = tile_code // nr_cols
    return (col_num, row_num)


def angle_file_maker(annotation_folder, angle_annotations_folder, width_tile, height_tile, x_nadir,
y_nadir,
                     pixel_size, fly_height, x_diff, y_diff):
    for annotation_file in os.listdir(annotation_folder):
        angle_txt = open(angle_annotations_folder + "angle_" + annotation_file, "w")
        lines = [line.rstrip('\n') for line in open(annotation_folder + str(annotation_file))]

        tile_id = re.sub(r'(.*_)', '', annotation_file)
        tile_id = re.sub(r'(.txt)', '', tile_id)
        col_num, row_num = tilecode_to_colrow(int(tile_id), 2)  # Change number of columns if needed!

        for box in lines:
            label, x, y, w, h = box.split()
            x_abs = (float(x) * width_tile) + (col_num * 608) + x_diff
            y_abs = (float(y) * height_tile) + (row_num * 608) + y_diff
            a2 = (abs(x_abs - x_nadir) * pixel_size) ** 2
            b2 = (abs(y_abs - y_nadir) * pixel_size) ** 2
            distance = sqrt(a2 + b2)
            angle = degrees(atan(distance / fly_height))

            angle_txt.write(
                str(label) + " " + str(x) + " " + str(y) + " " + str(w) + " " + str(h) + " " + str(angle) + "\n")
        angle_txt.close()


# Input and output paths
annotation_folder_gt = “path to ground truth bboxes folder”
annotation_folder_pred = " path to predicted bboxes folder”
angle_annotations_folder_gt = "path to new angle ground truth bbox folder”
angle_annotations_folder_pred = “path to new angle predicted bbox folder”

# Image dimensions, change where needed
```

```
orig_x = 4000 # Change if necessary. Original = 4000, 0.5 = 2000, 0.31 = 1240
orig_y = 3000 # Change if necessary. Original = 3000, 0.5 = 1500, 0.31 = 930
crop_x = 3648 # Change if necessary. Original = 3648, 0.5 = 1824, 0.31 = 1216
crop_y = 2432 # Change if necessary. Original = 2432, 0.5 = 1216, 0.31 = 608
width_tile = 608
height_tile = 608

# Needed for angle_file_maker(), change where needed
x_nadir = orig_x/2
y_nadir = orig_y/2
x_diff = orig_x - crop_x
y_diff = orig_y - crop_y
resize_factor = 0.31
original_pixel_size = 0.004  # In meters
pixel_size = original_pixel_size / resize_factor
fly_height = 10

angle_file_maker(annotation_folder_gt, angle_annotations_folder_gt, width_tile, height_tile,
                 x_nadir, y_nadir, pixel_size, fly_height, x_diff, y_diff)
angle_file_maker(annotation_folder_pred, angle_annotations_folder_pred, width_tile, height_tile,
                 x_nadir, y_nadir, pixel_size, fly_height, x_diff, y_diff)
```

8.6 Annex VI: Script to test angle dependent performance

```
# Title: Angle Dependent Performance Algorithm
# Author: Sebastian Paolini van Helfteren & Gert Kootstra
# Date: 30/03/2019
# For more information, contact the author (sebastianpvh95@gmail.com)

### Libraries ###
import cv2
from matplotlib import pyplot as plt
import numpy as np
import csv
import os
import re

# Class to represent lines in the standard form (ax + by + c = 0)
# and to perform some calculations with them
class ABCLine:

    def __init__(self):
        self.abc = np.zeros(3)

    def __init__(self, pt_1, pt_2):
        v = pt_2 - pt_1
        v_norm = v / np.linalg.norm(v)
        self.abc = np.zeros(3)
        self.abc[0] = v_norm[1]
        self.abc[1] = -v_norm[0]
        self.abc[2] = -self.abc[0] * pt_1[0] + -self.abc[1] * pt_1[1]

    def setPointAngle(pt, alpha):
        self.abc = np.zeros(3)
        self.abc[0] = np.sin(alpha)
        self.abc[1] = -np.cos(alpha)
        self.abc[2] = -self.abc[0] * pt[0] + -self.abc[1] * pt[1]

