Pierre Husebø Chambenoit

Automatic strawberry detection

A comparative study of traditional computer vision and deep learning for detection of strawberries.

Master's thesis in Cybernetics and Robotics

Supervisor: Pål Johan From

June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Pierre Husebø Chambenoit

Automatic strawberry detection

A comparative study of traditional computer vision and deep learning for detection of strawberries.

Master's thesis in Cybernetics and Robotics
Supervisor: Pål Johan From
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Norwegian University of
Science and Technology

Summary

This thesis explores the world of computer vision in order to detect strawberries in images. Three different methods of object detection are presented, implemented and compared in this thesis. The first one is based on traditional computer vision and uses primarily a segmentation algorithm to detect strawberries. The two other ones are based on a deep learning framework that uses a single pass of a neural network to detect strawberries in an image. The question this thesis intends to answer is; which of the two methods, traditional computer vision and deep learning, is the most suited to detect strawberries in images?

The theory behind each method is first explained to understand how they work and can be used. Then, the implementation is described and the results are presented. Finally, the methods and results are discussed individually and compared in order to conclude and discuss what further work can be done by Saga Robotics [1]. The code implemented in this thesis can be found in Appendix A. Appendix B contains links to three videos showcasing the results of each method implemented in this thesis.

The best results obtained in this thesis are a mAP of 87.1% and an average segmentation accuracy of 86.6% for YOLOv3-strawberry, the YOLOv3 object detection system [2] trained specifically on strawberries. The thesis concludes that the deep learning methods implemented are more suited to detect strawberries in images than the traditional computer vision method implemented.

Sammendrag

Denne oppgaven undersøker hvordan datasyn kan brukes til å detektere jordbær i bilder. Tre forskjellige datasynsalgoritmer legges frem, implementeres og sammenlignes med hverandre i denne oppgaven. Den første er basert på tradisjonelt datasyn og bruker hovedsakelig en segmenteringsalgoritme for å detektere jordbær. De to andre er basert på et rammeverk for dyp læring som bruker dype nett for å detektere jordbær. Spørsmålet denne oppgaven prøver å svare på er som følger; hvilke av de to metodene, tradisjonelt datasyn og dyp læring, egner seg best til å detektere jordbær i bilder?

Teorien bak hver metode forklares først for å forstå hvordan de fungerer og kan brukes. Så blir implementasjonen av hver metode beskrevet og resultatene blir lagt frem. Til slutt diskuteres metodene og resultatene for å kunne konkludere og diskutere hva Saga Robotics kan gjøre videre for å forbedre resultatene. Koden implementert i denne oppgaven finnes i Appendix A. Appendix B inneholder lenker til tre videoer som viser resultatet av hver metode implementert i denne oppgaven.

De beste resultatene i denne oppgaven er en mAP på 87.1% og en gjennomsnitlig segmenteringsnøyaktighet på 86.6% for YOLOv3-strawberry, en versjon av YOLOv3 [2] trent spesifikt på jordbær. Denne oppgaven konkluderer med at metodene basert på dyp læring som ble implementert egner seg bedre til å detektere jordbær enn metoden basert på tradisjonelt datasyn som ble implementert.

Preface

This thesis is made possible by the computer and workplace provided by NTNU [3]. The tools and libraries used in this thesis are Python [4], OpenCV [5], NumPy [6], Google Colab [7] and Darknet [8].

I wish to thank Pål Johan From from Saga Robotics [1] for supervising this project and Anastasios Lekkas for facilitating the project at NTNU. I also wish to thank Yuanyue Ge and Ya Xiong from Saga Robotics for providing images and videos of strawberries that are the backbone of this thesis. At last, I wish to thank Astrid Avdem for motivating me throughout this semester.

Contents

Summary **i**

Sammendrag **ii**

Preface **iii**

Table of Contents **vi**

1 Introduction **1**
1.1 Problem description . . . 1
1.2 Background and motivation . . . 1
1.3 Objective and method . . . 2
1.4 Outline . . . 2

2 Theory **3**
2.1 Segmentation with traditional CV . . . 3
2.1.1 HSV-segmentation . . . 3
2.1.2 Canny edge detection . . . 5
2.1.3 Contour detection . . . 10
2.2 Detection with deep learning . . . 12
2.2.1 Classification with CNNs . . . 12
2.2.2 Detection with YOLOv3 . . . 17
2.3 Evaluation metrics . . . 20
2.3.1 Detection . . . 20
2.3.2 Segmentation . . . 21
2.3.3 Frames per second . . . 22

3 Implementation **23**
3.1 Tools and libraries . . . 23
3.1.1 Python . . . 23
3.1.2 OpenCV . . . 23

3.1.3 NumPy . . . 24
3.1.4 Darknet . . . 24
3.1.5 Google Colab . . . 24
3.2 Source code . . . 25
3.2.1 Detection . . . 25
3.2.2 Segmentation . . . 26
3.2.3 Labeling . . . 27

4 Results **29**
4.1 Detection examples . . . 29
4.1.1 Traditional computer vision . . . 29
4.1.2 YOLOv3-tiny-strawberry . . . 30
4.1.3 YOLOv3-strawberry . . . 31
4.2 Evaluation metrics . . . 32
4.2.1 Detection . . . 32
4.2.2 Segmentation . . . 33
4.2.3 Frames per second . . . 33

5 Discussion **35**
5.1 Traditional computer vision . . . 35
5.1.1 Python vs. C++ . . . 35
5.1.2 mAP calculation . . . 35
5.1.3 Limitations of method . . . 36
5.2 Deep learning . . . 36
5.2.1 YOLOv3 vs. YOLOv3-tiny . . . 36
5.2.2 Testing on video . . . 37
5.2.3 Limitations of dataset . . . 37
5.3 Traditional CV vs. deep learning . . . 38
5.3.1 Detection metrics . . . 38
5.3.2 Segmentation metrics . . . 38
5.3.3 Frames per second . . . 39
5.3.4 Similarities . . . 39
5.3.5 Differences . . . 39

6 Conclusion **41**
6.1 Conclusion . . . 41
6.2 Further work . . . 42

Bibliography **43**

List of Tables **45**

List of Figures **48**

Abbreviations **49**

Appendix **51**

Chapter 1

Introduction

1.1 Problem description

Saga Robotics [1] in Ås are working on automating the picking of strawberries in greenhouses. They already have a robot equipped with a picking tool, but the detection software can not detect strawberries with sufficient precision. The robot has to be able to detect the position and depth of the ripe strawberries in order to pick them. This Master's thesis explores the use of traditional computer vision versus the use of deep learning to detect the ripe strawberries.

Which of the two methods, traditional computer vision and deep learning, is the most suited to detect strawberries in images?

1.2 Background and motivation

Computer vision has been around since the late 1960's, and has evolved with the evolution of computers. It started out with algorithms to recognize edges and lines, optical flow and motion estimation. Nowadays, computer vision with neural networks can recognize the most complex shapes and one single neural network may recognize up to thousands of different objects [9]! This thesis handles simple red shapes that stand out from the green and dark background. It is interesting to see if traditional computer vision can stay relevant in this new age, or if the newer convolutional neural networks outperform the old methods.

The personal motivation to work with this project comes from my fascination for the agriculture. For several summers I worked at a farm where I saw how overworked farmers are. In addition, there are fewer and fewer farmers in Norway. The agriculture is one of the main industries that lay behind on automation. The automation of the agriculture has a big potential and may solve the problem of overworked and diminishing number of farmers. I believe it is only a question of time before the agriculture becomes mostly automated, and I wish to contribute to that change.

1.3 Objective and method

The main objectives of this thesis are to detect strawberries in images and to determine which of traditional computer vision and deep learning is most suited for the task. To accomplish these objectives, the following methods have been implemented:

- Implementing an algorithm based on traditional computer vision to segment strawberries in images.
- Implement a labeling script that utilizes the segmentation algorithm to label images with bounding boxes.
- Implementing YOLOv3 [2] and YOLOv3-tiny with the labeled images to detect strawberries in images.
- Comparing the methods implemented in this thesis.

1.4 Outline

In addition to this first chapter, this thesis contains five other chapters, each described below:

- Chapter 2 explains the theory behind the methods used in this thesis. The first section explains the traditional computer vision methods in details, whereas the second section explains the neural networks used in this thesis in a more generalized way.
- Chapter 3 presents in its first section the different tools and libraries used in this thesis. In its second section, it explains how these tools have been used to implement the methods and where the code can be found.
- Chapter 4 presents the results obtained by the different detection methods by showing example images in its first section and comparison metrics in its second section.
- Chapter 5 discusses the results presented in chapter 4 and compares the traditional computer vision against the neural networks. It also discusses certain limitations of each method.
- Chapter 6 concludes this thesis and discusses further work that can be done by Saga Robotics to improve the results.

Chapter 2

Theory

This chapter presents the methods used in this thesis and goes through their theory. First, the methods of traditional computer vision, or CV, are presented. Then, the newer methods of deep learning are presented. Finally, evaluation metrics that can be used to compare the methods are introduced.

2.1 Segmentation with traditional CV

This section presents methods that can be used to segment strawberries in images. First, the RGB and HSV color models are introduced and used in combination with a threshold segmentation to segment strawberries from the background. Then the Canny edge detector [10] is presented to find edges between strawberries in clusters. Finally, a contour detection algorithm [11] is introduced to separate strawberries in clusters.

Figure 2.1: Strawberry before thresholding (left), after RGB-thresholding (middle) and after HSV-thresholding (right). The thresholding is done without filtering.

2.1.1 HSV-segmentation

In greenhouses that cultivate strawberries, the color palette is mostly made up of green and blue from the plants and the background, but there is also some red from the fruits. This

section will show how segmentation can separate the red fruits from the background, as is shown in Figure (2.1).

Threshold segmentation

Segmentation consists in partitioning an image into segments and to classify the segments into one or several classes. In this case, the strawberries must be segmented from the background. Threshold segmentation consists in setting an upper and a lower threshold T_u and T_l, then for every pixel intensity $I_{i,j}$ in the image, the following operation is performed:

$$I_{i,j} = \begin{cases} I_{i,j}, & \text{if } T_l \leq I_{i,j} \leq T_u \\ 0, & \text{otherwise} \end{cases}$$

RGB- to HSV-conversion

The RGB color model consists of three values, red, blue and green. These values can be added together to create most colors of the visible spectrum. Because of this additive property, computers use the RGB color model to handle images; a pixel in an RGB image can be represented on a LED screen by changing the intensities of the corresponding red, blue and green LED's on the screen.

The RGB color model is good at representing colors on a screen, but it is not intuitive to the human mind. Intuitively, high values of red should only be red, but as is seen in Figure (2.2), high values of red might be red as well as they might be white. A threshold segmentation is the equivalent of cutting out a smaller cube from the cube in Figure (2.2). If the lower left quadrant of the left side of the cube was to be segmented, the white would not be segmented, but there would still be some orange and pink since the segmentation is linear. A better segmentation would be to segment a sphere with center in the most red corner, or use polar coordinates instead of Cartesian coordinates. This is what the HSV color model intends to do.

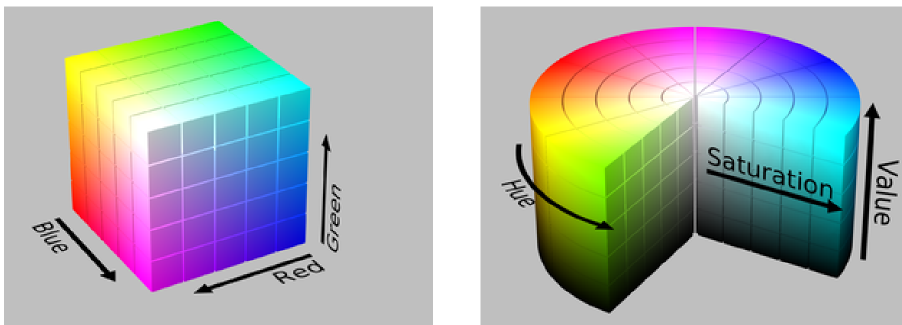


Figure 2.2: The RGB color spectrum in Cartesian coordinates (left) and the HSV color spectrum in cylindrical coordinates (right) [12].

