NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

PEDRO FRANCISCO CÂNDIDA OLIVEIRA DA SILVA FONTE

BSc in Computer Science and Engineering

AUTOMATIC COUNTING AND GEOREFERENCING OF FRUITS FROM VIDEO

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

AUTOMATIC COUNTING AND GEOREFERENCING OF FRUITS FROM VIDEO

PEDRO FRANCISCO CÂNDIDA OLIVEIRA DA SILVA FONTE

BSc in Computer Science and Engineering

Adviser: Fernando Pedro Reino da Silva Birra
Assistant Professor, NOVA University Lisbon

Co-advisers: Carlos Augusto Isaac Piló Viegas Damásio
Associate Professor, NOVA University Lisbon

João Carlos Gomes Moura Pires
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Maria Armanda Simenta Rodrigues Grueau
Associate Professor, NOVA University Lisbon

Members: Teresa Cristina de Freitas Gonçalves
Associate Professor, University of Évora

Fernando Pedro Reino da Silva Birra
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2023

Automatic counting and georeferencing of fruits from video

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.17) [30].

Para os meus pais, ***Maria Elisabete Silva*** *e* ***João Fonte****, e irmão* ***João Fonte*** *por todo o apoio ao longo do meu percurso académico*

Acknowledgements

I would like to thank my adviser, Dr Fernando Birra, and co-adviser, Dr Carlos Damásio, for their guidance and feedback throughout this project. Thanks to Dr Cláudia Soares for her help and knowledge in this research area.

This work was supported by a grant under the project *Smart Farm 4.0 (POCI-01-0247-FEDER-046078)* and funded by Compete2020 and FEDER. I want to thank all the entities that made this possible.

I would also like to thank my parents, Maria Elisabete Silva and João Fonte, and my brother, João Fonte, for supporting, inspiring, motivating, and helping me during my academic education.

Abstract

As the world population grows, so does food consumption and consequently the need for more efficient food production. Agricultural producers benefit from new technologies that keep up with these needs. Therefore, a proposal for counting and georeferencing fruits in an automated and accurate way is presented. This study was focused on apple fruit. However, the methods can be applied to other fruits.

Since the technique explored in this dissertation is video-based, for the sake of efficiency and easy access to tools, certain challenges arise, such as lighting at the recording location, camera angle, and the specific appearance of different types of fruit. These variables bring difficulties to the process of developing the technology described.

The proposed approach to solving these challenges involves the development of a system that uses computer vision techniques in conjunction with methods for detecting, tracking, counting, and georeferencing fruit in video. This system was built using artificial intelligence methods, trained by a dataset consisting of images and videos of fruits collected in orchards.

The best video fruit counting results were obtained using the YOLOv8 detection model with the BotSORT tracking system, which presented an accuracy of 87% compared to the estimated number of apples indicated by producers.

The main result obtained was the exploration and analysis of new techniques that make the counting process more efficient, using previously performed research. The development of this project sought to deepen the knowledge of this type of methods in order to contribute to the future development and implementations of this technology.

Keywords: Computer vision, Object detection, Object tracking, Georeferencing, Kalman Filter algorithm, Hungarian algorithm

Resumo

A população mundial encontra-se em crescimento, resultando num maior consumo de comida e na consequente necessidade de maior eficiência da sua produção. Os produtores agrícolas beneficiam de novas tecnologias que acompanhem estas necessidades. Para tal, é apresentada uma proposta de contagem e georeferenciação de frutos, de forma precisa e automatizada. Este estudo é focado principalmente em maçãs. No entanto, estes métodos podem ser aplicados a qualquer fruto.

Visto que a técnica explorada neste projeto baseia-se em vídeo, por uma questão de eficiência e acesso a ferramentas, surgem certos desafios como luminosidade no local de filmagem, ângulo da câmara e a aparência específica de diferentes tipos de fruto. Estas variáveis dificultam o processo de desenvolvimento da tecnologia descrita.

A abordagem proposta para solucionar estes desafios implica o desenvolvimento de um sistema que utilize técnicas de visão computacional em conjunto com técnicas de deteção, rastreamento, contagem e georeferênciação de frutos em vídeo. Este sistema foi construído através de métodos de inteligência artifical, treinados por um conjunto de dados composto por imagens e vídeos de frutos, coletados em plantações.

Os melhores resultados da contagem de frutos em vídeo foram obtidos utilizando o modelo de deteção YOLOv8 com o sistema de rastreio BotSORT, que apresentou uma precisão de 87% face ao número estimado de maçãs indicado pelos produtores.

O principal resultado obtido foi a exploração e análise de novas técnicas que permitam tornar o processo de contagem mais eficiente, utilizando investigações anteriormente realizadas. O desenvolvimento deste projeto procurou aprofundar este tipo de métodos, de forma a contribuir para o futuro desenvolvimento e implementações desta tecnologia.

Palavras-chave: Visão computacional, Deteção de objectos, Rastreamento de objectos, Georeferenciação, Algoritmo Kalman Filter, Algoritmo de Hungarian

Contents

List of Figures

List of Tables

List of Listings

Acronyms

ADAM	Adaptive Moment Estimation *(pp. 17, 18, 22, 37, 38)*
AI	Artificial Intelligence *(pp. 2, 3, 31)*
AP	Average Precision *(pp. 12, 16)*
CNN	Convolutional Neural Network *(pp. ix, 4–7, 12, 23)*
CNNs	Convolutional Neural Networks *(pp. 3, 4, 7, 22, 25)*
DL	Deep Learning *(pp. 2, 3, 15, 28, 56, 63, 77)*
Faster R-CNN	Faster Region-based Convolutional Neural Network *(p. 4)*
FCN	Fully Convolutional Network *(pp. 23–25)*
FN	False Negatives *(pp. 15, 16, 21, 47, 54, 56, 58, 60, 66–68)*
FP	False Positives *(pp. 15, 54, 56, 58–60, 66–69, 75)*
GANs	Generative Adversarial Networks *(p. 10)*
GPS	Global Positioning System *(pp. 9, 48, 78)*
IoU	Intersection over Union *(pp. 16, 53, 65, 66)*
KF	Kalman Filter *(pp. xi, 3, 8, 9, 18–21, 23–25, 44, 47)*
KLT	Kanade Lucas Tomasi *(pp. 18, 23–25)*
mAP	Mean Average Precision *(pp. 15, 16, 53, 55, 56, 65, 66, 68)*
MLP	Multi-layer Perceptron *(p. 6)*
MNIST	Modified National Institute of Standards and Technology dataset *(p. 6)*
ReLU	Rectified Linear Unit *(p. 6)*
SfM	Structure from Motion *(pp. 23–25)*

SGD Stochastic Gradient Descent *(pp. 17, 18, 37–40)*

SIFT Scale Invariant Feature Transform *(pp. 24, 25)*

SSD Single Shot Multibox Detector *(p. 4)*

TP True Positives *(pp. 15, 58, 66–68)*

YOLO You Only Look Once model *(pp. xi, 2–4, 12, 14, 15, 17–19, 24, 25, 29, 35–37, 39–42, 44–46, 53, 62, 64–70, 74–77)*

1 Introduction

1.1 Motivation

As global demand for food increases, so does the urgent need for the betterment of agricultural productivity. Precision agriculture[1] has emerged as an important tool to address this challenge by enabling farmers to monitor and manage their crops more efficiently. One of the key elements of precision agriculture is the ability to accurately count and locate crops in fields. In recent years, computer vision techniques have been widely used to detect and count objects in images and videos, allowing farmers to improve industry productivity and reduce costs [14, 2].

The constantly increasing worldwide population is a crucial challenge for agriculture. With an estimated 9.8 billion humans by 2050, an increase in food demand is consequently expected [34]. Not only this but the reduction of arable land, the decrease in the number of agriculture workers, the escalation of climate-related disruptions, and the increase of pests are all key challenges to agriculture. Some studies suggest that farmable land could be halved in the next quarter century, increasing the necessity for new ways to increase production efficiency. Labour shortages result in a small number of workers overseeing hundreds of thousands of acres, increasing the need to find new ways to make their job easier. Up to 40% of all crops worldwide are lost due to pests, resulting in the necessity to better identify and eliminate them before they cause severe damages [34].

This project was supported by a research grant included in the project "Smart Farm 4.0 (POCI-01-0247-FEDER-046078)" and funded by Compete2020[2] and FEDER[3]. This project consists of a collaborative effort between several entities to optimize the use of resources, sustainability and resilience of agricultural systems, resulting in higher profitability and added value.

These entities aim to design and develop innovative and accessible solutions in the context of Industry 4.0 for application in the agricultural sector.

[1] Precision agriculture is an agricultural resource management strategy that collects, processes, and evaluates data and offers insights to help farmers optimize and increase soil quality and productivity.

[2] https://www.compete2020.gov.pt/

[3] https://portugal2020.pt/glossario/feder-fundo-europeu-de-desenvolvimento-regional/

The invention and adoption of new technologies will be crucial to overcome the challenges mentioned above. AI in agriculture is already highly valued, but it is expected to increase over the following years exponentially. There are already applications of computer vision techniques in precision agriculture, precision livestock farming, autonomous farm equipment, and crop monitoring, among others [34].

Technology in agriculture is mainly used to automate farming through machines and computer systems to reduce human labour, errors and time. However, this requires significant investments that most farmers do not have access to.

Currently, counting and georeferencing fruit in orchards is mostly done manually, which is time-consuming and error-prone. Using computer vision techniques to count and georeference fruit in videos can revolutionize orchard management. It can help growers increase crop yields and reduce costs, improve agricultural sustainability, and reduce the amount of effort that small teams of workers have to oversee big crops.

This study aims to develop a system for automatic counting and georeferencing of fruit in videos, which can detect and count fruit in an orchard and determine their location. The system is based on computer vision and artificial intelligence techniques, including object recognition and tracking, using video data collected in an orchard for testing.

This solution allows farmers to gain insight and knowledge on their crops without making significant investments through accessible tools, such as a smartphone, to record, detect and count fruit. This allows farmers to better study their results and organize their production. Projects such as this one can also enable farmers to save time and increase efficiency by decreasing the excessive number of workers to count entire orchards.

1.2 Main Contributions

The main contributions of this project are studying new alternatives for fruit counting that are affordable to use and accessible to every producer in order for them to manage their productions better and not rely their production yield on mathematical estimations.

This project also aims to help agricultural producers have better control over their orchards and indirectly improve their management of resource usage, such as water and pesticides, allowing for more sustainable production.

The contributions rely on training Deep Learning (DL) models (in this case YOLO) in order to be possible to detect fruit in an orchard and track them so that counting of the production is provided. With this in mind, it also presents a set of datasets used for training these models, which can be used for future research problems in this area.

Models used were YOLOv7[56] and YOLOv8[52], so with this, it is also presented a performance comparison between the two models and the accuracy in this area when subjected to the same training dataset. Several state-of-the-art object trackers are evaluated with object detection models to understand their relevance in solving this task.

Another contribution is a methodology for georeferencing fruit using videos recorded with data such as GPS location and camera orientation. With this data, it is possible to

pinpoint detected fruit and their exact location, allowing for a cheap and easy way to know the overall density distribution over the orchard.

The principal contribution of this work is to allow the usage of new models to test and evaluate the usage of modern technology such as AI, DL and video processing algorithms for more precise production. This study focuses primarily on an Apple estimation. However, it can be applied to any fruit estimation, only having to change the training dataset of the model for the required fruit.

The YOLOv7 model showed above 88% for Precision, 86% for Recall, and 91% for mAP when tested in a large testing set with over 1000 new and unseen images. Tested on the same testing set, the YOLOv8 model showed above 87% for Precision, 85% for Recall, and 91% for mAP. The YOLOv7 model presented above 70% accuracy in detecting apples in authentic images from Portuguese orchards. The YOLOv8 in the same images showed an accuracy improvement of 5% compared to the previous model. These images were manually counted, including almost entirely occluded fruit in the trees. Regarding detecting and tracking fruit in videos for counting purposes, both models showed some under-counting and over-counting depending on the video conditions. However, upon visualisation of the resulting videos, the YOLOv8 showed more consistency and accuracy in detecting and tracking fruit than the previous version.

1.3 Document Structure

The following document is divided into the following chapters:

- **Chapter 2 - Fundamental Concepts:** In this chapter, fundamental concepts are explained, which are necessary for the implementation of this project. Concepts such as the difference between object detection and object tracking (Section 2.1), CNNs (Section 2.2), the KF (Section 2.3), Data Augmentation (Section 2.4), and the Hungarian algorithm (Section 2.5) are explained.

- **Chapter 3 - Related Work:** In this chapter, existing projects for fruit detection are presented, using the implementation of concepts described in Chapter 2.

- **Chapter 4 - Implementation:** This chapter dives into the details of how the solution to count and georeference fruit in video was implemented.

- **Chapter 5 - Results and Discussion:** This chapter exposes the results and evaluation of the methodologies used in Chapter 4.

- **Chapter 6 - Conclusion** - This chapter sums up all the study performed and briefly describes challenges encountered, future work and improvements that can be performed.

2

Fundamental Concepts

2.1 Object Detection and Object Tracking

Object detection and object tracking are two crucial tasks in computer vision, which have a wide range of applications in various fields, such as surveillance, robotics, and autonomous vehicles [13].

Object detection is a computer technology related to computer vision and image processing that involves identifying the presence, location, and type of objects in an image or video. This task typically consists in training a model to recognize objects of interest (such as humans, buildings, or cars) and then applying the model to new images or videos to identify these objects. Well-researched domains of this technology include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance. There are several techniques for these purposes, including template matching, feature-based methods, and Deep Learning-based methods [41].

Deep learning-based methods, such as Convolutional Neural Networks (CNNs), are now the state of the art in object detection. These methods can be trained through large datasets of labelled images, learning to recognize objects in new images with high accuracy. Some popular CNN-based methods for object detection are Faster R-CNN, YOLO, and SSD [26].

Object tracking, on the other hand, involves identifying and tracking the motion of objects in a video over time. This task usually consists of detecting the object in question in a frame of the video and tracking it in subsequent frames. Object tracking algorithms can be divided into two main categories: Single Object Tracker and Multiple Object Tracker. Single Object Trackers focus on tracking a single object, while Multiple Object Trackers can track multiple objects simultaneously. There are several techniques for object tracking, including correlation filter-based tracking, deep learning-based tracking, and particle filter-based tracking [41, 7].

As Figure 2.1 shows, it is necessary that the object in question is always detected first so that it can then be tracked between different frames of a video. This means that an

Figure 2.1: Steps Involved in Object Tracking [8].

object cannot be tracked without first being detected.

The challenge with using object detection as a standalone tool is that when using a video to detect objects of interest, if an object is detected in a particular frame of the video, there is no guarantee that this same object will be detected in subsequent frames. This leads to some difficulties in counting objects, as the detector might indentify the same object as being different.

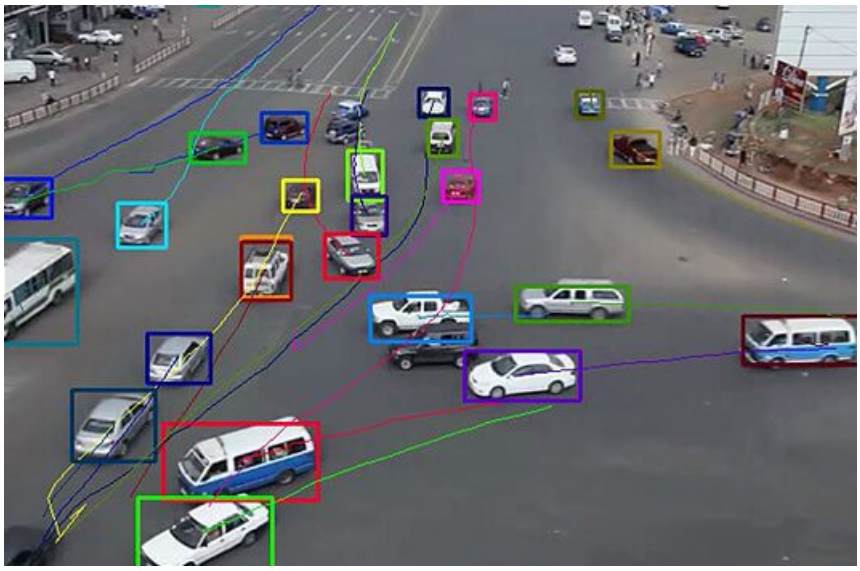


Figure 2.2: Real-time object tracking of cars [36].

The main advantage of object detection and additional object tracking is that the model identifies not only the desired object but also its unique features, allowing it to track it continuously in different frames of a video. This can be seen in Figure 2.2, where a line from previous video frames tracks each car.

Thus, Object Detection alone is not sufficient for counting fruit in a video. Since we are analyzing video and trying to count fruit, we need to consider both the position of the object we are trying to detect and that it is different from all the others so that double-counting of the same object is eliminated.

2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Network (CNN) is a type of deep learning algorithm designed to process and analyze data with a grid-like topology, such as images, videos, and audio signals [2]. It uses the mathematical operation of convolution, which extracts features from

the input data. What differentiates these artificial networks from standard Multi-layer Perceptron (MLP), is the existence of Convolutional Layers as hidden layers that use filters to transform the input data into output data for feature extraction. This is useful because these filters are able to identify patterns, important for object detection in images and videos [40]. A simplified CNN architecture for MNIST classification is illustrated in Figure 2.3, in which the basic functionality can be broken down into four key areas: the input layer, convolutional layer, pooling layer, and fully-connected layers. The input layer will hold the pixel values of the image. The convolutional layer, formed by neurons, will determine its output through the calculation of the dot product between local regions of the original image, received as input, and the weights present in the neuron's filters. The Rectified Linear Unit (ReLU) applies an 'elementwise' activation function, just like sigmoid activation function, to the output of the activation produced by the previous layer. The pooling layer will perform a downsample along the spatial dimensionality of the given input, reducing the number of parameters within that activation. Finally, the fully-connected layers will attempt to produce class scores from the activations to be used for classification. It is also suggested the usage of ReLu between these layers to improve performance [40].

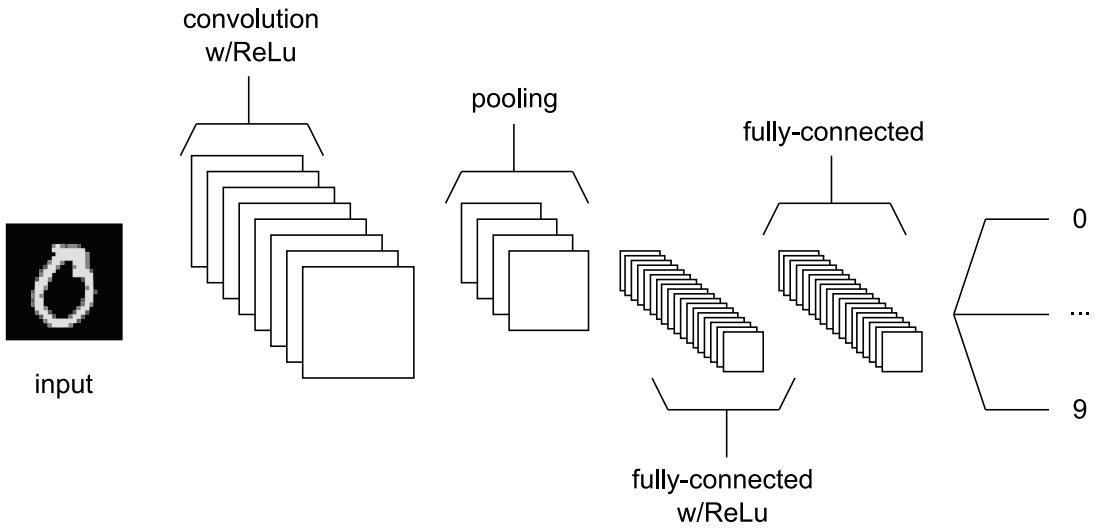


Figure 2.3: A simple CNN architecture, comprised of five layers [40].