    # Predict the y-value based on the x-value
    def predictY(self, x):
        return ((-x * self.abc[0] - self.abc[2]) / self.abc[1])

    # Predict the x-value based on the y-value
    def predictX(self, y):
        return ((-y * self.abc[1] - self.abc[2]) / self.abc[0])

        # Calculate the distance from the point pt to the line (shortest distance, perpendicular to the
line)

    def distance(self, pt):
        return (pt[0] * self.abc[0] + pt[1] * self.abc[1] + self.abc[2])

    # The absolute distancen
    def distanceAbs(self, pt):
        return (abs(self.distance(pt)))

    # Project a point to the line
    def projectPoint(self, pt):
        d = self.distance(pt)
        n = self.abc[0:2] * d
        return (pt - n)


# Some functions to read the annotation files:
def getLines(line_file):
    lines = [line.rstrip('\n') for line in open(str(line_file))]
    return (lines)


# Transforms the relateive representation to absolute values with respect to the image
# Output is a tupple with class, top-left point, bottom-right point, center, width and height
def bbToAbsoluteCoordinates(bb, width_tile, height_tile):
    c, x, y, w, h, a = bb
    x0 = x * width_tile - w * width_tile / 2
    y0 = y * height_tile - h * height_tile / 2
    x1 = x * width_tile + w * width_tile / 2
    y1 = y * height_tile + h * height_tile / 2
    x = x * width_tile
    y = y * height_tile
    w = w * width_tile
    h = h * height_tile
    return ((c, x0, y0, x1, y1, x, y, w, h, a))


# Get bounding-box info from a text file with space-separated columns.
# 5 columns: 1. class, 2. center_x, 3. center_y, 4. width, 5. height
# The center position and widht and height are relative to the width and height
# of the image. This function uses bbToAbsoluteCoordinates(..) to calculate
# the absolute coordinates and size of the bounding box in pixels
def getBBInfo(bb_file, width_tile, height_tile):
    bb_abs = []
    with open(bb_file, 'rt') as f:
        reader = csv.reader(f, delimiter=' ', quoting=csv.QUOTE_NONNUMERIC)
        for row in reader:
            bb_abs.append(bbToAbsoluteCoordinates(row, width_tile, height_tile))
    return (bb_abs)

# Function to get the column and row number of the tile from the tile_code
def tilecode_to_colrow(tile_code, nr_cols):
    col_num = tile_code % nr_cols
    row_num = tile_code // nr_cols
    return (col_num, row_num)


# Get line equations ax + by + c = 0 for the given tile
def getLinesForTileABC(lines, tile_code, width_tile, height_tile, dif_x, dif_y):
    col_num, row_num = tilecode_to_colrow(tile_code, 6)

    lines_abc = []
    for line_i, line in enumerate(lines):
        # Get the begin and end point of the line in the original image
        x1, y1, x2, y2 = line.split()
        # Calculate the begin and end point for the modified and tiled image
        minx = dif_x + col_num * width_tile
        miny = dif_y + row_num * height_tile
        x1 = float(x1) - minx
        y1 = float(y1) - miny
        x2 = float(x2) - minx
        y2 = float(y2) - miny

        # Calculate the ax + by + c = 0 standard form of the line
        lines_abc.append(ABCLine(np.array((x1, y1)), np.array((x2, y2))))

    return (lines_abc)

# Functions to evaluate the detection performance:
# Function to calculate the intersection of union of two bounding boxes
def bb_intersection_over_union(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou


