NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

JOSÉ RAFAEL RIJO TRÊPO BATE

Bachelor in Electrical and Computer Engineering

APPLYING DEEP LEARNING TO ESTIMATE FRUIT YIELD IN AGRICULTURE 4.0 SYSTEMS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
november, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

APPLYING DEEP LEARNING TO ESTIMATE FRUIT YIELD IN AGRICULTURE 4.0 SYSTEMS

JOSÉ RAFAEL RIJO TRÊPO BATE

Bachelor in Electrical and Computer Engineering

Adviser: Ricardo Alexandre Fernandes da Silva Peres
Invited Professor, NOVA University Lisbon

Co-advisers: José António Barata de Oliveira
Assistant Professor, NOVA University Lisbon

Sara Vanessa Oleiro Araújo
Researcher, UNINOVA - CTS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
november, 2021

Applying Deep Learning to Estimate Fruit Yield in Agriculture 4.0 Systems

*Para os meus pais **Carla** e **Joaquim**, irmãos **Ivânia** e **Rúben**,*
*e namorada **Carolina**, sem vocês nada disto teria sido possível.*

Abstract

Over the last few years, with the advances in Information and Communications Technology (ICT) and the increasing human needs, industry has been reshaping itself. A new industrial revolution is emerging, and it is called Industry 4.0. This revolution intends to digitize the market and make it as intelligent as possible. As the history tells, every time there is an industry revolution, the agricultural sector benefits from it. Agriculture 4.0 is ongoing, and it is marked by intelligence and data. It aims to make the agricultural sector more efficient, that is: producing more outputs (such as food, fibers, fuel and other raw materials) while using less inputs (e.g. water, fertilizers, pesticides). Additionally, it envisions to promote food security, by reducing food loss and waste during the "Farm to Fork" journey.

A major challenge in the agricultural sector is forecasting food storage and marketing activities prior to harvesting. Nowadays, most farmers manually count fruits before harvesting, in order to estimate the production yield of their fields, as a means to manage storage and marketing activities. Manually counting fruits in large fields is a laborious, costly and time-consuming effort, which is often also error prone. A consequence of this outdated methodology is that it leads to food wastage, which can affect food security.

The developed work along this dissertation is an entry point to a system that is capable of estimating the production yield of a whole orchard, while being capable of respecting the required time constraints of each case study. With data taken with a smartphone, the developed system was able to accurately estimate the number of fruits present in tree sides, registering accuracies up to 98%. The high accuracy and speed results were possible due to the combination of state-of-the-art object detection and tracking techniques. To achieve this, a large model of Scaled YOLOv4 was combined with an online Multiple Object Tracking (MOT) framework based on Simple Online and Realtime Tracking with a Deep Association Metric (Deep SORT). Furthermore, this results validated the viability of implementing a proposed system, capable of estimating the fruit yield of a whole tree and, consequently, the production yield of the whole orchard, that is both low in complexity, easy-to-use, fast and reliable.

Keywords: Agriculture 4.0, Fruit Tracking, Production Yield.

Resumo

O avanço das ICTs, juntamente com as necessidades humanas, está a proporcionar uma nova revolução industrial, designada de Indústria 4.0. Esta revolução visa uma digitalização do mercado, assim como torná-lo mais inteligente. Sempre que uma revolução industrial toma lugar, o setor agrícola beneficia disso, herdando as tecnologias que fazem parte de tal revolução. A agricultura 4.0 está em progresso, e é marcada por inteligência e informação. Esta revolução tem como objetivo tornar o setor agrícola mais eficiente, isto é: produzir mais (por exemplo comida, fibras, combustível e outras matérias-primas) com menos (e.g. água, fertilizantes, pesticidas). Adicionalmente, esta revolução visa a promoção da segurança alimentar, através da redução da perda e do desperdício de comida.

Um grande problema no setor agrícola reside no planeamento de armazenamento e marketing de alimentos, antes da sua colheita. Na realidade, a maioria dos agricultores realiza o processo de contagem de frutos, do seu campo agrícola, manualmente, a fim de planear o espaço necessário para armazenar os mesmos e planear as suas vendas. A contagem manual de frutos é uma tarefa dispendiosa, que consome uma grande porção de tempo, tediosa, e propícia a erros. Uma consequência desta metodologia de trabalho é o desperdício alimentar, o qual leva ao comprometimento da segurança alimentar.

O trabalho desenvolvido ao longo desta dissertação é um ponto de partida para um sistema que é capaz de estimar o rendimento de produção de um pomar inteiro, e ao mesmo tempo capaz de respeitar as restrições temporais de cada caso de estudo. Através de dados adquiridos com um smartphone, o sistema desenvolvido é capaz de estimar o número de frutos presentes em faces de árvores, registando eficácias tão altas como 98%. Os resultados obtidos foram possíveis devido às técnicas implementadas, que contaram com a combinação de metodologias de estado de arte de deteção e rastreamento de objetos. Um modelo da arquitetura Scaled YOLOv4 foi combinado com uma framework baseada em Deep SORT capaz de rastrear múltiplos objetos numa sequência de imagens. Os resultados obtidos validam a viabilidade da implementação de um sistema proposto, que ambiciona ser simples, fácil de usar, rápido e fiável na contagem de frutos de uma árvore inteira e, consequentemente, na estimação do rendimento de produção de um pomar inteiro.

Palavras-chave: Agricultura 4.0, Rastreamento de Frutos, Rendimento de Produção.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

AGV Automated Ground Vehicle i, 2, 7, 10, 11, 26, 30
AI Artificial Intelligence i, 1, 5, 7, 8, 12, 15, 30
ANN Artificial Neural Network i, 11, 14, 16

CAP Common Agricultural Policy i, 3
CNN Convolutional Neural Network i, 16, 17, 24, 37, 38, 40, 49
COCO Common Objects in Context i, 5, 16, 17, 44, 50
CSPNet Cross Stage Partial Network i, 38

DCNN Deep Convolutional Neural Network i, 27, 30
Deep SORT Simple Online and Realtime Tracking with a Deep Association Metric i, v, vi, ix, 5, 24, 25, 39, 40, 51, 74, 76
DL Deep Learning i, 2, 3, 4, 5, 11, 12, 16, 24, 29, 37, 38, 39, 44, 59, 67, 72

ICT Information and Communications Technology i, v, vi, 1, 2, 3, 27
IoT Internet of Things i, 1, 7, 8, 9
IoU Intersection over Union i, x, 22, 23, 24, 39, 56, 59, 62, 64, 65, 67

KLT Kanade-Lucas-Tomasi i, 28, 55

LOOCV Leave-one-out cross-validation i, 14
LSTM Long Short Term Memory i, 22

mAP mean Average Precision i, ix, x, xi, 49, 59, 61, 62, 63, 64, 65, 66, 67, 69, 74, 76
ML Machine Learning i, 11, 12, 13, 14, 16, 27, 30, 51
MOT Multiple Object Tracking i, v, 4, 5, 21, 22, 23, 24, 25

NAS Network Architecture Search i, 38
NIR Near-Infrared i, 26, 30

ONNX Open Neural Network Exchange i, xii, 51, 54, 57

RGB Red-Green-Blue i, 3, 26, 29, 30, 32, 34, 35, 36, 40
RGB-D Red-Green-Blue-Depth i, 26, 28, 30

SfM Structure-from-Motion i, 28, 30, 41
SORT Simple Online and Realtime Tracking i, 23, 24, 39, 40
SVM Support Vector Machine i, 11

UAV Unnamed Aerial Vehicle i, 2, 7, 10, 26, 30
UGV Unmanned Ground Vehicle i, 2, 10, 11, 26, 30

WSL Windows Subsystem for Linux i, 51, 53, 74

YOLO You Only Look Once i, xi, 16, 17, 23, 37, 49, 51, 53, 54, 55, 59, 61, 62, 63, 64, 74

Chapter 1

Introduction

In this section will be realized a context and background about the topic, its challenges as well as the motivation on and how to approach them. In the end will be presented the outline for this dissertation.

1.1 Context and Background

Agriculture is one of the oldest human activities worldwide, as humanity has been relying on agricultural activities, since ancient times, to ensure its own survival. Following Merriam-Webster dictionary definition, agriculture is "the science, art, or practice of cultivating the soil, producing crops, and raising livestock and in varying degrees the preparation and marketing of the resulting products"[1]. Agriculture is in constant evolution in order to meet humanity needs. It has gone through some revolutions and is currently going through the fourth, called by some sources as “Fourth Agricultural Revolution” [1].

The global human population has been continuously increasing and, as a consequence, more food is required [2]. To this end, a larger amount of food has to be produced, while maintaining its availability and nutritional quality across the world, thus ensuring food security. To achieve prime results, several problems must be addressed first [3]. New ICT approaches must be applied, in order to solve certain issues regarding ecological problems, food safety and security, lack of digitization, and inefficient agri-food supply/value chain, among others [4]. Thanks to the ongoing fourth industrial revolution (Industry 4.0), such challenges are now facing a good probability of being solved, through the fusion of emerging technologies such as Internet of Things (IoT), Big Data, Artificial Intelligence (AI) and robotics towards agriculture [5]. The integration of Industry 4.0 technologies in agriculture is propelling a new agricultural revolution, labeled as "Agriculture 4.0" [6].

Analyzing both industrial and agricultural production processes, it is possible to distinguish a clear difference on the lack of automation capability in the agricultural production, which is a consequence of poor digitization and intelligence in this field [7]. Agriculture 4.0 aims to change that, reforming the classical agricultural processes and

[1] https://www.merriam-webster.com/dictionary/agriculture

promoting the digital farming transformation through the integration of ICT systems in such processes. The Fourth Agricultural Revolution is ongoing, and many of these integrations are already being a success (see section Agriculture 4.0 Systems).

An essential application domain of the agricultural sector is crop monitoring [6]. With the recent technologies being applied to agriculture, Automated Ground Vehicle (AGV), Unmanned Ground Vehicle (UGV), Unnamed Aerial Vehicle (UAV), and already existing tractors, equipped with a wide variety of sensors and cameras, are being used to collect images of agricultural environments [8, 9]. These integrated platforms aim to fulfill tasks such as mapping of agricultural plots, pest detection, characterization and sorting of fruits, amongst others. State-of-the-art models based on Deep Learning (DL) are being proposed to solve such imagery tasks, and are showing themselves as a powerful solution to address these problems [10].

1.2 Motivation and Approach

Following the World Food Summit definition, "Food security exists when all people, at all times, have physical and economic access to sufficient, safe and nutritious food that meets their dietary needs and food preferences for an active and healthy life" [11]. To this end, an important task is to ensure that all produced food is properly stored and distributed. According to the United Nations, more than 30% of all produced food is not consumed due to two factors, food loss and wastage [4]. A portion of the wasted food is the reflection of the poor planning that is done prior to food harvesting, which often leads to miss storage. One way to minimize food wastage is through crop monitoring and estimation of the production yield. With the estimation of the production yield, it is possible to better plan agricultural activities, marketing activities, and achieve a more efficient storage system, thus avoiding food wastage. Nowadays, this estimation is manually done, which translates in a costly, tedious, laborious, time-consuming and error prone activity. This estimation is performed by counting the number of fruits in a sample of trees and statistically escalating that number to the whole orchard. When considering that there are orchards carrying hundreds of thousands of trees, this process may prove to be ineffective, as not all trees carry the same amount of fruits [12]. The same methodology is used in the surveyed literature, but instead of manually counting the number of fruits, a system is able to do it through complex image and other processing techniques. Although some of the techniques proposed by the literature are efficient in estimating the production yield, the majority of them require massive processing power, which, consequently, demands for large amounts of time to perform an accurate estimation of a row of trees [13]. The complexity and amount of time required by these solutions makes it unfeasible for them to estimate the production yield for large areas of agricultural crop.

Farmers and owners should be motivated by the agricultural digitization, as it is an enabler to assist with this and other crop challenges. On top of it, agricultural organizations

who try to adopt ICT technologies, benefit from initiatives like the Common Agricultural Policy (CAP). With all of this in mind, a fast and efficient system, capable of estimating the production yield of a whole orchard, in an autonomous manner, is necessary [14].

The Portuguese orchard in study, "Herdade Corte Romeira", located near Ajustrel (Beja), kindly agreed to provide access to their orchard along the implementation of this dissertation. This orchard is composed of 22 hectares of peach/nectarines, 23 hectares of apricot, 17 hectares of plums, 18 hectares of pears, and 8 hectares of apples. The fruit in study is a peach of the type nectarine "gardeta".

Figure 1.1: Portuguese orchard in study, "Herdade Corte Romeira".

During the guided visit to the Portuguese orchard, while having direct contact and talking with the actual farmers who know exactly what happens in their fields, it was concluded that a system able to autonomously count the number of fruits carried by a tree, and consequently estimate the production yield of the field, is essential. This system could be used not only to forecast how many fruits are to be harvested, but also in prior phases during the growth of the fruit, such as assisting in deciding whether it is necessary to apply chemical thinning[2] to a tree or not. Talking directly to the groups of people who do the actual job and knowing that this type of system is necessary was a huge motivation to conduct a study around it.

Having this into account, a framework that is capable of autonomously estimate the production yield of a whole orchard was proposed in chapter 3. For the proposed system to be implemented in its integrity, it is first necessary to assess the viability of using an ordinary smartphone to acquire Red-Green-Blue (RGB) data of a row of trees and use that same data to try and estimate the fruit yield in each side of that same tree row. The work developed during this dissertation was exactly to assess this premise. As such, RGB data of both sides of a tree row was acquired using Google Pixel 4A. This data was then used to detect and track fruits as a means to estimate the fruit yield in each side of the tree row. To perform the detection and tracking, state-of-the-art DL techniques were implemented,

[2] https://ag.umass.edu/fruit/fact-sheets/hrt-thinning-apples-chemically

which allowed for accurate and fast fruit yield estimation results. This system shown itself as being fast and reliable enough so that the remaining phases proposed in chapter 3 could be implemented in future work, which would allow for a total fruit count of a tree and consequent estimation of the production yield of a full orchard.

1.3 Research Questions & Hypothesis

To develop an easy-to-use system that can be both suitable for fast and accurate fruit yield estimation, four main Research Questions (RQs) must be addressed, namely:

- **RQ1:** What is the best and most convenient way to acquire the necessary data for the system to estimate the production yield of the crop?
- **RQ2:** How to detect every nectarine presented among all the nectarine trees?
- **RQ3:** How to avoid the overcounting of fruits in each side of a tree row, which is a consequence of a given fruit being considered multiple times along different viewpoints?
- **RQ4:** How to filter out the overcounted fruits when merging the fruit counts from both sides of a tree row?

For the first research question, it is proposed that an ordinary smartphone is used to acquire all the data. Although it greatly increases the complexity of the system due to poor photo quality, a smartphone is widely accessible by normal people, which makes it a viable choice. For simplicity, in the context of this work the production yield is considered as the number of counted fruits. The second question will be addressed using a state-of-the-art DL object detector. The third research question will be addressed using state-of-the-art MOT algorithms. The developed system in this dissertation fits into the first three research questions. Due to the seasonality of the fruits, along with the COVID pandemic situation and the six month time scope of this dissertation, the fourth research question must be implemented in future work, as a means to filter out the duplicated fruits when merging the counts from both sides of a tree row, and thus, obtain a full estimation of the production yield.

Chapter 3 proposes state-of-the-art methods to accomplish data acquisition, object detection, and accurate production yield estimation.

1.4 Main Contributions

The excitement around the image processing field has been growing, as has the research around it. DL-related solutions are presenting themselves as state-of-the-art when talking about object detection and tracking, and new ones are being released at an astonishing

pace. Continuously testing this new solutions is of importance, as they can increase the efficacy of systems relying on them.

One of the contributions of this dissertation was testing state-of-the-art solutions regarding object detection and object tracking towards the problem of estimating the production yield of a crop. A large Scaled-YOLOv4 model [15], which belongs to the current known best object detectors according to the Common Objects in Context (COCO) benchmark, was implemented to solve the fruit detection problem. For fruit tracking, FastMOT [16], a MOT framework based on Deep SORT, was implemented.

These DL techniques makes it possible for a faster and accurate estimation of the production yield. Therefore, unlike the already proposed systems, where the estimation of the whole orchard has to be statistically calculated, ours is an enabler to an estimation of the production yield of the whole orchard, which makes it less prone to error. To the best of our knowledge, it is the first time someone is implementing a state-of-the-art object detector (Scaled YOLOv4) and tracking (Deep SORT) techniques to detect and track fruits.

Lastly, the results obtained in estimating the fruit yield of tree row sides using an ordinary smartphone are a contribution itself. These results enable the continuity of the development of the remaining phases of the proposed system in chapter 3, which would allow for a full estimation of the production yield of a whole orchard.

1.5 Dissertation Outline

The current document is organized into six chapters, namely:

- **Chapter 1 (Introduction)** includes a context on the proposed work, the motivation for this dissertation, research questions and hypothesis, its main contributions, and the structure of the document;
- **Chapter 2 (Related Work and Technologies)** presents supporting concepts around Agriculture 4.0 and AI (which includes object detection and tracking techniques), and related work regarding estimating the production yield of an agricultural crop;
- **Chapter 3 (Proposed Framework)** proposes a framework capable of estimating the production yield of a whole orchard. This proposal includes how to implement the three-step process. That is, how to acquire data about the crop, how to implement a state-of-the-art object detection system to detect the fruits, and how to estimate the fruit yield of a tree side while avoiding the over count of fruits;
- **Chapter 4 (Implementation)** dives into the details of how the solution to count fruits in tree sides was implemented;
- **Chapter 5 (Results and Discussion)** exposes the obtained results as well as a discussion on what conclusions can be extrapolated from such results;

- **Chapter 6 (Conclusion and Follow Up)** sums up all the conducted work, answering the proposed RQs, and finally addressing a brief discussion of future work that can follow this dissertation.

Chapter 2

Related Work and Technologies

Human population is growing at a consistent rate and it only aims to keep this way. It is expected a population growth of 25% over the next 30 years [17], representing a total of 10 billion humans living in a small world by 2050 [18]. This will lead to an increase in the demand for countless human needs, especially food. The solution to this problem is to increase productivity in agriculture, while maintaining the high nutritional quality of agri-food products and protecting the natural ecosystems. Industry 4.0 brings emerging technologies towards agriculture, such as AI, IoT, Big Data and robotics (for instance, AGVs and UAVs) [4] are being presented to solve such huge food-demand problem in the coming years. The employment of these emerging technologies in the agricultural sector is often recognized as Agriculture 4.0 [6, 17, 19], smart farming [20] or precision agriculture. In the following sections, supporting concepts and related work to this dissertation will be addressed.

2.1 Agriculture 4.0

Since the beginning of times, agriculture, along with industry, has undergone four revolutions (figure 2.1) [7].

Starting on the Neolithic age with Agriculture 1.0 [21], marked by the shift from hunter-gatherer lifestyle to sedentary farming with plant and animal domestication. Agriculture 1.0 heavily relied on indigenous tools like pitchfork, sickle, and hoe for cultivation. Those practices stuck until the end of the 19th century, showing themselves as requiring a lot of manual labor and low productivity. The first industrial revolution (Industry 1.0) took place during the 19th century, being defined as the transaction from hand production (human force) to machine production (steam and water powered engines). As a consequence of Industry 1.0, a new revolution on agriculture emerged, named Agriculture 2.0. In this agricultural revolution, machinery assisted human labor in seedbed preparation, sowing, irrigation, weeding, and harvesting. With this, it was possible to save an enormous amount of time and work for the human side. Mechanized agriculture was a huge leap, as it greatly increased food production and reduced manual labor.

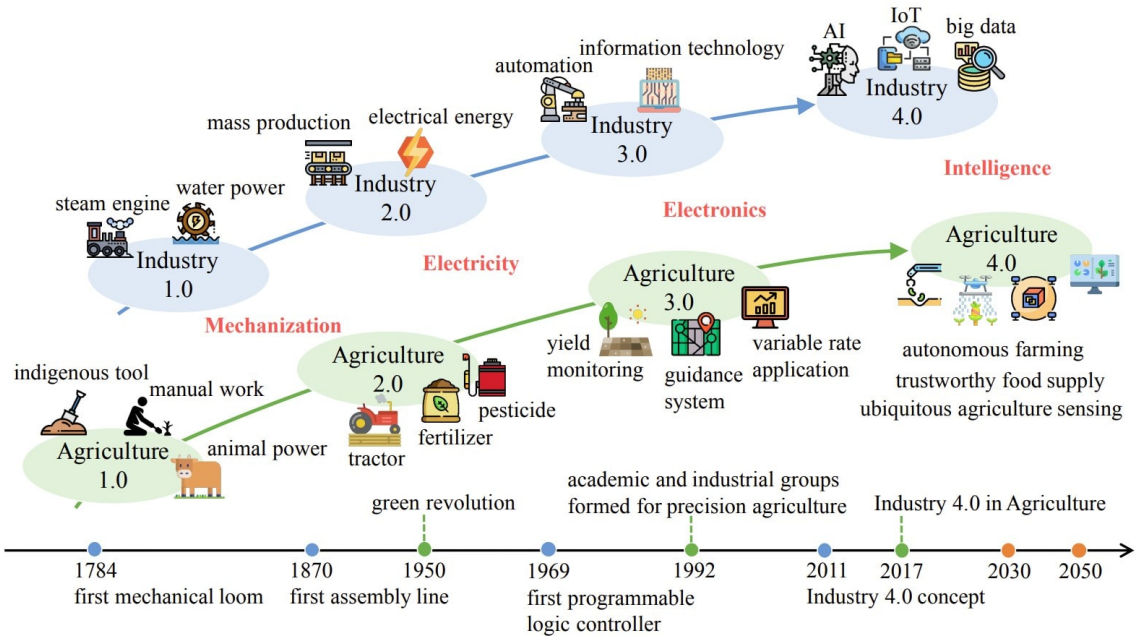

Figure 2.1: History of agricultural revolutions. Figure taken from [4].

The third agriculture revolution was the consequence of technologies and methodologies brought by the second (Industry 2.0) and third (Industry 3.0) industry revolutions. Industry 2.0 was defined by electricity, electronics, mechanical devices, and cars [22]. The new energy sources introduced by this industrial revolution, together with innovations in the transportation systems, significantly improved the efficiency and productivity of the agricultural sector, as it was an enabler for a better agri-food supply chain and for a mass production assembly line-based model. Industry 3.0 is mostly characterized by technological innovations, especially on electronics, which improved the automation capability of manufacturing equipment. During this period, green energy sources, such as photovoltaic power, wind power, and hydroelectricity solutions were (and still are) being explored. All these improvements brought by Industry 2.0 and 3.0 led to a recent agricultural revolution, known as Agriculture 3.0. Currently, we are amidst the era of Industry 4.0 and Agriculture 4.0. The fourth agricultural revolution is defined by intelligence, and, like the precedent revolutions, is a result of the Industry 4.0. The following subsections will introduce some applications brought by Agriculture 4.0.

2.1.1 Agriculture 4.0 Systems

Agriculture 4.0 is characterized by a fusion of emerging technologies brought by Industry 4.0, such as IoT for connectivity between "things", smart sensors and robotics to monitor and control the agricultural field, Big Data and AI approaches for data analysis, blockchain technology for food and process traceability, among many others. Agriculture 4.0 aspires to increase the crop productivity in a sustainable manner, i.e., considering economic, environmental and social aspects. To this end, there are several tools, techniques

and technologies that can be applied to achieve a more precise and smarter farming. This section aims to explore and explain some state-of-the-art techniques used in this field.

Internet of Things

The concept of IoT devices is to integrate the "things" in the physical world with the virtual world by using the Internet as the medium to communicate and exchange information [23]. These devices consist of embedded systems which interact with sensors (IoT sensors) and actuators (IoT actuators) through wireless connectivity. IoT sensors can be classified into five categories, namely: mechanical sensors, location sensors, optical sensors, airflow sensors and electrochemical sensors [24]. Composed by field programmable gate arrays or microprocessor, communication modules, memory and input/output interfaces, these sensors are widely used in Agriculture 4.0, as they possess key characteristics like power efficiency, memory, computational efficiency, portability, durability, coverage, reliability and cost, thus making them a suitable option for agricultural practices [23]. They are helping transforming agriculture into a more data-driven field, by gathering valuable data from the fields, such as soil temperature, moisture, electric conductivity, pH at various depths, air temperature, rainfall, leaf wetness, chlorophyll content of crop leaves, wind speed, dew point temperature[1], wind direction, relative humidity, solar radiation, atmospheric pressure, among many other parameters [23, 24].

Some key applications of IoT in agriculture are described below:

- Monitoring: For example, crop monitoring, is of huge importance as there are several environmental factors that can affect agricultural production. The acquisition of data such as the amount of rainfall, air and soil temperature, leaf wetness, air and soil moisture, salinity, solar radiation, pest movement, etc., enables optimal decision making processes. Hence, it is possible to improve the overall agricultural production and the quality of the farm products, to minimize risks and to maximize profits, always considering sustainable aspects (environmental, economical and social). Taking the soil radiation example: data acquired by IoT sensors gives information about the plants exposure to sunlight. That data is presented to farmers in a readable way, so that they can identify if their plants are properly exposed or over exposed to solar radiation. Another example is the collection of data related to plague movements, which is remotely fed to the farmers for pest control, avoiding big losses due to pests infestations and infections [25, 26]. Sensors integrated in agriculture-related machinery can be useful to collect relevant data, in instance, to establish fertilizing, irrigation, nutrition plans, or planning harvesting activities.

[1] "The dew point is the temperature the air needs to be cooled to (at constant pressure) in order to achieve a relative humidity (RH) of 100%. At this point the air cannot hold more water in the gas form." Source: https://www.weather.gov/arx/why_dewpoint_vs_humidity

- Control: For example, remote control, can be achieved using UGVs, UAVs, and other agriculture-related machinery. While monitoring is the practice of gathering relevant crop data, remote control makes a system react over that information. Based on available information provided by sensors, UGVs and UAVs can take proper actions in the fields, for instance, to automatically spraying the fields with pesticides, herbicides or other products [9, 27].

Agriculture-related machinery

In this context, agriculture-related machinery relates to UGVs, UAVs, and other agriculture-related transportation systems. These systems are being massively used in precision-spraying (e.g. water, pesticides, herbicides, and fertilizers), seed sowing and growth assessment, mapping, crop disease detection and yield estimation [8, 9, 28].

Some common applications of these systems in agriculture are:

- Mapping, which can provide useful information about the area of the farmland, soil conditions, and status of the crops. This information allows more profitable agriculture tasks, such as the agronomic control of homogeneous zones and the separation of fruit quality areas [29].
- Spraying application using UAVs can reduce pesticide use and maximize efficiency when compared to a speed sprayer or a wide-area sprayer [27]. Besides UAVs, UGVs can also be an effective solution to precision spraying, when equipped with cameras and smart algorithms capable of picking regions of interest [30].
- Crop Monitoring, which is essential for optimal production, and is used to predict the yield or quality of a crop via analysis of crop data. Usually this task is accomplished via satellite, which are typically used to scan very large farms. The problem associated with satellite utilization is it's poor precision of crop monitoring, when compared with other alternatives. Using UAVs to monitor a large farm is promising as it saves significant time and labor for the human side. Using smart and autonomous systems such as AGVs is useful in diagnosing insect pests, where early diagnosis is essential as it inhibits damage spreading quickly [31].
- Irrigation is precise when using smart transportation systems, as they can be equipped with multi-spectral cameras and heat sensors, which can identify areas where water is scar, avoiding over-irrigation [32].

For UAVs, they can be classified into two types of platforms: fixed and rotary-wing [8, 9]. A fixed-wing UAV is somewhat similar to an airplane, it flies via thrust and aerodynamic lifting force. It is mainly used for spraying and photographing over wide ranges. A rotary-wing UAV can be classified into helicopter and multi-rotor types. The helicopter type features a single, large and powerful propeller atop the aircraft. On the other hand,

the multi-rotor types have many smaller propellers, and are classified as quadcopter (four propellers), hexacopter (six propellers), and octocopter (eight propellers), and are used for extremely precise tasks. Helicopter types and octopters, like the fixed-wing, are mainly used for spraying. Quadcopters and hexacopters are used for field reconnaissance and mapping.