The HSV color model consists of three values, hue, saturation and value, as seen on Figure (2.2). This is a more intuitive model than the RGB color model since it separates

the intensity, or brightness, from the color. The HSV color model was first developed by computer graphics researchers in the 1970's [13] to mimic their perception of color. The hue decides the color, the saturation decides how light the color is and the value decides how dark the color is. Contrary to the RGB color model, only two of the channels, saturation and value, are changed by different lightning conditions. This makes a segmentation based on the HSV color model more adaptable to changes in lightning conditions. The transformation from RGB to HSV is done with the following equations:

$$
\begin{aligned}
&MAX = max(R, G, B) \\
&MIN = min(R, G, B) \\
&H = \begin{cases} 0, & \text{if } MAX = MIN \\ 60^\circ \cdot (0 + \frac{G-B}{MAX-MIN}), & \text{if } MAX = R \\ 60^\circ \cdot (0 + \frac{B-R}{MAX-MIN}), & \text{if } MAX = G \\ 60^\circ \cdot (0 + \frac{R-G}{MAX-MIN}), & \text{if } MAX = B \end{cases} \\
&H = H + 360^\circ \text{ if } H < 0^\circ \\
&S = \begin{cases} 0, & \text{if } MAX = 0 \\ \frac{MAX-MIN}{MAX}, & \text{otherwise} \end{cases} \\
&V = MAX
\end{aligned}
$$

Threshold determination

When using threshold segmentation, a threshold must be set that segments all the strawberries and only the strawberries. The threshold can either be constant for all images, or vary for every image with adaptive thresholding.

In the RGB color model, thresholds should be set by adaptive thresholding. The R, G and B values all vary with different lightning conditions. Therefore, a threshold set for a sunny day may not segment as well for a cloudy day. An example of an adaptive thresholding algorithm is Otsu's method [14]. For every image, Otsu's method creates histograms of the R, G and B values in the image to set thresholds.

In the HSV color model, thresholds can be set as constants for all images. Only the S and V values vary with different lightning conditions. An interval of H values will therefore cover most strawberries, from the darkest to the lightest, no matter the lightning conditions. This interval on the H value must be chosen to correspond to the ripe strawberries. An interval on the S and V values must also be chosen to respectfully discard the lightest and darkest objects that are not strawberries.

The HSV color model is often preferred to use in thresholding of color images because it is is invariant to lightning conditions.

2.1.2 Canny edge detection

A lot of the strawberries grow in clusters and are difficult to differentiate from each other. Threshold segmentation separates the strawberries from the background, but is not enough to separate them from each other, since they have similar colors. This subsection will

describe how filtering, Canny edge detection and closing are able to detect edges of strawberries.

Figure 2.3: Strawberry before (left) and after (right) filtering, Canny edge detection and closing.

Image filtering

The first step of Canny edge detection is to filter the image. Typically this is done with Gaussian filtering, but median filtering is used in this thesis. Both filters have edge-preserving properties, but the Gaussian filter is linear while the median filter is non-linear. The advantages and disadvantages of both are presented below.

The Gaussian filter is linear. For every pixel in the image, the Gaussian filter computes a weighted average of the surrounding pixels with a kernel of size $n \times n$. This kernel will return a value for every pixel on the image, and flatten unsmooth noise. The advantages of the Gaussian filter are that it is computationally effective and it filters out Gaussian noise.

The median filter is non-linear. For every pixel in the image, the median filter computes the median of the surrounding pixels with a kernel of size $n \times n$. The following example demonstrates how well the median filter preserves edges. Consider a one dimensional signal x with a clear edge and the output y_m of a one dimensional median filter of size $n = 3$:

$$\begin{aligned} x &= \begin{bmatrix} 2 & 1 & 3 & 245 & 253 & 250 \end{bmatrix} \\ y_m &= \begin{bmatrix} 2 & 2 & 3 & 245 & 250 & 251 \end{bmatrix} \end{aligned}$$

The signal on both sides of the edge has been smoothed out, and the clear edge has been preserved. On the other hand, consider a one dimensional Gaussian filter f of size $n = 3$ and the output y_g:

$$\begin{aligned} f &= \begin{bmatrix} 0.28 & 0.44 & 0.28 \end{bmatrix} \\ y_g &= \begin{bmatrix} 1.44 & 1.84 & 70.2 & 179.48 & 253 \end{bmatrix} \end{aligned}$$

The Gaussian filter has successfully smoothed out the signal, but the edge has not been preserved as well as with the median filter. The same is seen on Figure (2.4). The edges of the strawberry after median filtering are more sharp than after Gaussian filtering. The texture of the strawberry is more smooth after median filtering, since a median filter is good at eliminating salt and pepper noise, like the seeds on the strawberry. Further in this subsection about edge detection, a gray-scale image of the result of the median filter will be used because of these advantages.

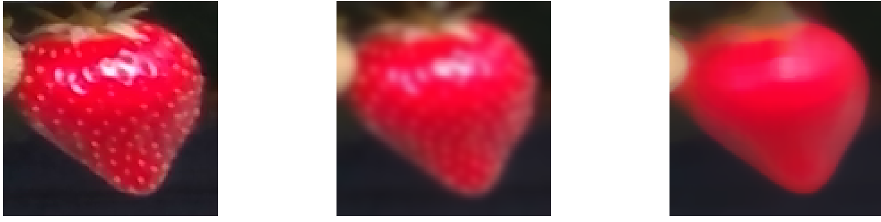


Figure 2.4: Strawberry without filter (left), with Gaussian filter (middle) and with median filter (right). Both filters have a kernel size of 19×19.

Edge detection

After filtering, the Canny edge detector does four more steps to find edges. The first step consists in finding the areas of steepest gradients in the image, as they probably will correspond to edges. Two Sobel kernels, one horizontal G_x and one vertical G_y, are used to find the intensity gradients in the image:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

G_x and G_y will respectively reinforce changes in the horizontal and vertical direction, as is seen in Figure (2.5). They can be compared to partial derivative in calculus and can be combined in the same way. For every pixel in the image, the Sobel kernels return a gradient G in a direction θ:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \Theta = \arctan 2(G_y, G_x)$$

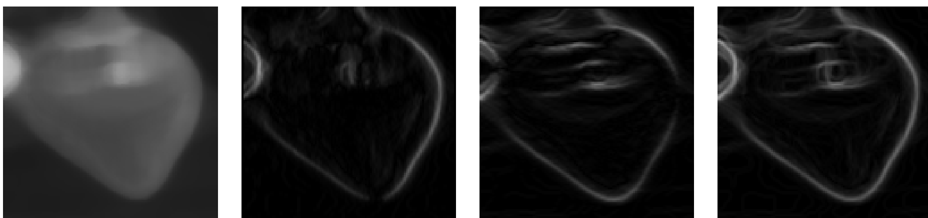


Figure 2.5: Strawberry in grayscale (left), result of horizontal Sobel operator (middle left), result of vertical Sobel operator (middle right) and combination of both results (right).

The second step is a non-maximum suppression operation. It reduces the width of the detected edges from several pixels to only one pixel. First, the edge direction of every pixel is rounded to the nearest of horizontal, vertical or a diagonal. Then for every pixel, the edge magnitude is compared to those of the pixels in positive and negative direction of the gradient. If the pixel has the largest magnitude, it is kept, or else it is suppressed.

a	b	**c**
d	↗	e
f	g	h

Table 2.1: Example of non-maximum suppression with a north-east gradient direction.

Table (2.1) is a representation of 9 pixels in an image. The middle pixel is the one undergoing a non-maximum suppression operation. Its gradient, represented as an arrow, may be pointing at any of the eight surrounding pixels. If the gradient direction is pointing in the north-east direction for example, the suppression will compare the middle pixel with the pixels in either direction of the gradient, pixels c and f. If the pixel intensity of the middle pixel is greater than the pixel intensity of both c and f, it is kept. If not, it is set to 0. This operation returns an image with one pixel thick lines of varying intensity, as seen in Figure (2.6).

Figure 2.6: Result from Sobel operators before (left) and after (right) non-maximum suppression.

The third step applies two thresholds to keep the edges with the strongest intensities and ignore the edges caused by noise. For every pixel in the image with detected edges, if its intensity is higher than the first threshold, it is marked as a high-intensity edge. If the intensity of the pixel is between the thresholds, the pixel is marked as a low-intensity edge. If the intensity of the pixel is below both threshold, the intensity is set to 0. In Table (2.2), T_h and T_l are respectively the high and low thresholds.

Pixel value	Classification of pixel
$P \in [T_h, 255]$	High-intensity
$P \in]T_l, T_h[$	Low-intensity
$P \in [0, T_l]$	$P = 0$

Table 2.2: Double threshold application.

The last step of the Canny edge detector goes through every low-intensity edge. If the edge is connected to a high-intensity edge, it is kept, or else it is suppressed. This results in eliminating the low-intensity edges caused by noise or by color changes. The gray-scale image with intensities is finally converted to a binary image, where ones are edges and zeros are the rest. Figure (2.7) shows the application of a double threshold and tracking by hysteresis.

Figure 2.7: Result from non maximum suppression before (left) and after (right) double threshold application and tracking by hysteresis.

Morphological operations

When detecting edges, the Canny edge detector will not detect every edges in their entirety. The morphological transformations presented in this subsection are a way of closing the edges that are not complete. Closing is an operation that consists of the morphological operations erosion and dilation. The binary image A and the kernel B in Equation (2.1) will serve as examples in this subsection. The binary image could be an output of the Canny edge detector as it has two edges represented by ones.

$$A = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \quad , \quad B = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \tag{2.1}$$

Dilation can be seen as an expansion of the shapes in a binary image, where shapes are the area with a value of one. A structuring element, or kernel, made up of ones slides over the binary image. For every pixel in the image that has a value of one, the kernel superimposes the image at that area. For a kernel size of 3×3, the edges of shapes will expand by one pixel and holes in the shapes will shrink by one pixel. The dilation of A by B returns:

$$A \oplus B = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \tag{2.2}$$

Erosion is the opposite of dilation. It can be seen as a shrinking of the shapes in a binary image. The same structuring element as in the dilation is used, but differently. For every pixel in the image that has a value of one, if the kernel is completely contained inside that area, the pixel is retained, or else it is set to zero. For a kernel of size 3×3, the edges of shapes will shrink by one pixel and holes in the shapes will expand by one pixel. The

erosion of $(A \oplus B)$ by B returns:

$$(A \oplus B) \ominus B = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \tag{2.3}$$

Closing is simply the erosion of a dilation by the same structuring element, $(A \oplus B) \ominus B$. Equation (2.3) is the result of closing A with the kernel B. The dilation in Equation (2.2) expands the shapes in A enough for the shapes to connect, then the erosion in Equation (2.3) shrinks the expansions that did not connect to anything. This results in the edges from A being connected, but still having a width of one pixel.

Figure 2.8: Result of hysteresis (left), result of dilation of hysteresis (middle) and result of closing, or erosion of dilation (right).

Figure (2.8) shows the steps in the closing operation. The edges from the Canny edge detector that were not fully connected are connected on the output of the closing operation.

2.1.3 Contour detection

When the strawberries in the image have been segmented from the background, the contour of each strawberry must be determined. For strawberries in clusters, inner contours must be determined in order to separate the strawberries from each other.

Border following

The algorithm used to find the contours is a border following algorithm from 1985 [11] that takes in a binary image and puts out a hierarchy of contours with their coordinates. The algorithm simply finds border pixels, which are pixels with ones next to pixels with zeros. Then, it follows these borders with a 4- or 8-connectivity. The 4- or 8-connectivity determines if the algorithm looks at all the 8 neighbouring pixels, or only at the 4 pixels in North, East, South and West direction.

Contour hierarchy

When the borders have been found by the border following algorithm, they are arranged in a hierachy tree, so that it can be determined if a contour is an inner or outer contour.

The outermost contours are of the first level. Then, the contours inside of the outermost contours are of the second level, and so on. In the case of this thesis, only the two outermost layers of contours are of interest. Figure (2.9) displays the contours for the binary image on the left. The outer contours, of first hierarchy level, are shown in the middle image. The inner contours, of second hierarchy level, are shown in the right image.