# Function to calculate the detection performance:

# First finds the TP, FN and FP. Then inspects FN (in/out row, size)
# Returns the matches, which a list of matching indices of gt and pred: (gt_i, pred_i)
def calcDetectionPerformance(gt_bbs, pred_bbs, lines_abc, iou_th, row_d, small_plant_area):
    # Calculate the IoUs for all combinations of gt_bbs and pred_bbs
    iou_table = np.zeros((len(gt_bbs), len(pred_bbs)))
    for gt_i, gt_bb in enumerate(gt_bbs):
        for pred_i, pred_bb in enumerate(pred_bbs):
            iou_table[gt_i, pred_i] = bb_intersection_over_union(gt_bb[1:5], pred_bb[1:5])
    # Find the matching ground truth and predictions
    matches = []
    while (1):
        # Get the largest value in the iou_table. If above iou_th, log the match and remove gt and pred
from table
        if (len(iou_table) > 0):
            try:
                (max_gt_i, max_pred_i) = np.unravel_index(np.argmax(iou_table), iou_table.shape)
            except ValueError:
                break
            if (iou_table[max_gt_i, max_pred_i] >= iou_th):
                # Log the match
                matches.append((max_gt_i, max_pred_i))
                # Remove max_gt_i nd max_pred_i from iod_table to prevent double-counting.
                # Set their iou-values with all other instances to 0.0
                iou_table[max_gt_i, :] = 0.0
                iou_table[:, max_pred_i] = 0.0
            else:  # No more matches above the iou_th, so stop
                break
        else:
            break

    # Get the TP, FN and FP
    tp = len(matches)
    fn = len(gt_bbs) - tp
    fp = len(pred_bbs) - tp

    # Analyse the FN to see whether they are in/out-of row and small/large
    fn_in_row_and_small = 0
    fn_in_row_and_large = 0
    fn_out_row_and_small = 0
    fn_out_row_and_large = 0

    detected_gt = [m[0] for m in matches]
    for gt_i, gt_bb in enumerate(gt_bbs):
        if (not (gt_i in detected_gt)):  # If the plant was not detected -> a false negative
            # Get the distance to closest line(row)
            # Sort the lines on distance from the bounding box and get the distance to closest
            sort_lines = sorted(lines_tile_abc, key=lambda obj: obj.distanceAbs(gt_bb[5:7]))
            point_line_d = sort_lines[0].distanceAbs(gt_bb[5:7])
            if (point_line_d <= row_d):
                if (np.prod(gt_bb[7:9]) <= small_plant_area):
                    fn_in_row_and_small += 1
                else:
                    fn_in_row_and_large += 1
            else:
                if (np.prod(gt_bb[7:9]) <= small_plant_area):
                    fn_out_row_and_small += 1
                else:
                    fn_out_row_and_large += 1
    return (
    matches, (tp, fn, fp), (fn_in_row_and_small, fn_in_row_and_large, fn_out_row_and_small, fn_out_row_and_large))


# Functions to show the results:

# Show an RGB image. Color channels need to be reversed, as OpenCV uses Blue-Green-Red order instead of RGB
def imshow_rgb(img_bgr):
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    plt.imshow(img_rgb)


def plotLinesABC(img_bgr, lines_abc, col):
    for line_abc in lines_abc:
        if (np.abs(line_abc.abc[1]) > np.abs(line_abc.abc[2])):
            x0 = 0
            x1 = img_bgr.shape[1] - 1
            y0 = int(np.rint(line_abc.predictY(x0)))
            y1 = int(np.rint(line_abc.predictY(x1)))
            cv2.line(img_bgr, (x0, y0), (x1, y1), col, 2)
        else:
            y0 = 0
            y1 = img_bgr.shape[0] - 1
            x0 = int(np.rint(line_abc.predictX(y0)))
            x1 = int(np.rint(line_abc.predictX(y1)))
            cv2.line(img_bgr, (x0, y0), (x1, y1), col, 2)


def plotBB(img_bgr, bbs, col):
    for bb in bbs:
        bb_rnd = [int(np.round(b)) for b in bb]
        cv2.rectangle(img_bgr, (bb_rnd[1], bb_rnd[2]), (bb_rnd[3], bb_rnd[4]), col, 2)


def plotDistanceBBLine(img_bgr, bbs, lines_abc, row_d):
    for bb in bbs:
        # Sort the lines on distance from the bounding box
        sort_lines = sorted(lines_tile_abc, key=lambda obj: obj.distanceAbs(bb[5:7]))
        # For the closest line, get projected point
        proj_pt = sort_lines[0].projectPoint(bb[5:7])
        # And the distance
        point_line_d = sort_lines[0].distanceAbs(bb[5:7])
        if (point_line_d < row_d):  # In row
            col = (255, 255, 255)
        else:
            col = (0, 128, 255)