As for the ground vehicles, UGVs and their autonomous version, AGVs, comes usually in varying forms, personalized for specific needs. As an example, [33] conducted an experiment of estimating the production yield, where the data acquisition was performed by "Shrimp", "a general purpose perception research ground vehicle", as cited by the authors. This ground vehicle is equipped with a variety of localization, ranging and imaging sensors, which makes it a good choice for imagery tasks, such as data acquisition. Besides UGVs and AGVs, a common and still valid application is to take advantage of the already existing tractors used by farmers, and equip them with the necessary material for specific tasks. On the one hand, using tractors is beneficial as they are often already available. On the other hand, their use can be a limiting factor for specific tasks such as taking more detailed photos / more viewpoints of fruit trees for later processing.

A revision on agricultural ground and aerial robots done in [34], show that the use of these in tasks such as monitoring plant health, ripeness, and soil moisture can be beneficial as they replace human labor on the field during adverse weather conditions, hence protecting their health. More studies show how ground vehicles can be of significant help in tasks such as performing monitoring and inspecting on soy and cotton crops [35], autonomous crop inspection [36], and autonomous cotton crop monitoring [37].

2.1.2 Image Processing in Agriculture 4.0 Systems

Agriculture has benefited of image processing methods in the past years. Classic image processing techniques, such as the appliance of preprocessing, segmentation, extraction of characteristics, either using Machine Learning (ML), Support Vector Machine (SVM) and/or Artificial Neural Network (ANN), have been proposed solutions for problems like pest detection [38], characterization and sorting of fruits, amongst others [39, 40]. Recently, a new technique is standing out, namely DL [10], which will be further detailed in section 2.2. DL is a sub-field of ML, and is similar to an ANN, but is "deeper" (has a bigger number of layers) than a normal ANN, which allows larger learning capabilities and thus higher performance and precision [41]. There has been a lot of research related to image processing using DL applied to agriculture, and it is being used for pest detection [42], insect classification [43], weed identification, soil and vegetation/crop mapping, plant recognition, fruit counting, crop type classification, among other applications.

2.2 Artificial Intelligence

> *"The art of creating machines that perform functions that require intelligence when performed by people." Raymond Kurzweil 1990.*

AI is usually defined as the science of making computers do things that require intelligence when done by humans[2]. AI has always been excellent at solving problems that are intellectually difficult for human beings, problems that can be described by mathematical rules. The field where computers lack the most is when they try to solve problems that are rather easy for humans but hard to formally describe, that is, problems that are solved by intuition, that for a human its solutions is trivial and automatic, like recognizing spoken words or faces in images. A proposed solution to this problem is to create a system that let's a computer learn from experience. Such systems reside in the sub-fields of AI, namely ML and DL [44–46].

2.2.1 Machine Learning

ML[3] can be defined as an ability that lets computers acquire their own knowledge by extracting patterns from raw data, enabling it to improve their accuracy over time without being explicitly programmed to do so. ML is based on defining statistical models (algorithms) which are as good as the amount of data they are fed. In the literature, there are four methods through which a machine can learn [47]: supervised, unsupervised, semi-supervised and reinforcement learning.

Supervised Learning

As the name hints, supervised learning is when a machine is supervised while learning [48]. In this method of learning, the machine is fed with labeled data, meaning that it is already tagged with the correct answer of what it represents. Suppose there is a machine ready to learn through a supervised learning algorithm, and the goal of the problem is for the machine to be able to classify whether a fruit is rotten or not. For the learning process, the machine is fed with data, which is composed of fruit decay signs such as "fruit has bruises? yes or no", "fruit has an unpleasant odor or taste? yes or no", and, because the learning model is supervised, at the end of each fruit decay signs there is a column that says if it is rotten or not. With this labeled data and through a statistical algorithm, the machine is able to module such a pattern that lets it identify if future fruits may or may not be rotten.

[2] http://www.alanturing.net/turing_archive/pages/Reference%20Articles/What%20is%20AI.html
[3] https://www.ibm.com/cloud/learn/machine-learning

Unsupervised Learning

On the other hand, in unsupervised learning, the machine works on its own to discover patterns in data [49]. In this type of learning method, the machine ingests lots of unlabeled data and uses complex algorithms to extract meaningful patterns and relationships in data that humans would miss. The task of the machine is mainly to group unsorted information according to similarities, patterns and differences. For this, it uses two types of algorithms, namely clustering and association. A good use of unsupervised learning is in email spam detection, where the machine can analyze huge volumes of emails and uncover the patterns and features that indicate spam. This type of learning is used when there are huge amounts of data and labelling all of it becomes impractical.

Semi-supervised Learning

Semi-supervised learning is in between supervised and unsupervised learning [50]. It is a process of learning where the machine uses a sample of a labeled dataset (supervised) to guide itself over an unlabeled dataset (unsupervised). Suppose there is a dataset consisting of labeled and unlabeled data regarding whether a fruit is rotten or not. A classifier can be trained with the labeled data, like it was discussed before, but there is still a huge amount of unlabeled data remaining, which could potentially increase the classifier accuracy. Through a semi-supervised learning technique, denominated as "pseudo-learning", it is possible to use the remaining unlabeled data to re-train the classifier. Using this technique, the already trained classifier will "pseudo-label" the unlabeled data, and then use this pseudo-labeled data to re-train itself. This process makes it possible to use unlabeled data, that would be thrown away due to the impracticability of labelling all of it, to increase the accuracy of a classifier.

Reinforcement Learning

Reinforcement learning is based on agent-reward-penalty learning, where the training and testing phases are intermixed [51]. The agent interacts with the environment, and, depending on its interaction, it can be rewarded or penalized, like a trial and error system [52]. The goal of the agent is to maximize its rewards along its path in an environment. A sequence of rewards in a given environment achieved by the agent results in a reinforced process, capable of solving a given problem.

The Learning Process

As mentioned before, in order for a computer to acquire enough knowledge so that it can solve a problem that is formally difficult for a human to describe, it needs to be trained, and that happens through the extraction of patterns from raw data. Therefore, the first thing to do while building an ML model is to select and prepare a training dataset.

Preparing a dataset consists in randomizing it, checking for biases that could impact the training, and divide it into two subsets, one for training, which will be used to train the model, and one for evaluation, which will test how good the model is and refine it. There are two types of data, labeled and unlabeled. Labeled data is sometimes used to call out features and classifications the model will need to identify. On the other hand, if the dataset is composed of unlabeled data, it means that the model will need to extract those features and assign classifications on its own.

Once the dataset is prepared, it is necessary to choose a way for the machine to learn, that is, the statistical algorithm to run on the training dataset. The chosen algorithm depends on whether the data is labeled or unlabeled. For labeled data, there are common types of ML algorithms such as regression, decision trees and instance-based algorithms (e.g. K-Nearest Neighbor). For unlabeled data, there are, mainly, clustering and association algorithms, which are more complex than the algorithms used for labeled data [47].

Once the model is built, the last step is to reevaluate and validate its accuracy, through methods such as bootstrapping or Leave-one-out cross-validation (LOOCV) [53, 54]. Building and training the ML model can be time-consuming and sometimes challenging, but, once it is done, classifying new data is as easy as feeding it to a computer and instantly obtaining a result. That being said, once all the steps are successfully accomplished, all there is left to do, is use it, for example, to support in decision making.

Artificial Neural Networks

When a human receives any kind of information, its brain processes it through a series of neurons, producing a certain output. ANNs are computing systems that are inspired by the human brain neural networks. The term "neural networks" used in ML doesn't actually refer to biological neural networks, they simply share some characteristics with biological neural networks, and for this reason, they are called as "Artificial Neural Networks"[4].

ANNs are generally composed by three layers. As it works as an input-output system, the first and last layers are the input and output layers. Between them there are the hidden layers, that perform the mathematical operations that help determine the output of the ANN. Those hidden layers manipulate the input data using mathematical operations, and, at the output layer, that data is converted into an output, through an activation function [55]. The hidden layers are composed by artificial neurons, that perform arithmetical operations in order to produce an output. All the layers are connected between them, and all those connections have associated weights. In most cases, ANNs are fully connected, which means that all the neurons of a given layer are connected to all the neurons of its neighbour layers and so on. In order for an ANN to give a desired output when presented with a certain input, it needs to be trained. Training an ANN consists in readjusting its weights after each iteration (input-output), through backpropagation. Lets say an ANN

[4] https://deeplizard.com/learn/video/OT1jslLoCyA

is supposed to give 1 as output, but instead it gives 0.6. In order to increase its precision, the weights are adjusted so that, in the next iteration, the result is slightly closer to 1. In this way, iteration after iteration, the weights are constantly adjusted in order for the neural network to achieve the best possible prediction accuracy. The train should be conducted in a way that avoids the neural network to overfit. This phenomena occurs when a neural network starts memorizing instead of learning. It happens when a neural network is presented with a certain dataset a huge number of times, and fits too much into that dataset. Technically speaking, overfit occurs when the weights are so biased to correctly classify the training dataset, that when the neural network is presented with another example from another dataset, it mostly miss classifies its classes.

A major problem of neural networks was the vanishing gradient problem [55]. For networks with one or two layers, this problem was not a big of a deal, but when using more than two layers, the network was rather hard to train, thanks to the vanishing gradient problem. As mentioned before, a neural network is trained through backpropagation and readjustment of weights. When training a neural net, one has to calculate the gradient of the loss with respect to weights of the network. When there are networks with many layers, the gradient with respect to weights in the first layers become small, like vanishingly small. Hence, vanishing gradient. This small gradient happens due to the type of activation functions used in the network. When back propagating a neural network, one has to use the chain rule, in order to find the gradients of upper layers. Using the chain rule, we have that the derivatives of each layer are multiplied through the network, starting on the output layer, all the way up to the input layer, in order to compute the derivatives of the initial layers. The problem is, when there are many n layers in the network and an activation function like the sigmoid function is used, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we make our way up through the network. As said before, this will make the gradient of the first layers become really small, which will result in a really small update in the weights of the initial layers of the network. This will make the network hard to train, since there is a vanishingly small change every time an epoch is made. This problem worsens with respect to the number of layers in the network. The more layers a network has, the worse the vanishing problem is.

Neural networks with more than one hidden layer are often said to be deeper neural networks, which are the type of artificial neural networks used in deep learning. Since these networks are composed of a large number of layers, they weren't of big attention due to the vanishing gradient problem.

2.2.2 Deep Learning

Thanks to the research on AI, more specifically on neural networks, new activation functions were proposed in order to avoid the vanishing gradient problem [56]. These new activation functions enabled the use of deeper neural networks, later named deep neural

networks. DL is a subfield of ML, which has been rapidly evolving due to the exponential improvements on computing power and storage systems. It achieves great results by learning to represent the world by simple, yet many, concepts. DL has showed itself as being a powerful tool to enable a computer capability of classifying, detecting and recognizing. In other words, it is enabling computers to understand a certain task that they are assign to do, rather than just following a couple of rules. The fields where DL is most used and shining the most is, ironically, the fields where humans and animals have no trouble in shinning, which is, for example, speech and image recognition [57–60]. Systems like Cortana, Siri, and Tesla autopilot are based on DL. Many authors refer to the term DL as being "*large deep neural nets*"[5].

Due to the astonishing and promising results DL has shown when used in image processing, this dissertation proposes a system based on a type of DL networks, namely convolutional neural networks.

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of neural network that is prepared to work with image and video data. Most generally, an CNN is a deep ANN that has some type of specialization that enables it to detect patterns and make sense out of them. This pattern detection is what makes CNNs so powerful for image analysis. Unlike regular neural networks, CNNs hidden layers are mostly composed of convolutional layers, hence the name, convolutional neural networks. The difference between a normal layer and a convolutional one, is that, while a normal neuron on a normal layer receives an input and transforms it in some way and outputs the result to the next layer, a convolutional layer also receives an input but the transformation is a convolution operation [61].

Detecting patterns in CNN layers refers to detecting image features. Those features vary from the most simple, like detecting edges and shapes, to the most complex, like detecting whole animals and cars. The level of complexity of features increases with the depth of the network.

There are several complex DL object detection models proposed by the community, which are based on years of research and improvement. The golden standard benchmark for evaluating object detection models is the COCO dataset. COCO dataset contains over 120,000 images for training and testing, with 80 classes that represent common objects. Usually, if a model does well on the COCO dataset, it is believed it will do well in specific tasks. According to the COCO dataset benchmark, the current best performing state-of-the-art object detector model is Scaled YOLOv4 [15].

Scaled YOLOv4

YOLO has been well known among the image processing community, ever since its first publication [62], as being one of the best and fastest state-of-the-art object detection

[5]Jeff Dean - Leader of Google AI.

model. In the last year, YOLOv4 was published [63], stating itself as being the most reliable when talking about velocity and detection quality. However, in low frequency, EfficientDet, among other models, performed better. Recently, Scaled YOLOv4 was proposed, clearing all doubts about what the best object detection algorithm is. This new version of YOLOv4 topped all the algorithms in the COCO dataset benchmark, and, for this reason, is now the best performing object detection model. Let us start by explaining what YOLO is and how it is built. We will then talk about the evolution of YOLO, and, finally, what were the enhancements it needed to become the current state-of-the-art object detection model, Scaled YOLOv4.

Back in 2016, when the first version of YOLO was published, it was considered the fastest object detection system ever made. The reason behind this lies in the way the algorithm is conducted. The most popular algorithms back then were based on sliding windows. Those algorithms would analyze multiple regions of an image, sliding a "window" all through the top left of an image to the bottom right of it. These type of systems required multiple iterations through an CNN, which translated in slow computation times, unpractical for realtime detection. Optimized models like R-CNN used "region proposals" to first generate potential bounding boxes in an image and then run a classifier over the proposed boxes. These systems also required heavy computational power, hence becoming extremely slow for realtime detection tasks. YOLOv1 proposed a new way to approach object detection. Using this system, You Only Look Once at an image, that is, the image passes once through an CNN to predict what objects it contains and what are their localization. YOLO system starts by dividing an image into a grid of SxS cells. If the center of an object falls into a cell, that grid cell is said to be responsible for predicting it. Each grid cell predicts B bounding boxes and, to each bounding box, a confidence score is assigned. This confidence score represents how likely it is for a box to be contained in an object. For each grid cell, the algorithm will compute the probability of each class being present on the cell. This probability is calculated only once for each class, independently of how many boxes are presented in that grid cell. The following step is to assign a class probability to each bounding box presented in a given grid cell. To do this, the conditional class probabilities and the individual box confidence predictions are multiplied. Figure 2.2 sums up this model.

The YOLO model CNN architecture is inspired in GoogLeNet model. The initial layers of this CNN are feature extractors, and the last layers are fully connected layers, used to predict the output probabilities and coordinates of the objects presented in the image. The YOLOv1 network architecture is composed by 24 convolutional layers followed by 2 fully connected layers.

Four versions of YOLO have been released. On YOLOv2 [64], the authors focused mainly on improving object detection localization, while maintaining classification accuracy. YOLOv3 [65], as cited by the authors, has a "bunch of small changes that make it better". YOLOv4 [63], released in the last year, was made so that everyone who owns a conventional GPU can train and achieve realtime, high quality object detection results.

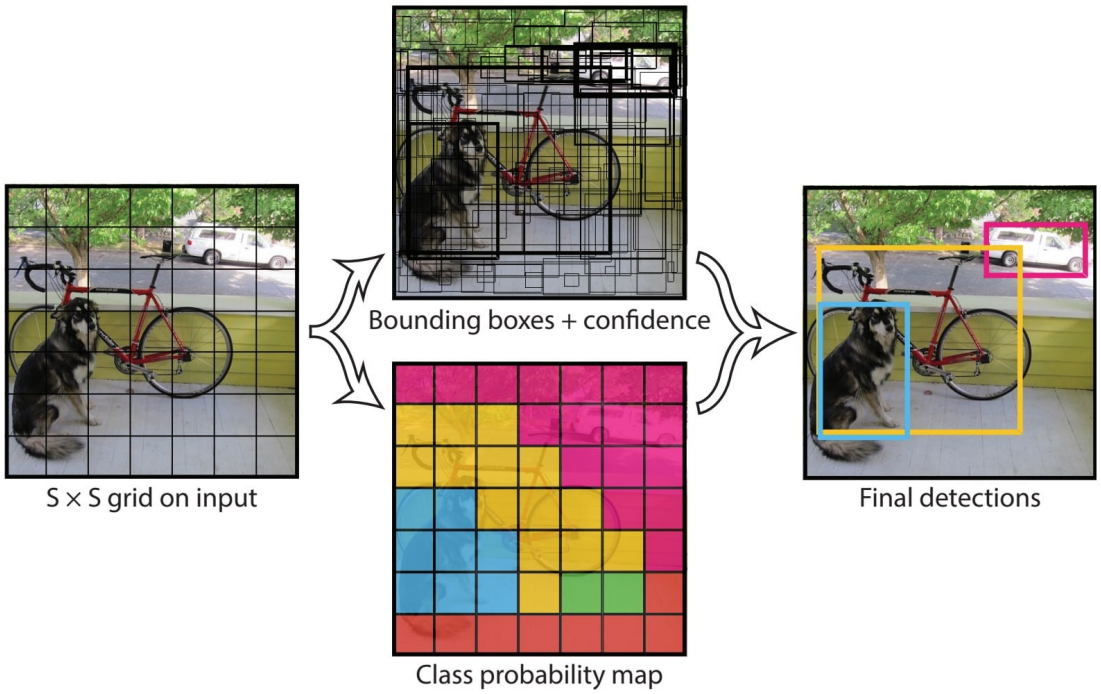

Figure 2.2: YOLOv1 object detection model. An input image is divided into a SxS grid cell. Each grid cell is responsible for predicting B bounding boxes, confidence score for those boxes, and C class probabilities. At the end, these bounding boxes are correlated with the class probability map, resulting in an encoded probability between a class appearing in the box and how well the predicted box fits an object. Figure taken from [62].

The authors tested and used state-of-the-art Bag of Freebies[6] and Bag of Specials[7] to increase backbone and detector quality. On top of this, the authors of YOLOv4 modified state-of-the-art methods, making them more efficient and suitable for single GPU training. Recently, Scaled YOLOv4 [15] was proposed, stating itself as the state-of-the-art object detector. As illustrated on figure 2.3, there is a version of Scaled YOLOv4 that tops every other possible detector models there are. Scaled YOLOv4 is the name given to the group of many "scaled" versions of YOLOv4. Those versions include YOLOv4-CSP, YOLOv4-Tiny and YOLOv4-Large models. There are three versions of YOLOv4-Large, namely YOLOv4P5, YOLOv4P6 and YOLOv4P7. YOLOv4P6 and YOLOv4P7 are scaled versions of YOLOv4P5, and they represent better accuracy but lower fps as they scale.

2.2.3 Object Tracking

Object detection has been around for many years, and a large study around the image processing field enabled it to achieve greats results. Object detection stands for the detection of one or more objects on a given image. Let us suppose there is a criminal

[6] Are methods that increase the model quality by changing the training strategy or increasing the training cost, without increasing the inference cost.

[7] Represents plugin modules and post-processing methods that only increases the inference cost by a small amount but can significantly improve the accuracy of object detection.

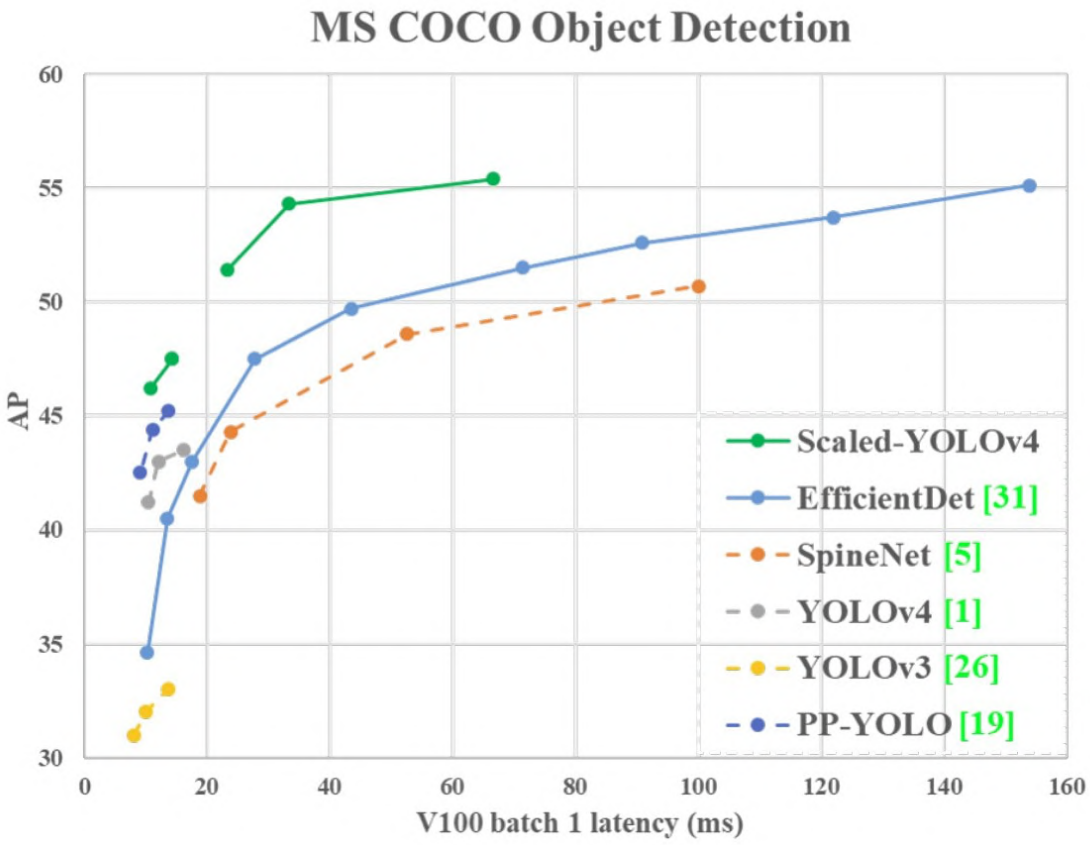

Figure 2.3: Comparison between Scaled YOLOv4 and other state-of-the-art object detectors. Figure taken from [15].

getting away in a car, there are cameras all over the city, we know object detectors exist, there is no way he can get away, right? It is not that simple. Object detection is great when it comes to detecting objects in a static world. A detector doesn't know that a certain object detected in frame n, is the same object that is detected in frame $n+1$, for the detector, these are two complete different identities. In the criminal example, a detector would be enough if there was only the criminal car in the whole city. Happily for the criminal, cities are full of cars, and it is rather easy for him to go unnoticed. This would be different if there was something that enabled us to merge our knowledge of detecting objects in the static world with information regarding the dynamic world, that is, temporal information. With such a system, that would be possible, not only to detect the criminal car in every frame, but also to attribute an ID to it, so that he couldn't go unnoticed, even with thousands of cars all around the city. This is known as object tracking, where an object detected in frame n, can be re-detected in frame $n+1$ and re-identified as being the same object from frame n. Object tracking has been widely used not only to catch criminals on the run, but also in tasks like tracking athletes in sports-related fields [66], tracking pedestrians [67], aircrafts [68], counting the number of different types of vehicles that pass through a certain region of interest [69], among other applications, like counting

fruits.

Over the next subsections, we'll dive into some challenges that arise when performing object tracking, the types of tracking methods there are and how they are performed, as well as state-of-the-art object tracking algorithms.

Tracking Challenges

Tracking is essentially identifying an object in a given frame n, give that object an identification number, and then be able to re-identify it in the next frames $n+1$, ..., $n+x$, while always attributing the same ID to it. While the core idea might seem simple, there are a lot of challenges that arise when trying to perform object tracking. The following list contains some of the most common challenges faced when performing object tracking:[8]

- Occlusion: The tracked object "hides" behind another object, staying occluded for a certain amount of frames (e.g. person being occluded by a passing bus);
- Identity switching: When two objects are being tracked and they cross each other, a problem arises: to which ID belonged each object (e.g. two persons crossing each other);
- Motion blur: An object is blurred for a certain amount of frames due to camera focus or motion of the object. During the frames which the object is blurred, it will not look the same, then, how to re-identify it during and after it has been deblurred (e.g. a football ball blurs for a certain amount of frames);
- Viewpoint variation: When the viewpoint of an object changes, it becomes hard to re-identify it due to changes in its appearance (e.g. a car being tracked and suddenly doing an U-turn);
- Scale change: Sudden object scale variations can lead the detector to fail, as the object suddenly becomes way bigger than it was on the previous frames (e.g. a football ball coming in the direction of the camera at high speed);
- Background clutters: When the colors and/or shapes of the background surrounding the object are identical to the object itself, it becomes harder to distinguish it from the background (e.g. fruits are often confused with leaves and trunk chunks);
- Illumination variation: If the illumination near an object is suddenly changed, its appearance can suffer big changes, becoming harder for the detector and tracker to identify it as the same object (e.g. a sudden shadow over an object caused by a cloudy sky);

[8] Sources of mentioned challenges:

- https://cv-tricks.com/object-tracking/quick-guide-mdnet-goturn-rolo/
- https://nanonets.com/blog/object-tracking-deepsort/

- Low resolution: Low object resolution means less pixels defining the object. This can happen either due to an object being too small, or the camera being too far away from it. When the number of pixels is low, it becomes hard for the detector to detect the object (e.g. a fruit is very small when comparing it to a whole tree, becoming a hard challenge to identify it);

- Non-stationary camera: If a camera is also in motion with respect to the object, it becomes even harder to predict its next position, as another variable (the camera motion) is added to the motion model equation (e.g. autonomous navigation).

Across time, there has been a huge research around object tracking and many solutions have been proposed to solve the listed challenges. Over the following subsections, methods proposed to solve these challenges and others will be analyzed.

Single and Multiple Object Tracking

There are two types of object tracking, single and multiple object tracking. In single object tracking, as the name hints, only one object is tracked along a sequence of frames, while in multiple object tracking, there is the challenge to track more than one object at once along a sequence of frames.

As might be expected, single object tracking is relatively easier when comparing to multi object tracking, as some problems, like distinguishing an object from another, don't necessarily arise. In fact, visual appearance could be enough to track a single object along multiple frames. The authors of [70] summarized and went through the single object trackers that were developed over the past decade, analyzing experimental results and discussing the development trends. The authors also dive into the definition of single object tracking, analyzing the components of general object tracking algorithms.

A more interesting type of tracking, for this dissertation scenario, is MOT. An MOT model has to be able to detect more than one object in a frame, and track all of them in the next frames, until they move out of sight. In [71], the authors discuss existing and recent approaches to MOT, the drawbacks of each approach, problems related to MOT research, and propose different directions for future research in this field. Similarly, the authors of [72] analyzed and summarized top-ranked MOT methods, and they also dive into each method.

On top of single and multiple object tracking, tracking can be done online or offline. In online tracking, the tracker uses present and past information to perform the tracking of one or more objects. This type of tracking is often used for realtime applications. Offline tracking is a slower type of tracking, where the model is fed with a recorded stream, and can look not only at n frames in the past, but also n frames into the future, using it to improve the tracking. As can be expected, this type of tracking is slower, but, due to the amount of information it can access, it usually outperforms online tracking.