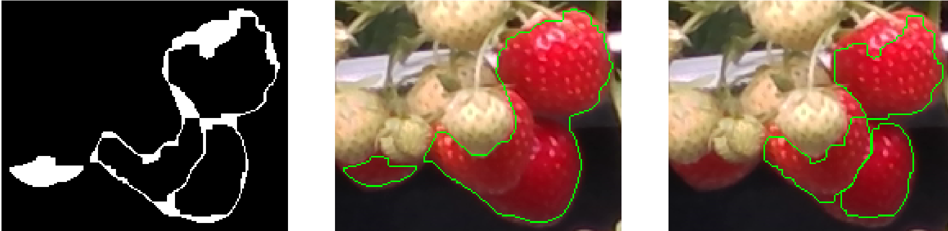


Figure 2.9: Result of filtering, Canny edge detection and closing (left), outer contours (middle) and inner contours (right).

The hierarchy tree of the contours is represented by a $n \times 4$-array where n is the number of contours. Every row of the array contains 4 elements, [Next, Previous, First_Child, Parent]. The hierarchy tree for the binary image in Figure (2.9) is shown in Table (2.3). The contour hierarchy makes it easy to iterate through the contours and arrange them in first and second hierarchy levels.

Contour	Next	Previous	First_Child	Parent
0	1	-1	-1	-1
1	-1	0	2	-1
2	3	-1	-1	1
3	4	2	-1	1
4	-1	3	-1	1

Table 2.3: Example of border hierarchy tree for binary image in Figure (2.9).

Minimum enclosing circle

Finally, the contour detection algorithm is used to separate strawberries in clusters from each other. If a contour of first hierarchy level is a single standing strawberry, it is kept, but if the contour is a cluster, its child's contours must be kept. For the hierarchy in Table (2.3) for example, the contours 0 and 1 are of first hierarchy level. Since contour 0 is not a cluster, it is kept. Since contour 1 is a cluster, its child contours 2, 3 and 4 are kept.

To determine if a contour is a cluster or not, the minimum enclosing circle for every contour is computed and their radiuses are computed. The mean value and standard deviation of these radiuses can be computed to find outliers. The outliers in the list of radiuses are identified as clusters of strawberries.

2.2 Detection with deep learning

In the previous section, different methods of traditional CV were introduced in order to segment strawberries in images. These methods successfully segment strawberries, but must be designed specifically for every task. It is interesting to see if newer methods of detection on images, like YOLOv3 [2], can be used in the case of strawberry picking. These methods are detailed in this section.

2.2.1 Classification with CNNs

The YOLOv3 framework is build around a convolutional neural network, or CNN, which is used to classify an object in a given image. An introduction to CNNs is given here.

Perceptrons

The smallest element of a neural network is the perceptron, as represented in Figure (2.10). A perceptron can act as a binary classifier to classify an object in one of two classes. It consists of n inputs, n weights, one bias and one output.

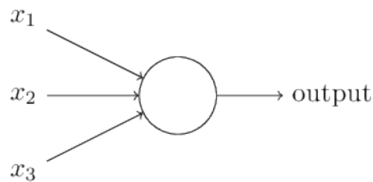


Figure 2.10: Model of a perceptron [15].

The perceptron creates an output y by multiplying a vector of inputs $\mathbf{x}$ with a vector of weights $\mathbf{w}$ and adding a bias b:

$$y = \mathbf{x} \cdot \mathbf{w} + b$$

In order to normalise the output, an activation function is used on the output. An example of an activation function is the sigmoid function, as seen in Figure (2.11). It squeezes inputs from $[-\infty, \infty]$ to $[0, 1]$ and creates a steeper gradient around 0, which pushes outputs to the extremities 0 or 1. This output is compared to a threshold in order to classify the object in one of two classes.

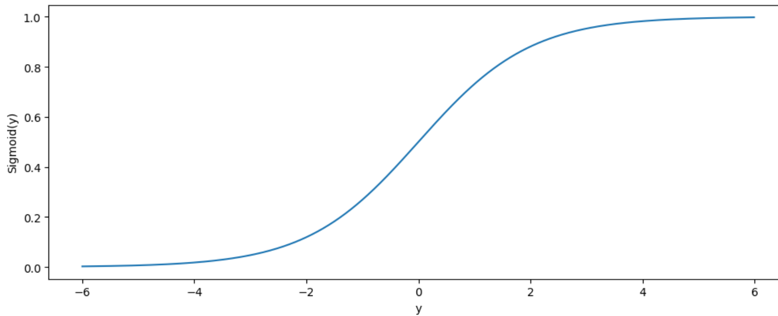


Figure 2.11: Sigmoid function used on the output of a perceptron.

In Figure (2.12), six objects from two classes are plotted in a graph. The two inputs x_1 and x_2 to the perceptron can for example be the weight and the height of the objects. The output of the perceptron becomes $x_1w_1 + x_2w_2 + b = 0$, which is the equation for a line, represented in red in Figure (2.12). Its gradient is $-\frac{w_1}{w_2}$ and its offset is b. By changing $\mathbf{w}$ and b, the perceptron changes the equation of the line.

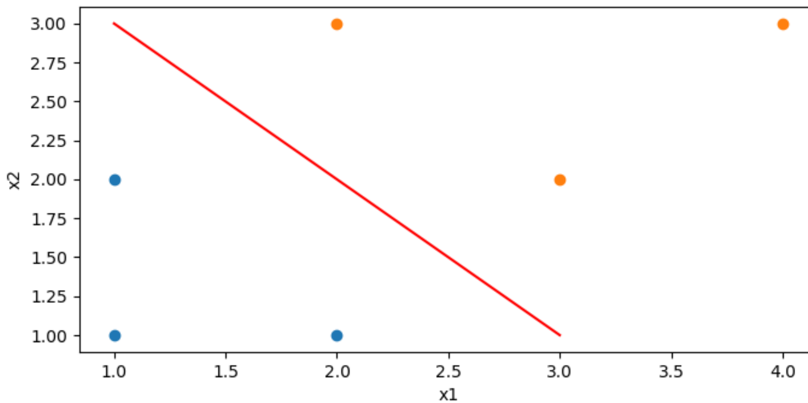


Figure 2.12: Parameters of a perceptron represented as a line between two classes.

In the example above, the separation line separates the two classes correctly, but for complicated problems, it may be more difficult to set $\mathbf{w}$ and b correctly. To solve this, the perceptron can be trained on training objects with back-propagation. This concept is crucial for the functioning of the neural networks used in this thesis. When the number of dimensions augment to more than four dimensions, it becomes impossible for humans to set a separation between classes.

Simply put, the back-propagation algorithm passes an object with known class, 0 or 1. By looking at the output of the perceptron and comparing it to the ground truth, the algorithm computes a loss value. Every weight and bias in the perceptron contributes to

this loss, so they should be adjusted accordingly. By using the chain rule on the derivation of the loss function, the contribution of every weight and bias can be calculated. A learning rate decides how much of this contribution must be adjusted for every weight and bias. This operation is repeated with many training objects over several epochs and returns a trained perceptron that hopefully can classify objects. An epoch is one pass of every training object.

By increasing the number of inputs to a perceptron, the number of dimensions augment, but the line remains linear with a single perceptron. For 3 inputs for example, the separating line would be a plane in a three dimensional space. To get a nonlinear separation, more perceptrons must be added.

Neural networks

Neural networks consist of many neurons, or perceptrons, in a network consisting of layers. The first layer is the input layer, the last layer is the output layer and the middle layers are the hidden layers. To explain neural networks, an example from *Neural networks and deep learning* [15] will be used in this subsection.

An example of a neural network is the one in Figure (2.13). The inputs are every pixel in a 28×28 gray image of handwritten digits, not all the input neurons are represented in the figure. Because of the sigmoid function, the output layers are 10 neurons that output the probabilities from 0 to 1 of the image depicting a certain digit.

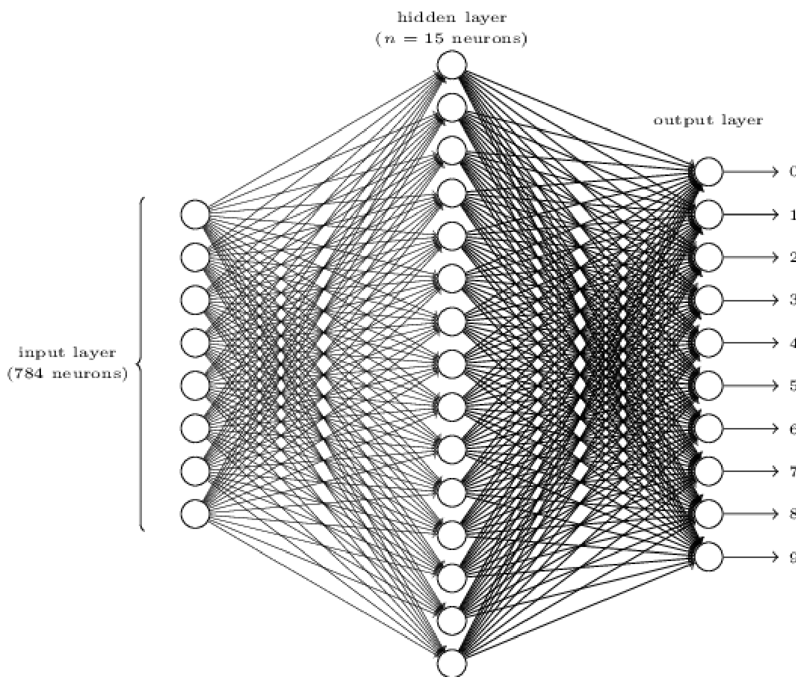


Figure 2.13: A fully connected neural network used to classify handwritten digits [15].

When passing an image of a handwritten 8 through the network, the optimal result would be to have an output of 1 on the output neuron annotated with 8 on the figure and outputs of 0 on the rest of the output neurons. However, all output neurons will most probably have other decimal values than the ground truth, 0 or 1. For every pass, a loss value is computed, as for the single perceptron in the previous subsection. Thanks to the chain rule, the contribution to the loss value of every weights and biases in the network can be computed and the weights and biases can be adjusted accordingly. Because of the chain rules however, the contribution of the first layers may disappear since differentiating a function many times ends up in 0.

In the start, all weights and biases are set randomly, but over the course of a training session, the network learns to classify digits. It can be seen as a statistical machine that learns similarities between objects of the same class. For every new object to classify, the network sees what similarities it has to every class and outputs a probability of belonging for every class based on these similarities.

The middle layers are called hidden layers because it is not easy to understand their functioning. Here is a guess of what may be happening. It may be that every neuron in the hidden layer learns a certain feature of the input image and activates itself if the input image has that feature. Digits are made up of a combination of 7 segments. Maybe 7 of the neurons in the hidden layer each look out for one of these 7 segments. The output neurons may then each look out for activations from a combination of certain neurons in the hidden layer. If the neurons for all 7 segments are activated, then maybe the output neuron for digit 8 is activated. If all but the middle segment are activated, then maybe the output for digit 0 is activated.

Fully-connected neural networks are good for simple tasks, but increase exponentially in complexity when layers are added because all neurons must be connected to all neurons in the previous and the next layer. For every new connection, another weight must be trained. Convolutional neural networks are a way of reducing this complexity.

Convolutional neural networks

This subsection uses another example from *Neural networks and deep learning* [15] to explain CNNs.

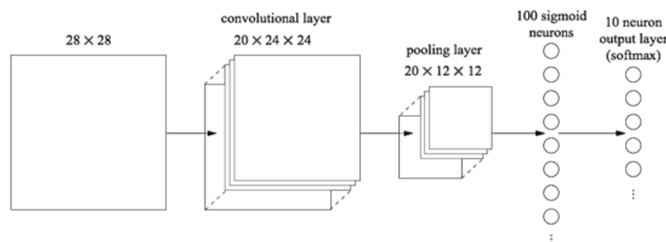


Figure 2.14: A convolutional network for classifying handwritten digits [15].