        # Round the coordinates to integers to plot
        proj_pt_rnd = tuple([int(np.round(x)) for x in proj_pt])
        bb_pt_rnd = tuple([int(np.round(x)) for x in bb[5:7]])

        # Plot the shortest line from bbox to closest line and plot the distance
        cv2.line(img_bgr, bb_pt_rnd, proj_pt_rnd, col, 2)
        txt = ("%1.1f" % point_line_d)
        txt_sz = cv2.getTextSize(txt, 1, 1, 2)
        txt_p1 = (bb_pt_rnd[0], bb_pt_rnd[1] - txt_sz[1])
        txt_p2 = (bb_pt_rnd[0] + txt_sz[0][0], bb_pt_rnd[1] - txt_sz[0][1] - txt_sz[1])
        cv2.rectangle(img_bgr, txt_p1, txt_p2, (255, 255, 255), -1)
        cv2.putText(img_bgr, txt, (bb_pt_rnd[0], bb_pt_rnd[1] - 5), 1, 1, (0, 0, 0), 1)


def plotPerformance(img_bgr, gt_bbs, pred_bbs, lines_tile_abc, performance):
    plotLinesABC(img_bgr, lines_tile_abc, (0, 255, 255))
    # Plot all true plant bboxes and all predicted bboxes
    plotBB(img_bgr, gt_bbs, (0, 0, 0))  # After TP plotting, the remaining FN are black
    plotBB(img_bgr, pred_bbs, (0, 0, 255))  # After TP plotting, the remaining FP are red
    # Plot the true positives (predictions that are correct)
    matches, confusion_mat, fn_analysis = performance
    tp_gt_bbs = []
    tp_pred_bbs = []
    for match in matches:
        tp_gt_bbs.append(gt_bbs[match[1]])
        tp_pred_bbs.append(pred_bbs[match[1]])
    plotBB(img_bgr, tp_gt_bbs, (255, 0, 0))  # Plant that has been detected as blue
    plotBB(img_bgr, tp_pred_bbs, (0, 255, 0))  # Correct detection plotted as green
    plotDistanceBBLine(img_bgr, gt_bbs, lines_tile_abc, row_d)


# Main code:

#############################################
######### PARAMETERS AND CONSTANTS #########
#############################################
# Three parameters for the evaluation of the detecting algorithm
row_d = 20          # Half-width of a row in pixels
iou_th = 0.25        # Minimum threshold on the IoU score to become a match
small_plant_area = 500 # Threshold for small/large plants

# Parameters about the dimensions of the original and cropped image
original_image_y_dimension = 3000
cropped_image_y_dimension = 2432
dif_y = original_image_y_dimension - cropped_image_y_dimension
original_image_x_dimension = 4000
cropped_image_x_dimension = 3648
dif_x = original_image_x_dimension - cropped_image_x_dimension

# The width and height of the tile images in pixels
width_tile = 608
height_tile = 608

# Total number of rows and columns
```

```
img_row_num = int(cropped_image_y_dimension / height_tile)
img_col_num = int(cropped_image_x_dimension / width_tile)

total_tile_num = list(range(img_row_num*img_col_num))

#######################################
######### READING IN THE DATA #########
#######################################
# Path to the data should contain subdirectories images, lines, ground_truth and predicted
path = “path to data to be processed”
tp_count = 0
fn_count = 0
fp_count = 0

fn_in_row_and_small_count = 0
fn_in_row_and_large_count = 0
fn_out_row_and_small_count = 0
fn_out_row_and_large_count = 0

lower_bound = 0
upper_bound = 45

for lines_file in os.listdir(path+"/lines"):
    image_nr_from_line = re.sub(r'(DJI_)', '', lines_file)
    image_nr_from_line = re.sub(r'(_lines_1.txt)', '', image_nr_from_line)
    image_nr = int(image_nr_from_line)   # Image to process
    # image_nr = 202

    for i in total_tile_num:
        tile_id = i               # Tile to process
        img_bgr = cv2.imread(path + ("/images/crop_DJI_%04d_%d.jpg" % (image_nr,tile_id)))