A CNN comprises multiple layers, each processing the input data and transforming it into a higher-level representation. The first layers of a CNN, known as the convolutional layers, apply filters to the input data, which extract features such as edges, textures, and shapes. These filters are usually a grid-like topology representing the feature we want to extract, in the form of a matrix. Consider, for example, an image of a seven (Figure 2.4), composed of a matrix of pixels. Using one convolutional layer, the edges of the object in the picture (left, right, top and bottom edges) can be extracted through the usage of filters, one for each edge. We consider these filters as a three-by-three matrix, using only

Figure 2.4: Input image of the CNN for identification of the number seven [16].

values between -1 and 1, where -1 corresponds to black, 1 corresponds to white, and 0 to grey. The matrices of these filters that extract the edges of the object in the picture are displayed in Figure 2.5, where the Figures 2.5(a), 2.5(b), 2.5(c), 2.5(d), display top, left, bottom and right edges, respectively. Since these filters consist of a 3 x 3 matrix, a window of 3 x 3 pixels is used to stride the original image (Figure 2.4) and compute the dot product between each window and the filter, one at a time. This dot product between two matrices will originate a single value, which will be the result of a pixel in the transformation image by that filter. This means that if an image is passed through a filter that extracts top edges, the top edges of the object inside this image will be highlighted [16].

The "dot product" is a term that might have multiple meanings. In this context, it is used to describe the operation of the sum of the element-wise products of each pair of elements in the two matrices (also called *Hadamard product*) [16].

This is visually represented in Figure 2.6. The filter that extracts the top edges (Figure 2.5(a)) from the input will originate a new figure (Figure 2.6(a)), where the top edges are identified with a brighter color (white). The same applies to the filters for left (Figure 2.5(b)), bottom (Figure 2.5(c)), and right (Figure 2.5(d)) edges, which originate new images where the left (Figure 2.6(b)), bottom (Figure 2.6(c)) and right (Figure 2.6(d)) edges are identified, respectivelly [16].

Every input transformed by "filters" in convolutional layers results in outputs to identify specific patterns, which are then passed through successive convolutional layers to identify other relevant features for what is desired to detect. Some CNNs may have pooling layers between convolutional layers that reduce the dimension of the input of the next layer, which allows for faster computation [40].

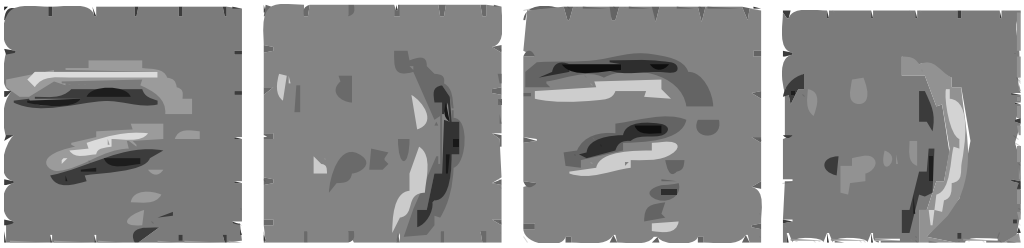
Successive convolutional layers work in a way that the features extracted by the first layers pass through to the following layers, where new features are extracted. For example, the first convolutional layers extract simple features like shapes, lines, and edges. Later layers apply previous knowledge to identify more in-depth characteristics such as eyes, ears, and fur so that we can identify, for example, a cat in the final layers [40].

The final layers of a CNN, known as the fully connected layers, are used to classify the input data based on the features extracted by the previous layers. Finally, the CNN

-1	-1	-1	-1	1	0	0	0	0	0	1	-1
1	1	1	-1	1	0	1	1	1	0	1	-1
0	0	0	-1	1	0	-1	-1	-1	0	1	-1

(a) Filter of top edges (b) Filter of left edges (c) Filter of bottom edges (d) Filter of right edges

Figure 2.5: Filters of the convolutional layer that extracts edge features from the original image. Panel (a) features the matrix for top edges extraction. Panel (b) features the matrix for left edges extraction. Panel (c) features the matrix for bottom edges extraction. Panel (d) features the matrix for right edges extraction [16].



(a) Output image from top edges extraction filter (b) Output image from left edges extraction filter (c) Output image from bottom edges extraction filter (d) Output image from right edges extraction filter

Figure 2.6: Output images with features highlighted due to filters. Panel (a) shows an original image with the top edges highlighted. Panel (b) shows an original image with the left edges highlighted. Panel (c) shows an original image with bottom edges highlighted. Panel (d) shows an original image with right edges highlighted [16].

is trained on a large dataset of labelled examples to recognize the patterns and features most relevant to the task at hand [40].

2.3 Kalman Filter

The Kalman Filter (KF) is a mathematical algorithm for estimating the state of a system from a set of noisy and uncertain measurements. The algorithm is used in various fields such as control systems, navigation, and estimation. It is particularly useful in systems where the measurements are noisy and uncertain and system dynamics are complex [10].

It uses a mathematical model[1] of the system and a mathematical model of the noise, associated with the measurements, to estimate the state of the system. This allows us to predict the future state of the system and correct the predictions with each new measurement [37, 51]. If we consider the movement of a car on a road, the state equation (mathematical model of the system) could be, for example, the position of the car, whereas

[1]A mathematical model usually describes a system through a set of variables and a set of equations that establish relationships between variables.

the noise equation (mathematical model of the noise) could be the wind forces affecting the car. In this common problem, we want an accurate estimate of its true state, even though we cannot directly measure it. This algorithm works by using the available measurements of the state system as well as the noise, to make the most accurate measurement of the true current state of the system [51].

The KF algorithm consists of two main steps: prediction and correction. In the prediction step, the algorithm uses the current estimate of the system's state, also called the "prior estimate", and the system's model, known as the "state transition model", to predict the state of the system at the next time step, known as "posterior estimate" [51, 3].

The prediction step estimates the state of the system at the next time step, based on the system's state at the current time step. This step uses the system model, which describes the dynamics of the system, to estimate the next state of the system. The prediction step also uses the control inputs, if there are any, to update the state of the system [3, 53].

In the correction step, the algorithm uses new measurements of the system to update the estimate of the next system's state, by comparing the predicted measurement with the actual measurement and adjusting the estimate accordingly [3, 53].

Suppose we are trying to track a moving object, like a car. This car has GPS technology to inform us about its location, but it is not always accurate and updates the location at certain intervals, not continuously. The KF algorithm uses the location updates from the GPS and the car's velocity to predict where the vehicle will be in the future. Then, when we get the new GPS location, we use it to correct its prediction. A more straightforward example would be to consider a ball thrown by a person, while we are attempting to predict where the ball will fall. Since we have little information at the beginning of the ball's trajectory, it is hard to understand where it will fall. However, as it moves through the air and gets closer to the ground, we can better estimate where the ball will hit the ground [51].

In object detection and tracking, the goal is to locate and identify objects within an image or a video. Once the object is detected, the algorithm needs to track its position and location as it moves through the scene. Tracking the position and location of the object is an essential step because, in this project the goal is to count fruit, meaning we need to avoid double counting as much as possible. Hence, the KF algorithm tracks the object by estimating its current position and velocity and then uses this information to predict the object's future position. It can then use new measurements of the object detections to correct its predictions and better their accuracy.

2.4 Data Augmentation

One of the main challenges in object detection is the limited amount of labeled data available for training. Data augmentation is a technique used to artificially increase the size of a dataset by applying various transformations to the existing data. This technique

is commonly used in machine learning and computer vision to improve the performance of models by increasing the diversity and robustness of the training data [50].

Data augmentation generates new training examples by applying various transformations to the images and their associated bounding boxes. These transformations can include:

- Geometric transformations such as rotation, scaling and flipping.
- Photometric transformations such as brightness and contrast adjustments and adding noise.
- Random cropping and padding.

Geometric transformations generate new training examples, which can help the model learn to detect objects regardless of their orientation or size. Photometric transformations help the model learn to detect objects that are partially obscured or in difficult poses. Random cropping and padding help with detecting objects that are not fully visible in the image or with different scales that were not present in the original dataset [50].

Applying these transformations to the training data exposes the model to a wider variety of images with different scales, orientations, lighting conditions, and noise levels, helping the model better adapt to new, unseen images during the testing phase, also improving the robustness of the model to variations in the input data [50].

In addition to these techniques, there are other advanced data augmentation techniques that can be used for object detection. For example, using Generative Adversarial Networks (GANs) to generate new images, or using synthetic data generated from 3D models [46].

Additionally, data augmentation can also help to reduce overfitting, which occurs when the model becomes too closely adapted to the training data and performs poorly on new, unseen data.

2.5 The Hungarian Algorithm

The Hungarian algorithm is used for the solving of assignment problems in polynomial time. An assignment problem can be seen as an optimization problem where we have j number of jobs and n number of workers, to perform those jobs. Each job is performed by only one worker. Each worker will then perform each specific job, at a different resource expenditure. The final goal is to find a worker for each job costing as few resources as possible. This kind of assignment can be visualized by a matrix where the number of lines is the number of workers (n) and the number of columns is the number of jobs (j). Each cell will then contain the cost of the n-th worker to perform the j-th job [49]. It is important to note that this algorithm solves the problem by considering a matrix that has the same number of rows and columns. For cases that do not satisfy this, a number of rows or columns filled with zero are added to the matrix, and then the algorithm is

performed normally. The Hungarian algorithm solves these assignment problems in five different steps, considering the matrix of the cost of workers per job [38]:

1. Subtraction of the smallest entry in each row from all the entries in the row. This will make the smallest entry in each row equal to 0.

2. Subtraction of the smallest entry in each column from all the entries in the column. This will make the smallest entry in each column equal to 0.

3. Find the smallest number of vertical or horizontal lines that can be drawn to go through all 0's in the matrix.

4. If the number of lines drawn is equal to the number of rows or columns, an optimal assignment is possible and the algorithm is finished, where each entry with a zero can be considered as an optimal solution for the assignment, considering that the zeros in the entries correspond to an optimal result and there may be more than one optimal solution. If the number of lines drawn is less than the number of rows or columns, the optimal solution is not yet found and we need to perform step 5.

5. Find the smallest entry not covered by any line. Subtract this value to any cell that is not crossed out by any line (uncovered cell), then add this value to each cell that is crossed by a horizontal and vertical line. Then we go back to step 3 and continue the algorithm.

These problems involving an adjacency matrix can also be transformed into bipartite graphs problems, with rows and columns considered as nodes and each matrix entry viewed as the cost of the edge [38]. Object detection and tracking problems fall into these assignment problems because we want to associate the object detected in the current frame with the same object detected in previous frames. In practice, in detecting and tracking objects, we want to associate each detection on a specific frame to a particular track, ensuring the two objects in different video frames are considered identical. Therefore, we can reformulate these problems to consider the matrix of the associations between the detections (for example, rows) and the trackings (for example, columns). Regarding the costs of the edges or the entries of the matrix, we have some options, such as [23, 15]:

- IOU (Intersection Over Union) considers that if the bounding box of an object detected in the current frame is overlapping a bounding box from a detected object in the previous frame, there is a probability that this object might be the same one;

- Shape Score, which considers that if the shape or size didn't vary too much during two consecutive frames, the score increases, giving a higher chance that they might be the same object;

- Convolution Cost runs a CNN on the bounding box and compares the result with the bounding box from a frame ago. If the convolutional features are the same, then the objects look the same.

It is not necessary to stick to one of the previous costs, enabling us to choose what best applies to our specific assignment problem. As long as we consider two lists of bounding boxes, one for the tracking and the other for detection, and create the adjacency matrix considering our specific measuring costs, we can use the Hungarian algorithm to solve our problem.

The Hungarian algorithm is helpful for problems such as object detection and tracking in computer vision techniques because this algorithm solves a high-complexity problem at a lower time complexity.

2.6 YOLOv7 Model

YOLOv7 is a state-of-the-art model architecture for object and real-time detection. It distinguishes itself for its high performance and accuracy in comparison with other state-of-the-art detection models. YOLOv7 has the highest accuracy, 56.8% AP, among all known real-time object detectors with 30 FPS or higher on GPU V100. YOLOv7 outperforms YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable DETR, DINO-5scale-R50, ViT-Adapter-B and many other object detectors in speed and accuracy. Moreover, the creators train YOLOv7 only on the MS COCO dataset from scratch without using other datasets or pre-trained weights [56].

When crafting an efficient neural network, designers often prioritize optimizing a subset of parameters—chiefly computation count, computational density, and the number of parameters. YOLOv7's architectural blueprint draws inspiration from ELAN (efficient layer aggregation network). The fundamental premise of ELAN lies in its meticulous control over the shortest and longest gradient paths, ensuring the effective convergence and learning of deeper networks [56, 44]. The core of ELAN's architecture is shown in Figure 2.7.

The input stream is directly channelled into the foundational block via 1x1 convolutions. Simultaneously, two additional connections are established by subjecting the input to convolutions with 2 and 4 sets of 3x3 convolutions, each maintaining the same channel multiplier. These outputs are amalgamated at the base block, followed by applying 1x1 convolutions to glean enriched information [56, 44].

The YOLOv7 paradigm refines the ELAN architecture, giving rise to E-ELAN (extended efficient layer aggregation network). E-ELAN incorporates the principles of expand, shuffle, and merge cardinality, enhancing the model's learning capacity while upholding the integrity of gradient flow paths. The modification enacted by YOLOv7 is focused exclusively on the architecture of the computational block, while the transition layer retains the foundational ELAN structure. E-ELAN (shown in Figure 2.8) leverages

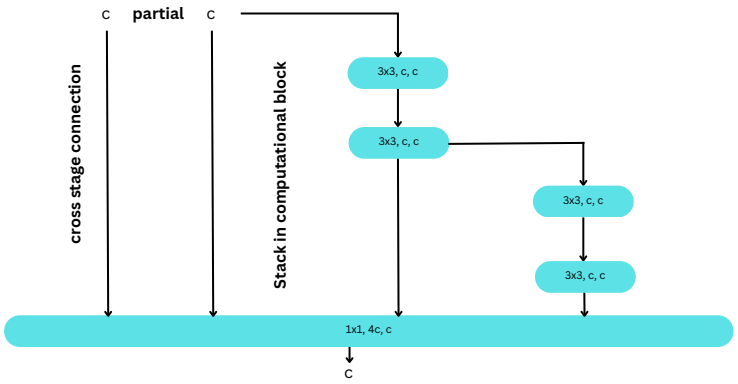


Figure 2.7: Architecture diagram of ELAN [56].

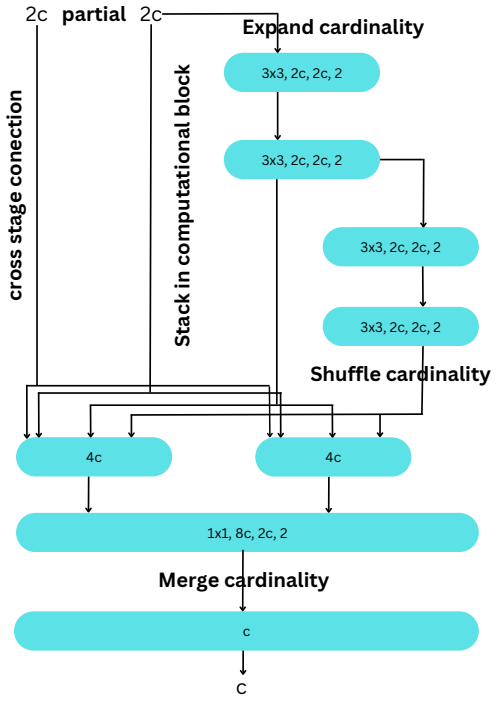


Figure 2.8: Architecture diagram of E-ELAN [56].

group convolution to amplify the channels and cardinality within the computational block. The uniform channel multiplier and group parameter are employed across all computational blocks within a computational layer. Post-group convolution, the feature maps originating from each computational block are organized into groups of size "g", which are subsequently concatenated. This culminates in applying a merged cardinality process, whereby shuffled group feature maps are combined [56, 44].

Model scaling emerges as a pivotal concept, serving as a pathway to augmenting a model's depth, image resolution, and architectural breadth. Scaling the depth involves adjusting the number of layers within the model, whereas scaling width corresponds to

manipulating the channel count within the architectural framework. The model architecture files delineate scaling factors for both depth and width. Given YOLOv7's concatenated architecture with other layers, scaling the depth parameter of a computational block necessitates a recalibration of output kernels. Consequently, the scaling of the width parameter must mirror the calculated kernel adjustment. This compound scaling strategy preserves the fundamental attributes embedded in the initial architectural design, maintaining its integrity and optimal structure [56, 44].

2.7 YOLOv8 Model

YOLOv8 is a new state-of-the-art computer vision model that introduces several significant advancements and refinements, enhancing its capabilities for object detection. While an official research paper for YOLOv8 has yet to be published, some insights have already been derived from available resources and repositories. Ultralytics developed this project, and the model YOLOv8 is a modified version of YOLOv5 CSPDarknet53 architecture [35].

A noteworthy update in YOLOv8 involves the modification of the backbone architecture. The C3 modules, the previous backbone component, have been replaced by the C2f modules. The C3 module refers Cross Stage Partial (CSP) Bottleneck with three convolutions, while the C2f module refers to a faster implementation of the C2 module, which is a CSP bottleneck with two convolutions. This transition aims to optimize feature extraction for improved object detection. Notably, the initial 6x6 convolution in the stem has been replaced with a more efficient 3x3 convolution, streamlining the early processing stages [52, 12].

In C2f, the outputs from the Bottleneck and a combination of two 3x3 convolutions with residual connections are consolidated. This differs from C3, where only the output from the last Bottleneck is used. This shift in the treatment of Bottleneck outputs enhances feature integration and representation, contributing to the model's overall performance [52, 12].

YOLOv8 introduces a series of convolutional refinements, focusing on the Bottleneck component. While the Bottleneck structure remains consistent with YOLOv5, a pivotal alteration is observed in the first convolution within the Bottleneck. Here, the kernel size has transitioned from 1x1 to 3x3. This change signifies a shift towards the ResNet block design introduced in 2015, aligning YOLOv8 with established architectural principles that have proven effective in deep learning [52, 12].

Adopting anchor-free detection is one of the most significant transformations in YOLOv8. Unlike its predecessors, YOLOv8 directly predicts the centre of an object, eliminating the need to calculate offsets from predefined anchor boxes. This approach enhances flexibility and efficiency in object detection, as it eliminates the manual specification of anchor boxes, which had posed challenges in earlier YOLO models. Anchor-free detection reduces the number of box predictions, leading to expedited Non-Maximum

Suppression (NMS) during the post-processing step, streamlining candidate detection refinement [52, 12].

YOLOv8 focuses on architectural enhancements and optimizes its training routine. Online augmentation is a critical aspect of training, where the model encounters slight variations of input images at each epoch. An important augmentation technique, mosaic augmentation, involves stitching four images together, exposing the model to diverse object locations, partial occlusions, and varying pixel environments. However, empirical evidence suggests that continuous use of mosaic augmentation throughout training can degrade performance. As a result, it is advisable to disable mosaic augmentation during the final ten training epochs, ensuring that the model converges effectively [52, 12].

Overall, this new state-of-the-art model showcases remarkable architectural improvements, including backbone evolution, convolutional refinements, the transition to anchor-free detection, and optimization of training routines. These enhancements collectively bolster the model's performance and represent a significant stride forward in object detection within the deep learning domain. This served as a motivation for implementing and evaluating this model in this research project.