A CNN solves some of the problems encountered with fully-connected neural networks. It consists of an input layer, convolutional layers, pooling layers and fully-connected

layers, as is shown in Figure (2.14). The first layer is the input layer, the second and third layers are the convolutional and pooling layers and the two last layers are the fully-connected output layers. In a more complicated network, more convolutional and pooling layers could be added in series, as well as more layers in the fully-connected layers. The fully-connected layers at the end serve as a statistical classifier as the one in the previous subsection.

Instead of being fully-connected with the previous layer, a convolutional layer is partially connected to its previous layer. Every neuron in the convolutional layer is connected to a $n \times n$-region of the previous layer, as is shown in Figure (2.15). The region is called the receptive field of the neuron. This ensures a local connectivity, which looks at the relation between pixels that are close to each others. Such a convolutional layer can be compared to the Gaussian filter explained in this chapter's first section.

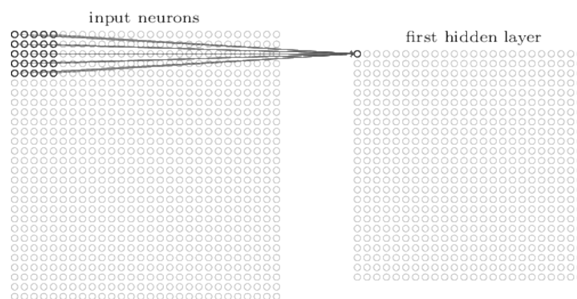


Figure 2.15: A convolutional layer [15].

All the weights and biases in a convolutional layer are shared. This reduces the number of trainable parameters and the computation time. By sharing the weights and biases, all neurons in a convolutional layer look for the same feature in the image. This is why the $n \times n$-kernel of weights used in a convolutional layer is often called a filter. The filter in Figure (2.15) may for example look like this:

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Such a filter would activate a neuron if the receptive field is a horizontal edge. The convolutional layer would output a feature map for this specific filter, or a spacial representation of where there are horizontal edges in the previous layer. The convolutional layer in Figure (2.14) has 20 filters that produce 20 different feature maps. During training, these filters will adjust their weights and biases to detect 20 different features in the input image and pass these feature maps forward.

After a convolutional layer comes often a pooling layer. The pooling layer reduces the size and resolution of the feature maps outputted by the convolutional layer, but also reduces the complexity of the network. Every neuron in the pooling layer is connected

to a $n \times n$-region of the previous layer, as is shown in Figure (2.16). In the case of a max-pooling layer, the neurons in the pooling layer output the maximum value of all the values in its receptive field, which ensures that the features in the feature maps are passed forward through the pooling layer. The pooling layer reduces the accuracy of the feature's position, but greatly reduces the complexity of the network.

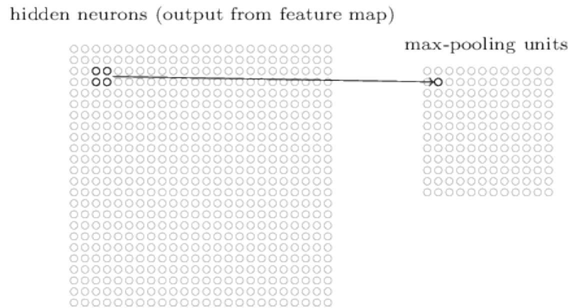


Figure 2.16: A pooling layer [15].

By adding convolutional and pooling layers in series, they output feature maps with features of features, and so on. Therefore, the early convolutional layers in a CNN recognize low level features while the later convolutional layers recognize higher level features made up of lower level features. These higher level features are then passed forward to the fully-connected layers, or classifier, that classifies the object in the image. For one image, a CNN can classify one object. In this thesis, the goal is to detect several strawberries in one image. The YOLO framework is capable to use a CNN for this purpose.

2.2.2 Detection with YOLOv3

Classification consists in assigning one class to one image, whereas detection consists in classifying several objects in an image and determining their position. Several detection networks do this by sliding a window over the image at different scales and classifying every window with a CNN. This means that there are many passes through the neural network per image and that increases the computation time. YOLO solves this by partitioning the image in a grid and passing the image once through the neural network. This is where the name YOLO, or You Only Look Once, comes from.

From classification to detection

YOLOv3 rescales the input image to 416×416, then divides the image into three grids of cells at different scales, as seen in Figure (2.17). For each of these grid cells, 3 bounding boxes, or anchor boxes, with different fixed dimensions are proposed and passed through a CNN.

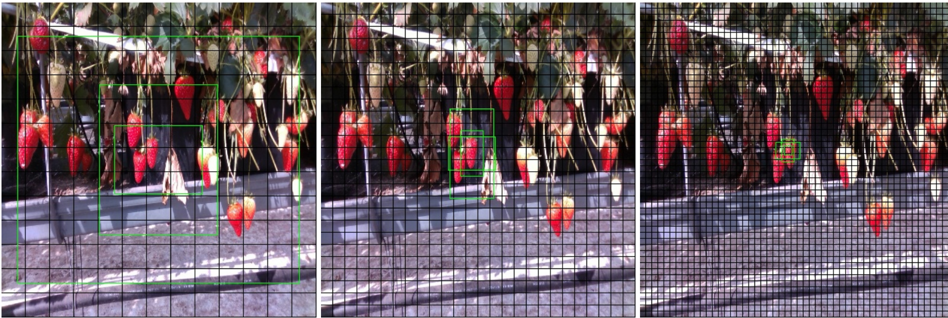


Figure 2.17: Detection on a grid cell with YOLOv3 at first (left), second (middle) and third (right) scale with three bounding boxes per scale.

The CNN used in YOLOv3 is called Darknet-53 and contains 53 convolutional layers. It has been trained by Joseph Redmon for classification on the ImageNET [16] dataset of a thousand classes. This results in a strong network that recognizes many different features at different layers of the network. Darknet-53 is modified from a classifier to a detector by removing the fully-connected layers at the end and replacing them with 53 additional layers that are used for detection at three different scales. This results in 106 layers in YOLOv3.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.18: Darknet-53 architecture [2].

The detection at three different scales happens at three different layers in the network, layer 79, 91 and 103. After the classifier Darknet-53, the feature map is downsized to a 13×13-grid until layer 79 to detect at the first scale. Then the feature map from layer 79 is passed through some convolution layers before being upsampled to a 26×26-grid and concatenated with a previous 26×26 feature map from layer 61 until layer 91 to detect at

the second scale. Then the feature map from layer 91 is passed through some convolution layers before being upsampled to a 52×52-grid and concatenated with a previous 52×52 feature map from layer 36 until layer 103 to detect at the third scale.

The detection at every scale is performed with 1×1 convolutional kernels on layer 79, 91 and 103 to output a vector per grid cell. The output vector consists of the bounding box coordinates, the objectness score and the class score for every of the 3 bounding boxes per scale. The output vector is as the following:

$$[t_x, t_y, t_w, t_h, p_o, p_c] * 3$$

The predicted bounding box coordinates t_x, t_y, t_w and t_h can be converted to true bounding box coordinates as in Figure (2.19). The objectness score p_o indicates which of the 3 bounding boxes have the highest IoU. The bounding box which has the highest IoU for a given grid cell has an objectness score of 1, the others have an objectness score of 0. The class score p_c indicates the probability of the object in the bounding box to belong to a given class.

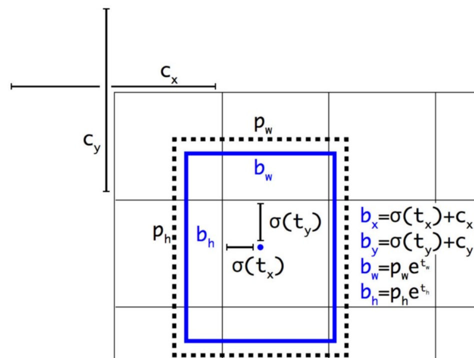


Figure 2.19: Conversion from predicted to true bounding box coordinates [2].

This results in one possible detection for every of the grid cells in Figure (2.17). Small objects, as well as big objects can be detected on a single pass of the network. Non-maximum suppression is performed on the resulting bounding boxes to remove duplicate bounding boxes.

YOLOv3-tiny

If YOLOv3 is too slow because of its 106 layers, YOLOv3-tiny is a faster but less precise alternative. YOLOv3-tiny contains only 23 layers and is build around the Darknet classifier with only 8 convolutional layers. Instead of three scales, YOLOv3-tiny detects at two different scales, as is shown in Figure (2.20).

Figure 2.20: Detection on a grid cell with YOLOv3-tiny at first (left) and second (right) scale with three bounding boxes per scale.

2.3 Evaluation metrics

To compare the different networks and methods used to detect strawberries, different evaluation metrics are used. They are described in this subsection.

2.3.1 Detection

Mean average precision, or mAP, is the universal metric used to evaluate the detection performance of neural networks. The mAP is the mean value of the average precisions, or AP, of every class. In the case of a single class, as in this thesis, the mAP equals to the AP. Here is how the mAP is calculated for a single class.

First of all, the intersection over union, or IoU, must be computed for every prediction in an image. If a predicted bounding box overlaps with a ground truth bounding box, the area of intersection and area of union can be computed. As the name would suggest, the IoU is given by the following equation:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

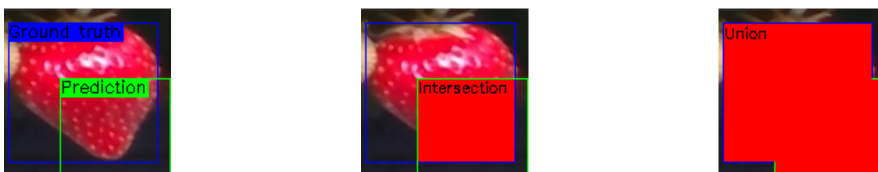


Figure 2.21: Intersection and union for a prediction.

Depending on the value of IoU, the prediction is either a true positive (TP), a false positive (FP), a true negative (TN) or a false negative (FN). In the case of detection, a true positive is when IoU ≥ 0.5 and a false positive is when IoU < 0.5 or the bounding box is a duplicated one. A false negative is when IoU ≥ 0.5 but the prediction is the wrong class. The precision and recall of a prediction are given by the following equations:

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

A precision-recall curve can be plotted and the mAP for a single class will be the area under the curve. During training, the mAP is calculated for every 100 epochs on the test set. This is to avoid overfitting of the neural network. If the loss keeps on decreasing, but the mAP begins to decrease after hitting a maximum, then the training should be stopped. When the mAP on the test set diminishes, it means that the neural network has become to specific to the training set.

Another metric used for evaluation of neural networks is the average loss. The loss function in neural networks is used during training to improve the weights and biases with back-propagation. It can also be used as a metric to evaluate how good the network has adapted to the training dataset. For YOLOv3, the loss consists of the classification loss, the localization loss and the confidence loss. The classification loss is the same as for a normal CNN. It is the sum of the mean square error of the class conditional probabilities for every grid cell. The localization loss is the sum of the mean square errors of the location and size of the bounding boxes for every grid cell. The confidence loss is the sum of the mean square errors of the confidence score for every grid cell.

A last metric used for evaluation of neural networks is the average confidence score. This metric is simply the average of the confidence scores in the test images. It is correlated with the average loss since the loss function contains the sum of the mean square errors of the confidence scores. A lower average loss will indicate a higher average confidence score.

2.3.2 Segmentation

While it is good at evaluating detection, mAP is not suited to evaluate segmentation since it needs a confidence score and bounding boxes. Here are some metrics that will be used to evaluate segmentation, the IoU, the accuracy and the precision. First, the notion of true and false positives and negatives must be adapted to segmentation.

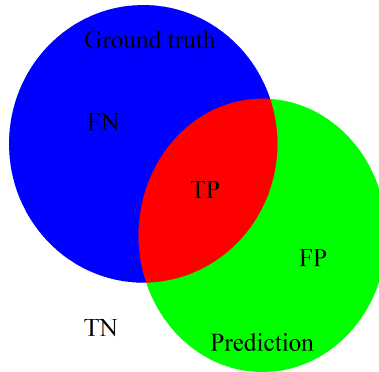


Figure 2.22: Representation of true and false positives and negatives on a prediction.