        # Get crop rows
        line_file = path + ("/lines/DJI_%04d_lines_1.txt" % image_nr)
        lines = getLines(line_file)
        lines_tile_abc = getLinesForTileABC(lines, tile_id, width_tile, height_tile, dif_x, dif_y)

        # Get ground truth annotations
        gt_file = path + ("/gt_angle/angle_crop_DJI_%04d_%d.txt" % (image_nr,tile_id))
        gt_bbs = getBBInfo(gt_file, width_tile, height_tile)
        gt_bbs_angle = []
        for bb_gt in gt_bbs:
            if lower_bound < bb_gt[9] <= upper_bound:
                gt_bbs_angle.append(bb_gt)

        # Get predicted plants
        pred_file = path + ("/pred_angle/angle_crop_DJI_%04d_%d.txt" % (image_nr,tile_id))
        pred_bbs = getBBInfo(pred_file, width_tile, height_tile)
        pred_bbs_angle = []
        for bb_pred in pred_bbs:
            if lower_bound < bb_pred[9] <= upper_bound:
                pred_bbs_angle.append(bb_pred)
```

```
##########################################################
######### EVALUATION OF THE DETECTION METHODS #########
##########################################################
    # Calculate prediction performance
    performance = calcDetectionPerformance(gt_bbs_angle, pred_bbs_angle, lines_tile_abc, iou_th,
row_d, small_plant_area)
    matches, confusion_mat, fn_analysis = performance
    tp, fn, fp = confusion_mat
    fn_in_row_and_small, fn_in_row_and_large, fn_out_row_and_small, fn_out_row_and_large =
fn_analysis

    tp_count += tp
    fn_count += fn
    fp_count += fp

    fn_in_row_and_small_count += fn_in_row_and_small
    fn_in_row_and_large_count += fn_in_row_and_large
    fn_out_row_and_small_count += fn_out_row_and_small
    fn_out_row_and_large_count += fn_out_row_and_large

print(path)
print("TP: %d, FN: %d, FP: %d" %(tp_count, fn_count, fp_count))