The Tracking Process

According to Prof. Dr. Laura Leal-Taixé, author of the famous Siamese CNN [73], there are three main components in an online MOT model:

1. Track initialization: Having into consideration that in online tracking it is not possible to look into the future, the MOT model starts by running an object detector over the first frame n. The output detections from the object detector will be used to initialize the object tracks. As expected, this detector must be trained to detect the objects that are pretended to be tracked (e.g. nectarines). After this first step, the MOT model proceeds to processing the new frame, frame $n+1$;

2. Prediction of the next position of each object: Once the MOT model starts processing the new frame $n+1$, it is going to try and predict the next position for each of the previous detected objects. Because the model is only processing the second frame, it doesn't have much information about how each object moves, hence it must make some assumptions. One option is for the MOT model to assume that the object has not moved, hence the model will use the bounding box coordinates from the detected objects on step 1 on the new frame. As the model keeps processing the frames, it starts building a model to best predict what the next positions for each object will be, this is called a motion model. The classic and most standard way of building a motion model is to use a Kalman filter [74, 75]. Advancements in MOT have shown that using recurrent neural networks like Long Short Term Memory (LSTM) [76] can be promising, as they learn the motion model in a data driven way, enabling it to produce favorable results. Once the model has the predicted bounding boxes for the next positions, be it through a motion model or not, it moves on to the third step;

3. Matching predictions with detections: Once the model has predicted where it thinks the objects moved to, from frame n to frame $n+1$, it passes frame $n+1$ through the detector. Now, there are two types of bounding boxes, the predicted ones and the detected ones. The model tries to match the predicted bounding boxes with the detected bounding boxes with the help of an appearance model. The matching occurs through a measurement (e.g. Intersection over Union (IoU) between detection and prediction boxes) that indicates how similar the set of detections from frame $n+1$ are with the predictions that are coming from the motion model. Once the matching is done, the objects that "pass" the matching process are attributed the same ID as they had in the previous frame, while others that might not "pass" the matching process, or objects the model has never seen before, will receive new IDs.

Some MOT models are more complex than others. The authors of [71] dive into all the components that might be part of any MOT model.

Simple Online Realtime Tracking

Simple Online and Realtime Tracking (SORT) [77], is a state-of-the-art online MOT model. SORT follows the paradigm of tracking-by-detection, where the model detects objects each frame and associate them in realtime. SORT is comprised of four key components:

1. **Detection:** The first stage is identical as the "track initialization" explained in the last section. The model first starts by detecting objects that are wished to be tracked, and shares this detections with the second stage. In the original paper, the authors used Faster R-CNN detection framework [78]. As the authors mentioned, the network architecture itself can be swapped, and as we'll see, YOLO architecture can be integrated with SORT. The ability to swap detectors is an important feature of SORT, as the quality of detection has a significant impact on tracking performance;

2. **Estimation Model:** The second stage receives the detections and the model tries to propagate them using a linear constant velocity model. The estimation model will predict the new position for each of the detected objects by using the Kalman filter framework to optimally solve the velocity components of the linear constant velocity model. If, for example, a previous detection falls into occlusion and no detection was associated to it, SORT will simply predict its position using the linear velocity model;

3. **Data Association:** Once predictions have been made, it's time to associate them to new detections. Once the detector detects all the possible objects in the frame, the SORT model will then compute the IoU distance between each detection and all predicted bounding boxes. This computation is solved optimally using the Hungarian algorithm [79]. Finally, the model filters estimated assignments that had an IoU distance lower than a certain threshold, known as IoU_{min}. Once this step is done, the tracking is finished for the new frame, and all that is left is destroying unused identities.

4. **Creation and Deletion of Track Identities:** When objects enter and leave the image, new trackers have to be created and old ones destroyed. For creating new trackers, the authors consider any detection with an overlap less than IoU_{min} as being an untracked object. In this scenario, the tracker is then initialized using the detected bounding box and the velocity set to zero. This new tracker undergoes a probationary period as a means to prevent false positives. Trackers are destroyed if they have not been detected for T_{Lost} frames. For the SORT algorithm, the authors recommend setting T_{Lost} to 1, as the constant velocity model is a poor predictor of complex dynamics and the authors focus in frame-by-frame detection, not being worried object re-identification.

DL has improved tracking algorithms by providing us with better detectors, and by adding more temporal and feature complexity. In the following section, we'll see how SORT was improved with the help of deep learning techniques.

Deep SORT

The SORT algorithm is a great approach to online MOT, but it often fails in tracking objects through long periods of time when considering artifacts. The authors of Deep SORT [80] introduced another distance metric, based on the appearance of the object, to improve the performance of SORT. This new version of SORT, Deep SORT (figure 2.4), enables occluded objects to be tracked for longer periods of time, thus reducing the number of ID switches.

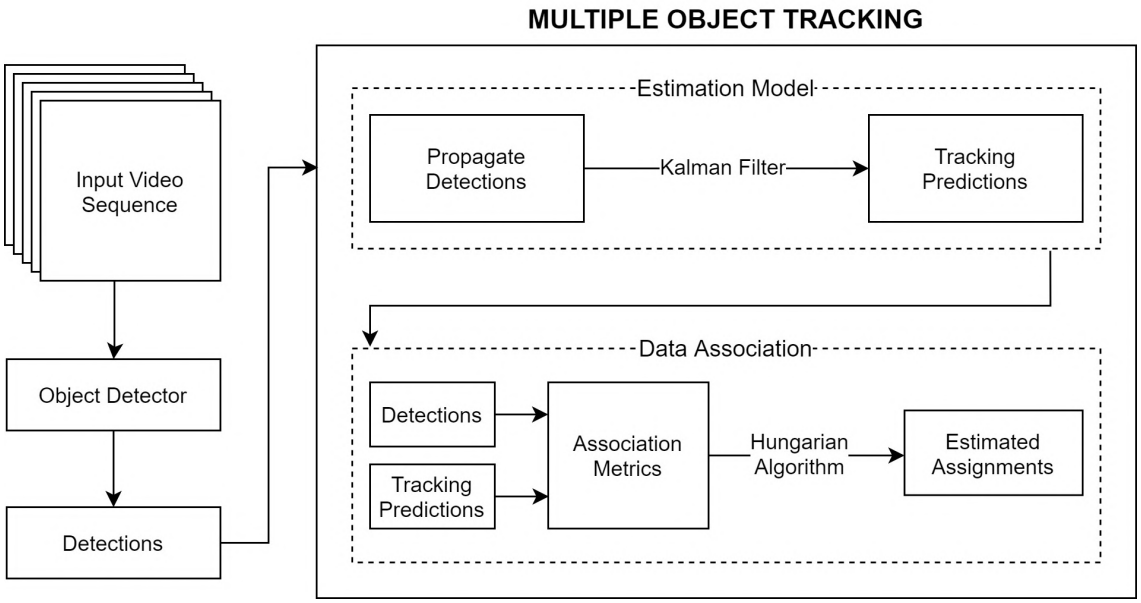

Figure 2.4: Simplified implementation architecture of Deep SORT. Adapted from [80].

As described in the previous subsection, in SORT, there is an object detector outputting detections, a Kalman filter tracking these detections and identifying missing tracks, and the Hungarian algorithm solving the association problem. While SORT uses IoU as the criterion for associating detections to tracking, Deep SORT makes the associations by comparing the feature similarities between an object in the current frame, and its past tracks. To obtain this features, an CNN is trained on a given dataset (ideally a dataset that contains objects similar to the ones to be tracked) and the classification layer is stripped from it, leaving us with a dense layer. The next time an object is passed through this CNN, the dense layer will output an appearance feature vector, which serves as an "appearance descriptor" of the object. On each frame, the model will save the appearance descriptor for each object in a gallery, and compare it with the already saved appearance descriptors of that object. Now, the squared Mahalanobis distance is used

to provide information about possible object locations based on motion (thanks to the Kalman filter), and cosine distance is used to measure the distance between the appearance descriptor of an object on the current frame and its appearance descriptors that are saved on the gallery on the past frames. Thanks to these two metrics, Deep SORT, while still remaining simple to implement, became the state-of-the-art algorithm for online MOT.

2.3 Related Work

In order to estimate the production yield of a field, it is necessary to perform four main tasks, consisting of: a) acquiring a dataset, b) detect fruits in a sequence of frames, c) count fruits on both sides of a row of trees (2D planes), and finally d) merge the counting of both 2D planes while filtering the resulting duplicate fruits (3D plane). In this section, work related to object detection in agriculture will be analyzed in respect to the four main topics. They are divided as follows: 1) Data Acquisition (that is, how data is acquired and what type of data it is); 2) Detection Methods (how the community is handling fruit detection); 3) 2D Plane Fruit Counting (what techniques are being used to avoid fruit double count in the 2D planes); and 4) The Merge of the 2D Planes (how to merge both 2D planes and filter the resultant duplicated fruits).

2.3.1 Data Acquisition

Outdoors data acquisition is a hard and challenging task, especially when performed under real agricultural field conditions. One of the biggest barriers of outside data acquisition is the variations in illumination, which are mainly caused by different weather conditions and shadows. Another challenge of data acquisition in agriculture field is the occlusion of fruits, caused by foliage, branches, and clustering of fruits. In [81], the author suggested that the occlusion of fruits can vary depending on the type of fruit. These barriers are problematic as they lead to miss count of fruits, and, consequently, poor fruit yield estimation. Therefore, solutions to these problems must be addressed. [82] proposes that data are acquired several times throughout the day, which can significantly improve fruit count results. In [83], the authors applied data augmentation over the acquired data, which helped to overcome detection problems due to illumination conditions, distance of the fruits, and fruit occlusion. Another solution to avoid illumination issues is capturing the dataset during night time, as suggested in [84]. During image acquisition, the authors used ring flashes around the two lenses as active lighting, thus avoiding shadows or overexposure of light. The dataset in [85] was captured during nighttime, using a complex flash system and expensive cameras (figure 2.5 a)). While this improves the quality of the dataset (figure 2.5 b)), greatly increasing results (97% accuracy), it becomes an expensive and complex system to implement in an agricultural environment. On top of this, capturing information during nighttime is not convenient as it is past work time. In order

to avoid fruit occlusion, a solution was proposed in [14], by increasing the number of viewpoints of images of a tree, in an attempt to visualize all fruits. The counterpoint of this method is the overcounting of fruits, which happens when a given fruit is considered more than once across multiple images. The authors of [86] presented a multi-modal approach, using color model RGB and Near-Infrared (NIR) spectral information. In this work, both detections from RGB and NIR images were merged together, showing better results than when used separately. The authors suggest that the combined use of RGB and NIR information is an efficient way to avoid fruit occlusion. In fact, a revision in literature on Red-Green-Blue-Depth (RGB-D) sensors done in [41], suggested that multi-modality sensors may be beneficial in fruit detection due to the complementary information they offer, regarding various aspects of the fruit and background. Therefore, this revision concluded that fruit detection, when done with the fusion of multiple types of information, produced better results than when done with only one type of information. Besides improving fruit detection results, the use of depth sensors can also help in localizing fruits, to prevent fruit overcount.

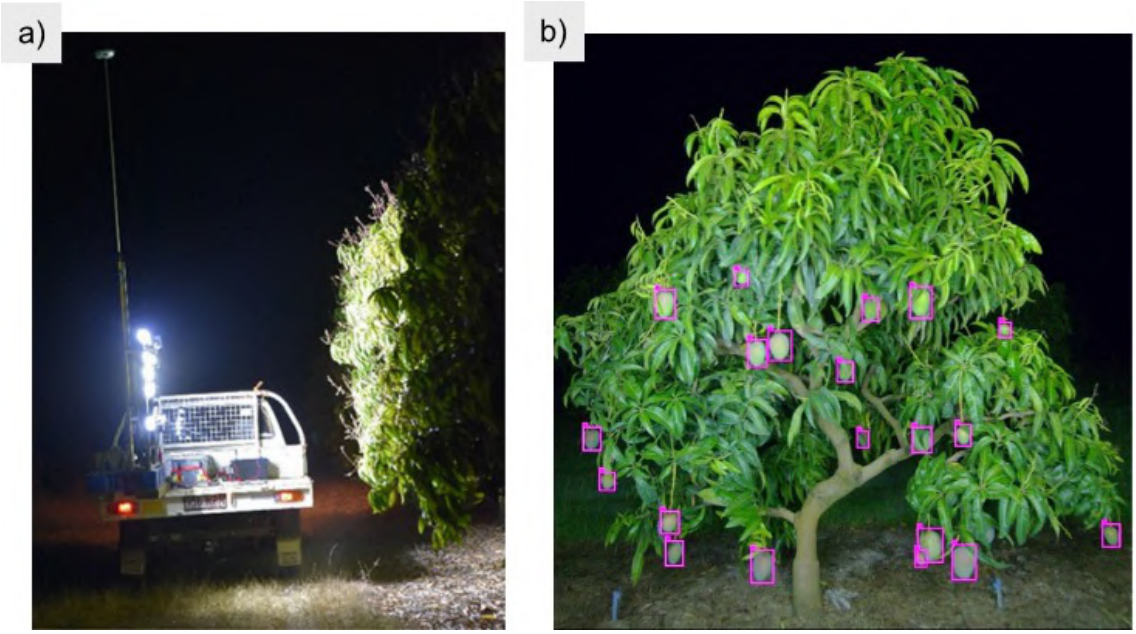

Figure 2.5: Complex data acquisition system. Figure taken from [85].

As to the physical systems that are used to acquire the data, there are mainly two: aerial (UAVs) and ground (UGVs, AGVs, and tractors) vehicles. In the one hand, the use of UAVs, in contrast with ground vehicles, isn't time-consuming, labor intensive, and unviable for orchards with a high density of cultivated area, as suggested in [83]. The authors also defend that UAVs have been progressively implemented in Agriculture 4.0 systems, and have been showing themselves as being an alternative that can be rapidly implemented, stating advantages like their flexibility and repeatability of the results they can obtain. In this same work, UAVs were used to scan rows of a field, taking one picture

per side of a tree. This method avoids fruit overcounting, but favors fruit occlusion, as there are only two viewpoints per tree. On the other hand, the use of ground vehicles can be beneficial and some are already available in most farms, like tractors.

2.3.2 Detection Methods

Prior to fruit yield estimation, it is necessary to automatically detect the fruits in the acquired images. This process has been done with image processing techniques.

With the goal in mind to leverage the use of ICTs in the agricultural field, more precisely the adoption of image processing techniques to count the number of fruits in a given orchard, several methods have been developed throughout time. The very first studies on this field are marked by traditional pixel-level segmentation techniques. However, the limited studies that have conducted accurate fruit detection results under these techniques were done for fruits in controlled glasshouse environments. Several problems arise when such techniques are applied in outdoor environments, mainly due to the high variability in the appearance of the target objects [86].

As time goes by, new techniques emerged, like the use of Deep Convolutional Neural Network (DCNN) in fruit detection. Techniques based on DCNNs have been rapidly growing [87], and the results they produce are getting better. Researchers hope that these technologies can complement farmers' knowledge and intuition by assisting them in laborious and error-prone tasks, which is the case of estimating the production yield of a crop production. [14] and [87] reviewed several DCNN algorithms and their respective F1 scores[9]. Among the reviewed papers by the authors, some of them related to fruit detection got F1 scores higher than 90%, which suggests the potential of DCNNs when applied to agriculture.

A detector is as good as the data it is fed. Applying techniques to increase the training data will most likely produce better results. These techniques are known as data augmentation [88]. Some techniques proposed in [88] to augment data are: a) adding noise levels based on Gaussian distribution [89]; b) adding blur to the dataset [89], as an attempt to represent the lack of auto-focus functionality in the collection of data samples; c) varying the contrast of images; and d) applying affine transformation, which simulates images rendered from different camera positions and projections.

2.3.3 2D Plane Fruit Counting

One could think that once the detection of the fruits is done, the next simple final step to perform is to sum the total count of detected fruits and that would be the estimation of the production yield. Unfortunately, it's not that easy. One of the biggest issues while estimating the production yield is the overcounting of fruits, which is a consequence of considering the same fruit multiple times across different images. One solution to this

[9] F1 score is a common evaluation metric in ML.

problem is to not have overlapped information between pictures. The problem associated with this approach is that it favors fruit occlusion. In this manner, while considering acquiring a dataset by taking multiple pictures with overlapped information (a video), of a tree, it is necessary to keep track of each fruit among these images, as a means to avoid counting the same fruit more than once along different frames and viewpoints.

Object tracking techniques have been used to solve this 2D-plane fruit overcounting issue. The authors of [90] tracked fruits using the Hungarian algorithm and Kalman filter to determine the objective cost, with Kanade-Lucas-Tomasi (KLT) feature tracker filling in the gaps. In [91], the tracking was performed using multiple images from multiple viewpoints and a sequence of geometric operations, as a means to triangulate and locate the fruits in the 3D plane every single frame. LiDAR was then used to associate the tracked fruits with the corresponding tree. The main problem of using geometric operations to locate every single fruit in the 3D plane is that it often demands for huge processing power, slowing down the whole process of fruit yield estimation. The authors of [14] reviewed more systems that use tracking to avoid the overcounting problem when considering multiple frames of multiple viewpoints.

2.3.4 The Merge of the 2D Planes

After tracking and identifying every single fruit in both 2D planes (front and back of the trees), there are cases where the same fruit is identified in both planes. This occurrences can cause an over estimation of the production yield. In order to obtain an accurate estimation, these duplicated counts need to be corrected.

To address this problem, [13] proposed the combination of instance segmentation neural networks and Structure-from-Motion (SfM) for apple detection and 3D location. The process consisted on first detecting and segmenting the fruits, followed by 3D point cloud generation of detected apples, using SfM photogrammetry, and finally projecting the 2D image detections onto 3D space and filter the previous count with this new 3 dimensional view. The authors state that the main advantage of this methodology are the reduced number of false detections, as the authors cross validate the detections across the multiple viewpoints, and the higher detection rate, as multiple viewpoints is a solution to fruit undercount due to occlusion. Another approach to avoid fruit overcounting is to locate every single fruit, and, if two fruits fall into the same location, they are considered as being the same fruit [92]. [41] reviews various types of RGB-D sensors and methods used for fruit localization.

Although merging both 2D planes while filtering out the duplicated fruits increases the fruit yield estimation accuracy, some techniques require huge processing power and time [13].

2.3.5 Other Literature Contributions and Results

A summary of related work is present in table 2.1. It summarizes some of the already cited work, as well as other contributions which can be taken into account along the development of this dissertation.

Table 2.2 contains results obtained from previous works. The column "Type" differ in the type of fruit yield estimation that is performed. These types are divided as follows:

- **Stationary:** Counting fruits in random fruit images, simply to assess the accuracy of a certain object detector or technique;
- **WO/ Correction:** Counting the number of fruits in both sides of a tree row;
- **W/ Correction:** Counting the total number of fruits in a row of trees, applying some technique to correct the fruits that are counted more than once form both sides of a tree row.

2.4 Challenges & Gaps

There are three major challenges that must be addressed during the implementation of this dissertation. These are: a) implementing a system capable of scanning the fruit trees, while not being too complex; b) training an DL detector capable of detecting all the present fruits in a given image while avoiding issues such as illumination and fruit occlusion; and c) avoiding the overcount of fruits when detecting and counting through multiple viewpoints per tree, as a means to count fruits on tree sides. Along the reviewed work, three main gaps were identified. Firstly, the works that produced accurate results often had their data taken with a dedicated camera and/or in some kind of controlled environment (like using flashlights during nighttime), or acquiring spatial data on top of the RGB one. Secondly, the time it takes for them to estimate the production yield, which makes it unfeasible to perform estimations over large areas of agricultural crop. And lastly, in some of the surveyed work, there is the need of an IT specialist following the estimation process, which results in a non-autonomous estimation of the production yield. To this end, this dissertation aims to fill these gap, by developing and validating the reliability of a system capable of both detecting nectarine fruits with the highest accuracy and speed possible, while filtering the overcounted fruits on tree sides when considering multiple viewpoints per tree. This validations is a door opener to further implement the remaining proposed framework, which would allow for a total fruit count of a tree, and, consequently, the estimation of the production yield of the whole orchard.

Table 2.1: Summary of surveys and applications of object detection and production yield estimation in agriculture.

Reference	Year	Contribution
[84]	2013	Captured the dataset during night time using a ring flash around the two lenses as active lighting, in order to avoid illumination issues.
[30]	2015	Developed a prototype AGV/UGV capable of automatically or remotely survey farmland, detect pest diseases and spray pesticide as necessary. eAGROBOT, as baptized by the authors, includes a camera that captures pictures of plants, AI based embedded algorithms to help the robot pick regions of interest and perform image processing techniques on them to identify the type and extend of diseases, a sprayer mechanism that sprays the desired pesticide, and has Wi-Fi connectivity for necessary external support.
[86]	2016	Presented a multi-modal approach, using RGB and NIR spectral information, as a means to avoid fruit occlusion.
[14]	2019	Proposed the increase in number of viewpoints of a tree when acquiring a dataset, as an attempt to visualize all fruits and thus avoiding fruit occlusion.
[82]	2019	Proposed that images are acquired throughout multiple times a day, in order to avoid illumination and occlusion issues, thus improving results.
[13]	2020	Proposed the combination of instance segmentation neural networks and SfM for apple detection and 3D location, as a means to avoid fruit overcount.
[34]	2020	In this revision work, the authors reviewed several implementations of autonomous aerial and ground robots for monitoring farms and controlling pesticides, soil moisture, and measuring crop ripeness. Some of this tasks were accomplished using image processing techniques based on ML.
[41]	2020	Review of the literature on RGB-D sensors and their positive impact in fruit detection. Reviews various types of RGB-D sensors and methods used for fruit localization.
[83]	2020	Applied data augmentation over a dataset, as a means to overcome detection problems due to illumination conditions, distance of the fruits, and fruit occlusion. Suggested that the use of UAVs present more flexibility and produce more repeatability in their results.
[87]	2020	Reviewed several DCNN algorithms and their respective F1 scores. Some of these algorithms achieve F1 scores higher than 90%.
[88]	2020	Suggested that data augmentation is an effective technique to increase detector accuracy. In order to augment the data, the author proposed some techniques to be applied to the dataset images, such as the addition of noise levels based on Gaussian distribution, adding blur, varying the contrast and applying affine transformation.

Table 2.2: Review of literature results.

Reference	Year	Type	Fruit	Overlap	Sensor Type	Accuracy
[93]	2016	Stationary	Orange	-	RGB	91.3%
[85]	2019	Stationary	Mango	-	RGB	97.0%
[94]	2020	Stationary	Chilli	-	RGB	92.7%
[12]	2016	WO/ Correction	Apple	No	RGB	81.0%
[95]	2019	WO/ Correction	Mango	Yes	RGB	62.0%
[91]	2016	W/ Correction	Mango	Yes	RGB & LiDAR	98.6%
[92]	2016	W/ Correction	Apple	Yes	RGB & ToF	82.0%
[90]	2018	W/ Correction	Apple	Yes	RGB	96.7%
[90]	2018	W/ Correction	Orange	Yes	RGB	99.8%
[13]	2020	W/ Correction	Apple	Yes	RGB	80.0%

Chapter 3

Proposed Framework

As discussed before, there is an obvious need for a system capable of estimating the production yield of an orchard, in order to give farmers the chance to better manage their storage and planning their marketing activities accordingly, thus avoiding mismatches between supply and demand, therefore ensuring food security. It has been seen in chapter 2 that systems have been proposed to address the issue of production yield. However, most of these systems rely on complex image acquisition and processing techniques, making them expensive (both in price and time) and non-user-friendly for real world scenarios.

With this in mind, a system that is capable of autonomously estimate the fruit yield of an orchard is proposed. As can be seen in further detail below, the system was designed to perform the estimation "on the go". Due to the properties of the chosen technologies, which makes it a fast and reliable system, it is possible to perform a continuous estimation of the production yield, with tools accessible to anyone and anywhere.

In the next section 3.1, the overall functioning of the system will be proposed, as well as how it can assist farmers in better managing their marketing activities and fruit storage in a simple and easy manner. The system will then be further detailed in section 3.2.

3.1 Framework Overview

As seen before, the proposed literature systems to estimate the production yield take RGB and, in some cases, spatial data of a certain tree row to then count the number of fruits in it by using high demanding processing techniques. Then, they escalate that estimation to the whole orchard using a relation similar to

$$y = \frac{a}{b} \times b_t,$$

where y is the resultant estimation of the production yield of the whole orchard, given in number of fruits; a corresponds to the fruits already counted; b to the number of rows which fruits have been counted; and b_t to the total number of rows in the orchard of that specific fruit. Ideally, the number of counted rows should be equal to the number of

existing rows ($b = b_t$), which means that all the fruits of the orchard were counted. The problem associated with this methodology is, in an orchard where can potentially exist hundreds of tree rows, counting the fruits belonging to one row and then escalating it to the remaining hundreds can result in a poor estimation of the production yield, as not all trees carry an evenly distributed number of fruits. The current proposed systems in the literature lack in the ease of use and the time it takes to perform the estimation of the production yield, limiting these systems to estimate the fruit yield of a couple of rows. The main objective is to try and get b as close as possible to b_t, therefore, a user-friend, reliable and fast system capable of counting as many rows as possible should be implemented.

The system proposed below enables that a continuous estimation of the production yield is performed. This is done with the help of two smartphones attached to both sides of a tractor. While the tractor is moving through the rows of trees, the smartphones are continuously streaming to a server both video, cardinal, and GPS data of the trees. The server will then process the available data and estimate the fruit yield of the available tree rows. The more rows the tractor pass by, the closer b gets to b_t and, consequently, the more accurate the system will predict the estimation of the production yield. This system needs two main characteristics to work, these are accuracy in counting the fruits per row and speed in doing this process, so that ideally all rows can be processed. The proposed framework is depicted in the diagram of figure 3.1.

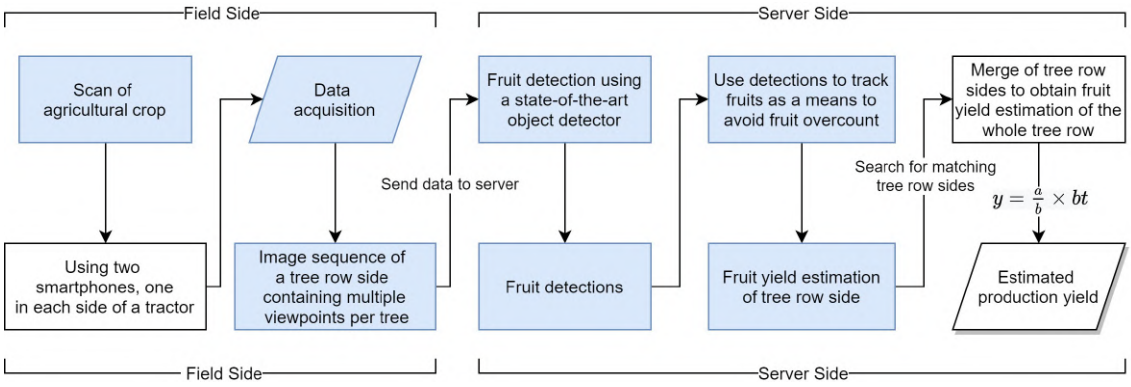

Figure 3.1: Diagram of proposed framework. The developed work focuses around the blue colored boxes.

Analyzing the diagram above, it can be perceived that the system is divided into five "stages", which purposes are to: acquire data of a tree row side and send it to a server, detect each fruit in every frame of the acquired data, use that detections to track and estimate the fruit yield in that tree row side, search for the matching tree row side to finally merge both sides and obtain the estimation of the production yield of the whole tree row, and then "contribute" with that row estimation to the total orchard estimation (contribute with a row estimation so that b gets closer to b_t).

3.2 Framework Proposal

To implement a fruit yield estimation system that is user-friendly, reliable and fast, five main questions have to be addressed:

- How to continuously acquire data of the nectarine trees with a simple and inexpensive solution?
- How to detect every fruit in the acquired images, given the challenges of outside data acquisition?
- How to identify a fruit that appears in a sequence of frames, thus avoiding fruit overcounting?
- How to know which tree row sides belong to the same tree row?
- How to merge the count of both tree row sides and obtain the estimation of the production yield for that tree row?