The positives are the pixels that the segmentation has identified as strawberries and the negatives are the pixels that have been identified as background. They are true if the prediction is true and false otherwise. With this rule, the pixels of a segmentation can either be a true positive (TP), a false positive (FP), a true negative (TN) or a false negative (FN), as is shown in Figure (2.22). For every image, an IoU, an accuracy and a precision can be computed:

$$\begin{aligned} \text{IoU} &= \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \\ \text{Accuracy} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \end{aligned}$$

IoU is the intersection over the union and is a measure of how much overlap there is between the prediction and the ground truth. It is a metric that combines the accuracy and precision in one. The accuracy is the intersection over the ground truth and indicates how much of the ground truth is covered by the prediction. The precision is the intersection over the prediction and measures how much of the prediction is covered by the ground truth.

2.3.3 Frames per second

Frames per second, or FPS is the last metric used to evaluate the speed of the detection algorithms in this thesis. It is a measure of how many frames the algorithm can process per second, as a frequency. To find the average processing time t per image, the inverse of the frequency is computed. The higher the FPS, the lower the execution time and the higher the speed:

$$\text{FPS} = \frac{1}{t}$$

Chapter 3

Implementation

This chapter presents the tools and libraries used in this thesis, Python, OpenCV, NumPy, Darknet and Google Colab. Then it presents the source code implemented in this thesis.

3.1 Tools and libraries

Several tools and libraries have been used in this thesis. This section explains the fundamentals of these tools and libraries and explains why they were chosen.

3.1.1 Python

Python [4] is a high-level open source programming language created by Guido van Rossum in 1991. It was chosen in this thesis because of its ease of implementation and its many libraries like OpenCV and NumPy.

3.1.2 OpenCV

OpenCV [5] stands for Open source Computer Vision and is a library for C++, Python and Java developped by Intel. It is used in this Master's thesis to segment the strawberries with traditional CV. OpenCV is widely used and documented which makes its use and debugging easy. The code is open source, so the source code is available for everyone for free. Its purpose is to give everyone access to optimized CV functions so that developers don't need to reinvent the wheel for each CV task.

The OpenCV source code is written in C++, so when it is run in Python or Java, it really runs the encapsulated C++ code. This makes OpenCV equally fast on all platforms, as long as the OpenCV-functions are used for the CV tasks. However, as soon as functions with nested for-loops are implemented in Java or Python, the code gets much slower than it would if implemented in C++. An example for segmenting in Python is to use the OpenCV-function cv2.inRange() instead of iterating through every pixel in Python. The code will instead iterate through every pixel in C++, which is faster than in Python.

The segmentation code in this thesis has been written in Python for ease of implementation and testing. As long as too many for-loops are avoided, the code will not be too slow. It is possible to rewrite the code in C++ if the code in Python is too slow.

3.1.3 NumPy

NumPy [6] is an open source library in Python. It provides an array object and tools to interact with these array objects. Python uses lists to handle arrays of object. A list is resizeable and can contain object of different types. NumPy arrays can only contain objects of the same type and are not resizeable. This makes them faster and take up less memory than Python lists. Instead of being an array containing pointers to different objects like a Python list, a NumPy array is one array of the same object type in memory, like in C++.

3.1.4 Darknet

Darknet [8] is an open source neural network framework developed by Joseph Redmon at University of Whashington. It can be used to train or use classifiers, but also detectors. Classifiers can identify one object in one image, but detectors can identify and locate several objects within an image. Since the goal of this thesis is to locate possible strawberries in an image, some of the detectors of Darknet have been implemented in this thesis. A detector can be seen as a classifier that works on subparts of the image.

Darknet is most known for its YOLO neural network. In short terms, it is a classifier that classifies every cell in a $m \times n$-grid of the image. YOLOv3 uses the Darknet-53 as classifier, which has been trained on 1000 classes. YOLOv3 has then been trained by Joseph Redmon on 80 classes. By stripping the weights of the last convolutionnal layers of the network, it can be trained to detect custom objects, like strawberries. This is what has been done in this thesis.

Darknet automatically uses data augmentation, so that the user doesn't need to think about it. This gives different results for every training session, even if the training data and configuration files are the same. Therefore, the network should be trained several times with the same parameters, to ensure that good results are not just random coincidences caused by different data augmentations.

Darknet has been chosen as neural network framework because it is easy to implement for custom objects and its YOLOv3 model scores good in comparison to other neural network frameworks [2]. It is also good at detecting many small and big objects in images because of its different scales. Since there may be many small and big strawberries in images from greenhouses, YOLOv3 should be a good choice.

3.1.5 Google Colab

Training neural networks on CPU's is not a viable option, as the computing takes too much time. The use of a GPU instead of a CPU reduces the training time drastically, in an order of magnitude. Google Colaboratory [7] is an open source service developed by Google that runs in Python. It gives the user the ability to switch between GPU and CPU

processing. All the processing happens in the cloud, so training sessions can be supervised with any device, regardless of processing power.

Google Colab has been used in this thesis to download Darknet, to train it and to test it. Since all computing happens in the cloud, Google Colab can in an instant download big datasets from other cloud services, like Google Drive.

Drawbacks of Google Colab are that the CPU and GPU may be shared with other users if the demand is high, and that the runtime resets itself every 24 hours. This is for example to avoid people from mining cryptocurrency. Advantages of Google Colab are that it is free, it gives access to a GPU with all the necessary libraries installed and it is an easy to use "plug and play"-service.

3.2 Source code

The source code implemented in this thesis is found in Appendix A. Here is a description of the three different folders found in Appendix A.

3.2.1 Detection

The "detection"-folder contains a Google Colab Notebook that can be opened in Google Colab. First, the Notebook downloads the trained models of the detection networks from a public Google Drive folder. This includes configuration files, data files and trained weights files. Then, the Notebook clones Darknet from a Github repository, moves the models of the detection network to Darknet and makes Darknet. Finally, the Notebook tests the models on an image and on a video.

To train YOLOv3, a Github repository [17] was cloned and the instructions at that repository were followed, with some adjustements:

- A "strawberry" folder with images and corresponding text files of labels must be made. This has in this thesis been done with the labeling script described below. For each image "name.jpg", a text file "name.txt" contains a line for every bounding box in the image. Every line in the file has this format: <object-class> <x_center> <y_center> <width> <height>. In the case of this thesis, object-class is 0 for every bounding box since there is only one class. x_center, y_center are the relative coordinates of the center of the bounding box and width and height are the relative width and height of the bounding box. This folder with images and labels is moved to darknet/data/.

- A configuration file "yolov3-strawberry.cfg" of the network to train is made, based on "yolov3.cfg". The number of batches is set to 64. The number of subdivisions is set to 16 to avoid memory error in Google Colab. In a testing setting, the number of subdivisions and batches are set to 1. The number of max_batches is set to 4000. The number of steps is set to 3200,3600. For each [yolo]-layer, the number of classes is set to one and the number of filters in the preceding [convolutional]-layer is set to 18. Finally, "max=200" is added to the last [yolo]-layer for the network to be able to detect more than the default 10 objects per image.

- A text file "strawberry.names" containing the name of every class is made. It contains one word: "strawberry". Two files containing the names of training and test images are created and named "train.txt" and "test.txt". On every line, they contain the path of every image. These three files are moved to darknet/data/.

- A text file "strawberry.data" summing up all the data information is created. It contains the number of classes and the path to "train.txt", "test.txt", "names.txt" and where to save the weights.

- The concatenated weights of YOLOv3 must be downloaded. Concatenated means that the last convolutional layers of the weights have been deleted in order for the network to be able to retrain its last layers on strawberry images. The concatenated weights are downloaded [18] and are moved to darknet/.

The training session starts with the following command in Google Colab:

```
1   !./darknet detector train data/strawberry.data cfg/yolov3-strawberry.
    cfg darknet53.conv.74 -dont_show -map >> yolov3-strawberry.log
```

This command saves a graph of the training session like the ones in Appendix C as "chart.png" in the darknet folder. The output of the training session is saved to a log-file in the darknet folder, in this case "yolov3-strawberry.log".

3.2.2 Segmentation

The "segmentation"-folder contains two python program files, "main.py" and "segmentation.py":

- "main.py" reads images from a "images"-folder, segments them and draws proposed bounding boxes on them before saving the images to a "results"-folder.

- "segmentation.py" contains the functions used by "main.py" to segment and detect the strawberries in each image.

First, the segmentation module converts the image from the RGB color model to the HSV color spectrum. Then, it filters the HSV-image with a median filter and segments the image with the cv2.inRange() and cv2.bitwise_and() functions.

Then, a Canny edge detection is performed on the saturation and value channels of the segmentation output with the cv2.Canny() function. The outputs of the Canny edge detectors are added together with the cv2.bitwise_or() function. A closing operation is then performed with cv2.morphologyEx() on the Canny edge detection output to close the gaps in the edges.

The cv2.findContours() function is then used to detect the contours in the binary edge image. For every contour of first hierarchy level over a certain size, the radius of the minimum enclosing circle is computed with the cv2.minEnclosingCircle() function. The mean value and standard deviation are computed for these radiuses and all radiuses that are bigger than the mean by more than half a standard deviation are identified as strawberry clusters. Their children are identified as being strawberries.

For each of the contours that are assumed to be strawberries, a bounding box is drawn on the image with cv2.boundingRect() and cv2.drawContours(). The bounding boxes are also added to a list of bounding boxes, which is the return object of the segmentation module. This enables other modules to use the segmentation script to get the position of strawberries.

3.2.3 Labeling

The "labeling"-folder contains three python program files, "main.py", "segmentation.py" and "utilities.py":

- "main.py" first reads images from a "images"-folder, segments them and draws proposed bounding boxes on them. Then, it waits for the user to decide which bounding boxes to save and discard with keyboard inputs before letting the user draw additional bounding boxes on the image. Finally, "main.py" saves the bounding boxes in YOLO-format in a "labels"-folder, one labels text file per image. The filename and path of every image is saved in "images.txt".
- "segmentation.py" is the same as in the "segmentation"-folder.
- "utilities.py" contains the functions used by "main.py" to receive user input and save bounding boxes to files.

Chapter 4

Results

This chapter presents the results obtained with the three methods implemented in this thesis, with traditional CV and deep learning. First, examples of each detection are presented. Then, the evaluation metrics used to compare the methods are presented in tables.

4.1 Detection examples

This section presents examples of each detection on an image. Videos of the detections are available in Appendix B. The image and video presented here were not a part of the training set for the YOLOv3 models and were not used when designing the traditional CV algorithm.

4.1.1 Traditional computer vision

Figure (4.1) shows a result of HSV-segmentation, Canny edge detection and contour detection applied in series on an image. The blue contours are the contours of first hierarchy level and the green contours are the contours of second hierarchy level.

Figure (4.2) shows a result of detection performed on an image with traditional CV. The contours in Figure (4.1) were either identified as strawberries or strawberry clusters by looking at the radiuses of the minimum enclosing circles and finding outliers. For every contour identified as a strawberry, its bounding box was saved, and for every contour identified as a strawberry cluster, the bounding boxes of its children were saved. Some of the clusters were correctly identified and split up, but other clusters were too small to be identified as clusters. In this example, two of the detections are false positives, meaning that leaves were identified as strawberries.

Figure 4.1: Example result of segmentation, Canny edge detection and contour detection.