2.8 Metric Evaluation

Metric evaluation is considered a process to measure and quantify the quality of an DL model. The evaluation metrics that are more commonly used through the execution of this study are Precision, Recall, Mean Average Precision and F1 Score.

Precision is an evaluation metric that quantifies the accuracy of a model's Positive predictions. It assesses the proportion of correct positive predictions, thus pivotal in mitigating False Positives. Precision is computed by dividing the number of True Positives (TP) predictions by the total number of Positive predictions, englobing both True Positives and False Positives (TP + FP). Hence, this metric highlights the model's competence in minimizing wrong detections, as well as ensuring the degree of confidence in its affirmative detections.

Recall is central to object detection metrics, evaluating the model's capability to identify all pertinent objects within an image. It quantifies the model's effectiveness in capturing a particular class's actual instances. Recall is computed by establishing the ratio of TP predictions to the total number of existing positive instances (TP+FN) in an image. This measure takes into account both True Positives and False Negatives.

The Mean Average Precision (mAP) is a pivotal metric in object detection, serving as a robust measure to assess the performance of models tasked with detecting and classifying objects within images, particularly in scenarios involving multiple object classes. Its significance lies in its capacity to provide a comprehensive evaluation, considering variations in class-specific performance. At its core, the calculation of mAP encompasses several vital steps. Initially, Precision and Recall values are computed for each class across a range of confidence thresholds. This value is achieved by sorting the model's predictions

based on their associated confidence scores. Subsequently, interpolated precision values are determined at different Recall levels. Finally, each class's Average Precision (AP) is calculated as the area under the Precision-Recall curve. The mAP is derived by taking the mean of these AP values across all classes, thereby encapsulating the model's overall performance. mAP metric provides a balanced assessment of a model's performance by considering both Precision (the accuracy of Positive predictions) and Recall (the model's capacity to detect all relevant objects) across various confidence thresholds. However, since the Average Precision (AP) is the area under the Precision-Recall curve resulting in a single value, the Mean Average Precision (mAP) does not inform at which confidence threshold the model performs best, but only an overall performance across different confidence thresholds.

The nomenclature mAP@[IoU threshold] is a variation of the mAP metric prevalent in object detection models. This variation considers the Intersection over Union (IoU) value between predicted and ground-truth bounding boxes. By evaluating different IoU thresholds (for example, 0.5, 0.95), this metric measures object detection accuracy at varying levels of overlap between predicted and ground-truth boxes.

The F1 Score is a metric that helps understand which confidence score the model performs best. It is important to note that with higher confidence, the Precision might increase as the model detects more assuredly the proper objects to detect. However, it also might not detect all the instances in the image, increasing the number of False Negatives and decreasing the Recall metric. So, a good balance between Precision and Recall regarding the model's performance is desired.

The F1 Score is a singular metric combining Precision and Recall into a single value, offering a harmonious balance between these critical facets of model performance. This metric is calculated by determining the harmonic mean of Precision and Recall. The F1 Score provides a unified measure that optimally balances the Precision-Recall trade-off, ensuring a comprehensive appraisal of a model's performance. This metric also allows the analysis of the model's performance as the confidence threshold changes. Examining the F1 curve makes it possible to identify the confidence threshold at which the model achieves the best balance between Precision and Recall. The highest F1 Score on the curvature indicates the confidence score where the Precision-Recall trade-off is most optimised.

These metrics are essential tools to assess the performance of object detection models. They are used for validation to improve the model's performance or for testing and quantifying its performance.

3

Related Work

3.1 YOLO 2 for fruit detection in video using Kalman Filter

In this section, we will present the study made for automatic pear and apple detection using the deep-learning-based method referred to as YOLO in combination with the Kalman Filter algorithm to identify fruit in successive video frames [22].

In this report the data was acquired using an iPhone X, filming the fruits under the canopy. Considering object detection, they used YOLOv2, but other convolutional neural networks were also used, such as YOLOv3. The pears and apples in the videos were manually labelled for training. The software used for the annotation was MATLAB, as well as for object detection and tracking. The fruit counting system was evaluated using the final counting result, not the detection accuracy in the test image frames. There was some Data Augmentation performed, used for image variety. For this, the input image was flipped horizontally, color information was altered randomly, where red-green-blue was converted into hue-saturation-value changed by -20% to +20%, and the rotation of images was applied. Since the task at hand is object detection and not image classification, the location of bounding boxes to identify fruit was corrected based on the augmentation. For example, if the input image was rotated by 30º, the position of the bounding boxes was rotated also by 30º. The flipping of images horizontally was also implemented in this work with 50% probability. However, vertical flipping of images was not performed in this work [22].

The size of input images was 608 x 608 pixels, using a pre-trained YOLOv2 with an input size of also 608 x 608 pixels presented by Redmon and Farhadi (2017) [43]. Since the input size affects the detection accuracy, a smaller image input was also tested, for comparison. To test this smaller image input size they built a YOLOv2 network with a backbone network of Res-Net18 [21]. The parameters for learning such as initial learning rate, optimizer, and the number of epochs, were manually determined, while the accuracy was evaluated using the validation dataset. They also used Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (ADAM) optimisers [20]. The initial learning

rate[1], momentum[2], L2 regularization[3], and number of epochs, were 1x10^{-2}, 0.9, 1x10^{-4}, and 120, respectively. The minibatch[4] size was 4. For training, the authors used a personal computer with an Intel Core i7-9700F central processing unit, random-access memory of 16 GB, and a GeForce RTX2070 graphics processing unit (NVIDIA Corporation, USA). The evaluation metric for object detection was the average precision [22].

Considering the tracking of the object, since it is essential in order to prevent multiple counting on the same object, in this research, a Kalman Filter was applied to the sequence of the video frames, since it predicts the object's future location and it does so when the object failed to be detected during the tracking. KF was used as an estimator to predict and correct the state of the tracked fruits. To implement this, a tracking system where X_k is the state vector was considered. X represents the dynamic behavior of the object, where the subscript k indicates the discrete time. A parameter called deletion threshold, present in the KF was set to [5,5], meaning that if a confirmed track is not assigned to any detection 5 times in the last 5 tracker updates, then the track is deleted. Hence, if the fruit is not detected for five frames in a row, the fruit information is removed. In order to validate the effectiveness of the Kalman Filter in fruit counting, the KLT feature-tracking algorithm was used, where image features in each bounding box were extracted using the minimum-eigenvalue algorithm. For evaluation, a video that was not present in either the training or test data for the YOLO detector was used. Then, the pears were counted manually and compared with the result of the automated counting [22].

The results of the different methods through evaluation of the average precision for pear detection go as follows and are presented in Table 3.1. The method with YOLOv2 with a backbone of ResNet-18 presented an average precision of 0.95. The method YOLOv2 with the ADAM optimizer presented an average precision of 0.96. The method using YOLOv2 without the usage of data augmentation presented an average precision of 0.88. The method of YOLOv2 with SGD optimizer presented an average precision of 0.97, which has the same value as the method YOLOv3 with SGD optimizer [22].

Table 3.1: Average precision in pear detection with different methods [22].

Method	Average precision
YOLO v2 with backbone of ResNet-18	0.95
YOLO v2 with ADAM optimizer	0.96
YOLO v2 without data augmentation	0.88
YOLO v2 with SGD optimizer	0.97
YOLO v3 with SGD optimizer	0.97

These results only show how well the fruit was detected. The results for pear counting

[1]Initial learning rate is a hyperparameter that controls how much is changed in the model in response to the estimated error each time the model weights are updated [11]

[2]Momentum is a hyperparameter used in gradient-based optimization techniques to speed up learning in directions of low curvature, without becoming unstable in directions of high curvature [59].

[3]L2 regularization is a hyperparameter which adds a "squared magnitude"coefficient as penalty to the loss function [39].

[4]Minibatch is the number of training examples used in each training iteration [4].

are present in Table 3.2. The proposed method (YOLOv2 with Kalman Filter), the method with no data augmentation, and the method with YOLOv2 with a backbone of ResNet-18 for smaller input sizes were tested using the same test video with a total of 234 pears. The proposed method detected 231 apples. However, for evaluation, omission, double count, and wrong detection must also be considered. An object that was not a pear but was detected as one was regarded as a wrong detection. Considering this, the Double count and the Wrong detection numbers were deducted from the Detected Number, to obtain the correct detection number. The proposed method obtains a correct detection of pears of 226. The precision[5], recall[6], and F1[7] values for this method were 0.978, 0.966, and 0.972, respectively. Regarding the method with no data augmentation, the correct detection number of pears was 196, while the precision, recall, and F1 values were 0.951, 0.838, and 0.891, respectively. The method which analyzed the smaller input image size had a correct detection number of 211, while the precision, recall, and F1 values were all 0.902. Considering apple detection, the proposed method detected correctly 157 apples out of 170, and the model was evaluated by the same metrics as pear counting, with values of 0.935 for precision, 0.924 for recall, and 0.929 for F1 score, as shown in Table 3.3 [22].

Table 3.2: Result of the pear counting [22].

Method	Detected number	Omission	Double count	Wrong detection	Correct detection	Number	Precision	Recall	F1
Proposed Method	231	8	4	1	226	234	0.978	0.966	0.972
No data augmentation	206	38	8	2	196	234	0.951	0.838	0.891
With smaller input image size	234	23	2	21	211	234	0.902	0.902	0.902

Table 3.3: Result of the apple counting [22].

Method	Detected number	Omission	Double count	Wrong detection	Correct detection	Number	Precision	Recall	F1
Proposed Method	168	13	3	8	157	170	0.935	0.924	0.929

3.2 MangoYolo Approach for counting Mangos in videos

This project[57] proposed and tested a method involving deep learning for on-tree mango fruit detection, tracking and counting. In order to perform the detection, tracking, and counting, a deep learning detection algorithm based on mango fruit was used, called MangoYOLO[25], a KF to predict the position of fruit and avoid multiple counts of a single

[5]Precision as an indicator of the machine learning model's performance, refers to the number of true positives divided by the total of positive predictions. It is used to measure the model's accuracy in classifying a sample as positive.

[6]Recall is an evaluation method for machine learning models which is calculated by dividing the number of true positives by the sum of well-classified objects (addition between true positives and false negatives). It is used to measure the ability of the model to detect positive samples.

[7]F1 score measures the accuracy of the model. It is computed by using precision and recall.

fruit and the Hungarian algorithm, to correlate fruit between neighbouring frames, with the improvement of enabling multiple-to-one assignment.

To obtain the imaging of fruit, two dual-view images (one per tree side) for each tree were used. The videos were filmed at night, after sunset. The camera was attached to a vehicle with a distance of two meters to the canopy, moving at around five kilometres per hour, so that there were around thirty frames per tree and a shift of approximately twenty pixels per frame. The vertical speed was assumed zero and every tree was geo-located to an accuracy of two centimetres [57].

Considering fruit tracking and counting, as stated previously, the KF and Hungarian algorithms were used, with some alterations done to improve their effectiveness. The KF presented a challenge: in order for this algorithm to work, it requires at least four continuous updates (four measurements) to obtain a relatively accurate location prediction model. There may be some cases in a video where a single fruit only appears a few (< 4) times, making the standard KF not applicable to that fruit. To overcome this, an improvement was made in cases where the fruit appears less than four times. The speed of fruit that is needed to calculate the prediction for the standard KF would be borrowed from a neighbouring fruit that has been updated at least four times in the whole video. If no "stable" neighbour fruit can be found, as occurs in the first four frames of a video, then a predefined global speed was selected based on the assumption that the camera travels at five kilometres per hour on flat ground. The Hungarian algorithm was used to identify if a certain fruit detected in the current frame is the same as the one detected previously. However, this algorithm presented a challenge: considering A_1 and B_1 as two fruits detected in Frame 1, and in Frame 2 fruit A is not detected, fruit B is detected in a new position as B_2, and a new fruit C is detected at position C_2. The standard Hungarian algorithm may assign B_2 to A_1 and C_2 to B_1. To solve this issue, an improved Hungarian algorithm was proposed, in which [57]:

(i) the Hungarian algorithm is applied to tracked and new fruit to obtain one-to-one assignments;

(ii) the maximum distance threshold is applied to decorrelate the assignments with large distances;

(iii) the Hungarian algorithm is applied a second time to unassigned tracked fruit and new fruit;

(iv) where two tracked fruit have been assigned to the same new fruit (a 'multiple-to-one assignment'), only the assignment with a smaller cost (distance) is retained.

This improved Hungarian algorithm can fail to count a new fruit in the situation that the new fruit appears very close to a tracked fruit that is not detected in the current frame, as in the case of the issue stated above. However, such cases were found to be rare.

Regarding results, fruit from videos were also manually counted in order to compare with the proposed method. It is imperative to evaluate the impact attributed to the value of the number of frames in which a fruit was not detected before it was removed from the tracked list. To test this, a range of values from 0 to 50 were chosen, which at a setting of 0, the KF was not used and only the Hungarian algorithm was used to correlate the fruit. This setting to zero resulted in a high count (n = 357) in comparison with the human count of 192. The values were stabilized with the setting at 15 where the MangoYOLO counted 191 fruits. The distance threshold used in assessing the validity of the Hungarian algorithm also needed to be considered. This distance threshold was optimized by trying a range of values from 20 to 100 pixels, with a step of 5. A smaller value should result in an increased repeat count, whereas a larger value should result in an increased incorrect assignment of a new fruit to tracked fruit, leading to an underestimation of the count. The estimation was close to the human count at a setting of 60 pixels [57].

As stated above, the human count was 192, while the estimated count of fruit by the final proposed method was 197. As seen in Table 3.4, the proposed method resulted in an overestimate of 2.6% of the human count. Some errors were also noted (Table 3.4), such as the repeat count associated with temporary occlusion within a 15-frame interval with a value of 3; the false prediction (Repetitions due to FN in Previous Frames) due to an inaccurate prediction model, or the platform suffering an abrupt movement, had a value of 1; the missing of a previous detection (False Prediction of Position); the case when a new fruit appears close to a tracked fruit and the tracked fruit does not exist in the current frame, the new fruit is assigned to the tracked fruit and a count is missed, which had a value of -14 [57].

Table 3.4: Error attributions based on human assessment for fruit detection in 110 video frames using the tracking (MangoYOLO-Kalman Filter-Hungarian algorithm) method [57].

Human Count	Repeat Due to Occlusion in Previous Frames	Repeat Due to FN in Previous Frames	False Prediction of Position	Total Repeat Count	Missed Count Due to New Fruit Assigned to Old Fruit Position	Estimated Count
192 100%	3 1.5%	1 0.5%	15 7.8%	19 9.9%	-14 -7.3%	197 102.6%

It is important to retain from this project how the usage of both KF and the Hungarian algorithm worked together to improve the efficiency of a tracking algorithm for the tracking of fruit.

3.3 Using Synthetic Data for Fruit Counting

This work[42] aims to analyze the performance and accuracy of a neural network to count fruit trained entirely on synthetic data. One of the objectives was to reduce the overhead of labelling the training samples for the object counting problem by creating a synthetic

dataset for training. This allows for a high quantity of training samples with minimum effort. The training parameters were then tested on real images.

Synthetic images were generated by a blank image with a size of 128 x 128 pixels and then filled entirely with green and brown coloured circles to simulate the background and the tomato plant, which are later blurred by a Gaussian filter. The next step was to create variable-sized tomatoes in the image by drawing several circles of random sizes in random positions on the image. For training, twenty-four thousand images were generated, whereas for testing two thousand and four hundred images were generated. The synthetic tomato images were generated with some degree of overlap along with variation in size, scale, and brightness in order to incorporate the possible complexities in real tomato images [42].

Regarding the network used in this project, a novel deep-learning architecture for counting fruits based on CNNs and a modified version of Inception-ResNet are presented. A convolutional layer attempts to learn to filter two spatial dimensions and a channel dimension in a 3D space, simultaneously. The Inception model makes this process easier and, therefore, it empirically appears to be capable of learning richer representations with fewer parameters. The Inception model would independently look at cross-channel correlations and at spatial correlations [42]. Considering that the size of the objects in the images varies, an architecture that can capture features at multiple scales is required. To do this, the Inception-ResNet-A [54] layer is modified. Inception Res-Net combines the ideas of Inceptions, which capture features at multiple sizes by concatenating the results of convolutional layers with different kernel sizes and residual networks [21], which use skip connections to create a simple path for information to flow throughout a neural network. This architecture was used because of its high performance on several image recognition challenges [54]. Two modified Inception-ResNet-A layers follow the normal convolutional layers. The modified layer consists of three parallel layers concatenated into one, in which the result of this concatenation is added to the activations of the previous layer and passed through the rectified linear function. After the modified layers, a modified Inception reduction module is used to simultaneously reduce the image size and expand the number of filters [42].

Deep neural networks with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks, since they are slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural networks at test time. A technique for addressing this problem is Dropout. The key idea is to randomly drop units (along with their connections) from the neural network during training [24]. Sixty-five per cent of connections were randomly kept while training the network [42].

The network was trained for three epochs of twenty-four thousand synthetic images and to minimize the error, an ADAM optimizer was used [24]. The learning rate for the ADAM optimizer was set at a constant 1×10^{-3}. For the cost function, the mean squared error was used [42].

The experimental results with synthetic data (2400 synthetic images) produced a mean squared error for the count of about 1.16. The network was trained with three different dropout values (50%, 65% and 80%) to find the lowest value for the mean square error. Considering this, the 65% value for the dropout was chosen and it was noticeable that the network converges quickly, hence why the network was trained only for three epochs. The results were compared with several methods, namely area-based technique, shallow neural network, and the network with the original Inception-ResNet [42].

The results are displayed in Table 3.5. Area-based techniques calculate the number of fruits based on the total area of fruits and individual fruit. The average accuracy of over one hundred images for this method is 66.16%. The shallow network consists of two convolutional layers and two fully-connected layers. The average accuracy for this method is 11.60%. In the third method, the original Inception-ResNet-A module was used instead of the proposed modified version. The average accuracy for this method is 76.00%. The proposed method by this paper resulted in 91.03% average accuracy. From the values described above and shown by Table 3.5, it is inferred that the proposed method is significantly better than the area-based method, since the area-based method is not scale invariant, and that occlusion by other tomatoes, foliage or branches will lead to a false count of the tomatoes [42].

Table 3.5: Average accuracy over 100 images [42].

Method	Average Accuracy (%)
Proposed Method	91.03
Area-based counting	66.16
Shallow network	11.60
The network with the original Inception-ResNet-A	76.00

3.4 Fully Convolutional Network for Fruit Counting

In the following work[29], a method for fruit counting is presented that combines deep segmentation, frame-to-frame tracking, and 3D localization to accurately count visible fruits across a sequence of images. First, a Fully Convolutional Network (FCN) is trained for the segmentation of video frame images into fruit and non-fruit pixels. Then, the fruits are tracked across frames using the Hungarian Algorithm where the objective cost is determined through a Kalman Filter (KF) corrected using the Kanade Lucas Tomasi (KLT) feature tracker. To correct the estimated count from the tracking process, the tracking results are combined with a Structure from Motion (SfM) algorithm to calculate relative 3D locations and size estimates to reject outlier and double-counted fruit tracks [29].

This network (FCN) differs from the standard convolution network (CNN) because it does not utilize any fully connected layers. Using only convolutional or deconvolutional

layers allows the network to perform segmentation tasks instead of classification or regression [29].