Proposing solutions to all these questions is fundamental in order to achieve a system capable of estimating the production yield of a field "on the go". Over the next subsections, we'll dive into all these questions while proposing some solutions based on the previous research of state-of-the-art technologies regarding image acquisition (section 2.3.1), object detection (section 2.2.2), object tracking (section 2.2.3), and other relevant related work (section 2.3).

Before implementing the whole system, it is first necessary to access the viability of using a smartphone to acquire the RGB data. That is, it is necessary to ensure if there is a system capable of estimating the fruit yield on tree sides given data acquired with an ordinary smartphone, which doesn't count with the highest resolution. To this end, the developed work in this dissertation focuses around the blue colored boxes in the diagram of the proposed framework (figure 3.1). RGB data was acquired with a smartphone in the agricultural field of "Herdade Corte Romeira", and a detection and tracking system to estimate the fruit yield of a tree row side was successfully implemented.

As will be further discussed, to the best of our knowledge, the results came out to be state-of-the-art, which is an exciting door opener to the development of the remaining proposed architecture, which should be further conducted in future work. The remaining implementation would be to acquire not only RGB, but also cardinal and GPS data with two smartphones attached to a tractor, automatically send that data to a server, integrate the developed system with that server, and finally merge the counts from both tree row sides and contribute with that merged count to the total estimation of the production yield.

3.2.1 Data Acquisition and Processing

In order to estimate the production yield, it is necessary to first acquire data of the actual field. Acquiring quality data in field conditions is a challenging task, as there are a lot of uncontrollable variables, like the sunlight, wind, rain, and even the conditions of the field itself. On top of it, the system should be easy to use, so that anyone without previous instructions can use it. A challenging task like acquiring data on an agricultural crop doesn't quite match with a simple system that anyone can use. The best solution that was found was to take advantage of something that nearly everyone uses everyday and are used to, namely a smartphone. A smartphone has all the five main characteristics needed for the proposed framework: it is a device that anyone knows how to use, it has a camera capable of acquiring RGB data, it has a built-in GPS system, it knows where north, south, west and east are, and it has a connection to the internet. On top of this, it is highly accessible world-wide. These were the reasons which motivated the use of a smartphone in the proposed framework. Although it has many advantages, there is a cost associated to it. RGB data acquired with a smartphone isn't as accurate as it would be if acquired with a dedicated camera. The price to pay is then frame resolution and quality. That is why it is so important to test if an actual system is capable to detect and count fruits using a smartphone, because if it isn't, the whole proposed framework loses its purpose.

Having this said, two smartphones should be attached to a tractor, both inside a box able to protect it from the sunlight and rain. Each smartphones should be placed in each side of the tractor, parallel to it, both with the main cameras pointing to the opposite direction of the tractor. With this disposition, the smartphones are able to scan two tree row sides when a tractor passes through a row of trees. Figure 3.2 is an illustration of the proposed data acquisition system.

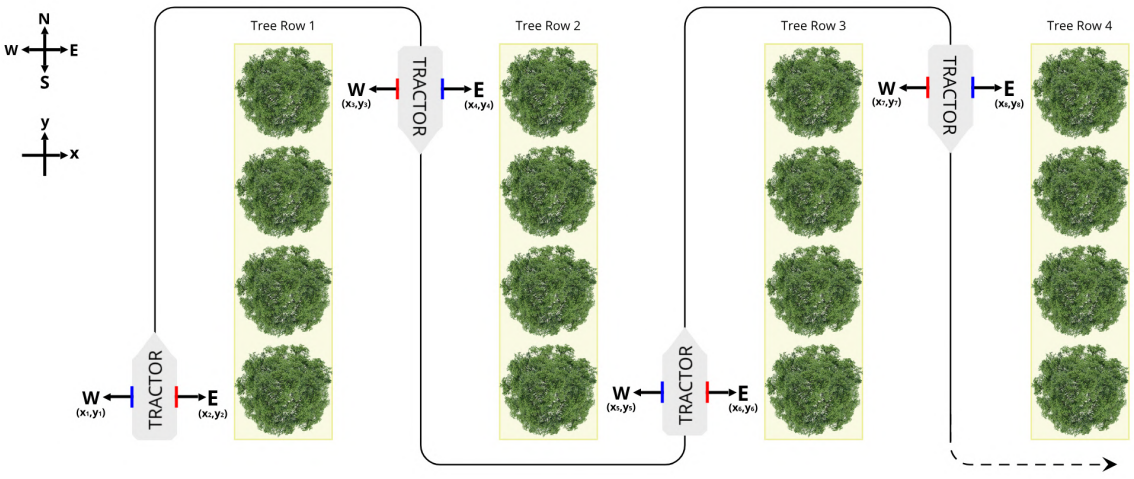

Figure 3.2: Proposed data acquisition system.

In figure 3.2 the tractor is represented by a grey pointy box subtitled "TRACTOR",

and the smartphones are represented using red and blue colored lines in the sides of the tractor. In this given example, the tractor starts at the left side of tree row one and moves in the line direction, which is merely illustrative, passing through tree rows 1, 2, 3, and 4. Both smartphones are continuously acquiring RGB data through the main camera, cardinal data through its magnetometer, and its coordinates through the global positioning system. Both smartphones are streaming this data to a server while acquiring it. This data will then be further used to count the number of fruits in it, as well as associate the fruits counted to their respective row. Subsection 3.2.3 will further detail how the system uses the cardinal and coordinate information to match two row sides into one unique row.

3.2.2 Tree Row Side Fruit Count

To estimate the production yield of a tree row, it is first necessary to know how many fruits are in both sides of that row. Using figure 3.2 as an example, once the tractor finishes passing through the left side of "Tree Row 1", the smartphone represented as a red line and pointing east will have scanned the whole left side of "Tree Row 1" and have it sent to the server. On the server side, using an object detector, the system will detect all the fruits that are present in the left side of "Tree Row 1" and use these detections to track them. As said before, the tracking is used as a means to avoid double counting a fruit that appears more than once along a sequence of frames. The "Tree Row Side Fruit Count" system is represented in figure 3.3.

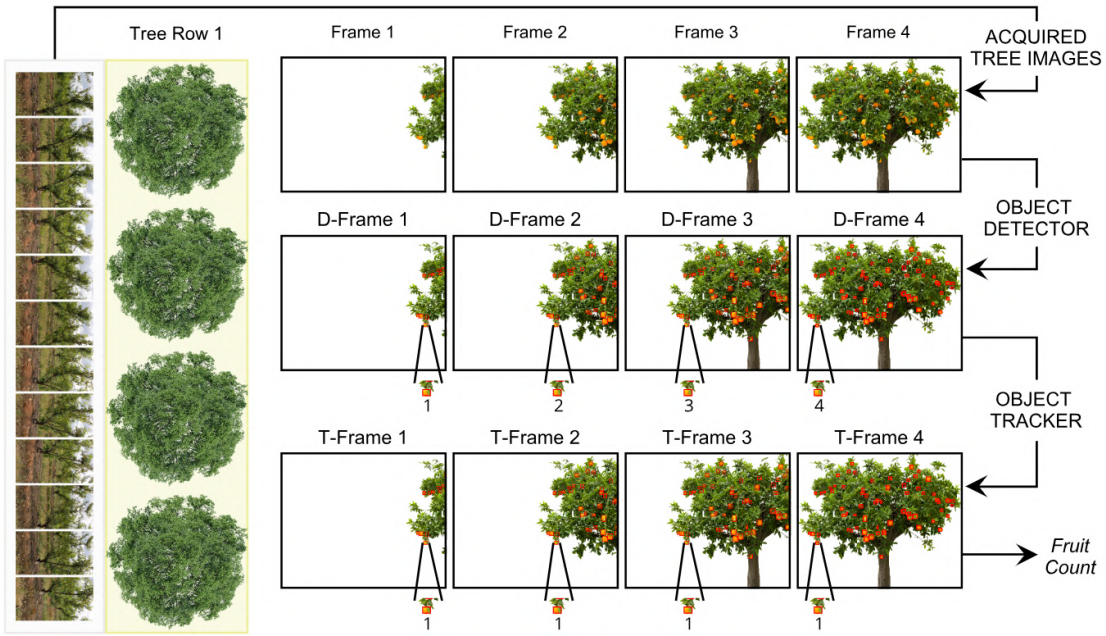

Figure 3.3: Methodology to count the number of fruits along a tree row side.

This figure gives a better understanding of what was said before. The acquired tree

images are passed through an object detector, which outputs "Detected Frames", or "D-Frames" for short. This frames contain detections of all the fruits in the acquired images. As it can be seen from the figure, in the D-Frames phase, one fruit was seen 4 times across all the 4 frames. The detector doesn't know this is the same fruit, for the detector, these are 4 complete different fruits. To avoid this overcount, a tracking system is used. The D-Frames are the input of this tracking system, which is capable of recognizing a given fruit through a sequence of frames. The output frames of this tracking phase are denoted as "Tracked Frames", or "T-Frames" for short. In this frames, it can be seen that the fruit that would've previously been counted 4 times, now always contains the same ID "1" along all the "T-Frames" 1, 2, 3, and 4, therefore avoiding overcounting. The output of this tracking phase represents the number of fruits in one side (in this case the left side of "Tree Row 1") of a tree row. Later in subsection 3.2.3, it will be discussed how the count from this side of the tree row will be matched with the opposite side of it.

Fruit Detection

Detecting fruits hanging in trees is already a challenge by itself as they are very small and their color is often easily confused with leaves and trunk chunks. The difficulty increases when considering that tree fruit images are taken in uncontrolled agricultural conditions, where there are often a lot of variation in illumination and shadows, among other challenges of outdoor data acquisition. On top of all this, the difficulty exponentially increases when the data is taken with an ordinary smartphone. Taking tree fruit images with a smartphone often causes fruits to become pixelated, which, together with the illumination, shadow, and other challenges, become even harder to distinguish it from leaves or trunk chunks.

Thankfully, DL has gifted the image processing field with incredible tools, capable of building the most reliable and fast object detectors. In this dissertation a detector of the YOLO family was trained, in the expectation of overcoming the challenges mentioned above. After analyzing the state-of-the-art, the chosen object detector was a scaled version of YOLOv4 proposed in [15], named "Scaled YOLOv4".

The YOLO object detection model is an one-stage object detector. These types of object detectors have three main components in their architecture. These are the backbone, the neck, and the head. The backbone is used to extract features of the input images. It is constituted of an CNN which is usually trained in some image database, like ImageNet[1], so that its weights are pre-adjusted to identify relevant features in an image. Backbone selection is of huge importance, as it hugely impacts on the model performance. After the backbone stage, comes the neck, which task is to mix and combine the feature layers formed in the backbone CNN for the detection step. The final block, named head, utilizes the YOLO algorithm explained back in subsection 2.2.2, and outputs a vector containing

[1] https://www.image-net.org/

the coordinates of the predicted bounding boxes, as well as its confidence scores and labels. These outputs is what is normally called "predictions".

Google research/brain used Network Architecture Search (NAS) to determine optimum parameters for width, depth and resolution on EfficientNet-B0. Later, using the NAS technique, [96] found standard optimum parameters for an object detector model. After surveying the literature for state-of-the-art detectors, the authors of Scaled YOLOv4 found that the backbone used in YOLOv4, CPSDarknet53, matched almost all optimal parameters established by [96]. For this reason, the authors developed model scaling technique based on YOLOv4, which would later became the current state-of-the-art object detector, Scaled YOLOv4.

First, the authors "CSP-ized" a lot of different aspects of the previous YOLOv4 model proposed by [63]. To "CSP-ize" is to convert an CNN into a Cross Stage Partial Network (CSPNet). An CSPNet is an CNN that is designed to efficiently run with similar performance relative to other CNNs. Replacing an CNN with an CSPNet increases the model accuracy, while reducing the inference time and computation costs. This resulted in a more accurate and faster version of YOLOv4, which the authors named "YOLOv4-CSP". Then, a network scaling approach was conducted, where the depth (number of convolutional layers), width (number of convolutional filters in a convolutional layer), resolution (image resolution size) and network structure were scaled. The authors baptized the scaled YOLOv4-CPS model as "Scaled YOLOv4". Scaling a network comes with a computational cost, which in Scaled YOLOv4 was paid with the "CSP-ized" process. In this way, the authors were able to achieve Scaled YOLOv4 models that are both accurate and fast in the inference process. They scaled YOLOv4-CSP both up and down, ending up with YOLOv4-Tiny and YOLOv4-Large models. YOLOv4-Tiny is a fast, light-weight version of YOLOv4, while YOLOv4-Large counts with three YOLOv4 large models, namely YOLOv4P5 and its scaled versions, YOLOv4P6 and YOLOv4P7.

Prior to detection, it is necessary to train the DL object detector. Training an object detector is an expensive computational task. Thankfully, Google offers an online computation service, Google Colab[2], which is used to train the detector. After training, the detector is able to detect the objects it saw during the training process, which, in this case, are "gardeta" nectarines. Once the detections are available they are fed to the tracking system.

Fruit Tracking

It is not viable to solely rely on detections to estimate the production yield. As mentioned in the above subsections, the detector doesn't keep track of a fruit along a sequence of frames, and will end overcounting them. For this reason, the detections of the fruits are

[2]"*Google Colabouratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.*" Source: https://colab.research.google.com

fed to the tracking system, which will be responsible to recognize the same fruits along a sequence of frames and estimate the production yield of a tree row side.

In fruit tracking, there are other challenges on top of the identified ones back in Fruit Detection. There are occasions where fruits appear during a large amount of time, and, in some of these occasions, they become partially or even fully occluded by leaves or trunk chunks. To this end, it is necessary a tracking algorithm that is able to overcome all these problems. On top of it, it also has to be fast so that a huge number of tree rows can be scanned, as a means to increase the quality of the estimation of the production yield. Once again, DL has provided us with tools that are able to increase tracking algorithms performance.

Ideally, with the use of a tracking algorithm, the system will remember the fruits it counts and, consequently, doesn't overcount them. This is illustrated in figure 3.3, where the fruit that was previously counted four times (represented in the "D-Frames"), is now only considered once across all tracked frames ("T-Frames"). The tracking algorithm proposed in this dissertation is based on the "track-by-detection" paradigm. Tracking by detection means that the tracking algorithm will use a previously trained detector to perform the detections of the object to be tracked. This is an elegant approach, as the detector can be switched without affecting the other parts of the system. This is useful as the object detector can be switched in the case of a better one eventually shows up.

As for the tracking system itself, it will be done using Deep SORT. As discussed before, Deep SORT is an extension to SORT. To create some context, let us resume how SORT works and how DL was used to improve and transform it into Deep SORT. SORT first starts by detecting fruits in the very first input frame, and attributes an ID and a track to each detected fruit. In this dissertation, these detections will be provided by a large model of Scaled YOLOv4. After the fruits are detected, SORT will use a velocity model to best predict what their next position will be. These predictions come in the form of bounding box, which will then pass to the second frame, where the fruits will be detected once again. The SORT model computes the IoU distance between each detection and all predicted bounding boxes. After it finishes computing all the IoU distances, it filters out the ones which had an IoU distance lower than a specified threshold IoU_{min}. Once this process is over, all the new fruits from frame one that made their way to frame two, should (hopefully) be re-identified with the same ID. This is how it is possible to avoid the overcounting while counting the number of fruits present in a tree row side. The last step is to destroy and create new tracks. The SORT model destroys tracks when it hasn't seen a fruit for T_{Lost} frames. In the other hand, it creates news tracks when it sees a fruit that it has never seen before. Although SORT is a reasonable object tracker, it often fails in tracking objects through long periods of time when considering artifacts. The authors of Deep SORT perform the association of detected fruits and its predicted boxes through a deep association metric. While in SORT the association metric used is IoU, in Deep SORT the association metric is based on the appearance of the object being tracked. Instead of calculating the IoU distance like in SORT, Deep SORT compares the features

similarities between an object in the current frame, and its past tracks. These features are obtained through a trained CNN dense layer. With this main change, the authors of Deep SORT turned the SORT algorithm into a reliable and fast tracking system.

3.2.3 Matching Tree Row Sides

After identifying and counting all fruits in a tree row side, it is necessary to merge the counts of that tree row side with the counts of its opposite side. This raises a problem. How does the system know which row side is the match of another row side? To solve this problem, a methodology to match rows sides is necessary.

This is where the GPS and cardinal data come into play. While acquiring the RGB dataset, the smartphone is also recording data regarding GPS coordinates and cardinal orientation. Looking at figure 3.2, it is possible to understand that RGB data that is captured while the smartphone is looking east (E), can only match with RGB data that is captured while the smartphone is looking west (W). Let us use the left side of "Tree Row 1" as example. When the smartphone finishes sending all the RGB, GPS and cardinal data of the left side of "Tree Row 1" to the server, it will search for another dataset that was captured while the smartphone was pointing west. Following the example in figure 3.2, at first, the system doesn't find any match, as there is only the data of the left side of "Tree Row 1". Now let us consider that the tractor just finished passing through all four tree rows and leaves to do another task. After the smartphones are done sending all the data, on the server side, there are now seven different datasets. Each one of this datasets corresponds to data taken from one side of the four scanned tree rows. In this example, both sides of all tree rows were scanned expect the right side of "Tree Row 4", because the tractor never passed through it. Now, how does the system match all the seven datasets so that it can further estimate the prediction yield of each row? Each data set is marked with a cardinal flag, which indicates if it was taken with the smartphone pointing east or west. The system will compare all datasets with flag "E" with all datasets with flag "W". If a comparison is validated in the following rule, it means that both datasets belong to the same tree row, and hence they may be merged.

if $X > ||x_{2avg}\_E| - |x_{3avg}\_W|| > Y$ **then**
    ROW SIDES MATCH
end if

In the above rule, "x_{2avg}_E" represents the average of all x points in a row side which data was captured with a smartphone pointing east (E), and "x_{3avg}_W" follows the same logic but belongs to a row side which data was captured with a smartphone pointing west (W). X and Y are predefined values. X represents the average distance between two tree rows and Y corresponds to a value larger than the tractor width. The X value is used to effectively find the matching row sides, while the Y value is used to make sure that

the system doesn't confuse matching side rows with side rows facing each other, which can happen due to the smartphones being too close to each other (a tractor of width away). If the absolute distance between "x_{2avg}_E" and "x_{3avg}_W" is lower than X, and higher than Y, it means that both row sides represent the same tree row, and so their fruit counts should be merged. Note that the absolute distance between "x_{2avg}_E" and "x_{3avg}_W" being lower than X means that the data was acquired with smartphones that were "a tree row away", which potentially indicates that they were acquiring data of the same tree row.

This distance comparison is possible thanks to the worldwide emerge of 5G, in specific in agricultural fields [97]. With the help of 5G, GPS will eventually become more accurate than ever, with object positioning accuracy down to the meter, decimeter and centimeter[3].

3.2.4 Fruit Count Correction

After matching both tree row sides to form a tree row, it is necessary to take care for fruits that might've been seen in both row sides. Simply summing both counts from both tree row sides can inflate the estimation of the production yield. This phenomenon is represented in figure 3.4. In this figure two frames taken from opposite sides of "Tree Row 1" were selected. The selected frames are represented in the top left and top right sides of the figure. The left frame represents a portion of the first tree taken from "Side 1", whilst the right frame represents the same tree but from the opposite "Side 2". In both of these frames, there is an area, represented inside a yellow box and further zoomed below both frames, where it is clearly demonstrated the discussed phenomenon. By looking at both this zones (zoomed images in the bottom of the figure), it is possible to identify four fruits that have been seen from both sides of "Tree Row 1". In this case, if simply summing the counts from both sides of "Tree Row 1" , the system would've counted 8 times this fruits, instead of 4. Unfortunately, the estimation of the production yield was not performed while capturing the dataset used in this dissertation, and, due to the seasonality of the fruits and the time scope of this dissertation, it was not possible to come back and take new data with the ground-truth fruit counts of that row. This makes impossible to validate the accuracy of simply summing the fruit counts from both row sides.

As seen before, some systems are proposed in the literature to address this overcount problem. Some authors reconstruct a 3D model of the whole tree row using SfM-related methodologies, while others map each fruit coordinates in the 3D plane using ToF sensors. The problem related with these techniques is that they require massive processing power, slowing down all the fruit count process, which makes the system unable to estimate the production yield for a high number of tree rows. One possible solution to correct this inflated fruit count is to use a statistical model. If the percentage of fruits that are overcounted in a tree row don't deviate a lot from other tree rows, it is possible to estimate

[3] https://www.ericsson.com/en/blog/2020/12/5g-positioning--what-you-need-to-know

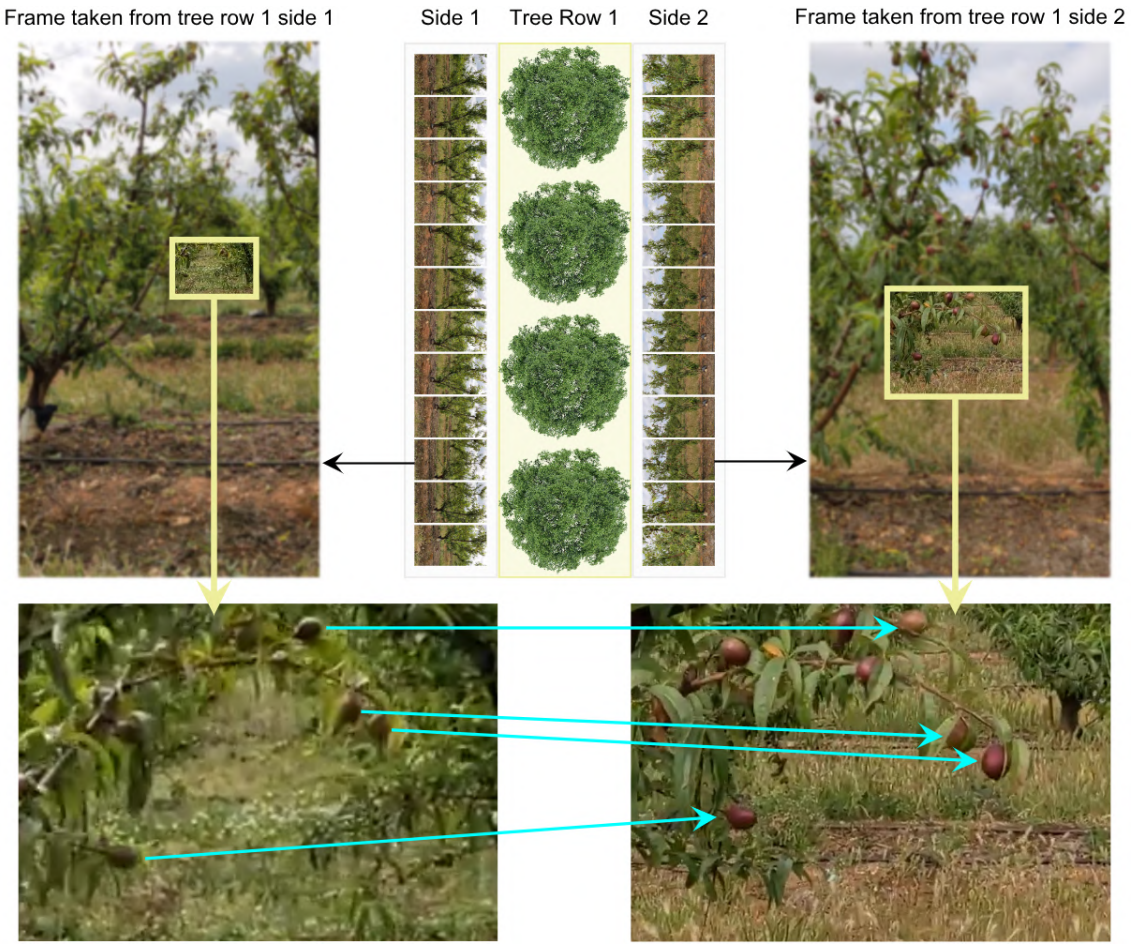

Figure 3.4: Illustration of the same fruits being seen from opposite sides of "Tree Row 1".

how many fruits are being overcounted per row, and thus correct that overcounted fruits. This is an interesting solution to be further analyzed in future work, as it would keep the speed proprieties of the system while hopefully maintaining good estimation accuracy.

3.2.5 Data Interpretation and Presentation

Once the counts from both tree rows are merged, the system has two main tasks left. Firstly it has to add the contribution of the counted fruits of the merged tree row to the total estimation of the production yield. As seen back in section 3.1, in $y = \frac{a}{b} \times b_t$, the more tree rows that are added to b, the closer it gets to b_t, and the more accurate the total estimation of the production yield becomes. Contributing with the fruit counts from a merged tree row is essentially increasing the value of b, hence, the more rows are scanned, the better. The second task the system has to perform is to present the actual estimation of the production yield of the whole orchard. There are a variety of ways through which this is achievable, like through a simple web application.

The proposed system shall be running indefinitely, so that every time the server receives data from the smartphones, it continues processing it and contributing to the

estimation of the production yield of the whole orchard. As said before, ideally, it is desired that $b = b_t$, which is the case where all the tree rows in the orchard have been processed.

3.3 Possible Framework Limitations

The use of the proposed framework should always be considered upon implementation. Depending on factors such as orchard field conditions or others, some limitations to the proposed framework may arise. The framework architecture was designed and thought having into consideration such limitations. Some of them are listed below, as well as the proposed solutions to address each one:

- It may be impractical to attach two smartphones to the tractor over specific conditions. As an example, when there isn't enough distance between tree row sides on a tree row, there may be cases where a smartphone will not be able to film the whole tree. Another possible scenario is when the smartphone is unstable during data acquisition, which results in imperceptible images. In such described use cases, or any others where the use of a tractor influences the acquired data in such a way that it can't be used, it is suggested that a human manually acquires the data, similar to what was done during this dissertation. This will not influence any of the other stages of the system, so it should be considered a valid solution of the identified limitation;
- The smartphone battery may be a limitation if the data is to be acquired for long periods of time. A solution to this use cases is to use power banks, which count with massive energy storage capacity, capable of charging multiple smartphones at once.

Chapter 4

Implementation

Along the implementation of the fruit yield estimation system, there are two main issues to be address, namely:

- Correctly detecting and classifying all the fruits frame-by-frame in a tree side;
- Avoid fruit double counting in the 2D plane along a tree side.

Achieving good accuracy in detecting and classifying fruits is a challenging task. Even for a human, trying to identify fruits in an image is often difficult due to lack of visibility, illumination issues, amongst other inconveniences. During the implementation, DL object detectors that achieved the highest scores in the COCO benchmark were tested and compared. These are YOLOv4 and its scaled versions, YOLOv4P5 and YOLOv4P6. There is also a larger version of Scaled YOLOv4, YOLOv4P7, which, unfortunately, demand huge computing capabilities to be trained, such as high GPU memory (more than 16GB). Such high demanding capabilities ask for better resources, such as training the models in AWS EC2[1] dedicated instances. These options do not come cheap, and, as such, Scaled YOLOv4P7 wasn't considered during this implementation.

After detecting each fruit in every image with the highest accuracy possible, the next step is avoiding fruit double count in the 2D plane. This means not counting the same fruit more than once in a sequence of frames of a tree. Figure 4.1 is a good representation of such an issue. To this end, a tracking framework (FastMOT [16]) based on DL was implemented.

4.1 Fruit Detection and Classification

Fruit detection and classification mainly consists of two steps: 1) creating a dataset and 2) training an DL object detector. In the following sections, there is an explanation on how both tasks were accomplished.

[1] https://aws.amazon.com/ec2/instance-types/#Accelerated_Computing

Figure 4.1: Same fruits on one tree across two frames.

4.1.1 Creating a Dataset

The dataset was acquired at "Herdade Corte Romeira" in April 2021, and consists of two videos from both sides of a row containing nectarine "gardeta" trees. In contrast to what was previously proposed, due to the lack of a tractor during the period of the visit, the dataset was acquired by a human with a smartphone (Google Pixel 4A). Nearly over 8000 fruits were identified in the acquired dataset.