print("Analysis of FNs:")
print("  - in row and small : %d" % fn_in_row_and_small_count)
print("  - in row and large : %d" % fn_in_row_and_large_count)
print("  - out row and small: %d" % fn_out_row_and_small_count)
print("  - out row and large: %d" % fn_out_row_and_large_count)
```

8.7 Annex VII: Error type relative to total number of ground truth labels

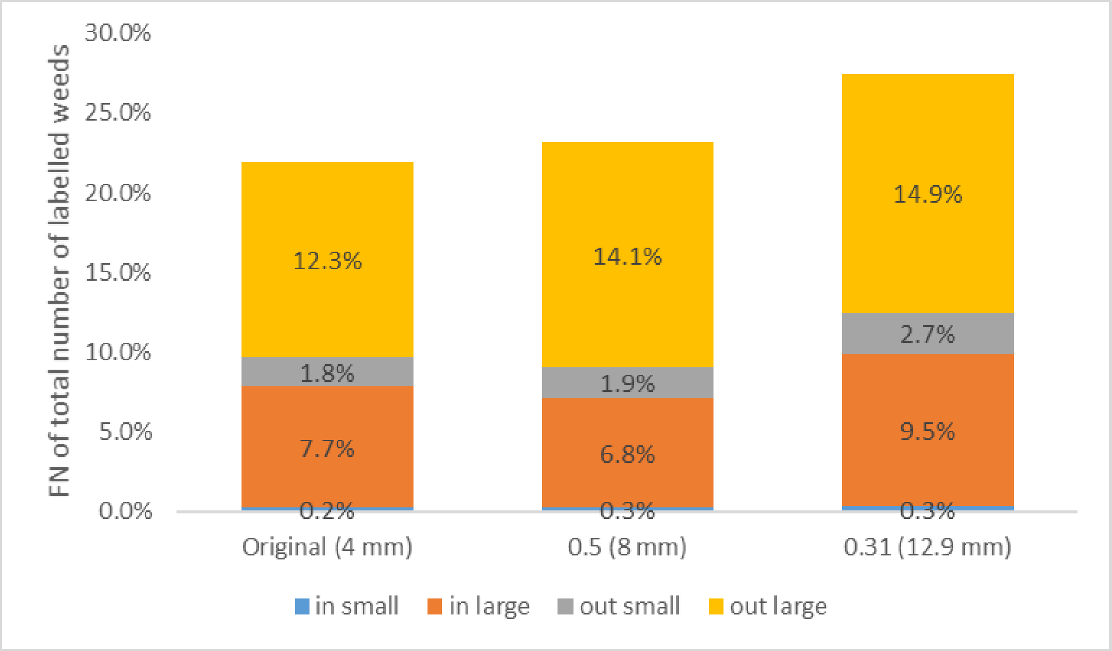


Figure 42: Percentage of error type relative to total number of labelled weeds per resolution.

8.8 Annex VIII: Angle residual plot for all tested resolutions