To keep track of fruit between image frames, it is necessary to assign the fruit detected in one frame to the next one. This can be viewed as a graph problem in which each edge would have an associated cost to it. To solve such problems, the Hungarian algorithm is the standard algorithm to use. In this research, the cost function is based on a KF-corrected optical flow estimate. The optical flow tracker used is the KLT feature tracker. The optical flow step uses FCN segmentation in the current image frame, while in the next frames, the FCN segmentation is only introduced in the Hungarian cost function. Using the KLT algorithm alone will sometimes yield a noisy flow where different fruits can be predicted to move in very different directions. This does not happen in most common cases, since the directions of optical flow should be similar between fruits. This problem is addressed using two techniques. First, the usage of KF to correct the optical flow estimates. Second, the usage of the velocity of every fruit as an initial guess of its optical flow for the KLT tracker. In this way, the cost function is built to consider the KF estimates.

The authors of this method still believe that even though it is possible to have counting of fruit, this count is still susceptible to double counting. To further correct this count, the fruit is localized in 3D and the relative sizes are estimated. The 3D reconstruction is obtained using an incremental Structure from Motion (SfM) algorithm. This algorithm matches features across image frames to estimate camera poses. A usual algorithm used to recognize features in images is Scale Invariant Feature Transform (SIFT) [31]. However, only SIFT features located close to the detected fruit regions are tracked. This reduction is done because tree environments are highly textured so that only the features close to the objects of interest are tracked, reducing computation time and improving the quality of the 3D reconstruction. After the localization of these fruits in 3D, it is possible to detect double-counted fruits by understanding that these fruits are close to one another within the 3D space.

Table 3.6 displays the count error metrics for both the orange and apple datasets. For the orange dataset, the uncorrected model has an L1 loss of 593, an error mean of 17.1%, and a standard deviation of 26.3%. After the 3D reconstruction correction step, the model has an L1 loss of 203, an error mean of -0.2% and a standard deviation of 7.8%, demonstrating a large performance gain in both accuracy and consistency. For the apple dataset, the uncorrected model has an L1 error of 672, an error mean of 8.5% and standard deviation of 4.7%. After the correction step, the model has an L1 error of 322, an error mean of 3.3% and a standard deviation of 4.1%.

3.5 Discussion

In the first section of this chapter, Section 3.1, a research project using the YOLO architecture and object detection model combined with the KF algorithm was presented. The authors also tested their model with and without data augmentation performed on the training

Table 3.6: *Error mean and standard deviation of the algorithm on the orange and apple datasets:* The correction from the 3D fruit localization significantly improves the algorithm performance by reducing both the error mean and standard deviation. The correction is effective, especially for the orange data set, in which the environment is not as carefully controlled, and the fruits have more variation in depth [29].

Measure	Oranges (Uncorrected)	Oranges (Corrected)	Apples (Uncorrected)	Apples (Corrected)
Count/Ground Truth	4049 / 3456	3449 / 3456	8622 / 7949	8215 / 7949
L1 Loss	593	203	673	322
Error Mean	17.2%	-0.2%	8.5%	3.3%
Error Std Dev	26.3%	7.8%	4.7%	4.1%

dataset. The model without data augmentation presented a precision of 0.95 against approximately 0.98 precision obtained with data augmentation for pear counting. These values reinforce the importance of data augmentation since it allowed an increase of around 3% in precision. This project also shows the importance of the KF since it allowed only four pears to be double counted, meaning that the fruit was being tracked relatively well.

Secondly, we analyzed a project for mango fruit counting in videos in Section 3.2. This project used an YOLO version created for detecting mangoes in combination with the KF and Hungarian algorithms. The proposed method presented an error for the estimated count of 2.6%, which is reasonably small, proving and reinforcing how the KF and Hungarian algorithms can improve the efficiency and accuracy of a model for object tracking in videos.

After this, we analyzed a reasonably different approach in Section 3.3. This project used a novel architecture based on CNNs and a modified version of Inception-ResNet trained by synthetic data for fruit counting. Since no real images of fruit were used for the training, it presented good results of 91% accuracy, showing that using synthetic data for training is a viable option. However, this should not be applied alone but rather in combination with authentic images. This aggregation between synthetic and authentic images might be helpful since there are various fruits with a green tone, where the construction of synthetic data alone may be more prone to error.

Finally, we analyzed a project for fruit counting in Section 3.4, using a FCN in combination with the KF, the Hungarian algorithm, and KLT tracker. It was stated that using the KLT alone may lead to a noisy flow when the fruit is moving in different directions. Since this was not the case, the KLT is a great tool to use in problems of fruit counting. To further increase accuracy in fruit counting, this project also located fruit by extracting features with a SIFT algorithm and 3D reconstruction with a SfM algorithm. The importance of using these methods was reinforced by an error of 0.2% in fruit counting of oranges and an error of 3.3% for apple counting, presenting impressive results.

4

Implementation

4.1 Materials and Methods

Various materials and methods were requisite to procure and define in the conduct of this research. This process included gathering video and images for model training and evaluation and employing necessary computer hardware and software tools. The latter proved indispensable for tasks such as data augmentation and image labelling. For the preliminary field study, videos were meticulously recorded from apple orchards near Alguber, in Cadaval, Portugal. The principal objective of this endeavour was to procure initial datasets pivotal for project initiation.

The recording involved a combination of devices: the GoPro 10, the GoPro 11, and a smartphone camera. This assortment was chosen to ensure diverse videos possessing distinct configurations and dimensions. The GoPro cameras captured footage of the apple orchards at a resolution of 3840 x 2160 pixels and a frame rate of 60 frames per second. In contrast, the smartphone recorded videos at 1920 x 1080 pixels and 30 frames per second. The GoPro cameras also captured auxiliary metrics encompassing GPS coordinates, altitude, prevailing weather conditions, velocity, and other pertinent data. These supplementary data points are slated for utilization in tracking and georeferencing fruits during the project's development phase.

The video capture scenarios encompassed diverse scenarios: some were shot from within a moving car, which maintained an approximate velocity of ten kilometres per hour, while others were taken on foot, involving a traversal alongside the orchard trees. Notably, the smartphone's capabilities were limited to recording with a linear lens, resulting in the acquisition of three videos from a moving car and an equivalent number while walking. In contrast, the GoPro cameras facilitated recording with both linear and wide lenses. The camera was directly pointing at the trees throughout most of the videos. However, some instances deviated from this norm, capturing videos at a 45º angle between the camera and trees, thereby introducing variability into the dataset and facilitating analysis of potential angle-induced distinctions.

Table 4.1 presents all the information of the videos recorded in Portuguese Orchards.

Table 4.1: Recorded videos configurations.

Camera	Lens	Movement	Angle	Nr of Videos	Total Duration
GoPro	Wide	Car	0º	4	24 min
GoPro	Wide	Walking	0º	13	34 min
GoPro	Wide	Walking	45º	4	8 min
GoPro	Linear	Walking	0º	4	8 min
GoPro	Linear	Car	0º	2	16 min
Smartphone	Linear	Car	0º	1	20 min
Smartphone	Linear	Car	45º	2	12 min
Smartphone	Linear	Walking	0º	3	11 min

GoPro cameras offer the advantage of utilising a wide lens and capturing telemetry data during video recording. In light of this capability, supplementary software was indispensable for the post-processing phase. A notable tool in this regard is Gyroflow[1], which facilitates the editing and post-processing of videos recorded with GoPro cameras. This software is adept at rectifying distortions inherent to wide lenses, correcting deformations that may arise from their usage, including the typical "fish-eye" effect. Gyroflow also proves instrumental in discerning key configurations within GoPro recordings, such as the camera's recording axis, pivotal for the subsequent georeferencing of fruit within video footage. Complementary to this, Telemetry[2] is a software solution that adeptly extracts data recorded by GoPro cameras, subsequently organising it into Excel files. This resource proves invaluable for the georeferencing process.

Albumentations[3], a Python library, emerges as a vital asset within computational tools. Tailored to computer vision, Albumentations facilitates swift and adaptable image augmentations. This capability significantly enhances the training models' performance and generalisation capacity, thus fostering the training process across a more diverse spectrum of data.

After these preparations, the need arose for proficient image labelling tools. Enter LabelImg[4], a Python library selected for its rapid and efficient image-labeling capabilities. This utility expedites the acquisition of authentic and meticulously annotated data, which is essential for training datasets. The choice of LabelImg is attributed to its user-friendly installation, ease of use, and compatibility with annotation configurations specifically tailored for YOLO (You Only Look Once) models.

Turning to the research methodology, a multi-faceted approach was adopted. The initial step involved constructing diverse datasets to comprehensively understand how the training dataset influences the object detection model's performance. Subsequently, model evaluation encompassed utilising videos and images independent of the training process. This methodology aimed to derive a realistic assessment of the model's performance. More

[1]https://gyroflow.xyz/
[2]https://goprotelemetryextractor.com/
[3]https://albumentations.ai/
[4]https://pypi.org/project/labelImg/

in-depth, the test set in the agglomeration dataset (seen in more detail in Section 4.2.4) was used as an additional validation set to understand the performance of the model and select the confidence threshold that better generalises the data. The real evaluation of the model will be performed on real-life scenarios using images and videos of Portuguese apple orchards. However, it is important to keep in mind that the confidence threshold selection is a guideline for the general use case of the model, meaning that in some specific orchards, a different confidence threshold might be more appropriate.

Different videos were employed (for evaluation), each characterised by distinct attributes such as lighting conditions, camera orientation, velocity, and fruit density on trees. This comprehensive range of scenarios was leveraged to discern optimal conditions that yielded peak accuracy and to appraise the model's performance across varied real-world contexts.

As delineated more comprehensively in Section 4.3.1 and substantiated by the results detailed in Section 5.1, the training approach adopted encompassed a phased strategy. Initially, the model underwent training based on a small dataset outlined in Section 4.2. Subsequently, this pre-trained model was deployed to facilitate the training process anew, leveraging the cumulative amalgamation of the first, second, and third datasets. This strategic two-step training methodology aimed to enhance the model's performance and accuracy. The initial dataset served as a preliminary gauge of the model's proficiency, while the subsequent datasets were designed to further refine its capabilities. The first dataset was included as an integral component within the merged dataset to ensure continuity of knowledge and avoid erasing prior insights.

In the realm of georeferencing fruit, the method's efficacy was juxtaposed with the calculated positional data provided by the software. This comparison was drawn against the actual fruit positions in the video footage, providing insights into the method's accuracy and reliability.

4.2 Dataset

The acquisition of a robust dataset is paramount when employing Deep Learning (DL) for various tasks, as it profoundly influences the quality of outcomes. Nonetheless, procuring a high-quality dataset remains an intricate challenge due to the substantial data appetite of DL models and their susceptibility to variances induced by training data quality.

Throughout this research endeavour, multiple datasets were employed to assess the model's efficacy. These datasets varied in size, encompassing both smaller and larger collections. Notably, all datasets used in this study exclusively featured images of apples. Selective data augmentation techniques were applied to enhance the training data volume and subsequently optimize model performance.

4.2.1 First Dataset

The first dataset utilized in this research primarily comprised images drawn from publicly accessible datasets designated for research purposes, generously provided by ROBOFLOW. Since this dataset already incorporated data augmentation, additional measures were deemed unnecessary.

The initial dataset was an amalgamation of diverse datasets sourced from ROBOFLOW. Each dataset included only images featuring apples. The amalgamation process necessitated adjustments to annotation files to adhere to the specifications of YOLO, the selected model framework. As a result of the distinct origins of the constituent datasets, a comprehensive fusion was essential. Several of these datasets featured augmented data, encompassing image rotations and manipulations in saturation and hue to simulate varied lighting conditions. In total, 1623 annotated images were incorporated.

The aggregated annotated images originated from the following five datasets, all accessible within Roboflow's repository:

- Appledetector Computer Vision Project: 68 annotated images of apples [5].
- dataset3 Computer Vision Project: 670 annotated images of apples in orchards [17].
- Apple yolo5 Computer Vision Project: 177 annotated images of apples in orchards [28].
- full indoors apple Computer Vision Project: 544 annotated images of apples from indoor trees [47].
- apple yolo5 2 Computer Vision Project: 164 annotated images of apples in orchards [27].

All images from these diverse datasets were aggregated into a singular folder to facilitate uniformity and compatibility with the chosen model. These datasets' images were 640x640 pixels, which is an image size compatible with the YOLO model.

The initial dataset exhibited commendable results in object detection and apple counting within orchards. However, due to its relatively limited size, measures were undertaken to enhance its scope and efficacy.

4.2.2 Second Dataset

To augment the authenticity of the dataset, images were annotated from videos obtained within Portuguese orchards, as delineated in Section 4.1.

Subsequently, several videos recorded in Portuguese apple orchards were systematically divided into frames, with a frame skip of 80 frames. This strategic approach aimed to procure diverse images, thereby introducing variability in apple positions. This yielded a total of over 400 authentic images from the video footage. Yet, subsequent data analysis warranted the exclusion of 23 images, rendering 377 fully annotated and suitable images

for utilization. The encompassing array of videos, each characterized by distinct attributes and configurations, ensured a substantial diversity essential for bolstering the model's performance for genuine and real-world scenarios. These videos spanned a spectrum, ranging from those exhibiting optimal lighting conditions (the ideal scenario) to those marked by suboptimal lighting (occasions where sunlight direction cast shadows on fruits), as well as videos encompassing images captured with a wide lens, both with and without lens correction.

Moreover, images featuring varying camera angles with the trees were included. Both pedestrian and vehicular perspectives were leveraged to incorporate images captured while walking and inside a moving car. This deliberate assortment aimed to capture scenarios where higher velocity might introduce distortions, juxtaposed with instances of minimal deformation due to slower movement.

The images drawn from GoPro videos exhibited a resolution of 3840x2160 pixels, while those extracted from the Android smartphone carried a resolution of 800x1300 pixels.

The distinct attributes of the videos translated to specific image counts:

- One hundred five (105) images were obtained under sunlight, characterizing situations where the camera filmed directly against the sunlight, resulting in shadows on fruits.

- Sixty-three (63) images captured instances where clouds obscured the sun, leading to darker tones due to diminished sunlight.

- One hundred thirteen (113) images were derived from videos with optimal light conditions, yielding well-lit images.

All the images mentioned above were sourced from GoPro cameras employing an ultra-wide lens. Then, the "fish-eye" effect was stabilized using Gyroflow except for images taken against sunlight, which lacked the ultra-wide lens effect stabilization.

In the context of smartphone-recorded videos, ninety-six (96) images were extracted from videos shot at a 45º angle between the camera and the orchard trees. This selection was informed by the smartphone's limitations in capturing the entire tree or a cluster of fruits simultaneously when filming directly towards the trees.

The resultant images and their respective annotations were systematically organized into distinct folders and partitioned into training, validation, and test sets. This division process was executed randomly, allocating roughly 80% of the images for training, 10% for validation, and another 10% for testing. This allocation was strategically chosen to maximize the training data pool while preserving a suitable portion for validation and testing.

4.2.3 Data Augmentation

Finally, the images belonging to the GoPro videos were subject to data augmentation to further increase the number of training images but also to artificially change the configuration of the image, increasing overall generalization to unseen data of the model. Each image was subject to three different data augmentation techniques, originating three more images while including the original image in the dataset. The Data Augmentation methods were performed using the Python library Albumentations referenced in 4.1.

The first data augmentation method used a combination of adding a random Gaussian Noise to the image with values between 1500 and 3500 and a random image rotation with a maximum 45º angle with a 50% probability. The addition of Gaussian Noise to the images was a data augmentation technique chosen due to its different applications in the AI research area, such as image classification, object detection, speech recognition and generative models, since it allows training models to more diverse images. The values chosen for the Gaussian Noise method (between 1500 and 3500) were obtained through experimentation and analysis to find the values that would change the picture enough to be able to achieve noticeable changes but also to easily identify the fruit while making sure that the maximum noise value would not make it impossible for a human to identify the fruit in trees with the naked eye. In this first data augmentation technique, a rotation of the image with a 50% probability was used to simulate some errors that might occur, for example, tilting of the camera while filming the orchards. The 50% probability of this rotation was also chosen to make it possible to have at least enough images subject only to Gaussian Noise.

The second data augmentation method used was a combination of a random change in Hue, Saturation and Value of colours in the image with a horizontal flip with 50% probability. The change in the Hue, Saturation and Value method was used to allow the model to learn different types of conditions and colours when considering fruit in orchards. Around the world, there is a vast and diverse production of different types of apple trees (also happens with other fruit), and they can change colours and have different tones of green and red. This will help the model improve performance on inference to different conditions and kinds of fruit. The maximum values chosen for Hue, Saturation and Value were 60, 30, and 40, respectively. These values were chosen based on experimentation in order to change the image to allow the model to identify the apple but also to not make drastic changes to obtain unrealistic cases. The Horizontal flip of the image with 50% probability was chosen to allow the model to adapt to different conditions, such as lighting and shadows, adding more diverse and robust data. 50% probability in this case was chosen to make variety between the light and shadow conditions proposed by this method.

The third data augmentation method was a combination of a random rotation between -60º and 60º with a random change in brightness of the image limited to 0.4 with a 50% probability. Same as the first data augmentation method, this rotation was chosen to

simulate a variety of different angles of data, allowing the model to better detect fruit and also simulate tilting of the recording camera. The change in brightness was used to simulate different lighting conditions during recording based on season, country and weather. The 50% probability was chosen to also obtain original pictures, which are only subject to rotation.

After data augmentation methods, the second dataset comprised 1309 images for the training set, 209 for the validation set, and 79 for the testing set. Both the validation and test sets contained some images from the validation and test sets of the first dataset mentioned along with labelled images from Portugal orchards to evaluate the model to more generalised data.

4.2.4 Third & Aggregation Datasets

The third Dataset built was an aggregation of five different datasets from ROBOFLOW. In total, these five datasets result in around eleven thousand images. These datasets were chosen because they only contained object detection annotation images of apples, had a high number of images of apple orchards and apple trees and had annotations for yolo models.

The five datasets are comprised as follows:

- Apple Computer Vision Project [58]: 1790 annotated images of apples in orchards. These images were divided into 68% for the training set, 1211 images, and 32% for the testing set, 579 images. This dataset was chosen for this project for having a high variety of images, such as side view of apple orchard trees, an up view of apple orchard trees using a drone, and a close view of apples. The high variety and quality of images will allow the model to be more generalized and identify apples in different environments. The images were not data augmented, which is a good sign due to a high quantity of raw images, in which data augmentation can be performed later to further increase data.

- Apple Computer Vision Project [19]: 2287 annotated images of apples in orchards. These images were stretched to 416x416 pixels and divided into 70% for the training set, 1601 images, 20% for the validation set, 457 images, and the remaining 10% for the testing set, 229 images. The dataset was chosen because it has a high quantity of side-view images of apple orchards. The images included a vast amount of apples in each tree and also had different lighting conditions. Some pictures were taken in bad lighting conditions, darkening their tone, while others had normal lighting conditions. The low lighting conditions present in the pictures of this dataset will help the model train to detect apples on cloudy days or even detect fruits that are in the shadow of leaves. The images present in this dataset were not subjected to data augmentation, which helps the model to train in raw data.

- Apples Computer Vision Project [6]: 154 annotated images of apples in orchards. These images suffered no data augmentation nor preprocessing method and were divided into 69% for the training set, 106 images, 19% for the validation set, 29 images, and the remaining 12% for the testing set, 19 images. Although the dataset is smaller than the other datasets of this aggregation stated so far, the dataset continues to be relevant for this research project. The images present in this dataset consist only of apple fruit in orchards with a high variety of angles. Each image has a high amount of fruit per tree which ends up compensating for the low amount of pictures.