Image Acquisition

The image acquisition process was performed under real field conditions, using an ordinary smartphone, while trying to simulate the motion of a tractor. Acquiring the data with a smartphone was not a random decision, as mostly everyone owns one and knows how to work with it. Figure 4.2 represents how the dataset was captured. This approach greatly increases the difficulty in the implementation, as the images are not as sharp as they would be if a high-end camera were to be used.

Figure 4.2: Dataset acquisition with a smartphone. a) and b) represent the different sides of the tree row.

Two videos, each one representing one side of a tree row, were acquired. Eight trees were captured in total. Later, using a software[2], images were extracted from both videos every 20 frames, which resulted in a total of 230 images.

Fruit Annotation

Labelling all the fruits along the 230 images was a laborious and arduous task, as each photo contained, on average, 35 fruits. Precisely labelling all the fruits was hard, as fruits can be confused for dim leaves (lack of illumination) and blurred trunk pieces (poor image quality). There are also green fruits that can easily be confused with leaves. The software used to label all the images was CVAT[3], a free solution.

[2] https://www.dvdvideosoft.com/pt/products/dvd/Free-Video-to-JPG-Converter.htm
[3] https://cvat.org/

The Training Dataset

The training dataset consists of train and validation images. Train images are images which the detector will constantly see during the training process, while validation images are only used to test the detector accuracy along this process. One of the eight trees was reserved for the validation set, while all the others are used for the training process. The validation tree will also be used to test the final system accuracy. The train set has a total of 211 images of nectarine "gardeta" trees, and the same amount of negative images (images without any nectarine "gardeta" fruit in it). Figure 4.3 a) represents an image of the training set and b) a negative image of the same set.

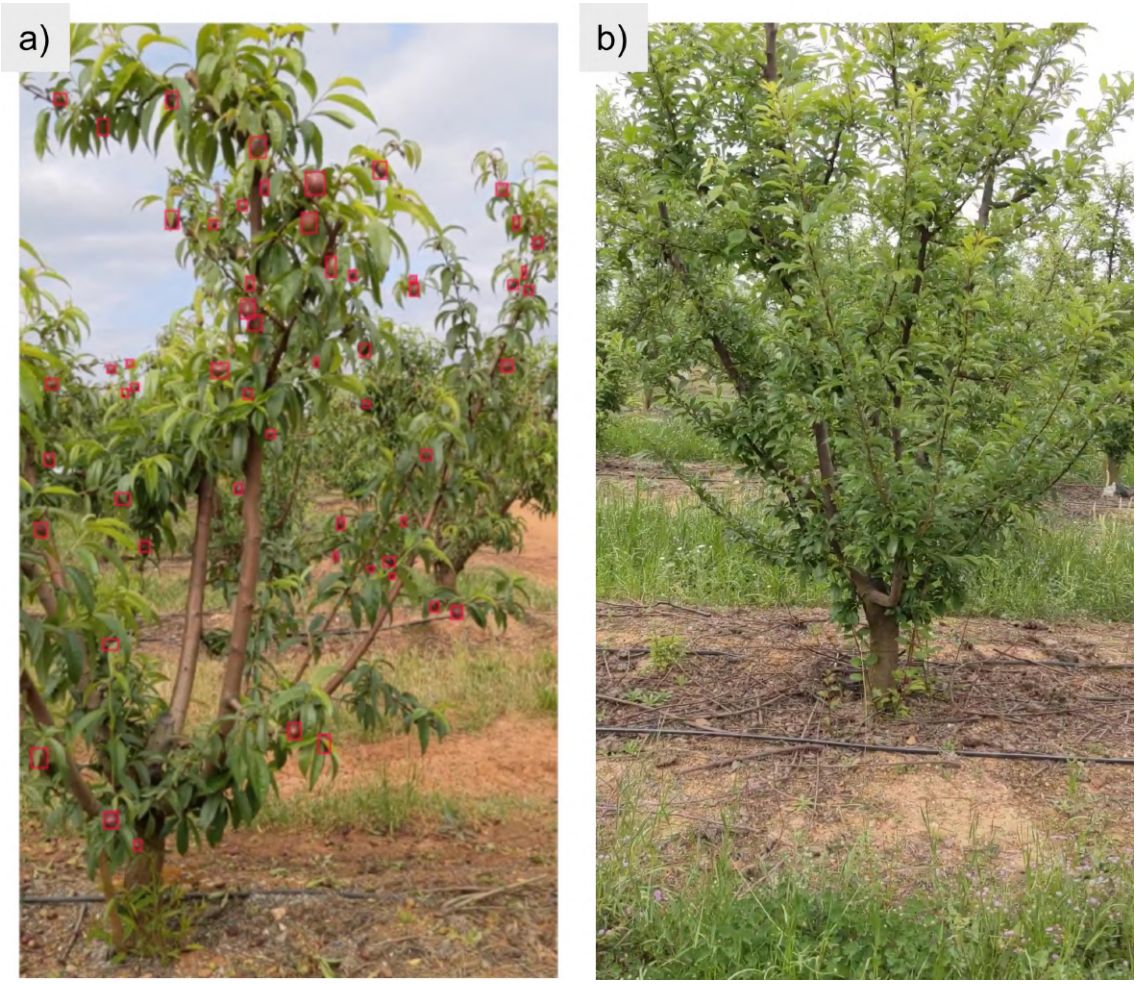

Figure 4.3: a) nectarine "gardeta" tree image and b) a negative plum black diamond tree image.

A rather curious event happened while testing different types of training processes. While training a variety of object detector models, these same models were identifying fruits that were missed during the manual labelling process. As such, the already labeled frames were manually relabeled considering this new fruit detections. Figure 4.4 is a representation of how the object detectors assisted in relabeling the dataset. This relabeled

dataset was then used during the training processes described further below.

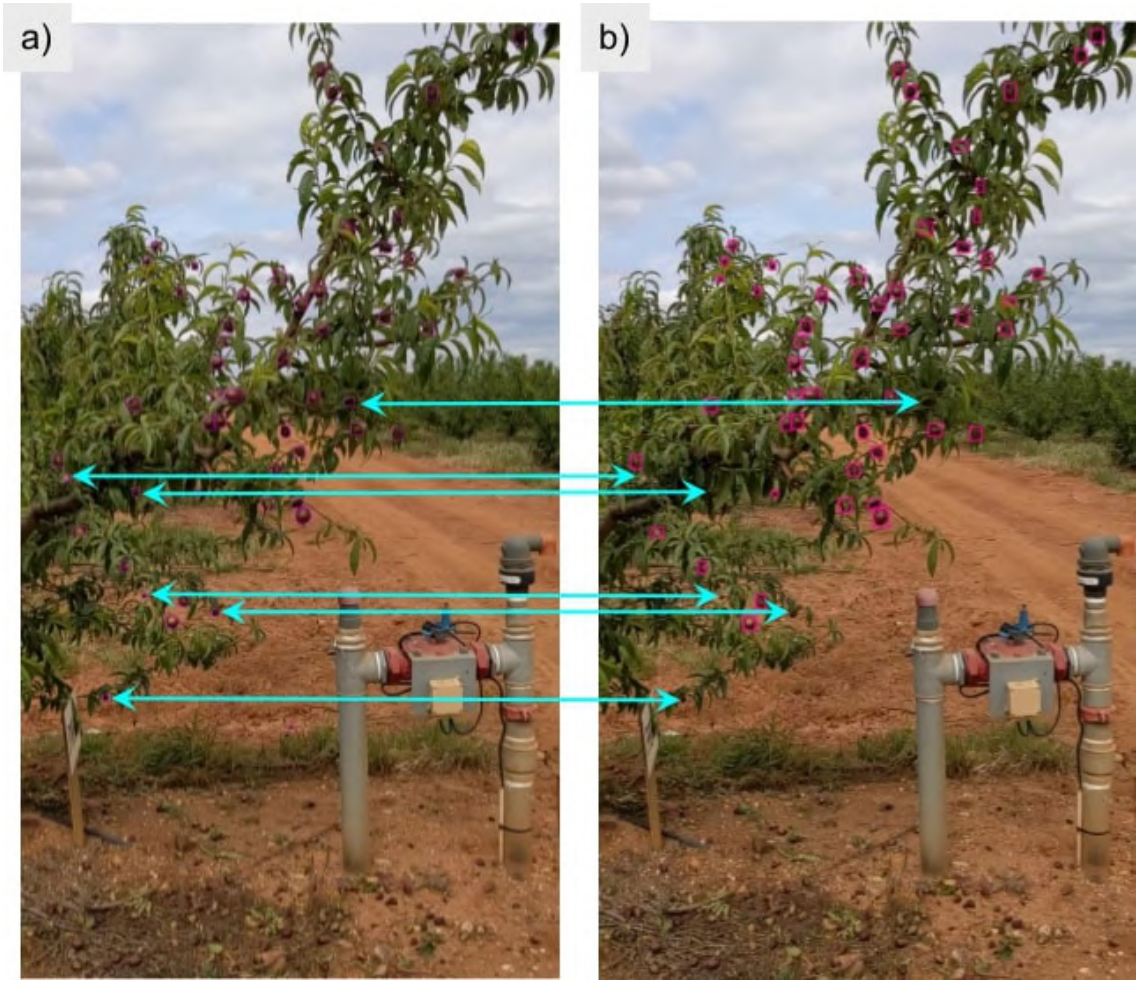

Figure 4.4: Using object detectors to manually assist labelling missed fruits. a) corresponds to automatic detections, while b) to the manually labeled fruits. In this particular frame, the detector helped in labelling six missed fruits.

4.1.2 Training YOLO

All the considered YOLO models were trained in google colab, using Alexey Bochkovskiy (known as AlexeyAB in GitHub) famous fork of darknet[4] framework[5]. The developed google colab file[6] was built so that it can be easily accessed and modified by anyone who wish to train a model supported by the AlexeyAB fork of darknet. The file also contains some tips on how specific parameters of the configuration files should be adjusted, as well as tips for general training.

[4]"Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation." Source: https://pjreddie.com/darknet/

[5]https://github.com/AlexeyAB/darknet

[6]https://colab.research.google.com/drive/1rjDHD0d3Mvtdetow_dcSw_ymfTMuWZP8?usp=sharing

Training Strategy

While training YOLO detector in the darknet framework, one has to look for the configuration file. This file has the structure definitions of the neural network to be trained. Among all possible definitions in the cfg file[7,8], there are some important parameters to respect before starting the training process. These parameters are:

- max_batches - is the number of iterations that will take place during training. According to AlexeyAB, for each class there must be 2000 iterations, but not less than 6000 in total;
- steps - define at which iterations the learning rate should be changed, according to a defined scale factor;
- width and height - represents the network size. Every image will be resized to this size during training and detection. Each value for width and height must be a multiple of 32;
- classes - number of classes in the dataset;
- filters - are the number of kernel filters to be used during convolution. This is one of the basic building blocks of an CNN, and its purpose is to help in the feature extraction process;
- subdivisions - will determine how many images the GPU can process at once. For better training performance, this parameter should be set to the lowest value possible, while avoiding CUDA out of memory error[9]. The max acceptable value for subdivisions is the specified batch size, which is normally defined as 64;
- letter_box - when set to "1", will keep the aspect ratio of loaded images during training. This is important in case the width is not equal to the height;
- random - when set to "1", will train YOLO for different input resolutions. This will find the input resolution which gives the best mAP results.

There are also parameters to ensure that data augmentation is performed during the training process. This includes randomly rotating images, randomly changing the saturation and brightness levels, randomly applying blur, randomly cropping images, among other operations.

Six different training processes were conducted using models of the YOLOv4 family. Each one with the same train and validation dataset, but with different configuration

[7] https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-%5Bnet%5D-section

[8] https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-different-layers

[9] Each GPU has its own number of images which it can process at once. This will depend on the image size, number of images, and GPU memory. If there are too many images, or large images, it is possible that a "CUDA out of memory" error might occur. In this case, the number of subdivisions should be increased, so that the GPU processes less images at once.

files/parameters. The chosen models to be trained were the default YOLOv4 model (with a custom configuration file proposed by AlexeyAB) and its scaled versions (YOLOv4P5 and YOLOv4P6), as they top the COCO benchmark. At the time of writing this dissertation, google colab best GPUs offer 16GB of memory on the paid plan. Unfortunately, this isn't enough to train the largest YOLOv4 model, YOLOv4P7, as it requires massive memory to train on the images of the current use case.

The chosen configuration parameters for each of the trained models are defined in table 4.1. YOLOv4 was trained with default settings "DS", that is, the default parameters in the original configuration file with little adjustments. YOLOv4P5 was trained three times with different configuration file parameters: one with DS; one with the width and height "WH" set so that they are as close to the original image resolution as possible; and the third one "WHR", where the width and height were also set to be as close to the original image as possible, but with the active random flag. Regarding YOLOv4P6, it was trained two times, with the same logic used in the training process of YOLOv4P5 being applied here. Unfortunately, it wasn't possible to train YOLOv4P6 with the random flag ("WHR") due to lack of GPU memory.

Before diving into the table, it is important to explain how the values for the filters parameters came up. For the default YOLOv4 model, it is defined by AlexeyAB[10] that

$$filters = (5 + classes) * 3 = (5 + 1) * 3 = 18$$

As for the YOLOv4PX versions[11], it is given by

$$filters = (5 + classes) * 4 = (5 + 1) * 4 = 24$$

Table 4.1: Configuration file parameter definition for all the trained models.

YOLO Model	v4 DS	v4P5 DS	v4P5 WH	v4P5 WHR	v4P6 DS	v4P6 WH
GPU	V100	V100	P100	P100	V100	P100
Train Time	12h	24h	93h	97h	30h	87h
Width	608	896	1024	604	1280	960
Height	608	896	1856	1472	1280	1792
Batch	64	64	64	64	64	64
Subdivisions	32	32	64	64	64	64
Filters	18	24	24	24	24	24
Letter Box	N/A	1	1	1	1	1
Random Flag	Yes	No	No	Yes	No	No

Beyond these parameters, in all the models, "max_batches" was set to 6000, "steps" was set to "4800,5400" and "classes" was set to 1.

[10] https://github.com/AlexeyAB/darknet#:~:text=L783-,change%20,-%5Bfilters%3D255
[11] https://github.com/AlexeyAB/darknet/issues/7838#:~:text=filters%3D%3C

4.2 Fruit Tracking

Achieving good accuracy in the fruit tracking process is crucial for the fruit yield estimation system. On top of being crucial, it is also a really challenging task to accomplish under real field conditions and with data taken with an ordinary smartphone. A state-of-the-art algorithm must be used in order to achieve the best possible results. That's why FastMOT was the chosen framework to be implemented [16]. FastMOT is a framework built on top of python that implements Deep SORT algorithm with some improvements. The developer of FastMOT swapped the feature extractor normally used in Deep SORT for a better ReID model, OSNet. The weights of the trained detector were converted to ONNX format[12], which were then converted once again by the FastMOT framework to a TensorRT file[13]. This enables FastMOT to perform detections using the TensorRT backend, which speeds up the inference times. Besides detection, the feature extractor replaced in Deep SORT also uses the TensorRT backend to speed up the whole tracking process. Similarly, algorithms, used in Deep SORT to perform the object movement predictions and associations between detections and predictions, were also optimized using Numba[14].

4.2.1 FastMOT

FastMOT supports, along other detectors, YOLOv4 and its scaled versions. The YOLO detectors previously trained will be used in order to test the accuracy of this tracking framework. In order to evaluate FastMOT effectiveness in tracking fruits and avoid fruit count duplication, it is necessary to set it up, configure it to run on our custom data, convert the trained model to ONNX format and, finally, run the fruit tracking system.

Setting up FastMOT

In the FastMOT[15] repository, a Dockerfile[16] can be found. This Dockerfile is used to set up the core dependencies needed to use FastMOT. In order to run the Dockerfile, nvidia-docker must be installed first on an Ubuntu environment.

To use an Ubuntu environment, a possible solution, which was implemented in this dissertation, is to use Windows Subsystem for Linux (WSL). WSL is a Microsoft project which lets Windows users run a Linux environment inside their Windows operating system, without the need to install virtual machines or dualboot[17]. After installing WSL[18],

[12] ONNX is an open format for ML models, which allows to interchange models between various ML frameworks and tools.

[13] A TensorRT file (.trt) is a file that results from the optimization of a neural network, which allows for ten times faster inference times. Source: https://developer.nvidia.com/tensorrt

[14] https://numba.pydata.org/

[15] https://github.com/GeekAlexis/FastMOT

[16] https://docs.docker.com/engine/reference/builder/

[17] https://docs.microsoft.com/en-us/windows/wsl/about

[18] https://docs.microsoft.com/en-us/windows/wsl/install-win10

one needs to install nvidia-docker[19].

After setting up both Ubuntu and nvidia-docker, it is finally possible to set up the FastMOT framework using the provided Dockerfile. This image requires an NVIDIA driver version >= 450 for Ubuntu 18.04 and >= 465.19.01 for Ubuntu 20.04. Assuming system requirements are met, running the following bash commands will set up FastMOT.

To begin with, it is necessary to clone the FastMOT repository from GitHub (listing 4.1).

Listing 4.1: Cloning FastMOT to a local directory.

```
$ cd $HOME
$ git clone https://github.com/GeekAlexis/FastMOT.git
```

The first command line sets the directory to the defined home directory and the second command line clones the git repository to a directory called "FastMOT".

After cloning the repository, it is necessary to build and run the docker image (listing 4.2).

Listing 4.2: Building and running a docker image.

```
$ docker build -t fastmot:latest .
$ docker run --gpus all --rm -it -v $(pwd):/usr/src/app/FastMOT \
-v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY \
-e TZ=$(cat /etc/timezone) fastmot:latest
```

After running both commands, a FastMOT docker container will be up and running inside the Ubuntu environment. Now it's necessary to build the YOLOv4 TensorRT plugin (listing 4.3).

Listing 4.3: Building YOLOv4 TensorRT plugin.

```
$ cd fastmot/plugins
$ make
```

Lastly, it's necessary to download a pretrained feature extractor model (OSNet) for Deep SORT to function properly (listing 4.4). This model doesn't come included with the repository due to its size.

Listing 4.4: Downloading pretrained OSNet model.

```
$ cd $HOME/FastMOT
```

[19] https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

```
$ ./scripts/download_models.sh
```

FastMOT is now successfully set up inside the WSL Ubuntu environment. Now, every time it is desired to use FastMOT, one simply needs to run the docker container using the "docker run" command provided above.

Following FastMOT set up, it is necessary to configure it to track custom classes.

Configuring FastMOT for Custom Model and Classes

FastMOT default configuration is set to track humans with YOLOv4 as the detector. For this dissertation use case, it is intended to track fruits (nectarine "gardeta") using the scaled version of YOLOv4, YOLOv4P5.

First, it is necessary to tell FastMOT what classes are to be tracked. In this case, it is desired to track nectarine "gardeta" (ng) fruits. To this end, the file "label.py" under FastMOT/fastmot/models must be edited. The code in listing 4.5 corresponds to the file for the current use case.

Listing 4.5: FastMOT custom class definition.

```
"""
90-class COCO labels
'unlabeled' (id = 0) is replaced with 'head' for CrowdHuman
These are different from the default 80-class COCO labels used by YOLO
"""

LABEL_MAP = (
    'ng'
)
```

Following class definition comes detector definition. Currently, FastMOT supports Single Shot Detector, YOLOv3 and derivatives, and YOLOv4 and derivatives. The detector to be used can be defined in mot.json file, which can be found under FastMOT/cfg directory. The detector to be used during this dissertation is from the YOLO family. To instruct FastMOT to use YOLOv4P5, it is necessary to edit the "detector_type" variable and set it to "YOLO" and set the variable "model" in "yolo_detector_cfg" to the desired YOLOv4P5. Still in "yolo_detector_cfg", the variable "class_ids" must also be changed accordingly to the class defined in "label.py", which in this case will be set to 0 (zero). The code used to configure FastMOT (mot.json) can be found in appendix A.

Not only the detector type needs to be defined, but also the detector configurations. YOLOv4P5 configurations can be edited in "yolo.py", which can be found under FastMOT/fastmot/models directory. This file contains all the definitions for the different types of detectors supported by FastMOT. For YOLOv4P5, the definitions to be edited are represented in listing 4.6.

Listing 4.6: FastMOT model configuration.

```
...

class YOLOv4P5(YOLO):
    ENGINE_PATH:    path to TensorRT engine (converted at runtime)
    MODEL_PATH:     path to ONNX model
    NUM_CLASSES:    total number of classes
    LETTERBOX:      keep aspect ratio when resizing
    NEW_COORDS:     new_coords parameter for each yolo layer
    INPUT_SHAPE:    input size in the format "(channel, height, width)"
    SCALES:         scale_x_y parameter for each yolo layer
    ANCHORS:        anchors grouped by each yolo layer

...
```

- ENGINE_PATH is the path to a TensorRT engine file, which will be automatically generated at runtime;
- MODEL_PATH is the path to an ONNX file. This file is generated from the conversion of both weights and configuration files from the detector training process. More details on how to proceed with the conversion can be found in Model Conversion to ONNX;
- NUM_CLASSES simply represents the number of classes to be tracked;
- LETTERBOX is a variable that tells the model to keep the aspect ratio of an image while resizing it during inference;
- NEW_COORDS is a parameter in the object detector configuration file that tells the model how the predicted bounding boxes should be calculated[20];
- INPUT_SHAPE is the dimensions used in the configuration file during the training process;
- SCALES corresponds to the parameter "scale_x_y" that can be found in the configuration file used for the training phase;
- ANCHORS is the anchors grouped by each YOLO layer according to the configuration file used during training. This anchors can be obtained from the "mask" and "anchors" variables of each YOLO layer in the configuration file. Each mask number serves as a pointer to a group of two anchors in the "anchors" variable. Say mask = 0,2 and anchors = a,b,c,d,e,f then, in FastMOT, ANCHORS = a,b,e,f.

[20] https://github.com/AlexeyAB/darknet/issues/6987#issuecomment-729218069

The chosen YOLOv4 model to be used along the implementation of this dissertation is "YOLOv4P5 WH", as it produced the best results upon training. The defined parameters for this model are represented in listing 4.7.

Listing 4.7: FastMOT "YOLOv4P5 WH" parameters.

```
class YOLOv4P5(YOLO):
    ENGINE_PATH = Path(__file__).parent / \
                  'yolov4_p5_w_1024_h_1856.trt'
    MODEL_PATH = Path(__file__).parent / \
                 'yolov4_p5_w_1024_h_1856.onnx'
    NUM_CLASSES = 1
    LETTERBOX = True
    NEW_COORDS = True
    INPUT_SHAPE = (3, 1856, 1024)
    SCALES = [2.0, 2.0, 2.0]
    ANCHORS = [[13,17, 31,25, 24,51, 61,45],
               [48,102, 119,96, 97,189, 217,184],
               [171,384, 324,451, 616,618, 800,800]]
```

The core configurations for FastMOT to work on custom classes is finished. Now, it is necessary to adjust specific tracking parameters to this dissertation use case.

Additional Parameter Tuning

All the parameter tuning takes place in "mot.json" file, which can be consulted in its entirety on appendix A. Over the next paragraphs we'll discuss which parameters were updated and why.

Parameter "resize_to" tells FastMOT to what size it should resize the input data to. This should be equal to the dimensions used in the configuration file of the chosen YOLO model. Since "resize_to" takes as arguments two variables, width and height, and given the chosen definitions for the trained YOLO model, this variable is set to [1024, 1856].

Regarding "video_io", the variables "resolution" and "frame_rate" must be adjusted accordingly to the input video settings. In this case, and knowing that "resolution" takes [width, height] as parameters, this variable is set to [1080, 1920] and "frame_rate" variable is set to 30. This indicates FastMOT that the input video runs at 30 frames per second at a resolution of 1080 width and 1920 height.

The variable "detector_frame_skip" instructs FastMOT how many frames it should skip per processed frame. As an example, if this variable is set to 7, for every processed frame, FastMOT will skip 7. The developer of FastMOT found this method to fasten the process of tracking, and uses KLT to fill in the gaps of the skipped frames. While it should work on some scenarios where the end goal is to track big and slow targets, skipping frames in small and fast moving targets like nectarines was tested and concluded that

produces awful results. As so, this variable is set to 1, so that every single input frame is processed.

Next comes "yolo_detector" parameters definition. In addition to the already mentioned variables ("model" and "class_ids"), variable "conf_thresh" should be set to the chosen detector IoU threshold, 0.3. As seen in the results chapter, this was the value for which the detector obtained better results in both quantitative and qualitative outcomes.

The variable "max_age" defined inside "multi_tracker" represents the max number of undetected frames before a track is terminated. As an example, if "max_age" = 30 and fruit with track ID 23 goes undetected for 30 frames, its track is terminated. This variable has to be chosen in respect with "detector_frame_skip", such that the product between "max_age" and "detector_frame_skip" is equal or close to 30, as recommended by the developer of FastMOT. Since "detector_frame_skip" was set to 1, "max_age" will be set to 30.

The variable "motion_weight" will be adjusted in respect to the obtained results. This variable represents a motion weight in the matching cost function. After many iterations, it was found that the value that performed the best qualitative and quantitative results was "motion_weight"=0.02. Lastly, the variable "conf_thresh" will be adjusted based on the chosen IoU threshold of the trained detector, that is, 0.3.

The default "kalmal_filter" parameters[21] of FastMOT are set to track humans. Nectarines are smaller by a few orders of magnitude. As such, and according to FastMOT developer, the correct way to define Kalman filter parameters is to think in terms of object size, as Kalman filter parameters are scaling factors[22]. As such, we assume that nectarines are roughly 6x times smaller than a human body. Considering that in issue #187[23], the developer of FastMOT assumed human heads to be 5x smaller than human bodies, a nectarine being 6x smaller than a body seems like a fair approximation. In fact, this approximation produced the best qualitative and quantitative outcomes. After applying this factor, we end up with the Kalman filter parameters represented in listing 4.8.

Listing 4.8: Custom Kalman filter parameter definition. On the left side of the arrow are the old parameters, while on the right side are the new, scaled ones.

```
"kalman_filter": {
                    "std_factor_acc": 2.25 -> 13.5,
                    "std_offset_acc": 78.5 -> 471,
                    "std_factor_det": [0.08, 0.08],
                    "std_factor_flow": [0.14, 0.14],
                    "min_std_det": [4.0, 4.0],
                    "min_std_flow": [5.0, 5.0],
```

[21] https://github.com/GeekAlexis/FastMOT/blob/ffc637fd95df20fc1a7ab22ead8cd3247c8e4178/fastmot/kalman_filter.py

[22] https://github.com/GeekAlexis/FastMOT/issues/187#issuecomment-907702450

[23] https://github.com/GeekAlexis/FastMOT/issues/187

```
        "init_pos_weight": 5 -> 30,
        "init_vel_weight": 12 -> 72,
        "vel_coupling": 0.6,
        "vel_half_life": 2 -> 12
    }
```

Model Conversion to ONNX

Now, it's necessary to convert the trained darknet model to the ONNX format. This conversion is necessary so that FastMOT is further able to convert the ONNX file to a TensorRT file (.trt), which allows for faster inference times.

To convert the trained darknet model to ONNX format, a script provided by the FastMOT framework should be used. This script is located under FastMOT/scripts, and it's named "yolo2onnx.py". In order to run it, some dependencies must firstly be installed (listing 4.9).

Listing 4.9: Dependencies installation to run ONNX conversion.

```
$ apt-get update && apt-get -y install sudo
$ sudo apt-get install protobuf-compiler libprotoc-dev
$ pip3 install onnx==1.4.1
```

After installing the dependencies, run the conversion script by adding the paths to the weights and configuration files from the detector training process (listing 4.10):

Listing 4.10: Darknet model conversion to ONNX.

```
$ ./scripts/yolo2onnx.py --config <path_to_cfg_file> \
  --weights <path_to_weights_file>
```

After running this script, an ONNX file will be generated under FastMOT/fastmot/-models directory.