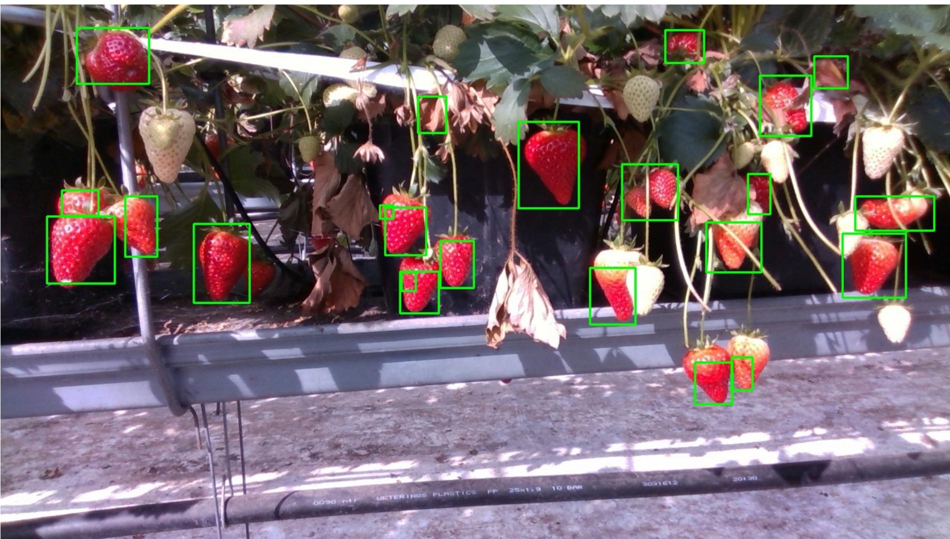


Figure 4.2: Example result of detection by drawing the bounding boxes of the contours in Figure (4.1).

4.1.2 YOLOv3-tiny-strawberry

Before training YOLOv3, YOLOv3-tiny was trained as a proof of concept because its fewer layers make it faster to train than YOLOv3. Figure (4.3) shows an example detection, Appendix C shows the average loss and mAP during training and Appendix B

provides a link to a video of the detection results. The mAP quickly reached 81%, but didn't improve for the rest of the training session. This is probably caused by the small size of the dataset. The small size of the dataset also explains the oscillations of the average loss. This limitation in the dataset is discussed in the next chapter.

The final weights were chosen at 2000 iterations with an average loss of 4.0510 and a mAP of 80.0%. Iteration 2000 was chosen because the mAP doesn't improve further and to avoid over-fitting of the network.

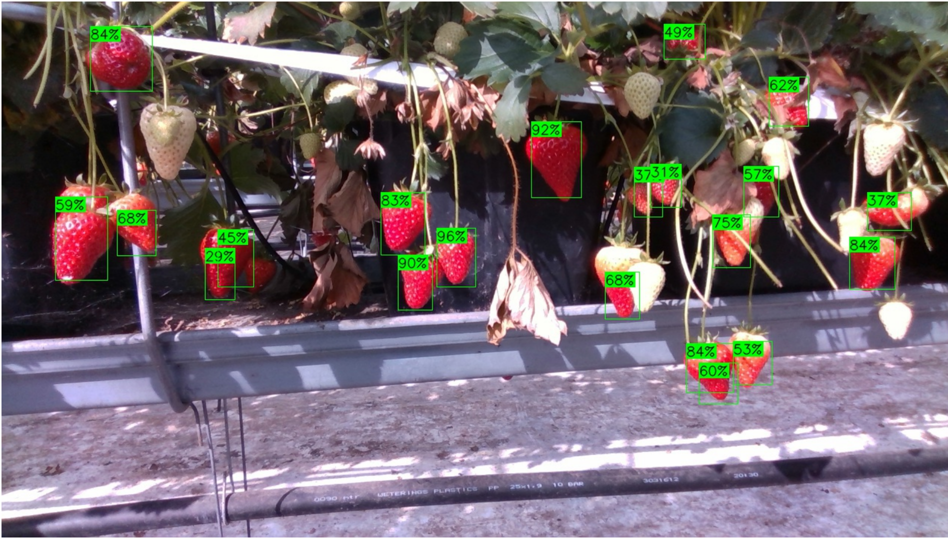


Figure 4.3: Example of detection with YOLOv3-tiny trained on strawberries.

4.1.3 YOLOv3-strawberry

Since YOLOv3-tiny-strawberry showed good results, YOLOv3 was trained on the same dataset and with the same parameters. Figure (4.4) shows an example detection, Appendix C shows the average loss and mAP during training and Appendix B provides a link to a video of the detection results. These metrics behaved similarly to the metrics of YOLOv3-tiny-strawberry because they trained on the same small dataset. The mAP reached 90% quickly, but didn't improve any further. The average loss had oscillations, but kept diminishing even if the mAP didn't improve. This is an indication of over-fitting, that the network is training too specifically for this dataset.

The final weights were chosen at 2000 iterations with an average loss of 1.7431 and a mAP of 87.1%. Iteration 2000 was chosen because the mAP doesn't improve further and to avoid over-fitting of the network.

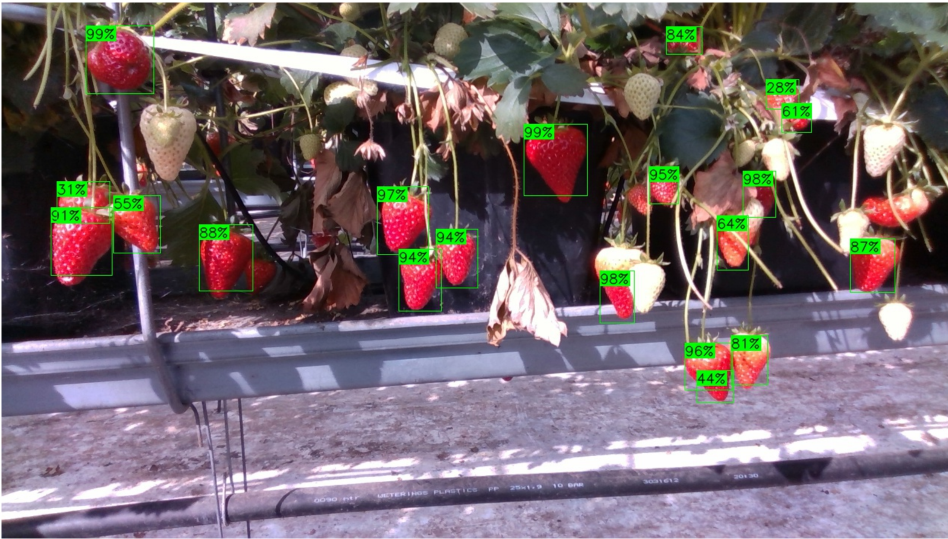


Figure 4.4: Example of detection with YOLOv3 trained on strawberries.

4.2 Evaluation metrics

This section presents the performances of each method with several evaluation metrics, the mAP, IoU, accuracy, precision and FPS.

4.2.1 Detection

The detection has been evaluated with the mAP. To compute the mAP of the traditional CV algorithm, a Github repository [19] was used. Since a confidence score is needed to calculate the mAP, but the detection doesn't output one, a confidence score of 100% was given to every detection. The mAP was calculated on the same dataset as YOLOv3 and resulted in 21.4%. For YOLOv3-strawberry and YOLOv3-tiny-strawberry, the mAP and the average loss from the training results in Appendix C were used. The resulting mAP, average loss and average confidence score for each method are shown in Table (4.1).

Method	mAP	Avg. loss	Avg. confidence score
Traditional CV	21.4%	-	-
YOLOv3-strawberry	87.1%	1.74	74.5%
YOLOv3-tiny-strawberry	80.0%	4.05	69.8%

Table 4.1: Mean average precision, average loss and average confidence score for the three methods implemented in this thesis.

When looking at the mAP, there seems to be a significant difference between the YOLOv3 models and the traditional CV method. This is discussed further in the next chapter.

4.2.2 Segmentation

As will be discussed in the next chapter, it seems that mAP is not suited to evaluate the performance of the traditional CV method. This subsection evaluates the segmentation performance of each method. A set of ground truths was created by segmenting the test images and deleting the false positives. Then, the inside of the bounding boxes of each detection method was segmented with the same thresholds. When compared to the ground truth, their performance could be computed. The results are shown in Table (4.2).

Method	IoU	Accuracy	Precision
Traditional CV	74.5%	75.0%	99.3%
YOLOv3-strawberry	86.1%	86.6%	99.4%
YOLOv3-tiny-strawberry	85.0%	85.5%	99.3%

Table 4.2: IoU, accuracy and precision for the different methods.

When evaluating the segmentation instead of the detection, traditional CV seems to perform better, but the deep learning methods are still better. This is discussed in the next chapter.

4.2.3 Frames per second

The speed performance of every method was tested in the Google Colab environment by testing the methods on a video. The resulting videos are given in Appendix B and the resulting FPS for each method are shown in Table (4.3).

Method	CPU	GPU
Traditional CV	3.0 FPS	3.5 FPS
YOLOv3-strawberry	0.1 FPS	16.6 FPS
YOLOv3-tiny-strawberry	1.1 FPS	41.3 FPS

Table 4.3: Speed performance of the different methods implemented in this thesis.

The different speed performances of each method with CPU and GPU usage are also discussed in the next chapter.

Chapter 5

Discussion

This chapter discusses the results presented in the previous chapter. First, it discusses traditional CV and deep learning separately, then compares them with each other.

5.1 Traditional computer vision

This section discusses the choices taken in the implementation of the traditional CV algorithm and its limitations.

5.1.1 Python vs. C++

Python and C++ were both considered as programming language for this thesis. Python ended up being used to implement the detection method with traditional CV because of its ease of implementation. Python is a higher level language than C++, so there is less to declare. It makes Python very good to test ideas and get results fast.

Python is slower than C++, in part because of its dynamic memory allocation and its dynamic typing, but this is solved by using the libraries NumPy and OpenCV. These use encapsulated C and C++ code, so instead of using for-loops in Python, the for-loops and the array handling can be used in functions from these libraries.

This thesis has served as a proof of concept to see if traditional CV can be used to detect strawberries in images. If this code is to be implemented in hardware, it should be translated to C++ to increase speed.

5.1.2 mAP calculation

In order to compare the detection with traditional CV and deep learning, the evaluation metrics for detection were first used. mAP is well suited for detections with deep learning, but is not suited for detections with traditional CV. There is a great difference in performance when evaluating the traditional CV method with detection metrics and segmentation metrics, as is seen in Tables (4.1) and (4.2). For detection of one class, the mAP

is given by the ratio of true positives over the sum of true and false positives. The more detections are identified as false positives, the lower the mAP. Two factors are increasing the number of false positives and thus lowering the mAP for the traditional CV method. First, the method detects several double detections which end up as false positives. In YOLOv3, these double detections are discarded by a non-maximum suppression on the confidence score, but this is not possible with the traditional CV since there is no confidence score. Second, the detections with an IoU less than 0.5 end up as false positives. Since the method is not trained to match the bounding boxes, as in YOLOv3, the dimensions of the predicted bounding boxes will vary more. Also, single standing strawberries that are falsely identified as clusters will be detected as several smaller bounding boxes. These will probably all end up as false negative and lower the mean average precision.

5.1.3 Limitations of method

The main limitation of this method is the separation of strawberries in clusters. It works in some cases, but not in others. For example, if a cluster has the same size as the other strawberries in an image, it won't be identified as a cluster since its mininum enclosing circle's radius is not an outlier. Also, if the picture has many clusters, they will raise the mean value and not be identified as outliers. Another case is if there are few strawberries in an image. Then the mean and standard deviation of the radiuses of the minimum enclosing circle may not find clusters as outliers.

It may be that this is not really a problem. In both cases, where a contour is either the contour of a strawberry or of a cluster, the inner or outer contour will be detected.

5.2 Deep learning

This section discusses the difference between YOLOv3 and YOLOv3-tiny and the limitations of the dataset used during training of these methods.

5.2.1 YOLOv3 vs. YOLOv3-tiny

Both YOLOv3 and YOLOv3-tiny were trained for object detection of strawberries in this thesis. Table (4.1) sums up the mAP, average loss and average confidence score calculations of each method. At first glance, it may seem that YOLOv3-strawberry far outperforms YOLOv3-tiny-strawberry, but the fact is that they both achieved high mAPs. By looking at the example detections in Figures (4.3) and (4.4), the differences are quite difficult to see, but when looking at Table (4.1), YOLOv3-strawberry has higher average confidence scores for each detection than YOLOv3-tiny-strawberry. According to the mAP and the average confidence score, both models are good, but YOLOv3-strawberry is even better than YOLOv3-tiny-strawberry.

When looking at the segmentation performance of YOLOv3-strawberry vs. YOLOv3-tiny-strawberry in Table (4.2), there is only a slight improvement in YOLOv3-strawberry over all metrics. In a real world setting, the bounding boxes of the predictions must be segmented to compute the depth of the strawberry in the bounding box. These metrics may therefore be a better measure of performance than the mAP.