- DS-PG2-Apples Computer Vision Project [32]: 4293 annotated images of apples in orchards. These images suffered a preprocessing resize to 640x640 pixels resolution. Data augmentation was also performed on this set of pictures in which the methods consisted of random crop between 0% minimum zoom and 20% maximum zoom, random rotation between -15º and +15º, random saturation changes between -40% and +40%, random brightness changes between -25% and +25%, random change in exposure between -15% and +15% and finally random blur was applied to a maximum of 1px. This dataset was chosen due to the high quantity and quality of images. Images were also divided into 87% for the training set, 3756 images, 8% for the validation set, 358 images, and 4% for the testing set, 179 images. All images include apple fruit in orchard trees from different angles and conditions, which are intended for the counting and georeferencing of fruit in orchards. Data augmentation performed in this dataset is also a benefit since it can induce more artificial variety and help the model generalise well to unseen data.

- DS-PG7-MinneApple Image Dataset [33]: 2400 annotated images of apples in orchards. The images present in this dataset suffered a preprocessing resize to 640x640 pixels resolution. Data augmentation was also performed on this set of pictures in which the methods consisted of random crop between 0% minimum zoom and 20% maximum zoom, random rotation between -15º and +15º, random saturation changes between -40% and +40%, random brightness changes between -25% and +25%, random change in exposure between -15% and +15% and finally random blur was applied to a maximum of 1px (equal data augmentation applied to the previous dataset). The images are divided into 88% for the training set, 2100 images, 8% for the validation set, 200 images, and 4% for the testing set, 100 images. The images present a vertical view, mostly common in pictures taken with smartphones, in which the side edges of the pictures are black to fit the square resolution of 640x640 pixels. This vertical view will allow the model to improve performance in detecting fruit in videos and photographs taken with smartphones, allowing agricultural productions to have a model that performs well in these circumstances in cases where high-resolution cameras and vehicles are not accessible.

This aggregation of datasets altogether had a total of 10924 images divided by 8774

images for the training set, 1044 for the validation set and 1106 for the testing set. The higher number of images in the testing set compared to the validation set is due to the first dataset aggregated having no images for the validation set.

In order to have the models trained on as much data as possible, the first, second, and third datasets mentioned previously were all grouped into a single dataset. Furthermore, the images that had no data augmentation performed were selected randomly, and the same data augmentation methods as the ones performed on the second dataset referred to in 4.2.3 were applied. The aggregation of the three datasets plus the additional data augmentation resulted in 12178 images for the training set, 1105 for the validation set, and 1168 for the test set.

4.2.5 Apple & Tree Dataset

One of the main questions that arose throughout this research project was the possibility of counting how many fruits per tree an orchard can have. In order to find a solution for this, the model was trained to detect not only fruit in videos but also trees. In this way, it would be possible to achieve an estimate of how many fruits per tree exist. For this reason, the second dataset mentioned previously was copied, and all the images were annotated once again but maintaining the apple annotations, only adding the tree annotations. Besides this, some images were also selected from the other datasets to be included for this specific case. The reason that only the dataset with the videos that were personally recorded and some images from the other datasets were used was that only those pictures added a clear view over a tree. Since the videos recorded consisted of trees side by side in an orchard, only these images were able to be used for this unique dataset. Although the other dataset contained a lot of fruit in orchard trees, most images lacked a broad and clear view of each individual tree. The total number of images present in this tree dataset was 402 images for training, 156 images for validation, and 55 images for testing. No data augmentation was performed on this dataset to evaluate the model on raw images.

4.3 Fruit Detection and Counting

The realm of computer vision has undergone remarkable advancements in recent years. One of the pivotal domains within computer vision is object detection. The usage of object detection in combination with computer vision algorithms provided a solid ground for the execution of this study.

Counting objects within videos presents a unique set of challenges. Videos encompass temporal and spatial variations, often featuring dynamic backgrounds, occlusions, varying lighting conditions, and object interactions. The accurate counting of objects requires a robust detection algorithm capable of addressing these complexities. Object detection

models have demonstrated exceptional competence in surmounting these challenges, such as those rooted in the YOLO framework.

The YOLO architecture introduced a paradigm shift in object detection, revolutionizing the field with its real-time processing capability. YOLO models process images or frames in a single forward pass, enabling rapid object localization and classification. This efficiency is crucial for real-time or near-real-time counting applications [12].

The significance of object detection extends beyond counting. In scenarios where objects are in motion, the ability to track their trajectories across frames adds a layer of understanding. Multi-object tracking algorithms built upon object detection results enable the reconstruction of object paths, yielding insights into movement patterns, interactions, behaviours and better fruit counting.

Considering this, the YOLO model was chosen to perform object detection of fruit in tree orchards. Two YOLO models were considered for this research project, the YOLOv7 and YOLOv8. These models were chosen for their breakthroughs in object detection performance and accuracy but also due to the lack of utilization in this area since these were recently released.

These two models were both trained under the same conditions and training data.

Additionally, a YOLOv7 model was also trained for object detection and tree detection. This variation was developed for agricultural producers to better understand and manage their crops, making it possible to understand which trees grow more or less fruit and relating this information to the amount of resources applied to each tree.

Considering the training of the models for object detection in videos, a few things need to be considered when choosing the image size. Typically, it is ideal for the model to be trained on an image size that will most often be used. If agricultural producers use full HD videos (1920x1080), the models should be trained with this resolution to achieve the best performance possible. Furthermore, all the training images should be the same size and equal to the size of the videos used for the task. However, this goal becomes more challenging due to the different datasets used for training. Firstly, we want a versatile model that can perform well in various scenarios rather than exceptionally well for a specific case while performing worse in other conditions. Different agricultural producers may use different types of cameras, resulting in a variety in the quality of the recorded videos. On the other side, it is also crucial for the model to perform relatively well in smaller video sizes since, with bigger video sizes, the amount of time to analyse a video also increases. Overall, we want a model that provides results no matter the situation. It is also essential to consider that training images should not be resized to a smaller size to avoid loss in image quality.

On the other hand, the increase in size of the training images will also severely increase the training time and computational resources necessary. The image sizes of the datasets were analysed to decide the appropriate size for most of our training models. All the images in the training set comprised resolutions between 416x416, 640x640, 720x1280, 1280x720, 1080x1980, 1980x1080, 2880x1620, and 3840x2160 pixels. Although this dataset

is extensive, it also has many different sizes, making it difficult to decide which image size is more beneficial. One thing that might be useful is that the YOLO models maintain the image's aspect ratio and do not resize when necessary.

With this idea in mind, we need a compromise between images not losing too much quality and lower training times. Considering all these options and resources, the image size in the following training models was 1080x1080 pixels, which would then be converted to 1088x1088 pixels since the YOLO models only accept images with width and height multiple to 32. Although this value would reduce relatively the quality of very high-resolution images, it also presented good results in performance and speed, as seen in the following sections and chapter.

4.3.1 YOLOv7 Training

The standard YOLOv7 model comprises over 30 million parameters and may be considered a medium size model. More extensive models require more computational resources, which may not be accessible to most agricultural producers. With this in mind, the medium size model suits the task well.

The model's training during this research project was divided into two sections. Firstly, the model of YOLOv7 was trained on the first dataset built, as referred to in Section 4.2.1. This first training set with around 1600 training images originates an excellent first model for usage in object detections in the context of fruit counting, more specifically in apple fruit counting. Secondly, the model already trained was then used as a pre-trained model to continue training, but this time using the aggregation of all acquired images as referred to in Section 4.2.4. This second set of training allows for a more in-depth evaluation of the model, using the most training data acquired, a dataset with over 14000 images. Furthermore, several other models were trained to test and compare different approaches and discover which model would perform best for this task.

In Table 4.2, the parameters used for the training of YOLOv7 models are described.

All the mentioned command parameters in Table 4.2 for training the model of YOLOv7 comprise the commands that seemed more relevant for evaluating how the training model could improve performance and accuracy in real situations. There were a few more commands that could be passed in training that should have been mentioned because they didn't result in a significant change in model training ability, such as selection of the GPU (only influences training speed) and changing project name, among others.

Table 4.3 shows the models and their training configurations. This table includes the dataset used, either the First dataset, referred in Section 4.2.1, the Second dataset, referred in Section 4.2.2, the agglomeration dataset, referred in Section 4.2.4, and the Apple & Tree dataset, referred in Section 4.2.5. In addition, this table also shows which initial weights were used (either a pre-trained model or not) and other configurations such as image size, additional parameters and number of epochs.

The initial hyperparameters used for all models were kept as standard with YOLOv7.

Table 4.2: Description of available parameters for training YOLOv7

Parameter	Description
–weights	Path for the model weights file, either a standard version or pre-trained model.
–cfg	Path to the model configuration file, which is needed for loading the model architecture. It also defines how many classes the model should train on.
–data	Path for a YAML file, which includes the directories for the datasets, which are training, validation and test sets.
–hyp	Path for YAML file containing the information for the hyper-parameters of the training model.
–epochs	Integer indicating the number of epochs the model should train for.
–batch-size	Integer indicating the batch-size for each epoch.
–img-size	Tuple of integers indicating the image size for training and validation set. Images are resized to a square shape with width and height given by this parameter while maintaining the aspect ratio.
–rect	Boolean indicating if images should be forced to be rectangular, meaning that the aspect ratio will not be maintained.
–evolve	Boolean parameter indicating the training is performed with evolving hyperparameters, that during the model's training, the hyperparameters are subject to changes by an optimization algorithm defining the best hyperparameters to improve performance and disregarding the need for manual hyper-parameters tuning.
–multi-scale	Boolean parameter indicating that the training images should have a varying image size. Each image will be assigned between +/-50% of its original training size randomly.
–adam	Boolean parameter that indicates the usage of ADAM optimizer instead of the default SGD.
–linear-lr	Boolean parameter indicating the use of a linear learning rate decay during training.

The principal hyperparameters were: 0.01 for the initial Learning Rate and 0.1 for the final Learning Rate, which means that the final learning rate of the training process will be the multiplication of the initial learning rate with the value chosen for the final learning rate, obtaining $0.01 * 0.1$; 0.937 for momentum; 0.0005 for weight decay; 3 for warmup epochs; 0.8 for warmup momentum; 0.1 for warmup bias learning rate. The standard optimiser used in YOLOv7 model is SGD.

The Yolov7-custom12 and Exp23 models were used to study how well the models could detect fruit until a more extensive dataset was built. The Exp25 and Exp26 were used to analyse how using a pre-trained model might affect performance. The rest of the models (Exp, Exp42, Exp43, and Evolve) were compared with Exp25 to study how changing training parameters might improve or decrease the model's performance. Considering the

Exp, instead of using SGD, ADAM was used, and consequently, the momentum parameter was changed to beta1 while maintaining the 0.937 value. The initial learning rate was also changed from 0.01 to 0.001.

The Exp42 model had the multi-scale parameter active, aiming to compare the model's performance to more varied image sizes. As such, the image size would vary randomly between 320x320 and 980x980 pixels. The Evolve model has the evolve parameter active, which enables the model to be subject to a hyperparameter tuning algorithm and compare the results to a more standardised solution. On the other hand, Evolve2 has both evolve and rect parameters active, allowing the study of how not maintaining the aspect ratio of the images might affect the performance and accuracy of the model.

The Exp-tree model was built to resolve the Tree detection problem, which is further explained in Section 4.3.3.

Table 4.3: YOLOv7 models trained characteristics.

Name	**Dataset**	**Weights**	**Image size**	**Additional Parameters**	**Epochs**
Yolov7-custom12	First	None	640x640	None	100
Exp23	Second	Yolov7-custom12	1088x1088	None	100
Exp25	Aggregation	Yolov7-custom12	1088x1088	None	75
Exp26	Aggregation	None	1088x1088	None	90
Exp	Aggregation	Yolov7-custom12	1088x1088	adam	100
Exp42	Aggregation	Yolov7-custom12	640x640	multi-scale	80
Exp43	Aggregation	Yolov7-custom12	1088x1088	linear-lr	100
Evolve	Aggregation	Yolov7-custom12	1088x1088	evolve	100
Evolve2	Aggregation	Yolov7-custom12	1088x1088	evolve & rect	100
Exp-tree	Apple & Tree	None	1088x1088	None	125

It is essential to consider that some manual hyperparameter modifications were performed as an initial step for model training. These modifications included different learning rates (initial and final), changes in the momentum parameter, and weight decay. However, these changes often showed inconsistencies, which were then disregarded, as the standard parameter values provided more consistent results during training.

The training of the model Yolov7-custom12 was performed on a personal computer with an NVIDIA GTX1060Ti graphics card, which, with a batch size of 4 and image size of 640x640 pixels, took around 20 hours to finish training. All the other model training was performed in a computer belonging to the cluster from NOVA School of Science and Technology with the usage of a node with an NVIDIA RTX3070 graphics card, with a batch

size of 2 and image size of 1088x1088 pixels, taking around 30 hours to finish training, except for the model Exp42, which with the varying image size took about 10 hours to complete training. In the case of model Exp23, since the dataset was also smaller, the model took 6 hours to train. The Exp-tree model was also trained on the cluster, but with a batch size of 4 and a smaller dataset, taking around 15 hours to complete training.

When defining the hyperparameters for our training model, it is necessary to include an initial learning rate (lr0) and a final learning rate (lrf). It is important to note that the final learning rate value is not the exact value of the final learning rate of training. Instead, it will be the multiplication of the value of the initial learning rate with the value chosen for the final learning rate. Furthermore, during training, the learning rate will start with the initial learning rate (lr0) value and slowly decay to the value of $lr0 * lrf$. This decay, by default, follows a strategy called cosine decay. This strategy shows that this decay is not linear, hence the experimentation of the parameter –linear-lr in Exp43, which enforces a linear decay between the initial learning rate and the final.

4.3.2 YOLOv8 Training

The standard YOLOv8 medium model, which was utilized in this experiment, comprises around 25.9 million parameters. Although it is smaller than the YOLOv7 model utilized in this research study, it has many features and improvements that may prove a better model fit for the task.

In the same way as the model YOLOv7, the YOLOv8 training was divided into two sections. Firstly, the model was trained on the first dataset with around 1600 images. Secondly, the first model was used as a pre-trained model for the training with the agglomeration dataset.

The number of training models executed with this YOLO version was relatively smaller due to its relatively new development, and many options present in YOLOv7 have yet to be available.

Table 4.4 shows the models and their training configurations, similarly as shown in Section 4.3.1 for YOLOv7. Since the model only detects one class, it was necessary to change the *custom_data.yaml*, which is a configuration file with the path to training, validation and test sets, as well as the number and names of the classes for the model to train on. As such, it is necessary to adapt this file to only one class, and the name of this class was *Apple*. Most hyperparameterization was left by default, and the optimizer was left as *auto*, which means the model will choose it based on its hyperparameters.

The hyperparameters and parameterization were equal to all models: 4 for the batch size; SGD for the optimizer; 0.01 for the initial learning rate; 0.01 for the final learning rate; 0.937 for the momentum; 0.0005 for the weight_decay; 3.0 for the warmup_epochs; 0.8 for the warmup_momentum; 0.1 for the warmup_bias_lr. The initial and final learning rates behave in the same manner as in YOLOv7 so that the actual final learning rate would be the multiplication between the initial and final learning rate values chosen. The Train6

and Train3 models were used for a study of how YOLOv8 performance is affected by the usage of a pre-trained model.

Table 4.4: YOLOv8 models trained characteristics.

Name	Dataset	Weights	Image size	Epochs
Train19	First	None	640x640	100
Train6	Agglomeration	Train19	1088x1088	100
Train3	Agglomeration	None	1088x1088	100

Train19 was performed on a personal computer with an NVIDIA GTX1060Ti graphics card with 16GB of RAM, which with a batch size of 4 and image size of 640x640 pixels, took around 14 hours to finish training. All the other model training was performed in a computer belonging to the cluster from NOVA School of Science and Technology with the usage of a node with an NVIDIA RTX3070 graphics card, with a batch size of 2 and image size of 1088x1088 pixels, taking around 18 hours to finish. The duration of training time of YOLOv8, when compared to YOLOv7, is relatively smaller. These smaller times happen due to the YOLOv8 being slightly smaller and having a slightly smaller inference time.

4.3.3 Tree Detection

As stated in Section 4.2.5, one possible information that would be favourable for fruit producers was to count the amount of fruit per tree. A simple proposed solution was to perform object detection to identify trees in video in the same way as with fruit. Consequently, having each tree and bounding boxes identified, it would be possible to detect and count every fruit close to each tree.

The precise identification of trees would provide more information about not only the counting of fruit but also about orchard management. This additional information would be important for resource management and identification of which trees are abundant in the orchard production and which trees do not produce at all, allowing for more precise agriculture solutions.

Accurate tree identification in the video would allow for better resource management, such as the water and pesticides applied to each tree, by estimating how many fruits a particular tree has and its size. This would help agriculture be more sustainable by optimizing resource utilization, reducing unnecessary waste, and minimizing environmental impact.

For the tree detection training, the Tree Detection dataset referred to in Section 4.2.5 was applied to the YOLOv7 model. This dataset comprises over 400 training images, 150 validation images and 55 testing images. This training was performed using the hyperparameters default of YOLOv7 as stated in Section 4.3.1. The model was trained for 300 epochs and used the optimizer SGD by default. The image size chosen was 1080x1080 pixels, which the model then changed to 1088x1088 to be a multiple of 32. This image size was chosen because most images were high-quality, with 3840x2160 pixels. In order

to have a compromise between image quality and high training speed, the 1080x1080 image size was chosen so that a lower image size would not cause a significant decay in quality, balancing it with the time that each epoch would take in order to minimize training duration.

Overall, the results of this method proved to be underwhelming, in the majority due to the lack of enough training data, as well as the difficulty involved in identifying a tree within an image due to its complexity, change in shape and colours, in addition to the different types of trees, described in Chapter 5.

4.3.4 Additional Programs for Fruit Counting

The YOLOv7 model alone does not support a method for counting objects detected in videos or images by default. Hence, there is a need to use tracking methods to identify objects with unique IDs and track them through consequent frames in a video. This tracking systems are explained more deeply for YOLOv7 in Section 4.4.

Additionally, when we perform the execution of detecting and tracking fruit in the video with the usage of a YOLO model, a folder with a file for every single frame of the video, after program execution, is created. Each line inside one of these files represents an object detected and tracked. Furthermore, this line will include the location and dimensions of the bounding box of the object, the confidence score, the class to which it belongs, and the unique ID number.

With these files with information about all the fruit detected in the video, an easy and fast-forward method to have a count of the different fruit present in the video is to search all files (or only the last few ones) for the fruit with the highest identification number. Since the ID numbers are continuous, the highest number, supposedly, is the last fruit detected and identified. However, after some tests, this is not what exactly happens since, in some early testing, videos of 1 and 2 minutes would have an excessive amount of ID numbers. What is happening is that although the ID numbers are continuous, it doesn't necessarily mean that there are no gaps between them, meaning that if the last fruit ID number is, for example, 50, the ID number 25 might not be identified. This often happens due to the tracking algorithm as they can disregard tracks through association and similarities, which is explained more in-depth in Section 4.4 for both YOLOv7 and YOLOv8 models.