Running FastMOT on a Custom Dataset

Once everything is set up, running FastMOT comes down to one bash command line. The used bash command is represented in listing 4.11.

Listing 4.11: Bash command to run FastMOT.

```
$ fname="v2_second_tree_only"
$ python3 app.py --input_uri custom/videos/$fname.mp4 --mot \
  --log eval/results/MOT20-$fname.txt --gui --output_uri \
```

```
results/results_$fname.mp4
```

Where "fname" is a local variable that is the name of the input video to be tracked, as well as the name of the output files, —mot is to instruct FastMOT to run multiple object tracker, —log is to log all the results to an txt file, and —gui is to prompt a live tracking display. Once this command is executed, FastMOT will take —input_uri video, process it using the defined settings and parameters, and output a video in the defined path and name –output_uri.

Chapter 5

Results and Discussion

The results of the implementation process of this dissertation are divided into two parts:

- First, results regarding the efficiency of the DL detector should be analyzed. That is, how well the trained detectors can identify fruits in frames they had never seen before. To analyze their accuracy, fruits along validation frames will be manually labeled and counted, to then be compared to the automatically detected and counted fruits by the trained YOLO detectors;
- Second, the accuracy of the implemented solution to count fruits in tree row sides shall be analyzed. For this analysis, fruits will be manually counted in two validation videos representing a validation tree. One video contains the front of that tree, and the other the back of it. The results from the manual counting will then be compared to the results given by the proposed tracking/counting system.

5.1 Frame-by-frame Fruit Count Accuracy

All the YOLO models were trained in google colab, using its available GPUs, Tesla V100 or Tesla P100, both with 16 gigabytes of dedicated memory. The combined training time of all the four different models was above 343 hours, with an average of 57 hours per train. The "YOLOv4P5 WH" model had the best overall performance. As can be seen in figure 5.2, all the models performed relatively well. The charts presented in this image were generated from the darknet framework while each model was training, showing the mAP and loss metric of a YOLO model along all the training iterations (6000). This mAP was calculated for an IoU threshold of 0.5. Over the next sections, it will be discussed how adjusting this threshold increased the overall quality of all the models.

5.1.1 Quantitative Outcomes

The performance of each trained model was assessed with two metrics, accuracy and mAP. The accuracy metric measures how accurately a model can predict how many fruits are

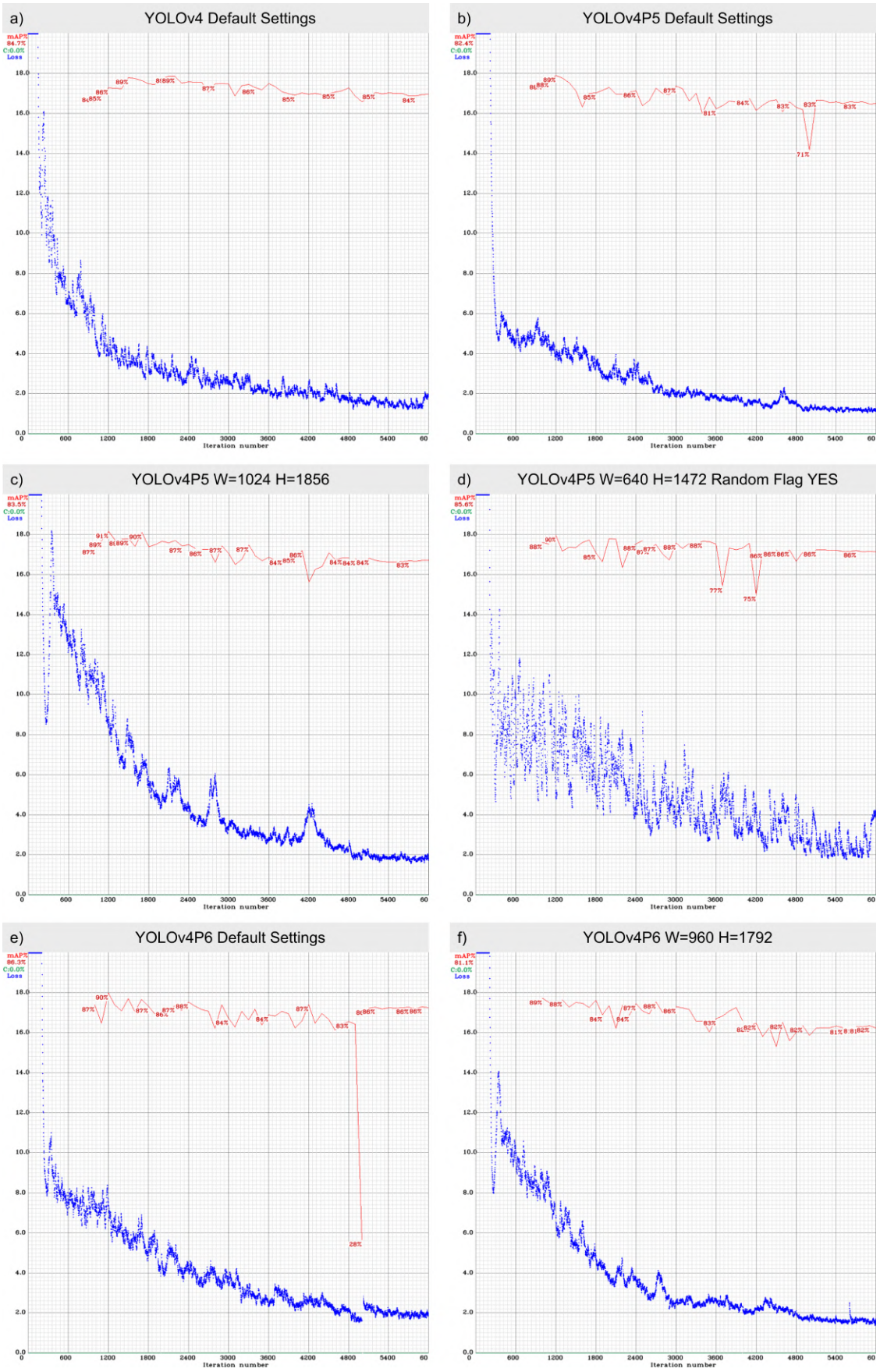

Figure 5.1: Charts generated by the darknet framework during training.

in a given frame, while the mAP metric determines how well the model is classifying the literal fruits. For this dissertation scenario, which end is to estimate the production yield of a crop, the most important metric is the one which tells how many "fruits" are being counted per frame, whether it is a fruit or not. This is the accuracy metric. Considering there is a small validation dataset (18 frames from both sides of a tree), it is not safe to choose the best model only considering the accuracy metric. Here comes the importance of considering and analyzing the mAP metric. Let us suppose a given model had an accuracy of 97%, but the mAP metric was just below 80%. This means that in that 97%, many of the detected objects weren't fruits, but rather leaves, trunk chunks, or any other object the detector thought was a fruit. Even though the probabilities are not that high, this scenario is possible when considering a small validation dataset. On the other hand, if the model has a high mAP and a lower accuracy, it is possible that it is detecting some fruits with high accuracy, but it is missing a lot of them, probably due to fruit occlusion. The chosen model was the one which had both high and similar accuracy/mAP.

In order to determine the average accuracy and mAP of all the trained models, six tables have been made (appendix B). These tables consist of six columns, where the first one represents the frame for which the accuracy and mAP will be calculated, the second one has the number of fruits that were manually counted in each frame, the third one the fruits that were automatically counted in each frame, and the last two the respective metrics in study. The last column "mAP@0.3" means that the mAP was determined using a threshold higher or equal to 0.3. By testing multiple threshold values, both quantitative and qualitative outcomes show that the 0.3 threshold produced the best results in terms of accuracy and mAP.

All the calculated and determined accuracy and mAP were put together in two separate tables, as a means to compare the results from the different YOLO models. Table 5.1 holds the accuracy, while table 5.2 holds the mAP values for all the trained models.

For all the models, without exception, the average mAP results were higher than the average accuracy. This conclusion is an indication that, although the detector is detecting with high accuracy the fruits presented in all the frames, it is skipping some of them. It is likely that this is happening due to poor photo quality, a consequence of the dataset being taken with an ordinary smartphone. Poor photo quality promotes fruit occlusion, fruit blur, fruit color not being so sharp, among other challenges for the detector. Another reason to justify such results, beyond the quality of the dataset, is its quantity. Although some fruits really are difficult to detect even when manually trying to identify them, an increased number of photos from different perspectives would probably increase both detector accuracy in predicting the number of fruits and its mAP. Section 5.1.2 will dive into more conclusions between the relation of the results from both tables, as visual results give a different perspective in this matter.

As can be concluded from the results itself, YOLOv4P5 with its width and height adjusted so that it would best match the original image size of the dataset photos ("YOLOv4P5 WH") had the best results both in accuracy and mAP. One might question why "YOLOv4P6

Table 5.1: Fruit count accuracy for the trained YOLO models.

YOLO Model	v4 DS	v4P5 DS	v4P5 WH	v4P5 WHR	v4P6 DS	v4P6 WH
Frame 1	93%	76%	97%	93%	97%	83%
Frame 2	92%	78%	86%	86%	89%	73%
Frame 3	93%	83%	95%	85%	98%	85%
Frame 4	100%	71%	86%	88%	97%	83%
Frame 5	91%	72%	94%	91%	91%	74%
Frame 6	77%	71%	81%	87%	85%	75%
Frame 7	83%	72%	94%	87%	91%	77%
Frame 8	96%	98%	93%	98%	93%	87%
Frame 9	80%	79%	95%	84%	89%	71%
Frame 10	81%	84%	84%	81%	88%	74%
Frame 11	97%	82%	95%	89%	92%	82%
Frame 12	87%	76%	97%	89%	92%	87%
Frame 13	89%	81%	98%	89%	91%	85%
Frame 14	91%	79%	91%	95%	93%	80%
Frame 15	95%	75%	91%	88%	89%	86%
Frame 16	94%	56%	81%	65%	75%	64%
Frame 17	82%	78%	93%	84%	87%	80%
Frame 18	94%	69%	78%	83%	81%	75%
Average	89.8%	76.7%	90.5%	86.9%	89.9%	79.0%

WH" didn't outperform "YOLOv4P5 WH", since the same logic is being applied to both models but one of them (P6) is a larger model, expected to achieve a higher mAP. A possible explanation for this results might be that, although P6 is a larger model than P5, it also requires more memory for the GPU to train it. As a consequence, the width and height of the trained v4P6 WH model was lower (W=960; H=1792) than the width and height of the trained v4P5 WH one (W=1024; H=1856), which is closer to the original width and height of the dataset images. It might also be due to bad luck. This last possibility could only be validated using a larger dataset for the training process.

Taking the chosen model "YOLOv4P5 WH" as an example, table 5.3 has been made and holds the mAP@0.3 and mAP@0.5 values for all frames for this model. As can be seen from it, without exception, all the mAP values are greater when the inference is done with an IoU threshold of 0.3. Further analysis on how decreasing the IoU threshold improved the results will be made in the next subsection.

Figure 5.2 gives a different perspective on the quantitative outcomes of both accuracy and mAP metrics. The first conclusion that can be drawn from these charts is that there are frames where the detectors struggle to precisely predict and count the fruits (e.g. frame 6 mAP and frame 16 accuracy, respectively), whereas there are others where the detectors can precisely predict them (e.g. frame 13 mAP and frame 8 accuracy, respectively). This occurrence is an indication on how the changes in light, shadows and perspective along multiple frames of the same tree can affect the detector accuracy and mAP metrics.

Table 5.2: mAP@0.3 for the trained YOLO models.

YOLO Model	v4 DS	v4P5 DS	v4P5 WH	v4P5 WHR	v4P6 DS	v4P6 WH
Frame 1	91%	95%	96%	89%	92%	89%
Frame 2	97%	93%	98%	96%	95%	89%
Frame 3	95%	96%	91%	95%	95%	96%
Frame 4	95%	92%	95%	94%	92%	96%
Frame 5	97%	92%	94%	96%	93%	94%
Frame 6	84%	79%	87%	86%	82%	86%
Frame 7	88%	91%	88%	89%	88%	78%
Frame 8	94%	93%	92%	93%	95%	91%
Frame 9	94%	96%	97%	94%	97%	90%
Frame 10	96%	94%	96%	94%	95%	89%
Frame 11	96%	95%	94%	96%	94%	92%
Frame 12	90%	92%	95%	92%	92%	92%
Frame 13	98%	97%	97%	98%	98%	98%
Frame 14	91%	92%	91%	90%	91%	88%
Frame 15	95%	95%	96%	96%	97%	95%
Frame 16	88%	89%	89%	90%	90%	82%
Frame 17	93%	92%	98%	91%	93%	94%
Frame 18	91%	91%	90%	89%	90%	93%
Average	93.0%	92.4%	93.5%	92.6%	92.9%	90.5%

This difference in performance between frames will be further discussed when analyzing the qualitative outcomes.

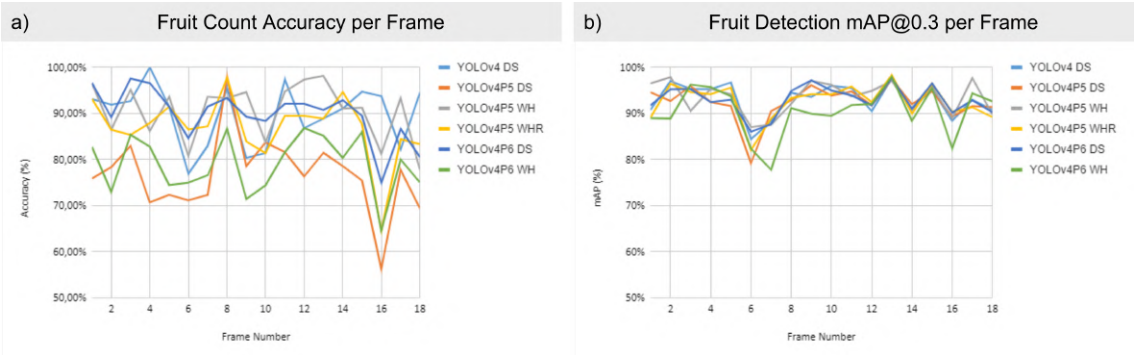

Figure 5.2: Accuracy and mAP metrics charts of all the trained models. Chart a) represents the fruit count accuracy, while chart b) the fruit detection mAP@0.3 along all frames.

Another interesting conclusion is that while on the mAP chart all the curves are close to each other, on the accuracy chart they deviate a lot more. This is a curious indicative to dive into in the next section, where the qualitative outcomes will be explored. Table 5.4 assists knowing in which frame the difference between the accuracy and mAP metric was the highest for the chosen model, "YOLOv4P5 WH". This table holds the difference between the accuracy and mAP metric for each frame. As it can be concluded,

Table 5.3: Comparison of mAP metric when inference is performed using the "YOLOv4P5 WH" model with a threshold of 0.3 vs 0.5.

Frame No.	mAP@0.3	mAP@0.5	Difference
1	96.49%	94.32%	2.17%
2	97.87%	97.87%	0.00%
3	90.56%	87.29%	3.27%
4	95.33%	88.16%	7.17%
5	94.13%	94.13%	0.00%
6	87.05%	81.18%	5.87%
7	87.55%	81.59%	5.96%
8	92.07%	89.34%	2.73%
9	97.08%	91.29%	5.79%
10	96.19%	96.19%	0.00%
11	93.70%	90.91%	2.79%
12	94.88%	94.88%	0.00%
13	97.05%	94.76%	2.29%
14	90.88%	90.88%	0.00%
15	95.83%	95.45%	0.38%
16	88.86%	86.57%	2.29%
17	97.67%	92.39%	5.28%
18	89.78%	85.49%	4.29%

the frame which has the biggest difference between the accuracy and mAP metric is frame number 10 (12.47%). This frame will be subjected to analysis in the following section, Quantitative Outcomes.

In order to confirm that there is an an overall bigger deviation in the accuracy metric when comparing to the mAP metric along all the models, a similar table to table 5.4 was made. Table 5.5 holds the standard deviation values of both accuracy and mAP metrics between all models and frames, as well as the overall standard deviation between both metrics. As it can be concluded, while the accuracy metric has an average SD of 6.93%, the mAP metric has an average SD more than three times lower, of 1.96%. This confirms that, comparing to the accuracy values, the mAP values do not change that much between models along different frames.

5.1.2 Qualitative Outcomes

The analysis of qualitative outcomes is as important as the analysis of the quantitative outcomes. All results presented in this section were obtained using the chosen YOLO model, "YOLOv4P5 WH". Three main topics will be explored:

- How changing the IoU threshold from 0.5 to 0.3 can positively affect the detection results;
- Why the detector shows high accuracy on some frames, while struggling on others;

Table 5.4: Difference between the accuracy and mAP metric for "YOLOv4P5 WH" model.

Frame No.	Model Accuracy	Model mAP	Difference
1	96.55%	96.49%	0.06%
2	86.49%	97.87%	11.38%
3	95.12%	90.56%	4.56%
4	86.21%	95.33%	9.12%
5	93.62%	94.13%	0.51%
6	80.77%	87.05%	6.28%
7	93.62%	87.55%	6.07%
8	93.33%	92.07%	1.26%
9	94.64%	97.08%	2.44%
10	83.72%	96.19%	12.47%
11	94.74%	93.70%	1.04%
12	97.37%	94.88%	2.49%
13	98.15%	97.05%	1.10%
14	91.07%	90.88%	0.19%
15	91.23%	95.83%	4.60%
16	81.25%	88.86%	7.61%
17	93.33%	97.67%	4.34%
18	77.78%	89.78%	12.00%

- What's happening in the frames where the mAP is way higher than the accuracy?

Regarding the first matter, it has been seen in the quantitative outcomes how decreasing the IoU threshold from 0.5 to 0.3 increases the mAP metric. But what is really happening when it comes to qualitative outcomes? Figure 5.3 a) and b) illustrates an example where on the left frame (denoted as frame a)), the inference was performed with an IoU threshold of 0.3, while on the right frame (frame b)), the inference counted with an IoU threshold of 0.5. To better understand the different types of detections happening in both frames, colored boxes were used, where each color represents one specific object being detected. Green stands for correctly detected fruits, blue stands for missed fruits, and red for all the other wrong detections, be it leaves, trunk chunks, etc. Analyzing both inferred frames: on frame a) there are a total of 47 correctly detected fruits, 11 missed fruits, and 3 wrong detections; while on frame b) 43 correctly detected fruits, 15 missed ones, and 0 wrong detections. The results came out as expected. That is, by decreasing the IoU threshold, fewer fruits were missed (47 vs 43) but more wrong detections were made (3 vs 0). Of all the tested IoU thresholds, this was the best not only in terms of the mAP metric, but also accuracy metric, as more fruits are correctly detected and wrong detections help to fill the missed ones.

For the second question, frames 13 and 18 were selected to analysis. While frame 13 had an accuracy of 98.15%, the highest among all frames, frame 18 had an accuracy of 77.78%, the lowest among all frames. Similar to the prior analysis, figure 5.4 a) and b)

Table 5.5: Accuracy and mAP metrics standard deviation when inference is performed using the "YOLOv4P5 WH" model.

Frame No.	Accuracy SD	mAP SD	SD between Accuracy and mAP
1	8.45%	3.05%	3.82%
2	7.13%	3.36%	2.67%
3	6.06%	2.08%	2.81%
4	10.44%	1.47%	6.34%
5	9.67%	1.83%	5.55%
6	5.89%	2.88%	2.12%
7	8.38%	4.54%	2.71%
8	4.14%	1.42%	1.92%
9	8.20%	2.77%	3.84%
10	4.57%	2.45%	1.50%
11	6.66%	1.45%	3.68%
12	7.01%	1.46%	3.93%
13	5.62%	0.43%	3.67%
14	6.84%	1.23%	3.96%
15	6.60%	0.69%	4.18%
16	13.59%	2.94%	7.53%
17	5.52%	2.33%	2.26%
18	8.50%	1.22%	5.15%
Average	6.93%	1.96%	3.51%

illustrates both frames (at the left, a) representing frame 13 and, at the right, b) representing frame 18) and its correct, miss and wrong detected fruits. Following the previous box color code, green stands for correctly detected fruits, blue stands for missed fruits, and red for all the other wrong detections. Analyzing both frames, for the highest accuracy frame, it didn't miss a single fruit, but wrongly detected 3 objects that were not fruits. As for the lowest accuracy frame, the detector missed 10 fruits and got 2 wrong detections. The main conclusion to draw from this analysis is that when there are frames with a low accuracy, it does not mean the detector is wrongly classifying many other objects, but rather it is missing a lot of them. This proves the premise that the quality of a photo have a huge impact in the detector accuracy in predicting the number of fruits in a photo. Another interesting analysis is represented in figure 5.4 a') and b'). a') has the 4 wrong detections of a), while b') has the 2 wrong detections of b). Looking at the "wrong detections" leaves a question unanswered, are those really "wrong detections", or rather fruits that weren't labeled due to the lack of resolution of the photo? This answer could only be answered by analyzing the tree in real life and concluding whether those are fruits or not. If any or all of the "wrong detections" are indeed fruits, this could be greatly affecting the detector training process as well as the values of both two final metrics, which would likely be a few percentage points closer to 100%.

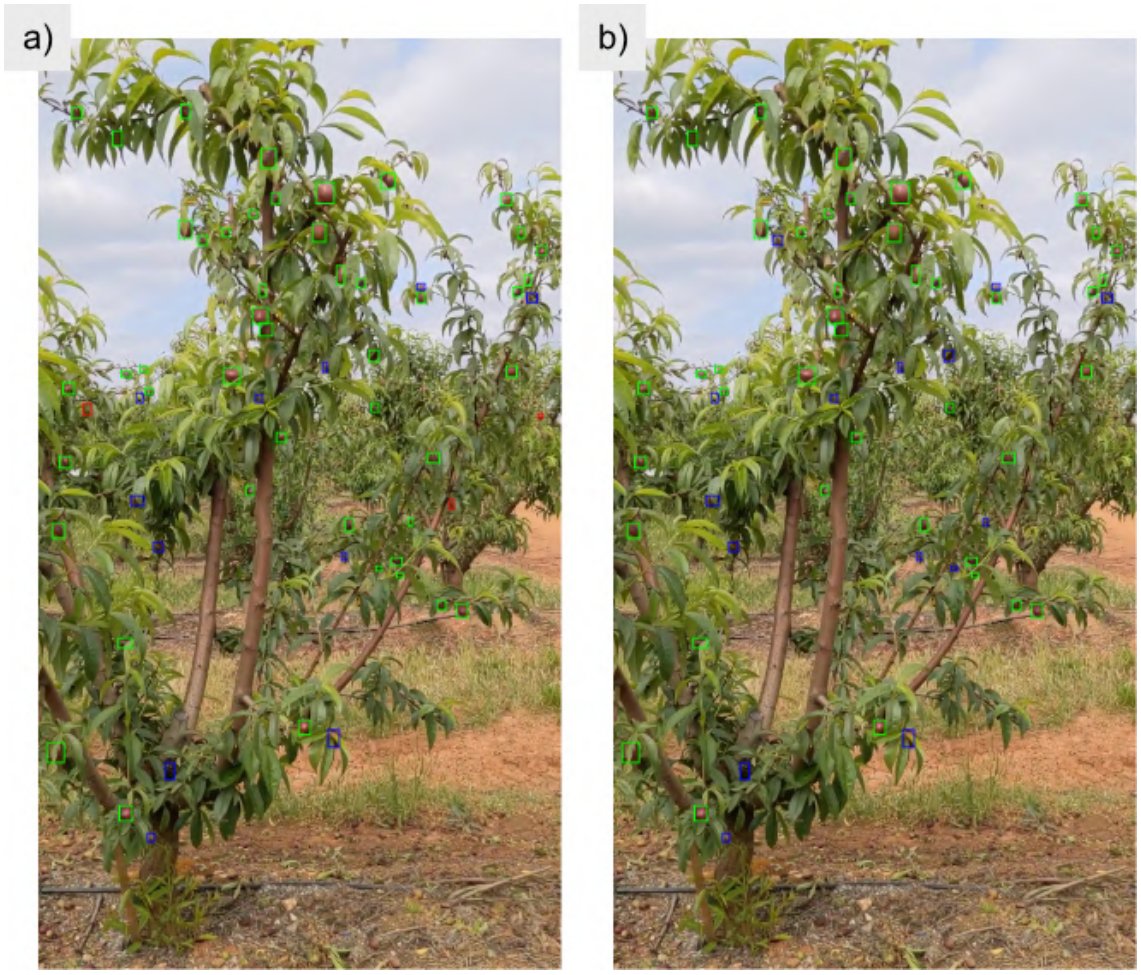

Figure 5.3: Comparison of inferencing the same frame with the same model but with different IoU thresholds. a) was inferred with an IoU threshold of 0.3, while b) with an IoU threshold of 0.5.

For the last question, the chosen frame for analysis was frame 10. This frame corresponds to the frame where the difference between the mAP and accuracy is the highest, having an mAP of bigger value than the accuracy. As before, figure 5.5 illustrates the correctly detected fruits as green, missed ones as blue and wrong detections as red. This results prove the point stated before. That is, if there is a high mAP and a low accuracy, the detector is likely to be classifying the fruits with high precision (only one wrong detection in all 36 detections), but missing a lot of them (8 in 43 fruits). The takeaway conclusion from this analysis is that, for this dissertation scenario where the end goal is to estimate the production yield of a crop, it is not safe to choose a model based only on its mAP metric. The model has to be chosen based not only on mAP, but also accuracy.

Lastly, there is one interesting conclusion when looking at all the inferred frames. In none of them did the detector considered the many fruits that were on the ground. Knowing if the detector would consider or not the fruits on the ground was a major concern upon implementation. The fact that it didn't show how mature and accurate DL

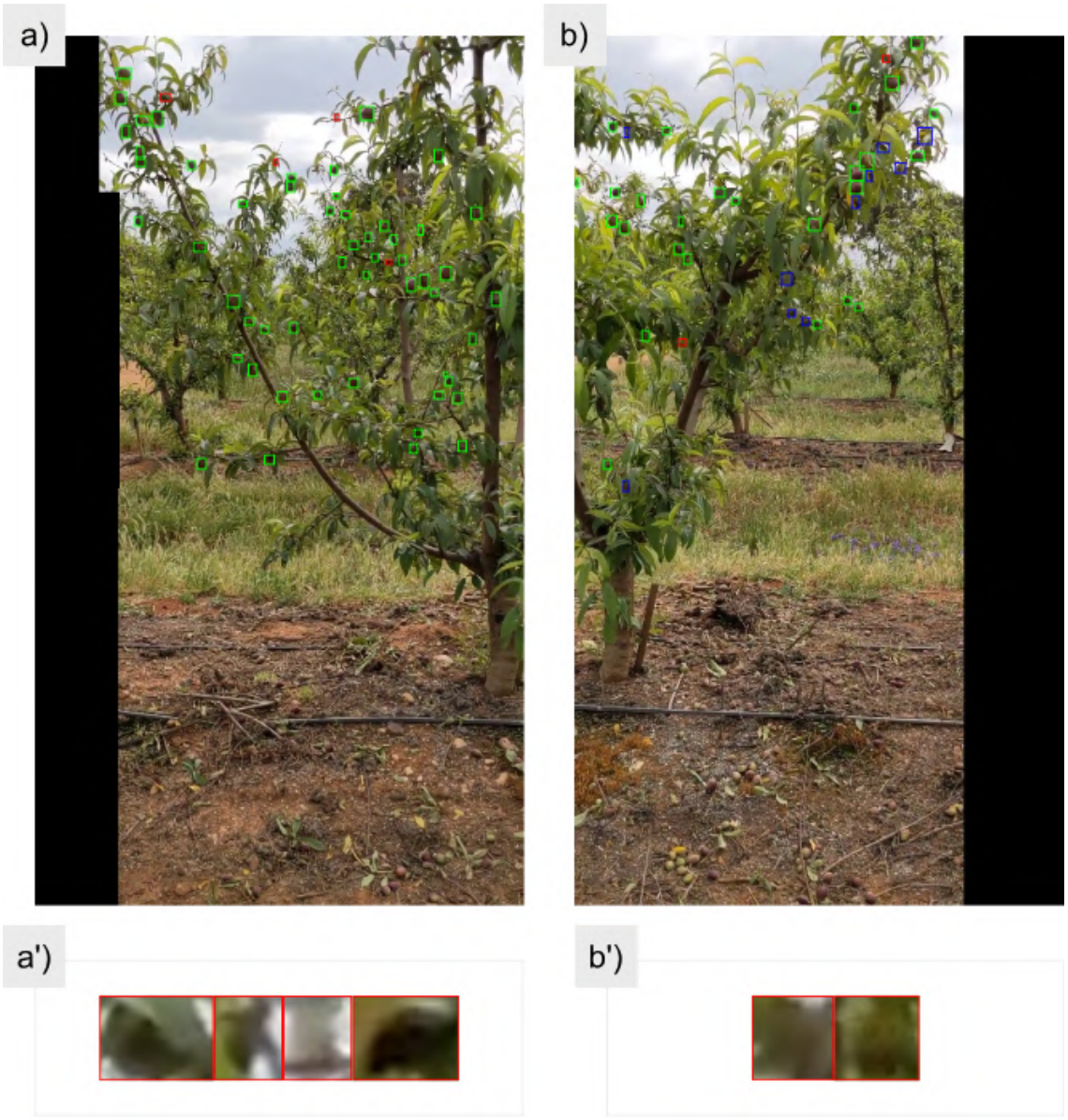

Figure 5.4: Inference of the highest accuracy frame, a), and lowest accuracy frame, b). a') and b') represents the 4 and 2 objects that the detector wrongly classified as fruits on a) and b), respectively.