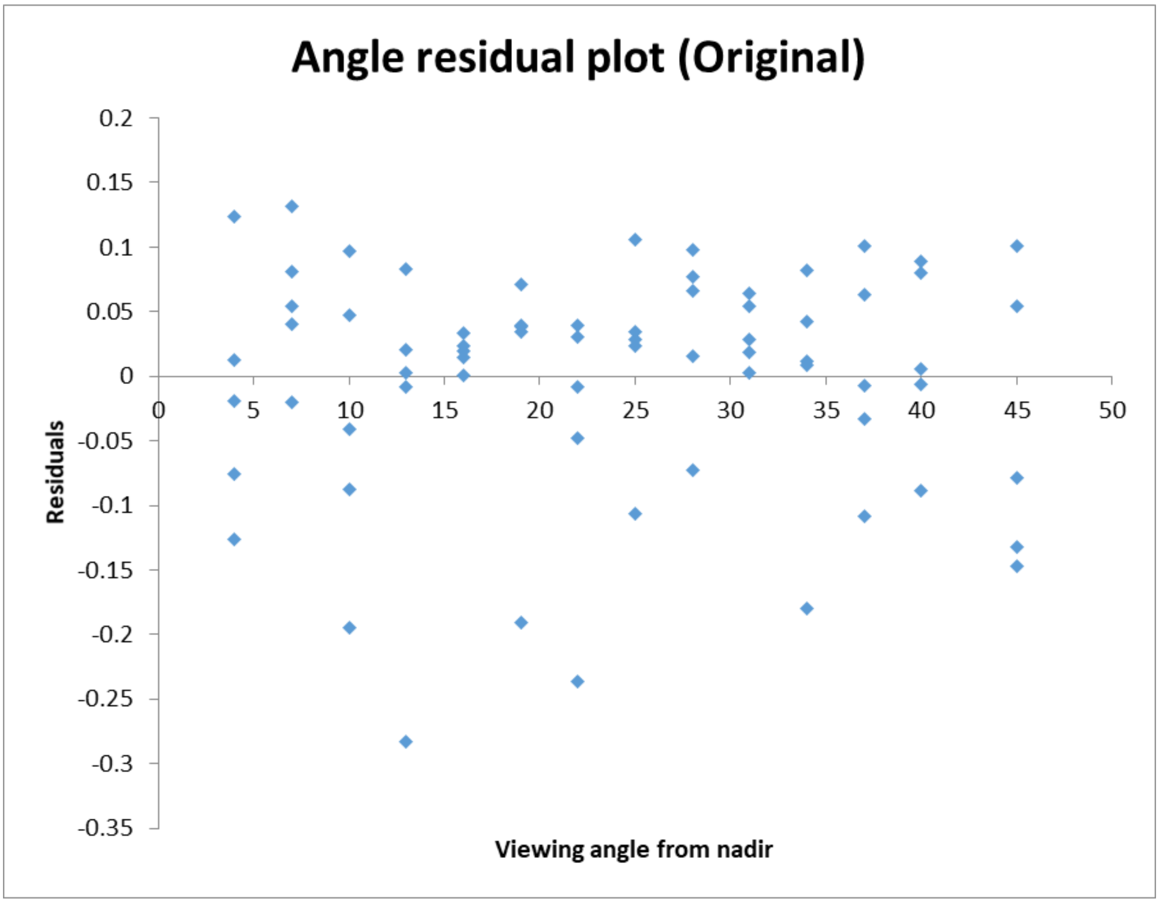


Figure 43: Angle residual plot for the original resolution tiles.

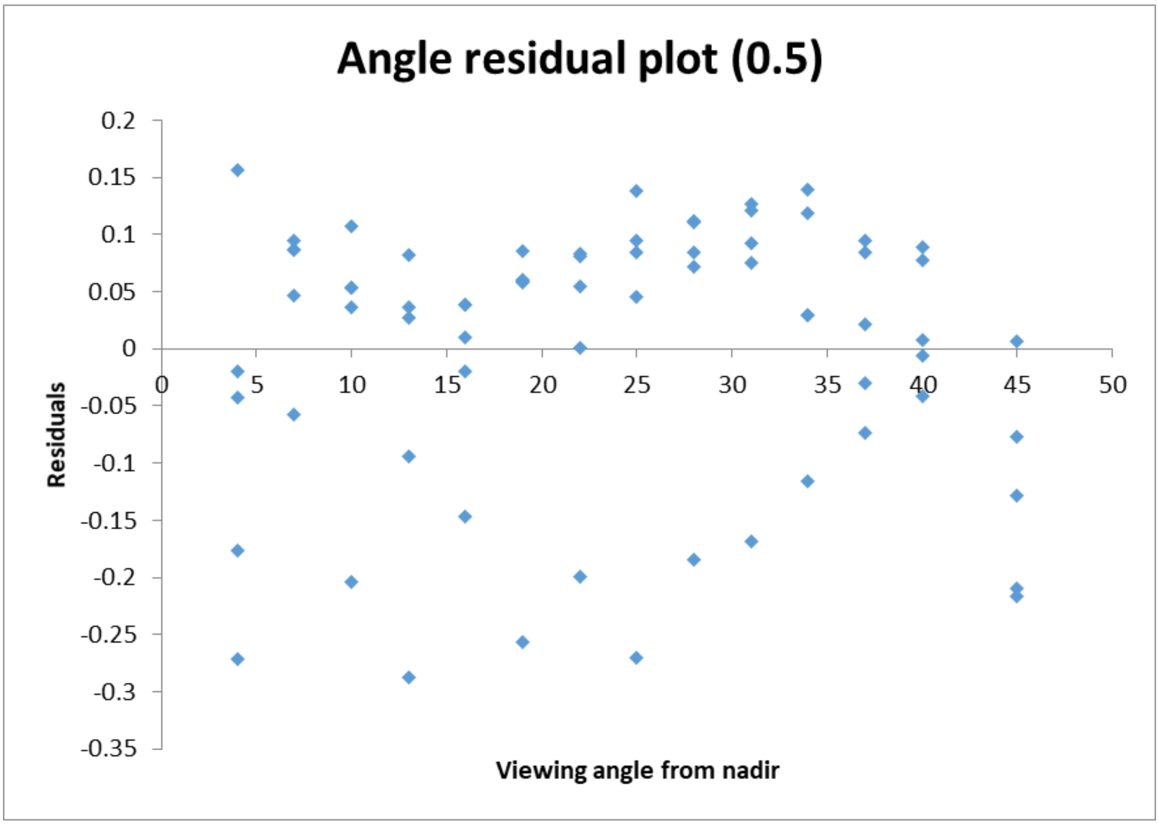


Figure 44: Angle residual plot for the 0.5 dataset tiles.

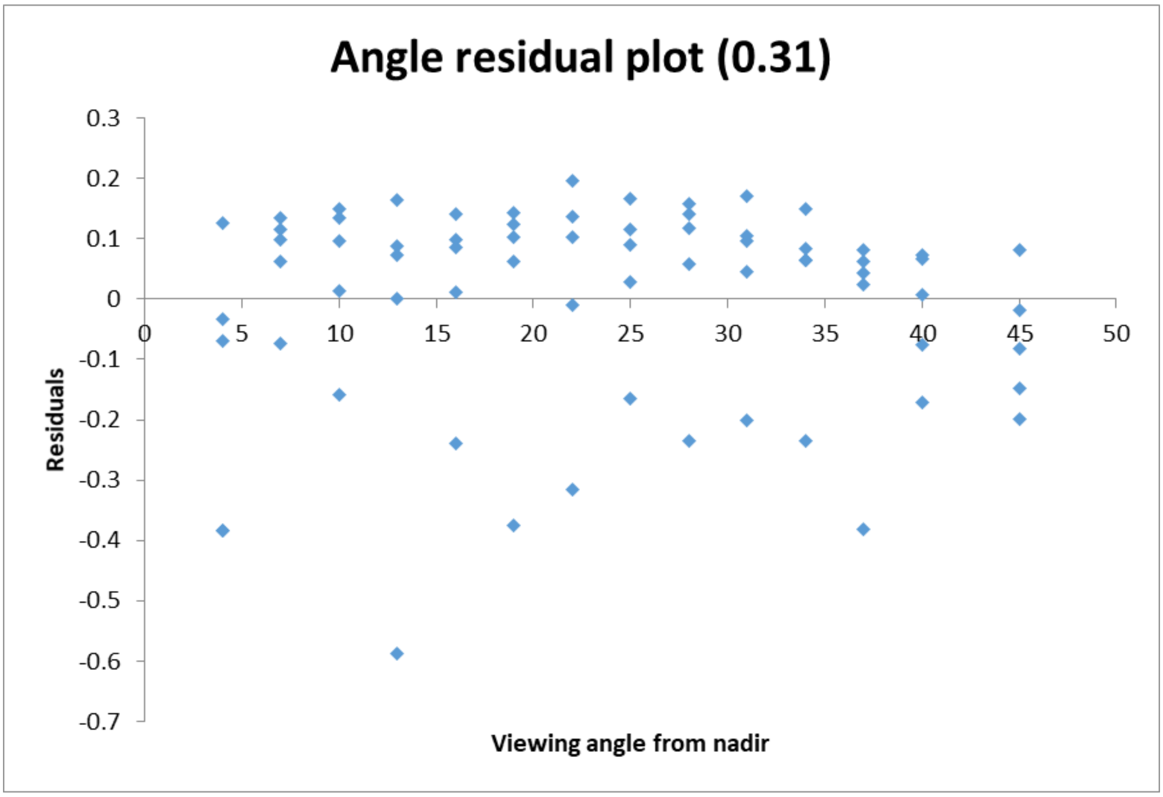


Figure 45: Angle residual plot for the 0.31 dataset tiles.

8.9 Annex IX: Data and results overview

Data used in this research can be requested by contacting the author. An overview of the data and documents containing results follows:

1. Original images and annotations; five folds.
2. Original tiles, aground truth annotations, predicted annotations and crop lines; five folds.
3. Orthomosaic tiles with corresponding annotations.
4. Error analysis excel sheet; used for the *performance per resolution test*.
5. Angle analysis excel sheet; used for the *viewing angle test*.
6. Orthomosaic analysis excel sheet; used for the *orthomosaic test*.
7. Weed detection evaluation Jupyter Notebook; used for performance per resolution test.

Lower resolution images can be created using the original images in combination with *8.3 Annex III*, but are also available if requested.