To solve this problem and have a more trustworthy fruit counting, the gaps between ID numbers must be taken into account. This problem is relatively easy to solve, but only after the execution of the detection and tracking of the video. Ideally, it would be better if this counting was made during the inference detection and tracking of the video to bring simplicity and efficiency to future applications of this method. However, trying to count fruit during the video's inference proved to be overcomplicated and slowed down the inference of each frame due to additional computation. Instead, counting fruit after every video frame has been analysed is more efficient and straightforward than during it. In this way, future applications of this project can have a faster inference speed, with every

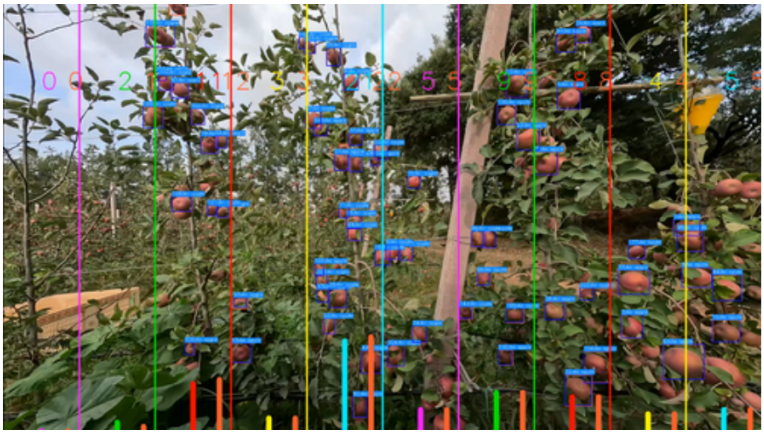


Figure 4.1: Frame of Video Detection model with 10 different zones. Coloured bars and numbers refer to Tracked Detections, and Orange bars and numbers refer to Detections.

data of the objects written in files and a video showcase with the detections of fruit, and then performing the counting of fruit using data files created by the inference program. The solution to this problem was written in an additional Python file, which can be run separately. Two Python functions are defined and seen in Listing 1: a function defined as func3 that goes through all frame files and extracts all the IDs of fruit present in the video, and a second function defined as find-missing that will identify the missing ID numbers from the files. The total number of fruits detected and tracked in the video is obtained by calculating the length of the set with all the track IDs. This result can also be obtained without looking for the missing IDs and simply returning the result. However, this step was performed to confirm that the missing IDs don't actually appear in the video.

To further evaluate the performance of the model in detecting fruit in video, there were some alterations to the *detect_and_track* Python file, which is responsible for the detection and tracking of fruit in video using the YOLOv7 model. The main goal of these alterations was to evaluate if there is an area of the frame (image) being analysed that is easier for the model to detect fruits. Since the camera is moving mostly horizontally, it was decided to divide the image into 10 equal columns from the bottom to the top of each image. At each frame, these columns would be drawn in the frame in the video by adding two bars and two numbers. These bars and numbers for each column identify the number of fruit being detected and the number of fruit being tracked consequently, as shown in Figure 4.1. Drawing in each video frame can be costly, so it is simply an auxiliary method used for representation and visualisation, which should not be used for faster inference times. The realisation of this zone study was only performed with YOLOv7 model since its goal is to better understand which zone of the image frame is easier for an object detection model to identify objects of interest.

Each time a fruit belonged to a specific zone, its counter would increase. The smallest, biggest, and average area of the fruit detected in a specific frame and the total of detections would also be registered. At each frame, every data previously referred to all the fruits is registered in a row of a *Dataframe* from *Pandas* Python library. After the model analyses

```
def find_missing(lst, initial, last):
    start = initial
    end = last
    #List of all the missing tracks
    return sorted(set(range(start,end+1)).difference(lst))

def func3():
    #Path to the folder where the files containing the IDs
    #and bounding box information
    files = os.listdir('labels')

    numbers = list()
    counter = 0

    biggest_number = 0
    smallest_number = 10000000

    n = len(files)

    for file in files:
        f = open('labels/'+file)
        lines = f.readlines()
        for line in lines:
            l = line.split(' ')
            fruit_number = int(l[0])

            if fruit_number > biggest_number:
                biggest_number = fruit_number
            if fruit_number < smallest_number:
                smallest_number = fruit_number
            numbers.append(fruit_number)

        counter += 1
        f.close()

    set_numbers = set(numbers)

    missing = find_missing(set_numbers, smallest_number, biggest_number)

    return missing, len(set_numbers)
```

Listing 1: Python functions that perform a final count of fruit in video

each frame, this *Dataframe* is converted to a table in an *CSV* file for future use in data analysis. It is important to reference that there are two *Dataframes* and two *Excel* files for the detected and tracked fruit, respectively.

Finally, some additional Python programs were created mainly for data visualisation, such as fruit counting and detections by the model. These visualisation methods include more in-depth graphs and tables analysed in Chapter 5.

4.4 Fruit Tracking

Counting fruit within orchards through video analysis presents a multifaceted challenge. Orchards are dynamic ecosystems where fruits are clustered on trees, often in close proximity. When employing computer vision techniques to count fruit, each fruit must be correctly identified and counted, ensuring no duplication or omission.

A recurring issue arises in the absence of tracking mechanisms - the risk of multiple counts for the same fruit in consecutive frames. As the video progresses, fruits can move due to wind, tree sway, or movement from the camera. Consequently, a fruit counted in one frame reappears in the next, leading to an overestimate of the fruit count. This inherent challenge threatens the reliability of fruit counting systems and the accuracy of data-driven decisions.

Tracking in the context of fruit counting is akin to assigning a unique identifier to each fruit and following its trajectory across frames. This crucial process mitigates double-counting risk by ensuring that a fruit, once recognized and tallied, is not erroneously re-counted in subsequent frames.

4.4.1 Yolov7 Tracking

In developing this research dissertation, a project by Munawar called Yolov7-object-tracking [45] was used. This project comprises the integration of the standard YOLOv7 model[56] with the Tracking algorithm SORT developed by Alex Bewley [9]. Adding this tracking algorithm with the object detection model allows the identification of each object in a video with a unique ID number. Besides this, tracking each object detected will remove one of the biggest challenges in this computer vision problem, the fruit duplicates in consecutive frames, making it a beneficial resource in the research.

SORT, which stands for Simple Online and Real-time Tracking, represents a fundamental visual object tracking framework implementation. This framework relies on basic data association and state estimation techniques. Its primary design is centred around online tracking applications with access to only past and current frames. Remarkably, SORT is adept at generating object identities in real-time as the tracking process unfolds. The SORT algorithm's fundamental concept also uses the Kalman Filter (KF) and the Hungarian algorithm to track objects of interest in a frame, analysing past and current frames [9].

It is important to note that SORT, while a valuable tool, has limitations. It does not handle scenarios involving object occlusion or re-entry into the frame. These situations often require more advanced tracking algorithms. However, SORT's primary purpose is not to provide an all-encompassing solution but to serve as a foundational platform for future tracker development [9].

The SORT algorithm allowed for some parameter alterations to fit the task. After experimentation between different small videos, it was possible to conclude that 120 for **max_age**, 0.015 for **iou_thresh** and 2 for **min_hits** produced the best results for dealing with ID-switching present in the system. The max_age parameter defines how many frames the tracker keeps tracking an object of interest. The iou_thresh represents the minimum IoU threshold required between two sequential detections to be assigned the same track. The min_hits is a parameter that counts how many consecutive frames a fruit needs to be detected before it is tracked.

Since counting fruit in orchards is very susceptible to occlusions due to leaves being on top of apples or even apples on top of each other, the SORT tracking algorithm may impose some drawbacks, such as double counting of fruit and misidentification. To solve this problem, the implementation of YOLOv7 with the tracking algorithm DeepSORT, developed by Faisal [18] was used.

DeepSORT is a new, improved version of the SORT tracking algorithm. It helps by mitigating the problem of occlusions and ID switches that may happen in the SORT algorithm. SORT performs very well in terms of tracking precision and accuracy. However, SORT returns tracks with many ID switches and fails in case of occlusion. This is due to the association matrix used. DeepSORT uses a better association metric that combines both motion and appearance descriptors. It can be defined as a tracking algorithm that tracks objects not only based on their velocity and motion but also on the appearance of the object [48].

Same as in the SORT tracker, DeepSORT provides some parameters that can be modified to improve performance. After some experiments, the same values for SORT parameters also worked best on the same DeepSORT parameters. However, an additional change made was to the maximum cosine distance parameter, which was increased to 0.9. This value improved the performance of the tracker in re-assigned tracks after fruit were occluded.

For the above purposes, a well-discriminating feature embedding is trained offline before implementing tracking. The network is trained on a large-scale person re-identification dataset, making it suitable for tracking context. To train the deep association metric model in the DeepSORT, the cosine metric learning approach is used. DeepSORT's paper states, "The cosine distance considers appearance information that is particularly useful to recover identities after long-term occlusions when motion is less discriminative." That means cosine distance is a metric that helps the model recover identities in case of long-term occlusion, and motion estimation also fails. These simple things can make the tracker even more powerful and accurate [48].

4.4.2 Yolov8 Tracking

Yolov8 made tracking a lot easier by coming up with two native trackers, Bytetrack and Bot-SORT. These two trackers comprise new state-of-the-art Multiple Object Trackers (MOTs) that solve many of the previously known problems and challenges of tracking. In this research project, these trackers' innovation in resolving occlusions and ID switch issues in detected objects will help drastically count and georeference fruit in the video. Combining these trackers with the new state-of-the-art model of YOLO models motivates the analysis of the performance and accuracy of these models and trackers in a realistic environment, such as fruit counting for agricultural purposes.

ByteTrack, titled "Multi-Object Tracking by Associating Every Detection Box", was presented at ECCV2022 by Yifu Zhang et al. Due to its versatile framework and inherent simplicity, this approach has garnered attention from subsequent researchers working on Multiple Object Tracking (MOT) solutions, such as Bot-SORT and SMILEtrack.

The core concept underlying ByteTrack is elegantly straightforward: it preserves low-scoring non-background bounding boxes for a secondary association step between the previous and subsequent frames based on their similarity to existing tracklets. This innovation significantly enhances tracking consistency by retaining pertinent bounding boxes that would otherwise be discarded due to low confidence scores, often attributed to occlusion or changes in object appearance. Importantly, ByteTrack's generic framework lends itself well to integration with various object detection techniques, including YOLO and RCNN, and instance association components such as Intersection over Union (IoU) or feature similarity [60].

Following the initial detection stage, the bounding boxes are categorised into high-confidence boxes, low-confidence boxes, and background boxes based on predefined upper and lower confidence score thresholds. While background boxes are appropriately eliminated in this process, high and low-confidence detection boxes are preserved for subsequent association stages [60].

In the second association stage, the low-scoring detection boxes are matched against the remaining unmatched predicted boxes from previous frames. The matching algorithm employed mirrors the first association stage, albeit with a lower matching threshold. This adjustment is based on the rationale that occluded boxes should exhibit poorer matches with boxes from previous frames. In cases where predicted boxes remain unmatched, they are designated as lost tracklets, whereas unmatched detection boxes are disregarded. The lost tracklets are retained for a specified number of frame durations and subsequently reintroduced into the active tracklet pool before undergoing Kalman filter prediction. This mechanism enables the tracker to recover some tracklets that may have been lost temporarily due to objects momentarily disappearing from the frame [60].

Bot-SORT is a new state-of-the-art Multiple Object Tracker that uses SORT-like trackers as a base but addresses their limitations and integrates them into the ByteTrack tracker. The release of this work showed that by adding improvements, such as camera motion

compensation-based features tracker and a suitable Kalman Filter (KF) state vector for better box localisation, tracking-by-detection could be significantly improved [1].

The first modification performed was on the KF. The Kalman Filter is widely used to model an object's motion within an image. Initially, SORT (Simple Online and Real-time Tracking) used a seven-part state vector to represent object properties, including position, scale, and aspect ratio. However, recent trackers have transitioned to an eight-part state vector, including height and aspect-ratio estimation. During experiments, it was found that estimating the object's width and height directly led to better tracking performance than estimation through height and aspect ratio. As a result, the Kalman Filter's state vector was defined differently, focusing on object position (x and y), width and height [1].

The second modification has to do with the compensation of camera motion. Traditional tracking-by-detection trackers rely heavily on the overlap between the predicted tacklets bounding boxes and the detected ones. In situations where the camera is susceptible to movements or motion, the bounding box location in the image plane can shift dramatically, increasing ID switches or False Negatives (FN). However, trackers in static camera scenarios can also be affected due to motion by vibrations or drifts caused by the wind, which, in very crowded scenes, ID switches can be a genuine concern. The solution for this problem follows the global motion compensation (GMC) technique. Firstly, image keypoints are extracted, followed by sparse optical flow for the feature tracking with translation-based local outlier rejection. In more detail, this means that for each consecutive frame in the video, features are extracted, and how these features move through the frames is tracked, allowing for a better understanding of the camera motion. Secondly, it uses a mathematical operation called *affine transformation* that, with sparse registration techniques, can ignore dynamic objects in the scene based on detections and potentially estimate the background movements and shifts more accurately. Thirdly, it is implemented a correction of the state vector, including the velocities term, which, when the camera is changing slowly compared to the frame rate, this correction can be omitted, making the tracking more robust to camera motion [1].

After some experimentations, both Bot-SORT and ByteTrack performed well with the standard parameters, changing only to 120, the value for how many frames an object of interest would be tracked. For both trackers, the threshold for the first association was 0.5, the threshold for the second association was 0.1, the threshold for initialising a new track in case the detection does not match any tracks is 0.6, and the threshold for matching track is 0.8.

4.5 Fruit Georeferencing

Applying georeferencing of objects through video can make agriculture production one step closer to precision farming, allowing farmers to pinpoint the exact location of each fruit and improving management practices. Through the seamless integration of advanced technology with geographic information systems (GIS), an approach to comprehensive

solutions for mapping and monitoring each tree in agricultural production is achieved. Zones where production is below standard can be identified, enabling better management of the resources applied.

With this in mind, as stated in Section 4.1, some videos of Portuguese apple orchards were recorded using GoPro cameras 10 and 11, which registered telemetry data. The GoPro cameras used were able to record several different types of data. However, the most relevant data acquired for the georeference of fruit were the GPS location of the camera, Camera Orientation (CORI) and altitude.

The telemetry data that the GoPro cameras provided brought up a new challenge. After extraction of data from the GoPro cameras using Telemetry software, as referred in Section 4.1, it was noted that not all data registered by the camera was recorded at the same rate as the recordings. The telemetry data exported from the camera using the Telemetry software comes in CSV files in which each value obtained for specific data is saved with the timestamp of the video in a column with label *cts*. This indicates that telemetry data is saved based on time steps and not based on the image frame. Having the telemetry data recorded at the same rate as the FPS the camera records allows for a more straightforward association process between each frame and its data. This means that if each frame recorded by the camera had exact telemetry data with camera position and camera orientation, georeferencing would be easier. However, Camera Orientation (CORI) is the only telemetry data that is recorded at the same rate as the frames per second of the video recording. On the other hand, the GPS coordinates and altitude of the camera are recorded with an interval of 100 milliseconds. A recording at 60 frames per second would mean an interval of 16 milliseconds between each frame, meaning that for each recording of GPS coordinates there would be 6.25 image frames between them, which is not ideal since we want GPS coordinates and Camera Orientation data for each image frame in order to have a more accurate georeference of fruit.

In order to solve this problem, two CSV files of telemetry data are required: one CSV file including the CORI data with its timestamp and one CSV file including both the GPS coordinates and the CORI data. The first one should contain a direct correlation between the timestamp of the data (*cts* column) and the frame rate. The second one should contain the mathematical technique *Interpolation* applied to the GPS coordinates and the Camera Orientation (CORI) values. The *Interpolation* between these two sets of data allows for a solution for the missing of GPS coordinates between frames since this technique estimates the values of missing data between the timestamp of frames imposed by the CORI data.

Since the Interpolation of data also performs the estimation of values for the CORI data between the timestamps of GPS data, it is then necessary to use the file containing only CORI values and timestamps to know which GPS data interpolated is necessary. This is done through the use of timestamps of CORI since it is equal to the frame rate. Although the *Interpolation* method produces a linear estimation of values for each frame, not absolute values, it was still found that it performed well since GPS data was not saved by the GoPro at equal timestamps, making this a feasible solution for the problem.

With the telemetry data for each specific image frame, it is possible to go more in-depth through the algorithmic and mathematical problem of georeferencing. Firstly, this problem can be viewed simplistically in a 3D world where a camera is filming the orchard at a certain point. The trees can consequently be seen in a plane perpendicular to the camera's direction and at a certain distance. This can be done in this simple manner because, in reality, the camera records successive images, and as a consequence, these images are represented in a 2D (width and height) plane. However, cameras do not film only a plane but, in fact, an infinite amount of planes, visualised in Figure 4.2 (caused by the FoV and focal distance), which in this particular case of orchards, we can have, for example, objects in the background at a much further distance than the trees. In reality, by watching a video, it is impossible to accurately determine the distance of an object without measuring it first.

Knowing the Field of View (FoV) of the GoPro cameras and the distance between the camera and the object of interest, it is possible to calculate the width and height of the window where all detected objects are present. This can be calculated through Equation 4.1, where the width (l) of the window (Figure 4.2) is obtained by multiplying 2 with the tangent of half the horizontal FoV of the camera with the distance between the camera and the fruit trees (d). The same can be applied to height by using the vertical FoV.

$$l = 2 \times \tan \frac{FoV}{2} \times d \tag{4.1}$$

After this, in a specific frame, the information of the bounding boxes' location and the Field of View (FoV) of the camera would only be necessary. Our object detection model already gives the information of the bounding boxes for each frame in text files. Having the window size, the location of bounding boxes, and the distance between the camera and window, it is possible to, frame by frame, identify where the fruit is placed since, in consequent frames, a single fruit will be placed on top of each of its representations. However, realistically, there are always some errors that might make the location of a fruit in 2 frames differ slightly. The two biggest challenges that arise are the transformation of the 2D location of an apple from all frames into a single 3D location and the need to know the distance between the camera and the fruit trees.

To overcome these challenges, it is still considered that we have a plane perpendicular to the direction of where the camera is pointing, and the horizontal and vertical axis of the window are parallel to the horizontal and vertical axis of the camera. The bounding boxes' location inside the window is obtained through the plane equation and the vertical and horizontal axis of the camera in case of rotation during recording. Instead of measuring the distance between the camera and the trees each time it is necessary to make a recording, the camera's focal distance (or length) is used. The focal length of a camera can be interpreted as the distance between the lens and the sensor that captures the image. It is usually measured in millimetres.

To explain the transformation of the 2D location of a fruit in a window frame into 3D,

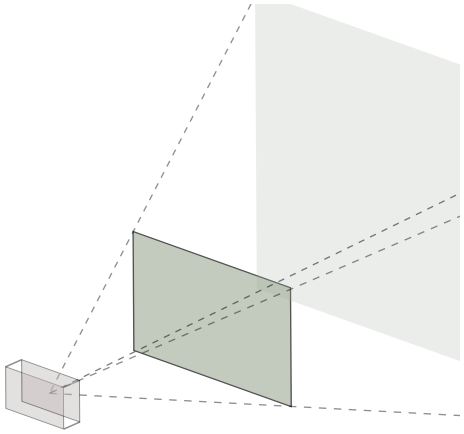


Figure 4.2: Representation of a camera filming FoV and the different planes.

let us consider only one fruit, identified in fifteen consequent frames. This means there are fifteen bounding boxes for this single fruit. For each frame, the idea is to imagine a straight line that starts in the camera, passing through the centre of the fruit's bounding box. This would mean there are 15 different straight lines that start in the lens and go to infinity. In perfect conditions, where there is no image-shifting or other noise to change the location of bounding boxes, the intersection of these straight lines is the real location of the fruit in 3D. However, these conditions are unrealistic, so instead the closest point to the intersection of straight lines is calculated. Since we have the location of the camera, the camera pointing direction, and the location of fruit bounding boxes inside the window plane at focal distance, it is possible to obtain an approximation of the real location of fruit using this method.