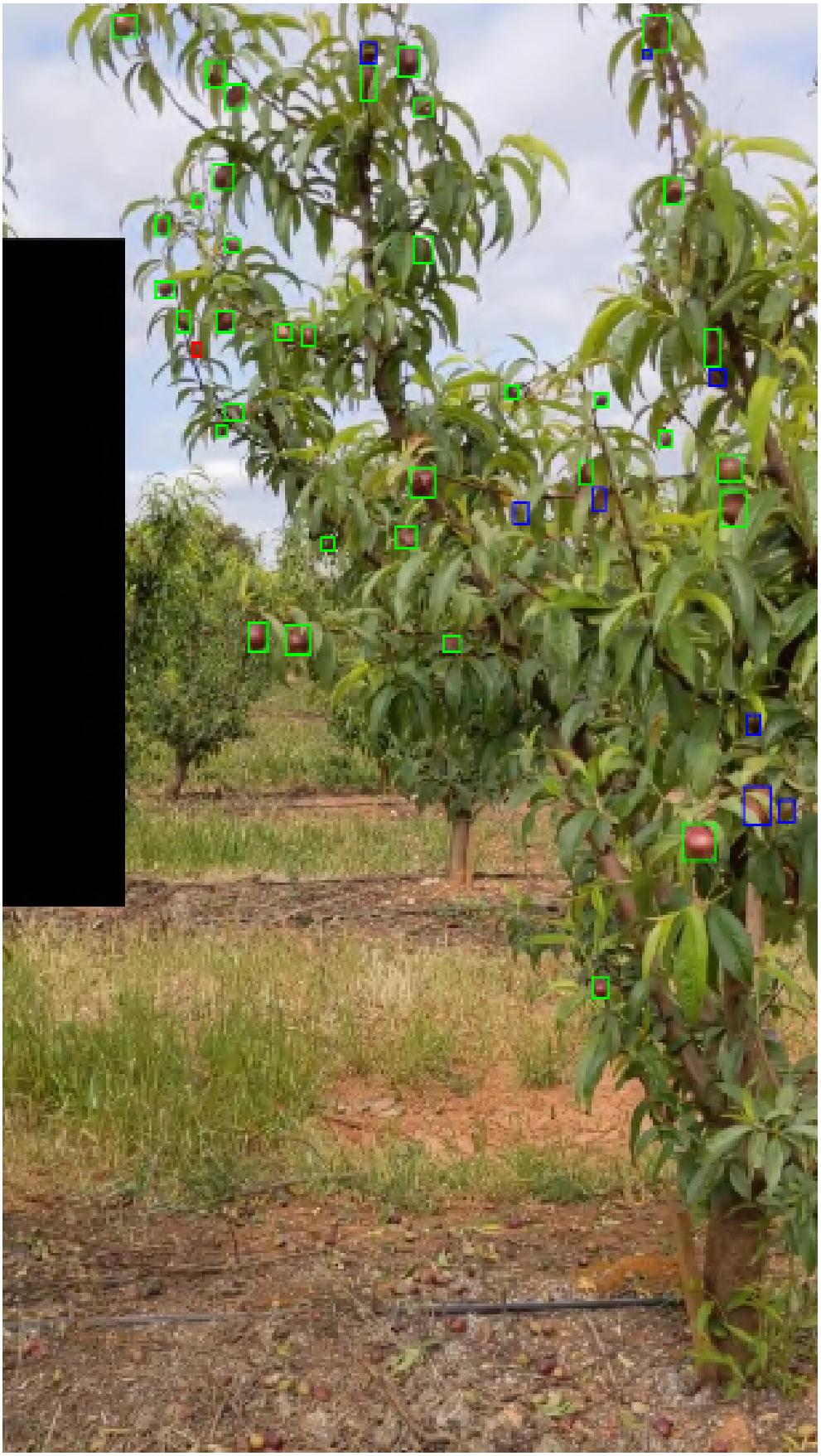

Figure 5.5: Inference of a frame with a high mAP and low accuracy.

object detectors can be, especially YOLOv4 models.

5.2 2D Fruit Yield Estimation

All the presented results were obtained using the mentioned FastMOT framework and using a computer with the following specifications: NVIDIA GeForce 970 GPU; AMD Ryzen 5 3600 3.6GHz CPU; and 16GB of RAM. All the 2D fruit yield estimation results presented below were obtained using the chosen, best performing, YOLOv4 model, "YOLOv4P5 WH".

5.2.1 Quantitative Outcomes

The results were obtained by passing two videos, each one representing one side of the validation tree, through the FastMOT framework. The output of this inference is the total number of counted fruits and the input video with the associated tracks. To obtain the quantitative results, fruits of both videos were manually counted. Let one video representing one side of a tree be nominated "v1" and the other video "v2". Although it is hard to manually count the number of fruits in a video with illumination issues, shadows, and not great resolution, 155 fruits were counted in video v1 and 123 fruits were counted in video v2. The implemented system was able to automatically count 161 fruits in video v1, and 120 fruits in video v2. This represents an accuracy of 96% for video v1 and 98% for video v2. This values can be considered a near-perfect fruit yield estimation if considering that there is an error associated to manually counting the ground truth number of fruits in the validation tree videos.

5.2.2 Qualitative Outcomes

The quantitative results were close to perfect, but the question is, is the system this accurate, or is it counting fruits it isn't supposed to, or counting things that are not fruits. When there isn't a big validation dataset, qualitative outcomes become of major importance to analyze. Analyzing these outcomes gives a perspective on how accurately the system is keeping track of fruits along all their appearance and how often they might be overcounted. They validate how viable the quantitative outcomes are. Figure 5.6 a) and b) give a general overview of the tracking system output. It represents a sequence of frames for various timestamps of the validation tree v1 and v2.

For the qualitative outcomes, two main aspects will be analyzed. These are: how well the system can keep track of the same fruit from the moment it gets into scene until it gets out; and how well the system handles fruit occlusion. To answer both questions, the output videos of the validation tree will be analyzed.

It is important for a fruit to always keep the same ID from the very beginning, when it enters the scene, until the very end, which is when it leaves the scene or becomes totally occluded. This assures that a fruit isn't counted more than once along its appearance

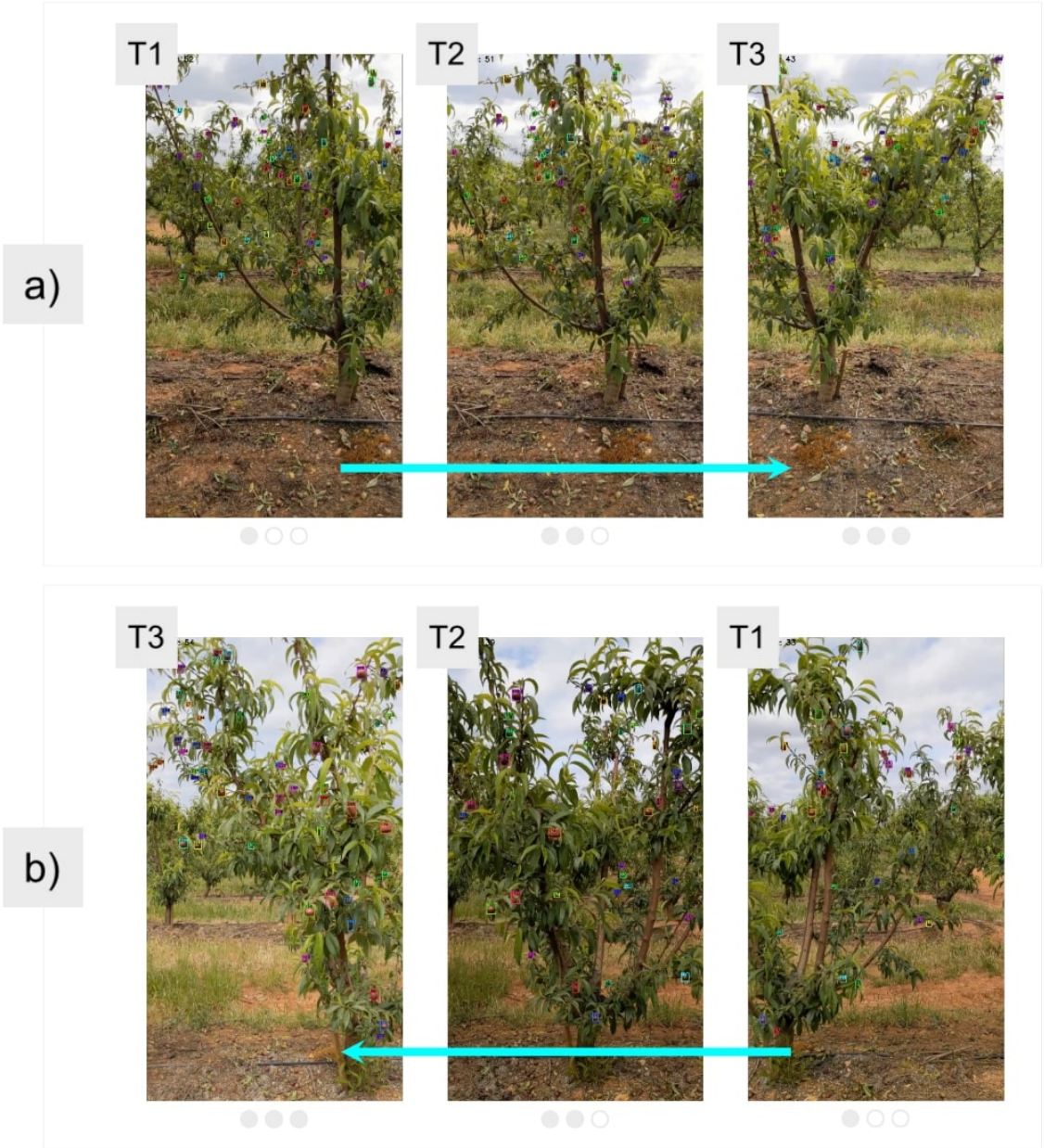

Figure 5.6: Sample of frames extracted from the output videos of the tracking system. a) frames were extracted from output video v2, which was acquired with a left-to-right motion, and b) frames were extracted from output video v1, which was acquired with a right-to-left motion.

in multiple frames. Figure 5.7 represents a row sampled at various timestamps of the tracked tree side v1. We'll focus on analyzing the tracks with IDs 61 and 39, since these run through the image in its full length. The system initialized its tracks at timestamp T1. On timestamp T2, the system loses track of ID 39 since it becomes partially occluded by leaves. Thanks to the ID re-identification system, after being partially occluded, the tracked fruit receives the same ID 39 on timestamp T3, avoiding fruit overcount. Similarly, the system loses track of fruit with ID 61 on timestamp T4 but is able to re-identify it with the same ID in timestamp T5. From this timestamp, where the fruits are located in the middle of the scene, until the very end, they always keep the same ID. On top of this, the remaining visible fruits in this sampled row with IDs 44, 7, 85, 124, 155, 203, 244, 245, 261, and 272 always keep their IDs throughout the whole video sequence, totally avoiding the overcount of fruits. After this analysis, it is viable to say that the developed system is robust against ID switches in a scenario where a fruit is present along all width of a scene.

It was previously seen that the system can handle partially occluded fruits. Next, it is important to analyze how it handles full fruit occlusion scenarios. When hundreds of fruits are being considered, it is easy to find one or another that hides itself behind tree leaves or chunks. To analyze this, fruit with track ID 137 of video v1 was chosen, as it became fully occluded twice during its tracking. Figure 5.8 illustrates its trajectory along both occlusions. The fruit becomes occluded in timestamp T2, which consequently conducts the system to lose track of it, and reappears in timestamp T3, where the system is able to re-identify it with the same ID. Next a longer occlusion occurs, in timestamp T4. As it can be seen, the system is able to re-identify it once again in timestamp T5. With the analysis of fruit ID 137 it is possible to conclude that the system is robust to full occlusions, avoiding fruit overcount.

5.3 Discussion of Results

In the orchard in study, "Herdade Corte Romeira", the estimation of the fruit yield is conducted manually and statistically. A human counts the number of fruits in one or a couple of trees, calculates the average number of fruits per tree, and then multiples it by the total number of trees in the orchard for a given fruit. This system is not only tedious, labour intensive and time consuming, but also obviously prone to error when considering that there are thousands of trees, as it is the case of the farm in study, where there are 22 hectares of nectarines, which reflects in a huge number of nectarine trees. The developed system is able to count fruits in tree row sides with data acquired with an ordinary smartphone, achieving accuracies up to 98% while being able to respect the time constraints of each use case.

Six DL detectors were trained, all with different settings. Of all the trained models, the model that performed the best was YOLOv4P5 with its width and height adjusted so that it would best match the input images width and height. This model obtained

Figure 5.7: Keeping track of a fruit when it enters the scene until it gets out. Row from tree side v1 sampled at various timestamps T1, T2, ..., T8.

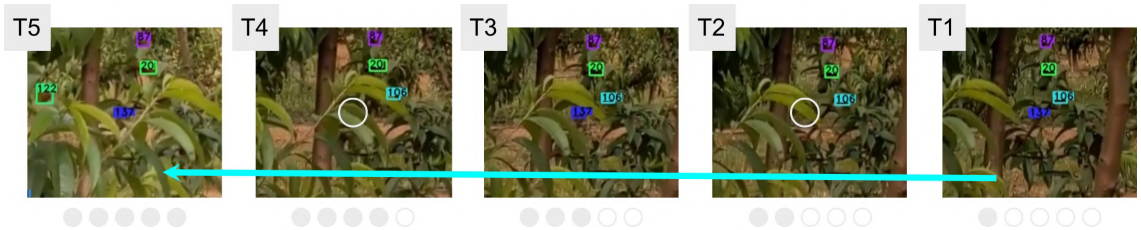

Figure 5.8: Fruit with track ID 137 suffering two full occlusions.

an average mAP and fruit count accuracy of 94% and 91%, respectively. This results could've been negatively influenced by the difficulty in the dataset labelling process, as there are occasions where it is hard, even when manually human labelling, to distinguish leaves and chunk pieces from actual fruits. Although the trained YOLOv4P6, a larger model than YOLOv4P5, was expected to obtain better results than YOLOv4P5, it didn't actually happen. The most feasible justification for this is that the YOLOv4P6 model couldn't be trained with a width and height close to the input images, due to lack of GPU memory. This could've affected the training process and, consequently, the mAP and fruit count accuracy results. When comparing to literature state-of-the-art results [85, 93, 94], where detector performance is analyzed using static images, with better quality and closer to the fruits, the obtained results are really satisfactory, considering the small training dataset and the fact that it was extracted from a video taken with an ordinary smartphone. One limitation while training the object detector really was the dataset size. The seasonality of the fruits, together with the time scope of this dissertation, prevented that more photos were taken during the development of this work as a means to enlarge the dataset and thus improve results. That being said, a possible improvement for future work is to increase the dataset size, which will most likely improve the overall results of all the work.

The chosen YOLO model was then used to count fruits in both sides of a tree. As a means to avoid fruit overcounting, a tracking technique, namely Deep SORT, was implemented. A framework named FastMOT was used to implement Deep SORT. As this framework requires an Ubuntu distribution, WSL was installed in a windows machine and, after some challenging setup, FastMOT was fully configured and ready to track on our custom dataset, "gardeta" nectarines. In order to validate the system quality, fruits were manually counted in both validation videos v1 and v2, obtaining a ground truth count of 155 and 123 fruits, respectively. Then, to obtain the number of fruits counted by the developed system, a piece of code on FastMOT files was edited so that it could print in the console the unique number of IDs counted when performing the tracking, which would represent the total number of fruits presented in one side of a tree. After passing both videos v1 and v2 through the FastMOT framework, it estimated that there was 161 fruits in video v1 and 120 in video v2. When comparing with the ground truth data, this translates to an accuracy of 96% for video v1 and 98% for video v2. These

are extremely satisfactory results, when considering that this system is able to output results given specific time constraints and that the validation dataset was acquired with a smartphone under uncontrolled lightning and climate conditions. The authors of [95] got an accuracy of 98% when counting mango's in tree row sides under controlled and perfect light conditions (using a 720W LED floodlight with a dedicated camera, at night-time), similar to figure 2.5. The developed system achieved the same results under uncontrolled climate and lightning conditions, without a dedicated camera and with a smaller fruit. This is an indicator of how promising a prototype using the developed system can be to automatically estimate the production yield of a full orchard, which is possible due to the intrinsic speed characteristics of the developed system. Similarly to before, better results would likely be achieved if a bigger dataset was available for training. It is important to consider that some fruits might've been missed or overcounted while counting the ground truth number of fruits in both videos, as manually counting fruits in fast and low quality videos is a hard assignment. This could potentially indicate that the system is being over or under rated. Unfortunately, the lack of a real life ground truth fruit count of both sides of the validation tree is not available to confirm the obtained results. Besides fruit count of both sides, it would be useful to have real life ground truth data of the whole validation tree, as a means to assess how well the system would've estimated the production yield of the whole tree without using a correction phase, that is, just by summing the fruit counts from both tree row sides. Having this said, in future work, it is suggested to gather a new validation dataset of the nectarines once they are again available pre-harvest, and count the number of fruits on each side of the tree, as well as the total number of fruits of the tree.

After going through a variety of challenges, both conceptual and technical, the initially proposed system was implemented with success. Overall, the system behaved as expected, and, to the best of our knowledge, with state-of-the-art fruit yield estimation results regarding tree row sides. The characteristics of the developed and proposed system allows for it to be escalated both horizontal and/or vertically, thus being able to obtain speed results that fit into the required time constraints of each use case. The obtained results opens doors for future research and development of the remaining proposed framework, which would allow to obtain the total fruit count of a tree, and, consequently, the total estimation of the production yield of the whole orchard.

Chapter 6

Conclusion and Follow Up

6.1 Conclusion

The estimation of the production yield is an essential and important task that is performed prior to fruit harvesting. This information gives farmers the possibility to better organize their storage system, so that ideally every fruit can be properly stored. On top of this, knowing how many fruits need to be harvested helps farmers better plan their marketing activates, such as matching supply and demand. This dissertation was an entry point to achieve a fast and reliable system capable of estimating the fruit yield of a whole orchard. The implemented system is capable of estimating the fruit yield of tree row sides with accuracies up to 98% while allowing for speeds that are able to fit into the time constraints of each use case.

The work done is divided into three parts. First a visit was conducted to "Herdade Corte Romeira", who kindly guided us through their orchard. This visit allowed to answer RQ1, where a dataset on nectarine trees was taken with a smartphone (Google Pixel 4A) carried by a human. Unfortunately a tractor wasn't available at the time of the visit, thus not being possible to take the data like it was previously proposed. The acquired dataset consists of images from both sides of a row composed of eight trees. Of all the eight trees, seven were used to train an object detector, and one was used to validate both detector and tracker results. Using the seven trees, models of the Scaled YOLOv4 family were trained, from which Scaled YOLOv4P5 performed the best, with an average mAP of 94% and average fruit count accuracy of 91%, thus answering RQ2. After validating the accuracy of Scaled YOLOv4P5, it was integrated with FastMOT, a framework that implements Deep SORT with some improvements, which was the answer to RQ3. With right parameter tuning, it was possible to achieve a fruit count accuracy of 96% on one side of the validation tree, and 98% on its opposite side. The counts from both sides of this tree could be summed in order to compare the results between this obtained merged count, and the total ground truth number of fruits carried by this validation tree, as a means to assess RQ4. Counting the total number of fruits carried by the validation tree from a video isn't accurate, and, as such, future work should include this validation.

To the best of our knowledge, the developed system presented state-of-the-art results regarding tree row side fruit count.

On top of the implemented solution, a system to accurately estimate the fruit yield of the whole orchard was proposed. In this system, two smartphones are attached to a tractor, and are continuously acquiring and sending data to a server. The server is processing this data and contributing with the fruit counts from tree rows to perform an accurate estimation of the production yield of the whole orchard. With this proposed system, instead of limiting the estimation to a couple of tree rows, it is possible to perform it using data of larger areas, thus increasing the overall accuracy of the fruit yield estimation of the whole orchard. For this system to work, it is crucial to assess the viability of counting fruits from a sequence of frames acquired with a smartphone. Thanks to this dissertation contribution, this was validated with state-of-the-art results, which should motivate the continuity of the development of the proposed system, allowing the estimation of the fruit yield of a whole tree and, consequently, orchard.

6.2 Follow Up

Following this dissertation, future work should include increasing the size of the dataset, so that there are both more data that enables better object detector training results and more data to better validate the developed system. While acquiring this new data, the fruits carried by the filmed trees should also be counted, as a means to validate how well the already developed system would perform in estimating the fruit yield of a tree row by simply summing both counts from both sides of that tree row, that is, without a correction phase.

As a means to determine whether the developed system is able to assist in chemical thinning control, the best performing object detector should be re-trained under a dataset consisting of unripened fruits, that is, smaller and greener fruits than the ones considered in this dissertation.

Following this, and giving the promising results achieved, future efforts should focus as well on the implementation of the remaining parts of the proposed system, which would be capable of estimating the production yield of an whole orchard.

Bibliography

[1] *Digital Farming: what does it really mean?* Tech. rep. T +32 (0)2 706 81 73 - F +32 (0)2 706 82 10. Diamant Building, Bld A. Reyers 80, BE-1030 Brussels: European Agricultural Machinery, Feb. 2017 (cit. on p. 1).

[2] *Global agriculture towards 2050*. Tech. rep. (+39) 06 57053354. Viale delle Terme di Caracalla, 00153 Rome, Italy: Food and Agriculture Organization, Oct. 2009 (cit. on p. 1).

[3] M. D. Clercq, A. Vats, and A. Biel. *Agriculture 4.0: The Future of Farming Technology*. Tech. rep. World Government Summit, Feb. 2018 (cit. on p. 1).

[4] Y. Liu et al. "From Industry 4.0 to Agriculture 4.0: Current Status, Enabling Technologies, and Research Challenges". In: *IEEE Transactions on Industrial Informatics* (2020), pp. 1–1. DOI: 10.1109/tii.2020.3003910. URL: https://doi.org/10.1109/tii.2020.3003910 (cit. on pp. 1, 2, 7, 8).

[5] I. Kovács and I. Husti. "The role of digitalization in the agricultural 4.0 – how to connect the industry 4.0 to agriculture?" In: *Hungarian Agricultural Engineering* 33 (2018), pp. 38–42. DOI: 10.17676/hae.2018.33.38. URL: https://doi.org/10.17676/hae.2018.33.38 (cit. on p. 1).

[6] S. O. Araújo et al. "Characterising the Agriculture 4.0 Landscape—Emerging Trends, Challenges and Opportunities". In: 11.4 (Apr. 2021), p. 667. DOI: 10.3390/agronomy11040667. URL: https://doi.org/10.3390/agronomy11040667 (cit. on pp. 1, 2, 7).

[7] I. Zambon et al. "Revolution 4.0: Industry vs. Agriculture in a Future Development for SMEs". In: *Processes* 7.1 (Jan. 2019), p. 36. DOI: 10.3390/pr7010036. URL: https://doi.org/10.3390/pr7010036 (cit. on pp. 1, 7).

[8] J. Kim et al. "Unmanned Aerial Vehicles in Agriculture: A Review of Perspective of Platform, Control, and Applications". In: *IEEE Access* 7 (2019), pp. 105100–105115. DOI: 10.1109/access.2019.2932119. URL: https://doi.org/10.1109/access.2019.2932119 (cit. on pp. 2, 10).

[9] P. Radoglou-Grammatikis et al. "A compilation of UAV applications for precision agriculture". In: *Computer Networks* 172 (May 2020), p. 107148. DOI: 10.1016/j.comnet.2020.107148. URL: https://doi.org/10.1016/j.comnet.2020.107148 (cit. on pp. 2, 10).

[10] A. Kamilaris and F. X. Prenafeta-Boldú. "Deep learning in agriculture: A survey". In: *Computers and Electronics in Agriculture* 147 (Apr. 2018), pp. 70–90. DOI: 10.1016/j.compag.2018.02.016. URL: https://doi.org/10.1016/j.compag.2018.02.016 (cit. on pp. 2, 11).

[11] F. Agriculture, D. E. D. (with support from the FAO Netherlands Partnership Programme (FNPP), and the EC-FAO Food Security Programme. *Food Security*. Tech. rep. World Government Summit, June 2006 (cit. on p. 2).

[12] C. Hung et al. "Autonomous intelligent system for fruit yield estimation". In: *Acta Horticulturae* 1130 (Dec. 2016), pp. 545–550. DOI: 10.17660/actahortic.2016.1130.82. URL: https://doi.org/10.17660/actahortic.2016.1130.82 (cit. on pp. 2, 31).

[13] J. Gené-Mola et al. "Fruit detection and 3D location using instance segmentation neural networks and structure-from-motion photogrammetry". In: *Computers and Electronics in Agriculture* 169 (Feb. 2020), p. 105165. DOI: 10.1016/j.compag.2019.105165. URL: https://doi.org/10.1016/j.compag.2019.105165 (cit. on pp. 2, 28, 30, 31).

[14] A. Koirala et al. "Deep learning – Method overview and review of use for fruit detection and yield estimation". In: *Computers and Electronics in Agriculture* 162 (July 2019), pp. 219–234. DOI: 10.1016/j.compag.2019.04.017. URL: https://doi.org/10.1016/j.compag.2019.04.017 (cit. on pp. 3, 26–28, 30).

[15] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2020. arXiv: 2011.08036 [cs.CV] (cit. on pp. 5, 16, 18, 19, 37).

[16] Y. Yang. *FastMOT: High-Performance Multiple Object Tracking Based on Deep SORT and KLT*. Version v1.0.0. Nov. 2020. DOI: 10.5281/zenodo.4294717. URL: https://doi.org/10.5281/zenodo.4294717 (cit. on pp. 5, 44, 51).

[17] L. T. Hickey et al. "Breeding crops to feed 10 billion". In: *Nature Biotechnology* 37.7 (June 2019), pp. 744–754. DOI: 10.1038/s41587-019-0152-9. URL: https://doi.org/10.1038/s41587-019-0152-9 (cit. on p. 7).