With a CPU, YOLOv3-tiny-strawberry processed approximately 1 frame per second, but YOLOv3-strawberry processed only 1 frame every 10 seconds. This will vary with the CPU and GPU used. Especially the GPU and CPU of Google Colab may be slower than a local GPU and CPU since they might be shared with other users of the service. This may explain the low CPU performance of the methods. With these numbers, only YOLOv3-tiny-strawberry can be used if there is no GPU available. With a GPU, YOLOv3-tiny-strawberry processed 41.3 frames per second and YOLOv3-strawberry processed 16.6 frames per second, so both can be used with a GPU.

The preferred model would be to use YOLOv3-strawberry because it achieved a lower loss, a higher mAP and higher confidence scores per detection. However if speed is an issue, YOLOv3-tiny-strawberry can be used instead of YOLOv3-strawberry with a small reduction in precision but a great increase in speed.

5.2.2 Testing on video

When using the trained YOLOv3 networks on video, the detection seems to lay behind when the camera is moving fast, as is seen in Appendix B. However, when the camera stops for example at 0:33, the detections are correctly aligned with the strawberries. This is not a problem when using the traditional CV algorithm. Luckily, the robot must stop whenever it must pick strawberries. This will give it time to adjust the detections before picking.

5.2.3 Limitations of dataset

The dataset used to train the YOLOv3 networks in this thesis contains only 250 images of strawberries taken in a greenhouse and the one used to test the networks contains 54 images. These 304 images were all taken on the same day so they all have similar lightning conditions. 304 is quite a low number of images to train a neural network of this complexity on. Usually, tens of thousands of images with a lot of variation should be used to train a network of this size. The low number of images can explain the oscillation of the average losses observed in the Figures in Appendix C.

In spite of these limitations, the trained networks achieve low average losses and high mAPs. This can have several explanations. First, the YOLOv3 models use data augmentation before training. This means that for every image in the dataset, similar copies are created. These copies are created by changing four different parameters, the hue, the saturation, the exposure and the angle of the image by a small increment. This results in a greater dataset with greater variations. Variation in the dataset is very important when training a neural network to avoid over-fitting.

Second, the problem at hand is quite a simple problem compared to other detection problems. The networks must only detect one single class, while the original YOLOv3 network is trained on 80 different classes. The strawberries to detect are all similar in color and shape and they stand out from the background. No other objects in the image have the same red color as the strawberries. In other detection problems, the objects to detect may for example be dogs of varying color and shape. The non convex shape of a dog is more complicated to recognize for the network than the mostly convex strawberry

shape. The simplicity of the detection problem is the second reason why the network performs well despite a low number of training images.

5.3 Traditional CV vs. deep learning

This section discusses the differences and similarities of traditional CV and deep learning with observations made in this thesis.

5.3.1 Detection metrics

The mean average precision, the average loss and the average confidence score for the three methods implemented in this thesis are summed up by Table (4.1). There is a significant difference between the traditional CV and the trained YOLOv3 models. First and foremost, the difference can be explained by the fact that the mAP metric is build for object detection where a confidence score is given for every detection. Second, as discussed previously in this chapter, the low score of the traditional CV algorithm can be explained by the mismatch of dimensions between the ground truths and the detections.

By looking at the mAP only, it seems clear that the trained YOLOv3 models are a lot better than the traditional CV method. However, mAP is a bad metric to measure the performance of the traditional CV algorithm and should not be used to compare it to the performance of the deep learning methods.

5.3.2 Segmentation metrics

By looking at the mAP and the videos in Appendix B, there seemed to be a mismatch between the mAP and the actual results. Therefore, the segmentation performance for each method was computed. They are summed up in Table (4.2). These metrics seem to represent the results observed in Appendix B better than the mAP. The first metric that stands out is the precision. It is approximately the same for all methods, which is probably caused by the way the ground truths were created. They were created by segmenting the test images with certain thresholds and removing the false segmentations of the background. The detected bounding boxes were segmented with the same thresholds, so the segmentation almost doesn't segments pixels not segmented by the ground truth. Since precision is the ratio of intersection over the detection, it is almost 100% for all methods.

Since the precision is approximately 1 for all methods and IoU is a combination of accuracy and precision, the IoU and accuracy are similar, as is observed in Table (4.2). Since they are similar, only the segmentation accuracy will be discussed further.

YOLOv3-strawberry performs the best with an accuracy of 86.6%, then comes YOLOv3-tiny-strawberry with an accuracy of 85.5% and finally comes the traditional CV method with an accuracy of 75.0%. Even if the traditional CV method is based on a segmentation algorithm and reaches a high accuracy, the trained YOLOv3 models outperform it with a good margin.

5.3.3 Frames per second

The number of frames per second achieved by the three methods implemented in this thesis are summed up by Table (4.3).

The performance was evaluated in Google Colab for all the methods and the resulting videos are linked in Appendix B. The trained YOLOv3 models have a great advantage when using the GPU, but the traditional CV algorithm doesn't improve noticeably. This is because the YOLOv3 models have been optimized to work with GPU, which the traditional CV method has not.

When using a CPU, the traditional CV algorithm is clearly better than the trained YOLOv3 methods. It processes 3 frames per second, which is tolerable as the robot doesn't move too fast. On another CPU, the traditional CV algorithm processed 10 frames per second, but the measurement from Google Colab is used to compare with the trained YOLOv3 models. YOLOv3-strawberry processes 1 frame every 10 seconds because of the number of operations it must do per frame, this is way too slow. Since YOLOv3-tiny-strawberry has fewer layers than YOLOv3-strawberry, it processes approximately 1 frame per second. Based on speed performance, traditional CV should be chosen when using a CPU.

When using a GPU, the trained YOLOv3 models far outperform the traditional CV algorithm. The traditional CV method processes 3.5 frames per second, half a frame more than with CPU. YOLOv3-strawberry processes 16.6 frames per second and YOLOv3-tiny-strawberry processes 41.3 frames per second! If a GPU is available, the trained YOLOv3 methods should definitively be used instead of traditional CV.

5.3.4 Similarities

In the traditional CV method, the RGB color model was a problem since its segmentation was linear. A solution to this problem was to use the HSV color model since its segmentation is spherical. Instead of segmenting out a smaller cube from the cube representation of the RGB model in Figure (2.2), the HSV segments out a sphere from the RGB-cube.

Interestingly, it may be that the neural network has learned to segment similarly to the HSV-segmentation. By iterating over the training data over and over again, the neural network has made a statistical model of the RGB-cube. For every R, G and B values in the cube, the neural network will activate different neurons, that will output a different class belonging. It will have stored a probability of these values being a part of a strawberry or not.

5.3.5 Differences

A big difference in the two methods is the time and complexity of implementation. The traditional CV method took several weeks to implement. There was a lot of trial and error in order for the model to give good results. A detection algorithm that worked for one image may not work for the rest of the images. If this segmentation algorithm was to be used on another fruit or vegetable, it would take a lot of time to tailor it to the new object.

The YOLOv3 models, surprisingly, took under a week to implement to get good results. The first day was used to label training and test data with the labeling script. The

second and third day were used to configure YOLOv3 in Google Colab. The fourth day was used to train a model of YOLOv3-tiny with the labeled data. The fifth day was used to train a model of YOLOv3. If YOLOv3 was to be used on another fruit or vegetable, it would again take under a week to implement it and obtain good results.

Another difference between the methods is that the traditional CV algorithm can run on a CPU, but YOLOv3-strawberry and YOLOv3-tiny-strawberry require a GPU to reach a high enough FPS. A CPU is much cheaper than a GPU, so price is important to consider when deciding which method to use. If the budget is tight, the traditional CV algorithm might be better.

A third difference is that the YOLOv3 models can be trained further while the traditional CV algorithm can not. The dataset used to train the YOLOv3 models in this thesis was small, but the trained models still achieved high mAPs. With a bigger dataset of thousands of images, the models could become even more precise and detect strawberries that are not detected with the current models. The traditional CV algorithm has little potential to improve further, at least not as easily as the deep learning models.

Chapter 6

Conclusion

This chapter concludes this thesis and discusses further work that can be done by Saga Robotics to build on this thesis. The conclusion answers the question from the problem description; which of the two methods, traditional computer vision and deep learning, is the most suited to detect strawberries in images?

6.1 Conclusion

This thesis has explored the world of computer vision in order to detect strawberries in images. Three different methods of object detection have been presented, implemented and compared in this thesis. The first one is based on traditional computer vision and uses primarily a segmentation algorithm to detect strawberries. The two other ones are based on a deep learning framework that uses a single pass of a neural network to detect strawberries in an image.

When using the mAP as performance metric, the deep learning methods far outperformed the traditional CV algorithms, but this is because mAP is not suited to evaluate the performance of the traditional CV method. However, when using the segmentation IoU as performance metric, the traditional CV algorithm performs well with an accuracy of 75.0%, but the trained YOLOv3 models perform even better with accuracies of 86.6% and 85.5%. Another argument in favor of the deep learning models is the speed and ease of implementation. In less than a week, a functioning neural network was implemented that could detect strawberries better than the traditional CV algorithm that took several weeks to implement. When using a CPU, the traditional CV method was the fastest with 3.0 FPS, but when using a GPU, the trained YOLOv3 models were the fastest with 16.6 and 41.3 FPS.

To conclude, the trained YOLOv3 models based on deep learning are more suited to detect strawberries in images than the method based on traditional CV. YOLOv3-tiny-strawberry can be used instead of YOLOv3-strawberry if speed is an issue.

6.2 Further work

Unfortunately, the code implemented in this thesis has not been tested on the actual robot. This is in part because the strawberries in Norway are ready to be picked in June, but the deadline for this thesis is the 3rd of June. The code has instead been tested on videos, with the resulting videos found in Appendix B.

I would encourage Saga Robotics to implement YOLOv3-strawberry or YOLOv3-tiny-strawberry on the robot with a GPU and use the weights trained in this thesis. They should then test this implementation on the robot when strawberries appear in June.

To obtain higher mAPs and confidence scores, Saga Robotics should create a bigger training dataset with the labeling script implemented in this thesis or with the labeling script found in AlexeyAB's Github repository [17]. The images to label should be taken at different times of different days to augment the diversity of the training set. The images should also be taken at different distances from the strawberries for the same purpose.

To improve the training performance, new anchor boxes should be generated that suit the new training data. They can be generated by using k-mean clustering on the bounding boxes of the training dataset. Darknet has a built-in function that can compute these anchor boxes [17] based on a labeled dataset. This will create bounding boxes that have similar shapes as the strawberries in the training set.

Finally, YOLOv3 or YOLOv3-tiny should be trained with this new dataset.

Bibliography

[1] Saga robotics. https://sagarobotics.com/.

[2] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[3] Norges teknisk-naturvitenskapelige universitet. https://www.ntnu.edu/.

[4] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–.

[7] Google colab. https://colab.research.google.com/.

[8] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[9] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[10] John Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8:679–698, 1986.

[11] KeiichiA be Satoshi Suzuki. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30:32–46, 1985.

[12] Wikipedia. HSL and HSV — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/HSL_and_HSV, 2019.

[13] George H. Joblove and Donald Greenberg. Color spaces for computer graphics. *SIGGRAPH Comput. Graph.*, 12(3):20–25, August 1978.

[14] Chen Yu, Chen Dian-ren, Li Yang, and Chen Lei. Otsu's thresholding method based on gray level-gradient two-dimensional histogram. In *Proceedings of the 2Nd International Asia Conference on Informatics in Control, Automation and Robotics - Volume 3*, CAR'10, pages 282–285, Piscataway, NJ, USA, 2010. IEEE Press.

[15] Michael A. Nielsen. Neural networks and deep learning. *Determination Press*, 2015.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[17] AlexeyAB. darknet. `https://github.com/AlexeyAB/darknet`, 2019.

[18] Joseph Redmon. Yolov3 weights. `https://pjreddie.com/media/files/darknet53.conv.74`.

[19] João Cartucho. Mean average precision. `https://github.com/Cartucho/mAP`, 2018.