This solution can be viewed as an optimization algorithm by minimizing the sum of squared distances between the point and straight lines. Johannes Traa has already proposed a solution by solving the closest point to the intersection of a set of lines as a Least-squares problem [55]. This solution can be applied to our 3D context by considering a point on a line:

$$a = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}^T, \tag{4.2}$$

and the direction vector of the line:

$$n = \begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix}^T, \|n\|_2 = n^T n = 1, \tag{4.3}$$

Hence, the line is of the form:

$$l = a + tn, -\infty < t < \infty. \tag{4.4}$$

It is then stated in the research paper that the squared perpendicular distance from a point to a line can be simplified to:

$$D(p; a, n) = (a - p)^T (I - nn^t)(a - p), \tag{4.5}$$

where the matrix $I - nn^T$ serves to project the vectors **a** and **p** into the space orthogonal to **n**.

Furthermore, it is then explained that considering distance D in a minimization problem in order of **p**, consequently providing a linear system of equations:

$$Rp = q, \tag{4.6}$$

$$R = \sum_{j=1}^{K} (I - n_j n_j^T), q = \sum_{j=1}^{K} (I - n_j n_j^T) a_j, \tag{4.7}$$

which can then be solved as a Least-squares problem.

This all can be implemented in Python and visualized in Listing 2.

To go more in-depth about how this Georeferencing of fruit was performed, the Camera Orientation (CORI) data is exported, which included the values **W**, **X**, **Y**, and **Z**. These values are represented in quaternions, which can create a rotation matrix that represents the same rotation. This rotation matrix can be multiplied by the coordinates of points or vectors in 3D, giving the new location of the point or vector after the rotation. This means that if we consider the vectors of the standard axis of a referential (X, Y, Z) and multiply it by this rotation matrix obtained by the telemetry data, it is possible to obtain the same axis of the camera and perform the georeference of fruit in the way that was previously stated.

The coordinates of GPS given by the cameras are in Latitude and Longitude. However, since the calculations of the window size and location of fruit are all performed in meters, the Latitude and Longitude, which are in WGS84, need to be converted to Portuguese ETRS89 systems in order to have the data in a cartesian system.

In summary, the georeferencing problem was resolved using estimated and real telemetry data for each frame. Then, using this data, the intersection of all straight lines between the fruit (located in the image window) and the camera was estimated, resulting in an estimate of the actual location of the fruit. Section 5.3 shows some results obtained from this method.

```python
def intersect(P0,P1):
    """P0 and P1 are NxD arrays defining N lines.
    D is the dimension of the space. This function
    returns the least squares intersection of the N
    lines
    """

    if P1.shape == (1,3):
        p = P1.transpose()
    else:
        #generate all line direction vectors
        n = (P1-P0)/np.linalg.norm(P1-P0,axis=1)[:,np.newaxis] # normalized

        #generate the array of all projectors
        #I - n*n.T
        projs = np.eye(n.shape[1]) - n[:,:,np.newaxis]*n[:,np.newaxis]


        #generate R matrix and q vector
        R = projs.sum(axis=0)
        q = (projs @ P0[:,:,np.newaxis]).sum(axis=0)

        #solve the least squares problem for the
        #intersection point p: Rp = q
        p = np.linalg.lstsq(R,q,rcond=None)[0]


    return p
```

Listing 2: Python function to calculate the closest point to the intersection of multiple straight lines in 3D.

5

Results and Discussion

This chapter presents the performance of all training models and an evaluation of the one that produced the best results during training and in realistic examples. The results for georeferencing are also presented.

A metric evaluation is first made into the testing set of the agglomeration dataset as referred in Section 4.2.4. This is performed as a validation step to have a broad view of the model's generalisation and achieve a guideline for the confidence threshold that will work in more varied data. Secondly, the model's performance and accuracy in realistic use cases for fruit counting in videos and images of Portuguese orchards are evaluated. The metric evaluation is done to all models in Sections 5.1.1 and 5.2.1. In contrast, the evaluation in realistic use cases will only be done using the models that perform best in the validation.

The validation step will comprise various metrics to evaluate the model's performance, such as Precision, Recall, mAP, and F1-score.

These metrics collectively serve as indispensable instruments for meticulously validating and evaluating object detection models. All the models were evaluated using these metrics in the same set of images to evaluate the model under the same use cases and select the confidence threshold that better fits the model in more generalised data.

5.1 Yolov7 Results

5.1.1 Yolov7 Quantitative Results

This section provides the quantitative results, showing the performance of every different model of YOLOv7 on the same testing set. It shows the numerical results and explains each of them to better understand how each variation of the model performs and which best fits the research task.

Table 5.1 aggregates the values of each model performance based on Precision, Recall, mAP with 0.5 IoU threshold (mAP@0.5), and mAP with IoU threshold values changing between 0.5 and 0.95. These results are also compared by testing the models on images with 1088x1088 pixels and 640x640 pixels.

The models that performed best were Exp25, Exp26, Exp43 and Evolve. The models Yolov7-custom12 and Exp23 performed worse to more generalised data when compared to the ones trained in the agglomeration dataset referred in Section 4.2.4. Across all the models trained, the percentage of False Negatives (FN) was higher than the percentage of False Positives (FP) due to the higher Precision in comparison with Recall.

An important feature to notice is that Yolov7-custom12 was trained on 640x640 pixels images, and on inference images of 1088x1088 pixels, it still showed a slight increase in the balance between False Negatives and False Positives with a higher F1 Score than 640x640 pixels images. This result highlights the importance of higher inference sizes, which might increase accuracy while maintaining good results at lower inference sizes. This point can also be demonstrated with Exp26, where the model was trained on 1088x1088 pixels images and still performed really well on 640x640 pixels images.

The Exp model on images with a size of 1088x1088 pixels showed an overall bad convergence compared to model Exp25. With a Precision of 80.5%, it showed a higher number of False Positives than the Exp25 model. The same happens with the number of False Negatives due to the smaller Recall value of 73.4%. Overall, this model performed worse on the image set than Exp25, with a value of mAP@0.5 significantly smaller. Regarding images with a size of 640x640 pixels, this model also performed worse than Exp25, with Precision, Recall and mAP@0.5 with values 76.2%, 69.6% and 75.7%, respectively. With this data, adding the ADAM optimiser did not improve the model's performance, making it relatively worse.

The Evolve2 model showed a decrease in performance in all metrics compared to the Evolve model. The only difference between the training of these two models is that Evolve2 uses the training parameter –rect, which forces all images to be rectangular, meaning that images that are not square will have some deformations. The performance of this model showed that maintaining the aspect ratio of the images and avoiding the deformation of images improves the model's overall performance.

The Exp-tree, contrary to all the other models, was validated on a set with generalised data, including visible trees as referred to in Section 4.2.5. This is done because it is the only dataset with labelled trees. The model didn't show very good results in the detection of fruit and trees. Using a smaller dataset to train the model for both apple and tree detection decreased its performance. The model performed better in apple detection with an F1 score close to 70% and performed worse in tree detection with an F1 Score of 30%. As expected, it was not able to detect trees effectively since trees have a much more complex set of features compared to apples.

After a first analysis of the results, there are a few things to point out regarding different training methods. The first thing to point out is the effectiveness of always trying to use the best quality of images and sizes without losing too much time with the right computation resources. This can be further explained with the model Exp42. This model was trained with image sizes of 640x640 pixels with the training parameter –multi-scale active, meaning that images would be selected randomly for a +/-50% difference in image size. However,

despite being trained on smaller images, this model showed no improvement in evaluation metrics to the Exp25 with an image size of 640x640 pixels, performing slightly worse, while Exp25 still performed much better in higher image sizes. This shows how vital the padding and maintaining the aspect ratio of images is, as explained in Section 4.3, developing why the image size chosen for training the models was 1088x1088 pixels.

Another essential factor is the comparison between Exp25 and Exp26, which shows that using a pre-trained model does not necessarily mean better performance. This is mainly because the pre-trained model was simply an apple detector trained in a smaller dataset. It is interesting to see that the changes are minor. However, to draw better conclusions, a further analysis will be performed.

Table 5.1: Metric results for all models

Model / img-size	Precision	Recall	mAP@0.5	mAP@0.5:0.95	F1
Yolov7-custom12 / 1088	0.714	0.647	0.682	0.326	0.68
Yolov7-custom12 / 640	0.731	0.641	0.681	0.309	0.67
Exp23 / 1088	0.704	0.677	0.695	0.314	0.69
Exp23 / 640	0.710	0.671	0.689	0.308	0.69
Exp25 / 1088	0.882	0.867	0.918	0.560	0.87
Exp25 / 640	0.887	0.837	0.899	0.530	0.86
Exp26 / 1088	0.896	0.872	0.916	0.573	0.88
Exp26 / 640	0.892	0.862	0.906	0.549	0.88
Exp / 1088	0.805	0.734	0.817	0.428	0.77
Exp / 640	0.762	0.696	0.757	0.380	0.73
Exp42 / 1088	0.832	0.825	0.885	0.514	0.83
Exp42 / 640	0.866	0.829	0.899	0.521	0.85
Exp43 / 1088	0.884	0.863	0.914	0.559	0.87
Exp43 / 640	0.886	0.842	0.898	0.528	0.86
Evolve / 1088	0.892	0.855	0.913	0.555	0.87
Evolve / 640	0.892	0.839	0.896	0.526	0.86
Evolve2 / 1088	0.796	0.718	0.799	0.408	0.75
Evolve2 / 640	0.748	0.657	0.727	0.352	0.70
Exp-tree / 1088	0.566	0.473	0.475	0.216	0.49
Exp-tree / 640	0.603	0.471	0.517	0.237	0.52

On further analysis of Exp25, Exp26, Exp43 and Evolve models, Appendix A shows the graphs for all the metrics considered. Both Exp25 and Exp43 showed very similar results. For the image size of 1088x1088, the Exp43 model showed higher Precision but a lower Recall. For an image size of 640x640, Exp43 showed a slightly lower Precision with a higher Recall value. However, the mAP value for Exp25 was higher for both image sizes. This result might indicate that the Exp25 model is more consistent across different confidence thresholds. The Evolve model, compared to the Exp25, showed a higher Precision but a lower Recall. This indicates that in situations where higher Precision is necessary, this model might be a better choice than Exp25. Exp26, compared to the other three models, showed higher Precision, Recall, and F1 score. This indicates that this model might be better suited for the solution of the problem.

With an image size of 1088x1088 pixels, the Exp25, Exp43, and Evolve models achieved an F1 Score of 87% at a confidence threshold of 53.3%, 51.3%, and 54.0%, respectively. These models all achieved 100% Precision at a confidence threshold of 90%, 90.9%, and 91%, respectively for Exp25, Exp43, and Evolve. The Exp26 model achieved an 88% F1 Score at 56.2% confidence threshold and 100% Precision at 90.5%. All four models achieved a maximum of 97% Recall at 0.00% confidence threshold.

With an image size of 640x640 pixels, the Exp25, Exp43, and Evolve models achieved an F1 Score of 86% at a confidence threshold of 47.3%, 44.1%, and 48.3%, respectively. These models all achieved 100% Precision at a confidence threshold of 89.7%, 90.9%, and 89.3%, respectively, for Exp25, Exp43, and Evolve. The highest Recall value for the Exp25 and Evolve models was 96%, while for the Exp43 model, it was 95%. The Exp26 model achieved an 88% F1 Score at 45.2% confidence threshold and 100% Precision at 89.8%. The highest Recall value for Exp26 is 96%.

After a more in-depth analysis of each model's performance, it is possible to conclude which one is a better fit for the task. It is essential to consider the context and priorities of the specific application. All four models analysed (Exp25, Exp26, Exp43, Evolve) had a higher Precision than Recall, meaning that all had more problems in reducing the number of False Negatives than the number of False Positives. Exp43 and Evolve can be disregarded because they perform very closely to the Exp25 model with the slight difference that Exp25 performed better with the highest mAP value. Comparing Exp25 and Exp26, we must look more in-depth at our task. The fundamental goal of using these DL models is to perform object detection of fruit for counting and georeferencing purposes.

Consequently, the model needs to identify all the objects of interest (fruit) present in each video/image, meaning that the goal is to minimise the number of False Negatives. A high value of False Negatives would mean that not all fruit were detected by the model, contributing to a lower value in the counting fruit task. However, it is also essential that all detections performed by the model are considered objects of interest (fruit), meaning that another goal is to minimise False Positives. A high number of False Positives would mean that the model would identify many objects that are not supposed to and, consequently, perform an over-counting of fruit.

Considering this, a model that best minimises both False Positives and False Negatives is necessary. The model best fit for this task is Exp26 since it had a better balance between Precision and Recall, with an F1 Score value of 88%, which was the highest of all values, obtaining the lower results for both False Positives and False Negatives in both images sizes of 1088x1088 and 640x640 pixels. However, to obtain these optimal results, it is necessary to run the inference videos at a detection confidence of 56.2% and 45.2% for image sizes of 1088x1088 and 640x640 pixels, respectively. Although Exp26 might perform better for a specific confidence threshold, Exp25 still has a slight increase in mAP value. This means that if agricultural producers need a more versatile model that can perform very well among different confidence thresholds for cases where more Precision or Recall is necessary, then the Exp25 model might be more appropriate. However, the difference

in overall performance across the confidence thresholds between Exp25 and Exp26 is so minor that the Exp26 model could also be used for these use cases.

The next step was to test the Exp25 and Exp26 models' performance in real-life scenarios based on the selected confidence threshold from the previous validation step. From the video of Portuguese orchards, 10 Frames were extracted with different characteristics. The Frame's characteristics can be comprised as follows (with ID for future reference):

- Frame1: This image aims to mimic the standard example for fruit counting, where no light conditions are difficulting the detection of apples and is extracted from a video recorded with the GoPro camera using a wide lens inside a moving vehicle, filming directly pointing at trees, with no stabilisation applied, so there may be some deformations on the sides of the image. This image has a size of 3840x2160 pixels.
- Frame2: This image was extracted from the same video as Frame1 at a similar time step. However, the Hue, Saturation, and Value were changed using a Video editor. These changes were applied so that the majority of Green and Blue present in the image was viewed as black and white while maintaining a bright red colour to evaluate if this pre-editing process would impact the model's performance (a representation of this effect can be seen in Figure C.2). The image size is 2880x1620 pixels.
- Frame3: This image was extracted similarly to Frame1, with the only modifications applied being wide lens stabilisation and decreasing the image quality. The image size is 2880x1620 pixels.
- Frame4: This image was extracted from a video recorded by the GoPro camera with a wide lens, walking alongside and directly pointing at the trees. This image had slightly bad lighting conditions as it was recorded against sunlight, although clouds slightly diffused the sunlight. The image size is 3840x2160 pixels.
- Frame5: This image was extracted from a video recorded by the GoPro using a wide lens, where the light conditions were favourable, and there were few occlusions. The image size is 3840x2160 pixels.
- Frame6: This image was extracted from a video recorded by GoPro using the linear lens, where the light conditions were favourable, and there were not many occlusions. The image size is 3840x2160 pixels.
- Frame7: This image was extracted from a video recorded with the GoPro camera using a wide lens inside a moving car, with favourable lighting conditions, the camera pointing with a 45º angle to the trees, and the wide lens later stabilized using Gyroflow. The image size is 2880x1620 pixels.
- Frame8: This image was extracted from video recorded with the GoPro using a wide lens inside a moving vehicle and pointing directly at the trees, later stabilized using

Gyroflow. This image tests the model's performance in terrible conditions, with low sunlight creating shadows among the fruit. The image size is 2880x1620 pixels.

- Frame9: This image was extracted from a video recorded using an Android smartphone, with favourable lighting conditions and pointing with a 45º angle to the trees. The image size is 1080x1920 pixels.

- Frame10: This image was extracted from a video recorded using an Android smartphone, with favourable lighting conditions and pointing directly at the trees. The image size is 1080x1920 pixels.

Table 5.2 shows the results for the detection of each image frame described previously for the Exp25 model using an inference image size of 1088x1088 pixels and a confidence threshold of 53.3%. This confidence threshold was chosen based on where the F1 value is highest, as described previously (visualised in Figure A.1(a)). This table represents the number of Apples that were detected correctly (TP), the number of detections that were miss-classified as Apples (FP), the number of Apples present in each image left to be detected (FN), the number of Double-Detections, which represents the number of fruit detected twice, and finally a comparison between the detected fruit and the total amount present in the Frame, manually counted.

Table 5.2: Performance of the model Exp25 in image frames extracted from videos with an inference image size of 1088x1088 pixels at a confidence threshold of 53.3%.

Image	TP	FP	FN	Double Detection	Detected/Human Count (%)
Frame1	102	0	49	1	102/151 (67.5%)
Frame2	92	0	45	0	92/137 (67.2%)
Frame3	98	1	39	1	98/137 (71.5%)
Frame4	67	0	15	0	67/82 (81.7%)
Frame5	51	0	23	0	51/74 (68.9%)
Frame6	62	0	11	0	62/73 (84.9%)
Frame7	113	0	57	1	113/170 (66.4%)
Frame8	87	0	39	1	87/126 (69%)
Frame9	34	0	11	0	34/45 (75.6%)
Frame10	24	0	5	0	24/29 (82.8%)
Total	730	1	294	4	730/1024 (71.3%)

Table 5.3 shows the results for the same experiment performed for the Exp26 model, as described previously. Compared with the Exp25 model, the Exp26 performed slightly better overall, with a 0.1% increase in performance. One thing to note was that Exp26 did not have as many double-detections as Exp25. However, both models' performance is so similar that it does not necessarily mean that one is better than the other. In fact, from the information shown in the tables, it is possible to conclude that Exp25 may be more suitable for certain types of videos while Exp26 is more suitable for others. From this, Exp25 may be better for video counting recorded with smartphones, as the performance

of Frames 9 and 10 are better in model Exp25. On the other hand, Exp26 performed better on Frames obtained from videos recorded with a wide lens on a moving vehicle and performed worse on the Frame with a 45º angle towards the Tree.

Table 5.3: Performance of the model Exp26 in image frames extracted from videos with an inference image size of 1088x1088 pixels at a confidence threshold of 56.2%.

Image	TP	FP	FN	Double-Detection	Detected/Human Count (%)
Frame1	107	0	44	0	107/151 (70.9%)
Frame2	92	0	45	0	92/137 (67.2%)
Frame3	101	1	36	1	101/137 (73.7%)
Frame4	67	0	15	0	67/82 (81.7%)
Frame5	50	0	24	0	50/74 (67.6%)
Frame6	58	0	15	0	58/73 (79.5%)
Frame7	110	0	60	0	110/170 (64.7%)
Frame8	92	0	34	1	92/126 (73%)
Frame9	31	0	14	0	31/45 (68.9%)
Frame10	23	0	6	0	23/29 (79.3%)
Total	731	1	293	2	731/1024 (71.4%)

There are a few interesting points worth mentioning from the results of these tests. Firstly, the enhanced method used in Frame2 by making the whole image black and white, except for the fruit (maintained red), did not affect overall performance, actually slightly decreasing it. This can reflect that the model, although trained on natural images, does not retain only information about objects that are red but, instead, a much more complex set of features. Secondly, it is shown that the model can benefit from not using a wide lens or later stabilising it. Frame3, a stabilised version similar to Frame1, produced better results after stabilisation. Frame6, extracted from a video using a Linear Lens, showed the best results with the Exp25 model and second best with the Exp26 model. With this in mind, both models can be used for different purposes, allowing agriculture producers to obtain better results based on the different types of videos they acquire.