[18] M. C. Hunter et al. "Agriculture in 2050: Recalibrating Targets for Sustainable Intensification". In: *BioScience* 67.4 (Feb. 2017), pp. 386–391. DOI: 10.1093/biosci/bix010. URL: https://doi.org/10.1093/biosci/bix010 (cit. on p. 7).

[19] L. Klerkx and D. Rose. "Dealing with the game-changing technologies of Agriculture 4.0: How do we manage diversity and responsibility in food system transition pathways?" In: *Global Food Security* 24 (Mar. 2020), p. 100347. DOI: 10.1016/j.gfs.2019.100347. URL: https://doi.org/10.1016/j.gfs.2019.100347 (cit. on p. 7).

[20] E. D. Lioutas and C. Charatsari. "Smart farming and short food supply chains: Are they compatible?" In: *Land Use Policy* 94 (May 2020), p. 104541. DOI: 10.1016/j.landusepol.2020.104541. URL: https://doi.org/10.1016/j.landusepol.2020.104541 (cit. on p. 7).

[21] M. A. Rapela. *Fostering Innovation for Agriculture 4.0*. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-32493-3. URL: https://doi.org/10.1007/978-3-030-32493-3 (cit. on p. 7).

[22] Y. Yin, K. E. Stecke, and D. Li. "The evolution of production systems from Industry 2.0 through Industry 4.0". In: *International Journal of Production Research* 56.1-2 (Nov. 2017), pp. 848–861. DOI: 10.1080/00207543.2017.1403664. URL: https://doi.org/10.1080/00207543.2017.1403664 (cit. on p. 8).

[23] O. Elijah et al. "An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges". In: *IEEE Internet of Things Journal* 5.5 (Oct. 2018), pp. 3758–3773. DOI: 10.1109/jiot.2018.2844296. URL: https://doi.org/10.1109/jiot.2018.2844296 (cit. on p. 9).

[24] S. Ratnaparkhi et al. "Smart agriculture sensors in IOT: A review". In: (Dec. 2020). DOI: 10.1016/j.matpr.2020.11.138. URL: https://doi.org/10.1016/j.matpr.2020.11.138 (cit. on p. 9).

[25] D. Pimentel, R. Zuniga, and D. Morrison. "Update on the environmental and economic costs associated with alien-invasive species in the United States". In: *Ecological Economics* 52.3 (Feb. 2005), pp. 273–288. DOI: 10.1016/j.ecolecon.2004.10.002. URL: https://doi.org/10.1016/j.ecolecon.2004.10.002 (cit. on p. 9).

[26] O. Debauche et al. "Edge AI-IoT Pivot Irrigation, Plant Diseases, and Pests Identification". In: *Procedia Computer Science* 177 (2020), pp. 40–48. DOI: 10.1016/j.procs.2020.10.009. URL: https://doi.org/10.1016/j.procs.2020.10.009 (cit. on p. 9).

[27] B. S. Faical et al. "Fine-Tuning of UAV Control Rules for Spraying Pesticides on Crop Fields". In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. IEEE, Nov. 2014. DOI: 10.1109/ictai.2014.85. URL: https://doi.org/10.1109/ictai.2014.85 (cit. on p. 10).

[28] A. D. Boursianis et al. "Internet of Things (IoT) and Agricultural Unmanned Aerial Vehicles (UAVs) in smart farming: A comprehensive review". In: *Internet of Things* (Mar. 2020), p. 100187. DOI: 10.1016/j.iot.2020.100187. URL: https://doi.org/10.1016/j.iot.2020.100187 (cit. on p. 10).

[29] M. Gašparović et al. "An automatic method for weed mapping in oat fields based on UAV imagery". In: *Computers and Electronics in Agriculture* 173 (June 2020), p. 105385. DOI: 10.1016/j.compag.2020.105385. URL: https://doi.org/10.1016/j.compag.2020.105385 (cit. on p. 10).

[30] S. K. Pilli et al. "eAGROBOT - A robot for early crop disease detection using image processing". In: *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*. IEEE, Feb. 2015. DOI: 10.1109/ecs.2015.7124873. URL: https://doi.org/10.1109/ecs.2015.7124873 (cit. on pp. 10, 30).

[31] M. Christiansen et al. "Designing and Testing a UAV Mapping System for Agricultural Field Surveying". In: *Sensors* 17.12 (Nov. 2017), p. 2703. DOI: 10.3390/s17122703. URL: https://doi.org/10.3390/s17122703 (cit. on p. 10).

[32] L. Santesteban et al. "High-resolution UAV-based thermal imaging to estimate the instantaneous and seasonal variability of plant water status within a vineyard". In: *Agricultural Water Management* 183 (Mar. 2017), pp. 49–59. DOI: 10.1016/j.agwat.2016.08.026. URL: https://doi.org/10.1016/j.agwat.2016.08.026 (cit. on p. 10).

[33] C. Hung et al. "A Feature Learning Based Approach for Automated Fruit Yield Estimation". In: *Springer Tracts in Advanced Robotics*. Springer International Publishing, 2015, pp. 485–498. DOI: 10.1007/978-3-319-07488-7_33. URL: https://doi.org/10.1007/978-3-319-07488-7_33 (cit. on p. 11).

[34] Z. Al-Mashhadani and B. Chandrasekaran. "Survey of Agricultural Robot Applications and Implementation". In: *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, Nov. 2020. DOI: 10.1109/iemcon51383.2020.9284910. URL: https://doi.org/10.1109/iemcon51383.2020.9284910 (cit. on pp. 11, 30).

[35] W. S. Barbosa et al. "Design and Development of an Autonomous Mobile Robot for Inspection of Soy and Cotton Crops". In: *2019 12th International Conference on Developments in eSystems Engineering (DeSE)*. IEEE, Oct. 2019. DOI: 10.1109/dese.2019.00107. URL: https://doi.org/10.1109/dese.2019.00107 (cit. on p. 11).

[36] M. S. A. Mahmud, M. S. Z. Abidin, and Z. Mohamed. "Development of an autonomous crop inspection mobile robot system". In: *2015 IEEE Student Conference on Research and Development (SCOReD)*. IEEE, Dec. 2015. DOI: 10.1109/scored.2015.7449304. URL: https://doi.org/10.1109/scored.2015.7449304 (cit. on p. 11).

[37] A. I. S. Oliveira et al. "On the Intelligent Control Design of an Agricultural Mobile Robot for Cotton Crop Monitoring". In: *2019 12th International Conference on Developments in eSystems Engineering (DeSE)*. IEEE, Oct. 2019. DOI: 10.1109/dese.2019.00108. URL: https://doi.org/10.1109/dese.2019.00108 (cit. on p. 11).

[38] C.-L. Chung et al. "Detecting Bakanae disease in rice seedlings by machine vision". In: *Computers and Electronics in Agriculture* 121 (Feb. 2016), pp. 404–411. DOI: 10.1016/j.compag.2016.01.008. URL: https://doi.org/10.1016/j.compag.2016.01.008 (cit. on p. 11).

[39] M. Jhuria, A. Kumar, and R. Borse. "Image processing for smart farming: Detection of disease and fruit grading". In: *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*. IEEE, Dec. 2013. DOI: 10.1109/iciip.2013.6707647. URL: https://doi.org/10.1109/iciip.2013.6707647 (cit. on p. 11).

[40] J. F. S. Gomes and F. R. Leta. "Applications of computer vision techniques in the agriculture and food industry: a review". In: *European Food Research and Technology* 235.6 (Oct. 2012), pp. 989–1000. DOI: 10.1007/s00217-012-1844-2. URL: https://doi.org/10.1007/s00217-012-1844-2 (cit. on p. 11).

[41] L. Fu et al. "Application of consumer RGB-D cameras for fruit detection and localization in field: A critical review". In: *Computers and Electronics in Agriculture* 177 (Oct. 2020), p. 105687. DOI: 10.1016/j.compag.2020.105687. URL: https://doi.org/10.1016/j.compag.2020.105687 (cit. on pp. 11, 26, 28, 30).

[42] E. C. Tetila et al. "Detection and classification of soybean pests using deep learning with UAV images". In: *Computers and Electronics in Agriculture* 179 (Dec. 2020), p. 105836. DOI: 10.1016/j.compag.2020.105836. URL: https://doi.org/10.1016/j.compag.2020.105836 (cit. on p. 11).

[43] E. Ayan, H. Erbay, and F. Varçın. "Crop pest classification with a genetic algorithm-based weighted ensemble of deep convolutional neural networks". In: *Computers and Electronics in Agriculture* 179 (Dec. 2020), p. 105809. DOI: 10.1016/j.compag.2020.105809. URL: https://doi.org/10.1016/j.compag.2020.105809 (cit. on p. 11).

[44] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Pearson Education, 2010 (cit. on p. 12).

[45] X. Jiang et al. *Deep Learning in Object Detection and Recognition*. Springer (cit. on p. 12).

[46] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016 (cit. on p. 12).

[47] A. Burkov. *The hundred-page machine learning book*. English. 2019. ISBN: 978-1-9995795-0-0 (cit. on pp. 12, 14).

[48] M. R. M. Talabis et al. "Analytics Defined". In: *Information Security Analytics*. Elsevier, 2015, pp. 1–12. DOI: 10.1016/b978-0-12-800207-0.00001-0. URL: https://doi.org/10.1016/b978-0-12-800207-0.00001-0 (cit. on p. 12).

[49] P. Wittek. "Unsupervised Learning". In: *Quantum Machine Learning*. Elsevier, 2014, pp. 57–62. DOI: 10.1016/b978-0-12-800953-6.00005-0. URL: https://doi.org/10.1016/b978-0-12-800953-6.00005-0 (cit. on p. 13).

[50] M. Sugiyama. "Semisupervised Learning". In: *Introduction to Statistical Machine Learning*. Elsevier, 2016, pp. 375–390. DOI: 10.1016/b978-0-12-802121-7.00044-3. URL: https://doi.org/10.1016/b978-0-12-802121-7.00044-3 (cit. on p. 13).

[51] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, 2012. ISBN: 026201825X, 978-0262018258 (cit. on p. 13).

[52] A. G. Barto. "Reinforcement Learning". In: *Neural Systems for Control*. Elsevier, 1997, pp. 7–30. DOI: 10.1016/b978-012526430-3/50003-9. URL: https://doi.org/10.1016/b978-012526430-3/50003-9 (cit. on p. 13).

[53] S. Borra and A. D. Ciaccio. "Measuring the prediction error. A comparison of cross-validation, bootstrap and covariance penalty methods". In: *Computational Statistics & Data Analysis* 54.12 (Dec. 2010), pp. 2976–2989. DOI: 10.1016/j.csda.2010.03.004. URL: https://doi.org/10.1016/j.csda.2010.03.004 (cit. on p. 14).

[54] R. Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In: 14 (Mar. 2001) (cit. on p. 14).

[55] I. H. Witten et al. "Deep learning". In: *Data Mining*. Elsevier, 2017, pp. 417–466. DOI: 10.1016/b978-0-12-804291-5.00010-6. URL: https://doi.org/10.1016/b978-0-12-804291-5.00010-6 (cit. on pp. 14, 15).

[56] A. Apicella et al. "A survey on modern trainable activation functions". In: *Neural Networks* (Feb. 2021). DOI: 10.1016/j.neunet.2021.01.026. URL: https://doi.org/10.1016/j.neunet.2021.01.026 (cit. on p. 15).

[57] A. B. Nassif et al. "Speech Recognition Using Deep Neural Networks: A Systematic Review". In: *IEEE Access* 7 (2019), pp. 19143–19165. DOI: 10.1109/access.2019.2896880. URL: https://doi.org/10.1109/access.2019.2896880 (cit. on p. 16).

[58] A. A. M. Al-Saffar, H. Tao, and M. A. Talab. "Review of deep convolution neural network in image classification". In: *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*. IEEE, Oct. 2017. DOI: 10.1109/icramet.2017.8253139. URL: https://doi.org/10.1109/icramet.2017.8253139 (cit. on p. 16).

[59] S. Sengupta et al. "A review of deep learning with special emphasis on architectures, applications and recent trends". In: *Knowledge-Based Systems* 194 (Apr. 2020), p. 105596. DOI: 10.1016/j.knosys.2020.105596. URL: https://doi.org/10.1016/j.knosys.2020.105596 (cit. on p. 16).

[60] M. Alam et al. *Survey on Deep Neural Networks in Speech and Vision Systems*. 2019. arXiv: 1908.07656 [cs.CV] (cit. on p. 16).

[61] I. Goodfellow, Y. Bengio, and A. Courville. "Deep Learning". In: http://www.deeplearningbook.org. MIT Press, 2016. Chap. 9, pp. 331–334 (cit. on p. 16).

[62] J. Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV] (cit. on pp. 16, 18).

[63] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV] (cit. on pp. 17, 38).

[64] J. Redmon and A. Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV] (cit. on p. 17).

[65] J. Redmon and A. Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV] (cit. on p. 17).

[66] Y. Zhang, Z. Chen, and B. Wei. "A Sport Athlete Object Tracking Based on Deep Sort and Yolo V4 in Case of Camera Movement". In: *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*. IEEE, Dec. 2020. DOI: 10.1109/iccc51575.2020.9345010. URL: https://doi.org/10.1109/iccc51575.2020.9345010 (cit. on p. 19).

[67] D. Kim et al. "Real-Time Multiple Pedestrian Tracking Based on Object Identification". In: *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*. IEEE, Sept. 2019. DOI: 10.1109/icce-berlin47944.2019.8966205. URL: https://doi.org/10.1109/icce-berlin47944.2019.8966205 (cit. on p. 19).

[68] A. Ellouze et al. "Multiple Object Tracking: Case of Aircraft Detection and Tracking". In: *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE, Mar. 2019. DOI: 10.1109/ssd.2019.8893202. URL: https://doi.org/10.1109/ssd.2019.8893202 (cit. on p. 19).

[69] V. Mandal and Y. Adu-Gyamfi. *Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis*. 2020. arXiv: 2007.16198 [cs.CV] (cit. on p. 19).

[70] Y. Zhang et al. "Recent advances of single-object tracking methods: A brief survey". In: *Neurocomputing* 455 (Sept. 2021), pp. 1–11. DOI: 10.1016/j.neucom.2021.05.011. URL: https://doi.org/10.1016/j.neucom.2021.05.011 (cit. on p. 21).

[71] W. Luo et al. "Multiple object tracking: A literature review". In: *Artificial Intelligence* 293 (Apr. 2021), p. 103448. DOI: 10.1016/j.artint.2020.103448. URL: https://doi.org/10.1016/j.artint.2020.103448 (cit. on pp. 21, 22).

[72] Y. Xu et al. "Deep learning for multiple object tracking: a survey". In: *IET Computer Vision* 13.4 (May 2019), pp. 355–368. DOI: 10.1049/iet-cvi.2018.5598. URL: https://doi.org/10.1049/iet-cvi.2018.5598 (cit. on p. 21).

[73] L. Leal-Taixé, C. C. Ferrer, and K. Schindler. *Learning by tracking: Siamese CNN for robust target association*. 2016. arXiv: 1604.07866 [cs.LG] (cit. on p. 22).

[74] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45 (cit. on p. 22).

[75] Q. Li et al. "Kalman Filter and Its Application". In: *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. IEEE, Nov. 2015. DOI: 10.1109/icinis.2015.35. URL: https://doi.org/10.1109/icinis.2015.35 (cit. on p. 22).

[76] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 22).

[77] A. Bewley et al. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)* (Sept. 2016). DOI: 10.1109/icip.2016.7533003. URL: http://dx.doi.org/10.1109/ICIP.2016.7533003 (cit. on p. 23).

[78] S. Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (cit. on p. 23).

[79] H. Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval Research Logistic Quarterly* 2 (May 2012) (cit. on p. 23).

[80] N. Wojke, A. Bewley, and D. Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*. 2017. arXiv: 1703.07402 [cs.CV] (cit. on p. 24).

[81] A. R. Jiménez, R. Ceres, and J. L. Pons. "A SURVEY OF COMPUTER VISION METHODS FOR LOCATING FRUIT ON TREES". In: *Transactions of the ASAE* 43.6 (2000), pp. 1911–1920. DOI: 10.13031/2013.3096. URL: https://doi.org/10.13031/2013.3096 (cit. on p. 25).

[82] J. Ma et al. "Estimating above ground biomass of winter wheat at early growth stages using digital images and deep convolutional neural network". In: *European Journal of Agronomy* 103 (Feb. 2019), pp. 117–129. DOI: 10.1016/j.eja.2018.12.004. URL: https://doi.org/10.1016/j.eja.2018.12.004 (cit. on pp. 25, 30).

[83] O. Apolo-Apolo et al. "Deep learning techniques for estimation of the yield and size of citrus fruits using a UAV". In: *European Journal of Agronomy* 115 (Apr. 2020), p. 126030. DOI: 10.1016/j.eja.2020.126030. URL: https://doi.org/10.1016/j.eja.2020.126030 (cit. on pp. 25, 26, 30).

[84] Q. Wang et al. "Automated Crop Yield Estimation for Apple Orchards". In: *Experimental Robotics*. Springer International Publishing, 2013, pp. 745–758. DOI: 10.1007/978-3-319-00065-7_50. URL: https://doi.org/10.1007/978-3-319-00065-7_50 (cit. on pp. 25, 30).

[85] A. Koirala et al. "Deep learning for real-time fruit detection and orchard fruit load estimation: benchmarking of 'MangoYOLO'". In: *Precision Agriculture* 20.6 (Feb. 2019), pp. 1107–1135. DOI: 10.1007/s11119-019-09642-0. URL: https://doi.org/10.1007/s11119-019-09642-0 (cit. on pp. 25, 26, 31, 74).

[86] I. Sa et al. "DeepFruits: A Fruit Detection System Using Deep Neural Networks". In: *Sensors* 16.8 (Aug. 2016), p. 1222. DOI: 10.3390/s16081222. URL: https://doi.org/10.3390/s16081222 (cit. on pp. 26, 27, 30).

[87] J. Naranjo-Torres et al. "A Review of Convolutional Neural Network Applied to Fruit Image Processing". In: *Applied Sciences* 10.10 (May 2020), p. 3443. DOI: 10.3390/app10103443. URL: https://doi.org/10.3390/app10103443 (cit. on pp. 27, 30).

[88] K. Kusrini et al. "Data augmentation for automated pest classification in Mango farms". In: *Computers and Electronics in Agriculture* 179 (Dec. 2020), p. 105842. DOI: 10.1016/j.compag.2020.105842. URL: https://doi.org/10.1016/j.compag.2020.105842 (cit. on pp. 27, 30).

[89] A. Das et al. "Image splicing detection using Gaussian or defocus blur". In: *2016 International Conference on Communication and Signal Processing (ICCSP)*. 2016, pp. 1237–1241. DOI: 10.1109/ICCSP.2016.7754350 (cit. on p. 27).

[90] X. Liu et al. *Robust Fruit Counting: Combining Deep Learning, Tracking, and Structure from Motion*. 2018. arXiv: 1804.00307 [cs.CV] (cit. on pp. 28, 31).

[91] M. Stein, S. Bargoti, and J. Underwood. "Image Based Mango Fruit Detection, Localisation and Yield Estimation Using Multiple View Geometry". In: *Sensors* 16.11 (Nov. 2016), p. 1915. DOI: 10.3390/s16111915. URL: https://doi.org/10.3390/s16111915 (cit. on pp. 28, 31).

[92] A. Gongal et al. "Apple crop-load estimation with over-the-row machine vision system". In: *Computers and Electronics in Agriculture* 120 (Jan. 2016), pp. 26–35. DOI: 10.1016/j.compag.2015.10.022. URL: https://doi.org/10.1016/j.compag.2015.10.022 (cit. on pp. 28, 31).

[93] Z. Malik et al. "Detection and Counting of On-Tree Citrus Fruit for Crop Yield Estimation". In: *International Journal of Advanced Computer Science and Applications* 7.5 (2016). DOI: 10.14569/ijacsa.2016.070569. URL: https://doi.org/10.14569/ijacsa.2016.070569 (cit. on pp. 31, 74).

[94] N. N. Bhookya, R. Malmathanraj, and P. Palanisamy. "Yield Estimation of Chilli Crop using Image Processing Techniques". In: *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE, Mar. 2020. DOI: 10.1109/icaccs48705.2020.9074257. URL: https://doi.org/10.1109/icaccs48705.2020.9074257 (cit. on pp. 31, 74).

[95] Wang, Walsh, and Koirala. "Mango Fruit Load Estimation Using a Video Based MangoYOLO—Kalman Filter—Hungarian Algorithm Method". In: *Sensors* 19.12 (June 2019), p. 2742. DOI: 10.3390/s19122742. URL: https://doi.org/10.3390/s19122742 (cit. on pp. 31, 75).

[96] I. Radosavovic et al. *Designing Network Design Spaces*. 2020. arXiv: 2003.13678 [cs.CV] (cit. on p. 38).

[97] Y. Tang et al. "A survey on the 5G network and its impact on agriculture: Challenges and opportunities". In: *Computers and Electronics in Agriculture* 180 (Jan. 2021), p. 105895. DOI: 10.1016/j.compag.2020.105895. URL: https://doi.org/10.1016/j.compag.2020.105895 (cit. on p. 41).

Appendix A

FastMOT Configuration Code

Listing A.1: FastMOT configuration code

```
{
    "resize_to": [1024, 1856],

    "video_io": {
        "resolution": [1080, 1920],
        "frame_rate": 30,
        "buffer_size": 10
    },

    "mot": {
        "detector_type": "YOLO",
        "detector_frame_skip": 1,

        "ssd_detector": {
            "model": "SSDInceptionV2",
            "class_ids": [1],
            "tile_overlap": 0.25,
            "tiling_grid": [4, 2],
            "conf_thresh": 0.5,
            "max_area": 130000,
            "merge_thresh": 0.6
        },
        "yolo_detector": {
            "model": "YOLOv4P5",
            "class_ids": [0],
            "conf_thresh": 0.3,
```

```
        "max_area": 800000,
        "nms_thresh": 0.5
    },
    "public_detector": {
        "sequence": "eval/data/MOT20-03",
        "conf_thresh": 0.5,
        "max_area": 800000
    },
    "feature_extractor": {
        "model": "OSNet025",
        "batch_size": 16
    },

    "multi_tracker": {
        "max_age": 30,
        "age_penalty": 1,
        "age_weight": 0.1,
        "motion_weight": 0.02,
        "max_feat_cost": 0.9,
        "max_reid_cost": 0.6,
        "iou_thresh": 0.4,
        "duplicate_iou": 0.8,
        "conf_thresh": 0.3,
        "lost_buf_size": 50,

        "kalman_filter": {
            "std_factor_acc": 13.5,
            "std_offset_acc": 471,
            "std_factor_det": [0.08, 0.08],
            "std_factor_flow": [0.14, 0.14],
            "min_std_det": [4.0, 4.0],
            "min_std_flow": [5.0, 5.0],
            "init_pos_weight": 30,
            "init_vel_weight": 72,
            "vel_coupling": 0.6,
            "vel_half_life": 12
        },

        "flow": {
            "bg_feat_scale_factor": [0.1, 0.1],
```

```
                "opt_flow_scale_factor": [0.5, 0.5],
                "feature_density": 0.005,
                "feat_dist_factor": 0.06,
                "ransac_max_iter": 500,
                "ransac_conf": 0.99,
                "max_error": 100,
                "inlier_thresh": 4,
                "bg_feat_thresh": 10,
                "target_feat_params": {
                    "maxCorners": 1000,
                    "qualityLevel": 0.06,
                    "blockSize": 3
                },
                "opt_flow_params": {
                    "winSize": [5, 5],
                    "maxLevel": 5,
                    "criteria": [3, 10, 0.03]
                }
            }
        }
    }
}
```

Appendix B

Appendix 1 Object Detector Results

Table B.1: YOLOv4 DS accuracy and mAP@0.3.

Frame No.	Manually Counted	Automatically Counted	Accuracy	mAP@0.3
1	29	27	93%	91%
2	37	34	92%	97%
3	41	38	93%	95%
4	58	58	100%	95%
5	47	43	91%	97%
6	52	40	77%	84%
7	47	39	83%	88%
8	45	47	96%	94%
9	56	45	80%	94%
10	43	35	81%	96%
11	38	37	97%	96%
12	38	43	87%	90%
13	54	48	89%	98%
14	56	51	91%	91%
15	57	54	95%	95%
16	48	45	94%	88%
17	45	53	82%	93%
18	36	34	94%	91%

Table B.2: YOLOv4P5 DS accuracy and mAP@0.3.

Frame No.	Manually Counted	Automatically Counted	Accuracy	mAP@0.3
1	29	22	76%	95%
2	37	29	78%	93%
3	41	34	83%	96%
4	58	41	71%	92%
5	47	34	72%	92%
6	52	37	71%	79%
7	47	34	72%	91%
8	45	46	98%	93%
9	56	44	79%	96%
10	43	36	84%	94%
11	38	31	82%	95%
12	38	29	76%	92%
13	54	44	81%	97%
14	56	44	79%	92%
15	57	43	75%	95%
16	48	27	56%	89%
17	45	35	78%	92%
18	36	25	69%	91%

Table B.3: YOLOv4P5 WH accuracy and mAP@0.3.

Frame No.	Manually Counted	Automatically Counted	Accuracy	mAP@0.3
1	29	28	97%	96%
2	37	32	86%	98%
3	41	39	95%	91%
4	58	50	86%	95%
5	47	44	94%	94%
6	52	42	81%	87%
7	47	44	94%	88%
8	45	42	93%	92%
9	56	53	95%	97%
10	43	36	84%	96%
11	38	36	95%	94%
12	38	37	97%	95%
13	54	53	98%	97%
14	56	51	91%	91%
15	57	52	91%	96%
16	48	39	81%	89%
17	45	42	93%	98%
18	36	28	78%	90%

Table B.4: YOLOv4P5 WHR accuracy and mAP@0.3.

Frame No.	Manually Counted	Automatically Counted	Accuracy	mAP@0.3
1	29	27	93%	89%
2	37	32	86%	96%
3	41	35	85%	95%
4	58	51	88%	94%
5	47	43	91%	96%
6	52	45	87%	82%
7	47	41	87%	89%
8	45	44	98%	93%
9	56	47	84%	94%
10	43	35	81%	94%
11	38	34	89%	96%
12	38	34	89%	92%
13	54	48	89%	98%
14	56	53	95%	90%
15	57	50	88%	96%
16	48	31	65%	90%
17	45	38	84%	91%
18	36	30	83%	89%

Table B.5: YOLOv4P6 DS accuracy and mAP@0.3.

Frame No.	Manually Counted	Automatically Counted	Accuracy	mAP@0.3
1	29	28	97%	92%
2	37	33	89%	95%
3	41	40	98%	95%
4	58	56	97%	92%
5	47	43	91%	93%
6	52	44	85%	86%
7	47	43	91%	88%
8	45	48	93%	95%
9	56	50	89%	97%
10	43	38	82%	95%
11	38	35	92%	94%
12	38	35	92%	92%
13	54	49	91%	98%
14	56	52	93%	91%
15	57	51	89%	97%
16	48	36	75%	90%
17	45	39	87%	93%
18	36	29	81%	90%

Table B.6: YOLOv4P6 WH accuracy and mAP@0.3.

Frame No.	Manually Counted	Automatically Counted	Accuracy	mAP@0.3
1	29	24	83%	89%
2	37	27	73%	89%
3	41	35	85%	96%
4	58	48	83%	96%
5	47	35	74%	94%
6	52	39	75%	82%
7	47	36	77%	78%
8	45	39	87%	91%
9	56	40	71%	90%
10	43	32	74%	89%
11	38	31	82%	92%
12	38	33	87%	92%
13	54	46	85%	98%
14	56	45	80%	88%
15	57	49	86%	95%
16	48	31	65%	82%
17	45	36	80%	94%
18	36	27	75%	93%