List of Tables

2.1 Example of non-maximum suppression with a north-east gradient direction. 8
2.2 Double threshold application. 8
2.3 Example of border hierarchy tree for binary image in Figure (2.9). 11

4.1 Mean average precision, average loss and average confidence score for the three methods implemented in this thesis. 32
4.2 IoU, accuracy and precision for the different methods. 33
4.3 Speed performance of the different methods implemented in this thesis. . 33

List of Figures

2.1 Strawberry before thresholding (left), after RGB-thresholding (middle) and after HSV-thresholding (right). The thresholding is done without filtering. 3
2.2 The RGB color spectrum in Cartesian coordinates (left) and the HSV color spectrum in cylindrical coordinates (right) [12]. 4
2.3 Strawberry before (left) and after (right) filtering, Canny edge detection and closing. 6
2.4 Strawberry without filter (left), with Gaussian filter (middle) and with median filter (right). Both filters have a kernel size of 19×19. 7
2.5 Strawberry in grayscale (left), result of horizontal Sobel operator (middle left), result of vertical Sobel operator (middle right) and combination of both results (right). 7
2.6 Result from Sobel operators before (left) and after (right) non-maximum suppression. 8
2.7 Result from non maximum suppression before (left) and after (right) double threshold application and tracking by hysteresis. 9
2.8 Result of hysteresis (left), result of dilation of hysteresis (middle) and result of closing, or erosion of dilation (right). 10
2.9 Result of filtering, Canny edge detection and closing (left), outer contours (middle) and inner contours (right). 11
2.10 Model of a perceptron [15]. 12
2.11 Sigmoid function used on the output of a perceptron. 13
2.12 Parameters of a perceptron represented as a line between two classes. . . 13
2.13 A fully connected neural network used to classify handwritten digits [15]. 14
2.14 A convolutional network for classifying handwritten digits [15]. 15
2.15 A convolutional layer [15]. 16
2.16 A pooling layer [15]. 17
2.17 Detection on a grid cell with YOLOv3 at first (left), second (middle) and third (right) scale with three bounding boxes per scale. 18
2.18 Darknet-53 architecture [2]. 18

2.19 Conversion from predicted to true bounding box coordinates [2]. 19
2.20 Detection on a grid cell with YOLOv3-tiny at first (left) and second (right) scale with three bounding boxes per scale. 20
2.21 Intersection and union for a prediction. 20
2.22 Representation of true and false positives and negatives on a prediction. . 22

4.1 Example result of segmentation, Canny edge detection and contour detection. 30
4.2 Example result of detection by drawing the bounding boxes of the contours in Figure (4.1). 30
4.3 Example of detection with YOLOv3-tiny trained on strawberries. 31
4.4 Example of detection with YOLOv3 trained on strawberries. 32

6.1 Average loss and mAP during training for YOLOv3-tiny. 52
6.2 Average loss and mAP during training for YOLOv3. 53

Abbreviations

AP	=	Average Precision
Avg.	=	Average
CNN	=	Convolutional neural network
CPU	=	Central processing unit
CV	=	Computer vision
cv2	=	OpenCV 2
FN	=	False negative
FP	=	False positive
FPS	=	Frames per second
GPU	=	Graphics processing unit
HSV	=	Hue, saturation, value
IoU	=	Intersection over union
mAP	=	Mean average precision
NTNU	=	Norges teknisk-naturvitenskapelige universitet
RGB	=	Red, green, blue
TN	=	True negative
TP	=	True positive
YOLO	=	You only look once
YOLOv3	=	YOLO version 3

Appendix

A. Source code

`https://github.com/pierrecham/TTK4900`

B. Video results

Traditional computer vision: `https://youtu.be/RVBzKU2U57s`
YOLOv3-tiny-strawberry: `https://youtu.be/7XlbTPyxWHU`
YOLOv3-strawberry: `https://youtu.be/d0QasyvV-v8`

C. Training graphs

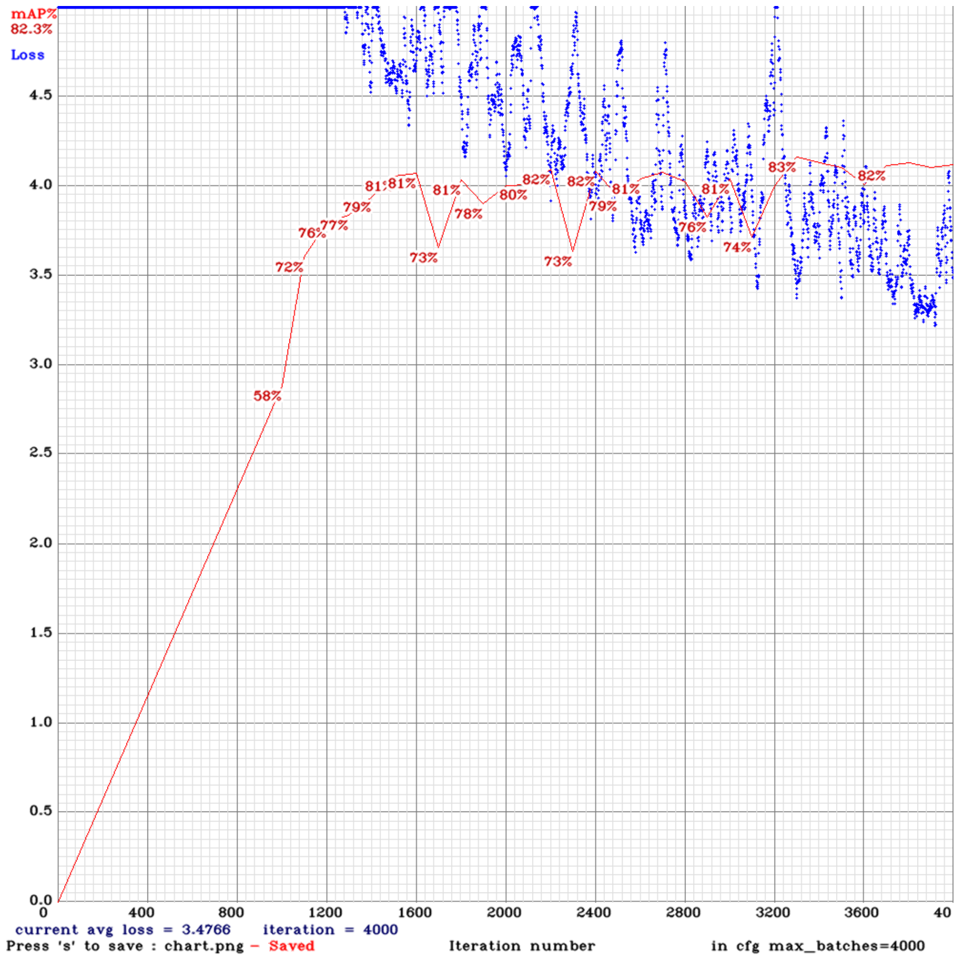


Figure 6.1: Average loss and mAP during training for YOLOv3-tiny.

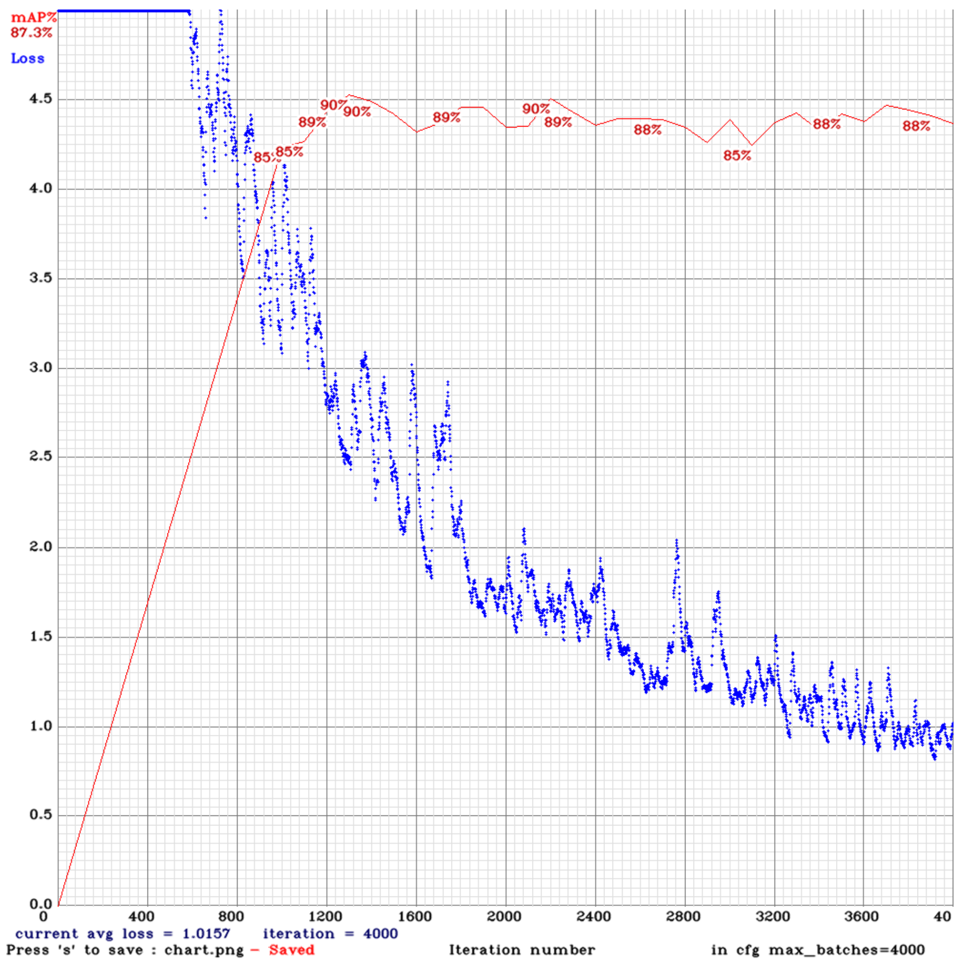
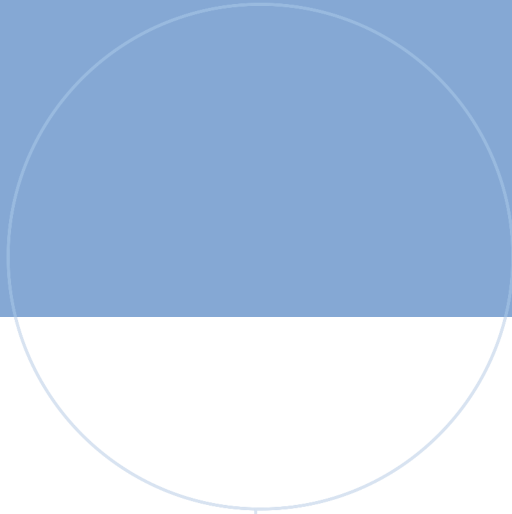
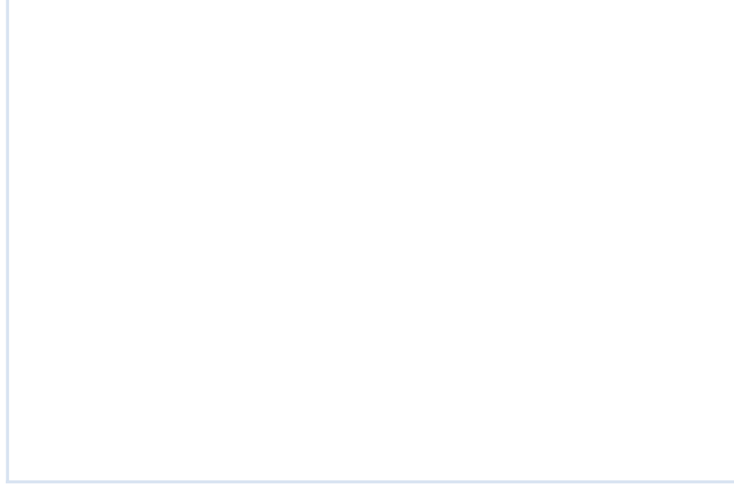


Figure 6.2: Average loss and mAP during training for YOLOv3.

NTNU
Norwegian University of
Science and Technology