Although these results provide some information about the performance of the models in real scenarios, they could be better than the metrics provided by the set previously performed. In order to further evaluate this phenomenon, one single test with the Exp26 (since both models have similar overall performance) on the same image frames was performed, but this time with a confidence threshold of 5%. The results of this test can be seen in Table 5.4. This table presents an extra column, FPA (False Positive Apple), which includes the number of apples correctly detected by the model. However, the apples were not considered for the counting because they belonged to trees behind the row where the camera was pointing. We can further evaluate the model's performance through a lower confidence threshold., making it is possible to see an increase in detections. However, this does not necessarily mean a better performance. The most important thing to notice is that the number of FP has increased slightly while the Recall metric has decreased.

Consequently, it is essential to note that the number of Apples detected that did not

belong to the front row of trees (FPA) was very high in specific frames. This indicates that while the low confidence threshold might decrease the number of False Negatives, it might increase the number of False Positives, challenging fruit counting. However, with the low confidence threshold, the Recall metric would be expected to be higher. After analysing the image frames used in testing, this phenomenon and why the model's performance is lower than expected is further discussed in Section 5.1.2.

Table 5.4: Performance of the model Exp26 in image frames extracted from videos with an inference image size of 1088x1088 pixels and confidence threshold of 5%.

Image	TP	FPA	FP	FN	Double-Detection	Detected/Human Count (%)
Frame1	127	1	0	24	5	127/151 (84.1%)
Frame2	107	0	1	30	1	107/137 (78.1%)
Frame3	114	0	1	23	5	114/137 (83.2%)
Frame4	80	19	3	2	2	80/82 (97.6%)
Frame5	69	20	1	5	1	69/74 (93.2%)
Frame6	70	8	0	3	3	70/73 (95.9%)
Frame7	160	1	1	10	9	160/170 (94.1%)
Frame8	116	0	1	10	5	116/126 (92.1%)
Frame9	41	0	0	4	2	41/45 (91.1%)
Frame10	27	0	0	2	0	27/29 (93.1%)
Total	911	49	8	113	33	911/1024 (89%)

The next thing that was evaluated was the number of detections for each zone of the image the model can detect, as referred to in Section 4.3.4. For this test, the model used was Exp26 due to a subtle increase in performance compared to Exp25. This test aims to better understand the model's efficacy in detecting fruit in specific image zones. For this test, three videos were selected with different configurations and can be comprised as follows:

- Video1: Frame3 mentioned in the previous test was extracted from this video. This video is relevant due to the movement from right to left through the orchard.
- Video2: Frame6 mentioned in the previous test was extracted from this video. The relevance of this video for comparison is that the video was recorded from left to right.
- Video3: Frame7 mentioned in the previous test was extracted from this video. This video is relevant for comparison since it was recorded with a 45º angle between the camera and the fruit trees, with a left-to-right motion.

Table 5.5 shows the results of this experiment for each of the videos mentioned. It is important to note that the numbers in this table represent the number of detections, not the unique fruit, which means that a fruit detected in zone 1 can also be detected in any other zone. Looking at the first video, it is possible to notice that the zones where more fruit was detected were in the middle. The left-most columns also detected slightly more fruit than

Table 5.5: Results of the detection of each zone in the three Videos selected

Zones	Video1	Video2	Video3
1	34622	56588	8604
2	35617	61512	9946
3	41017	65529	13174
4	43643	68297	17677
5	41144	70558	23568
6	40310	70810	31198
7	36037	70037	39188
8	30447	68193	42963
9	26222	63166	41904
10	23965	57076	31598

the right-most columns. This can indicate that the camera's movement from right to left may increase the detection performance on the left side while decreasing on the right side. Looking at Video2, it is possible to see this phenomenon, where the video recorded from left to right detected more fruit in the most-right columns. The phenomenon observed in this video may be more subtle than in the first due to the first video being recorded inside a moving car. However, since Video2 was recorded while walking, these results show the impact of velocity on this phenomenon.

Considering Video3, we do not have a more linear distribution throughout the zones. It is possible to see the number of detections increasing from Zone 1 to Zone 8 and then decrease to Zone 10. Since this video was recorded with a 45º angle between the camera and the trees, and after visualisation of the videos, it is possible to conclude that, in fact, in the model with this angle, it is possible to avoid most of the occlusions, while in the most left side, the leaves are occluding more fruit. However, most right-side columns will have apples represented in a smaller size, making it even more challenging to detect on the far right side.

With this evaluation, it is possible to have more insight into how the model performs and what conditions obtain the best results, such as the camera's trajectory's impact on the results and the recording with an angle to the trees, which might avoid some occlusions behind the tree. However, it is only possible to draw accurate conclusions on this topic with the various recordings of the same tree row, using different vehicle velocities, angles, and camera trajectories.

The last evaluation done for this model was the performance in combination with both tracking algorithms (SORT and DeepSORT) for fruit counting. Three videos with different characteristics were chosen for this task. These three videos were used in the previous test, and their relevance for this evaluation can be comprised as follows:

- Video1: This video includes good lighting, allowing every apple to be more visible without being occluded by shadow. Using a wide lens, the video was recorded directly pointing at the trees while in a moving vehicle. The wide lens was then stabilised to approximate the linear lens. The testing on this video allows insight

into the model's performance in better lighting conditions and how a moving vehicle might affect results. The orchard producer estimates 88 fruits per tree, and 106 trees are present in the video, so it is estimated that there are 9328 apples in this video.

- Video2: This video was recorded in a cloudy environment, where little sunlight creates shadows or highlights the apples. It was also recorded while walking with a linear lens of a GoPro and directly pointing at trees. The testing on this video allows insight into the model's performance in more neutral lighting conditions and a slower trajectory. There was no estimation of fruit per tree for this video. Consequently, several trees were counted, and it was estimated 31 fruits per tree. Since there are 250 trees, it is estimated that 7750 apples are in this video. Most trees in the video contained around 30 to 40 fruits, while others contained 10 to 20 and around 50.

- Video3: This video was recorded with favourable lighting conditions, inside a moving vehicle, and with a 45º angle towards the trees. The testing on this video allows insight into the model performance when tracking apples without directly pointing at trees. The producer in this orchard is the same as in Video1, so it is estimated to have 88 fruits per tree. Since there are 137 trees present in the video, it is estimated that 12056 apples are in this video.

Table 5.6 represents the summary of the video configurations used in this test. The movement represents if the video was recorded in a moving vehicle at 10 km/h or while walking. The camera angle at 0º represents that the camera was directly pointing at the trees.

Table 5.6: Video configurations used in tests for the model performance.

Video	Movement	Camera angle	Lens	Fruit Estimation
Video1	Vehicle	0º	Wide with stabilization	9328
Video2	Walking	0º	Linear	7750
Video3	Vehicle	45º	Wide	12056

Although evaluating the model's performance on more videos is imperative, getting an exact estimate of all the apples in a row of trees is challenging. For this reason, only a small selection of videos with different characteristics was used to provide more information about the differences in the tracking systems and how they benefit the solution for the task. The fruit in each video was estimated based on information obtained from the producers where the videos were taken. The fruit counting in each video was then obtained using an additional Python program using information from the model, as referred to in 4.3.4. It is important to note that both trackers were tested with the same conditions in YOLOv7. The confidence chosen was 56.2% based on the F1 score discussed in previous testing. The image size was 1088x1088 pixels based on the model's training.

Exp26 model, using only the SORT tracker, obtained a fruit count of 7500, 9457, and 7052 for Video1, Video2, and Video3, respectively. These values correspond to accuracy for Apple counting of 80.4%, 122%, and 58.5% for each video based on the estimated values, respectively. Through these results, we can see that in Video1, the model performed as expected, while in Video2, it provided an overcounting of around 20%, and in Video3, the model provided a low counting of fruit.

The model Exp26, using the DeepSORT tracker, obtained a fruit count of 7759, 7891, and 8881 for Video1, Video2, and Video3, respectively. These values correspond to an accuracy of 83.2%, 101.8%, and 73.7% for each video based on the estimated values, respectively. These results showed an increase in the model's performance compared to the SORT tracker.

These results between trackers showed inconsistencies between different videos. One video produced an overcounting of fruit, while the other two produced an under-counting. To further analyse what is happening in each of these videos, Section 5.1.2 comprises a qualitative analysis of the performance of the model with each tracker for each video to further understand the results.

5.1.2 Yolov7 Qualitative Results

Appendix C shows some figures of the frames mentioned in Section 5.1.1 with the detection done by the models. After some analysis of each image, it is possible to conclude more about the model's performance. The Figures show that most of the Apples in the image are detected. The illusion that the model might perform slightly worse is due to the occlusions that apples might suffer. The essential key factor is that every single frame was manually counted in each frame. This means that every single fruit was counted, including the occluded fruit. While a person has perception and context, a model does not. A person can look at a tree of apples and count an almost occluded apple by leaves. They can understand through context and perception that it is, in fact, an Apple.

On the other hand, a DL model can not perceive context as effective, meaning an almost occluded apple will not be identified as such. Consequently, the counting of fruit in a frame performed by the model was compared to the total fruit present in the trees, including almost entirely occluded fruit. This emphasises the importance of counting fruit throughout multiple frames or during a video since continuous frames allow occluded fruit to become visible.

Another essential aspect retained from the analysis of each frame was that in the case where the model over-detected fruit with a confidence threshold of 5% while having a congruent decrease in Precision, in most cases, it was already detecting actual fruit present in trees behind those in front of the camera.

To further understand how the model is performing in the fruit counting task, every video was observed with the detections performed by the model.

Regarding the analysis of the Exp26 model with SORT tracking in Video1, it was

evidenced that the model was detecting and tracking correctly the majority of apples that appeared in the video. The model only showed the problem stated previously, where it cannot detect fruits that are almost entirely occluded and are only perceived by a person through context. The SORT tracker was accurate when a fruit was detected and had no occlusions throughout its trajectory with almost non-visible ID switching. However, when a fruit was occluded for a set of frames and became visible, occasionally, the tracker would assign a new ID, identifying it as an unseen fruit. Occlusions are the biggest challenge, especially because, in this video, the bottom part of the tree is more crowded with leaves, making it more difficult for the model to detect fruit.

In comparison, the usage of DeepSORT increased the number of fruits counted. The DeepSORT showed positive results in dealing with occlusions. After a fruit was occluded, DeepSORT, most of the time, would identify it as the same fruit when it became visible. However, it imposed another challenge. The stabilisation of the video, although it corrected the wide lens effect, still produced different frame characteristics between the frame centre and frame edges. The stabilisation process inverts the wide lens effect, making the sides of the image artificially stretch. Since the DeepSORT tracker extracts object features to track, occasionally, when the fruit being tracked achieves the side of the frame where the stretch effect is more intense compared to the middle, the characteristics of the apple may differ. The tracker would identify the fruit as unseen and apply a new ID.

Regarding the performance of the SORT tracker with YOLOv7 in Video2, it was possible to see that the model detected almost every single fruit. However, in this particular video, in the beginning, due to the trees having less vegetation in comparison to the latter parts of the video, the model would detect several apples that belonged to trees that were on the row behind the one that was being recorded. Although these apples were being identified correctly, this was not an intended effect, causing an overcounting of fruit. However, it was seen that does not justify the 20% of overcounting. Another issue that was noticed was the label switching. Although the SORT tracker would track most apples correctly, often, when a fruit is occluded for a few consecutive frames, when it became visible again, the tracker would assign a new ID. This is the main problem of SORT, which DeepSORT helps resolve.

The DeepSORT tracker with YOLOv7 showed far better results. However, the problem of apple detection in the row behind was still present, as expected, since it is an effect of the model and not the tracker. Since this video was recorded with a linear lens, there were no deformations in the image frame. DeepSORT performed perfectly in this case, as no ID switching was noticeable. This is the leading cause for the drastic reduction of overcounting between trackers. These observations showed that DeepSORT improved significantly the ID switching present in the SORT and is a viable solution for this task.

Regarding the performance of the SORT tracker with YOLOv7 in Video3, the problems previously stated were confirmed. SORT tracker once again was able to track most of the apples that were visible and partially occluded. However, when a fruit is completely occluded or undetected for a set of consecutive frames, the tracker assigns a new ID.

The DeepSORT on this video counted more fruit than the SORT tracker. However, it does not perform better since the wide lens and stabilisation induce image stretching in the sides, performing ID switching between fruits when it reaches the sides of the image. Once again, the DeepSORT model performs relatively worse when there are changes in the object of interest due to image stretching.

Another thing understood from the visualisation of Video3 with both trackers was that the camera angle to the trees did not help the detection of fruit. This video was recorded from left to right with a 45º angle to the right (direction of forward movement). In some trees, the camera had a better angle towards the tree centre, allowing for the easier detection of apples present behind and inside the leaves. However, most times, the apples on the right side of the trees were not detected since the tree was being recorded from the left, and other trees and leaves would occlude the right side. This way, directly pointing the camera to the trees seems a more viable solution.

To further interpret these results for the model, Video1 and Video3 results are compared to the estimation of the producer. However, observing the videos made it possible to understand that the majority of visible and partially occluded fruit was detected and tracked (around 40 to 70 visible fruit per tree). In these two videos, due to the stretching of the image, DeepSORT presented an ID-switching problem more intense than SORT, producing an over-counting of visible fruit. In Video2, since the estimation of values was performed from what could be interpreted and counted as apples inside the video itself, the fruit counting behaved as expected. The great majority of apples were being detected and tracked. However, through the SORT tracker problem of ID-switching, it performed an over-count. On the other, through a linear lens, the DeepSORT tracker performed extraordinarily, with almost no ID-switching and achieved almost 100% accuracy.

5.2 Yolov8 Results

5.2.1 Quantitative Results

This section provides the quantity results analysis of the performance of the YOLOv8 models trained on the same testing set. The test set used for this evaluation was the one that belongs to the aggregation dataset referred to in Section 4.2.4. It was chosen due to the different amount of generalised data, which allows for a better comparison between models.

Table 5.7 shows the results of each model performance based on Precision, Recall, mAP with 0.5 IoU threshold (mAP@0.5), and mAP with IoU threshold values changing between 0.5 and 0.95 (mAP@0.5:0.95). These results are also divided by running the testing set with image sizes of 1088x1088 and 640x640 pixels to evaluate how inference size might influence results.

Train19, as expected, the model performed worse than all the YOLOv8 models tested. This is because the model was trained on a relatively smaller dataset than the others

and performed worse when tested in a large set and subjected to more generalised data. Regarding evaluating the model with an inference image size of 1088x1088 pixels, the Precision, Recall, and mAP@0.5 results were 72%, 62%, and 67%, respectively. With these results, we can conclude that for all the detections performed by the model, 28% were not apples (False Positives) and that for all the apples present in images, 38% were not detected (False Negatives). YOLOv8 provided a more informative confusion matrix than YOLOv7, providing more in-depth information about the number of False Negatives and False Positives. This model, using this image size, had a total of 5833 False Positives and 7574 False Negatives, against a total of 14577 True Positives. This is coherent with a higher Precision and lower Recall. Considering the analysis of this model with an image size of 640x640 pixels, it was possible to conclude that the Recall metric was higher and the Precision was slightly lower. In contrast, overall performance with the mAP metric increased. These values are expected since this first training was performed with an image size of 640x640 pixels, meaning that the model is more accustomed to images of this size, which, when evaluated in a higher image size, might perform slightly worse due to the model being used to the bounding boxes having a specific size inside 640x640 pixels images. In contrast, it was not trained to detect apples in a larger image, performing worse in testing with an inference size of 1088x1088 pixels.

Considering the Train6 and Train3 models, it is possible to conclude that using a more detailed dataset improved performance with more generalised data. Since these models were trained on an inference image size of 1088x1088 pixels, the results were better for this size. Considering the image size of 1088x1088 pixels, both models achieved the same Precision (87%), meaning they have a similar number of False Positives. In contrast, Train3 had a slight increase in Recall compared to Train6, meaning that Train3 obtained a lower number of False Negatives. This proved that Train3 might perform slightly better under these conditions due to the model being able to better fit the data. However, for the image size of 640x640 pixels, Train6 performed slightly better, with an increase in Precision (lower False Positives) and a slight decrease in Recall (higher False Negatives). Train6 for this image size presented an mAP slightly higher than Train3, indicating it might perform better to a more generalised amount of confidence and IoU thresholds. This improvement is due to using Train19 as a pre-trained model for the Train6 model. Since Train19 was trained on an inference image size of 640x640, using these weights, the Train6 model could perform better for lower image sizes while performing slightly worse for larger image sizes.

A more in-depth analysis is required to decide which models better fit our task (Train6 or Train3). Appendix B comprise the set of testing graphs and confusion matrices for train3 and train6 models.

Considering the model Train6 for the image size 1088x1088, the F1-Confidence Curve showed that this model had the highest F1 value of 86% at 42.3% confidence threshold. This F1 value indicates that the model achieves excellent performance considering the set's size, accurately identifying objects of interest while minimising False Positives and

Table 5.7: Metric results for all YOLOv8 models

Model / img-size	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Train19 / 1088	0.726	0.621	0.667	0.327
Train19 / 640	0.720	0.652	0.686	0.331
Train6 / 1088	0.874	0.851	0.914	0.594
Train6 / 640	0.869	0.837	0.898	0.565
Train3 / 1088	0.874	0.861	0.916	0.594
Train3 / 640	0.867	0.840	0.895	0.563

False Negatives. The confusion matrix of this test showed that the model counted 4417 False Positives, 2140 False Negatives and 20011 True Positives. The model also achieved 100% Precision at a 91% confidence threshold, meaning at this confidence, every detection performed by the model was guaranteed to be correct (0 False Positives). The same model testing images with a size of 640x640 pixels had an F1 value of 85% at a 34.3% confidence threshold, which shows a good score in minimising both False Positives and False Negatives. The confusion matrix for this case showed 3506 False Positives, 2838 False Negatives, and 19113 True Positives. These results indicate an increase in Precision and a decrease in Recall with lower image size testing. It is essential to keep in mind that these confusion matrices may not always reflect the values present in Table 5.7 as the confusion matrix is computed at a fixed confidence threshold of 25% while the values presented in Table 5.7 were computed at the maximum mean F1 confidence by the testing file.

Considering the model Train3 for the image size 1088x1088, the F1-Confidence Curve showed that this model had the highest F1 value of 87% at 41.8% confidence threshold. This F1 value indicates that this model performs slightly better than Train6, minimising False Negatives and False Positives. The confusion matrix indicates 4005 False Positives, 2158 False Negatives and 19993 True Positives. Compared with Train6, this model detected fewer False Positives (412 less), which indicates better Precision, and a slight increase in False Negatives (18 more), slightly decreasing Recall. This model also achieved 100% Precision at a 90.3% confidence score, indicating it can decrease the number of False Positives at lower confidence thresholds. Considering the image size 640x640, the F1-Confidence Curve indicates that the highest F1 value is 85% at 32.8% confidence threshold. This indicates that this model at this image size performs relatively similarly to the Train6 model at minimising False Positives and False Negatives. In this case, the confusion matrix indicates a total of 3416 False Positives, 2885 False Negatives, and 19266 True Positives. These results indicate an increase in Precision and a decrease in Recall against the same model with an image size of 1088x1088 pixels. However, in comparison with model Train6 for the exact image size, it is possible to see a reduction in False Positives (90 less) and an increase in False Negatives (47 more), consequently indicating more Precision and less Recall than the Train6 model for this image size.

To decide which model better fits the task, we would have to think about what is the main goal to achieve. As seen for YOLOv7 in 5.1, the model's primary importance

is having the least amount of False Positives and False Negatives. For this reason, the Train3 model would better fit this task. Not only this, as opposed to what happened with YOLOv7, this model still performed better based on the mAP metric, indicating a better performance across different confidence thresholds. However, if agricultural producers used this model on smaller images for faster inference times, the Train6 might be more suitable for different confidence thresholds. In summary, the model that performs best might depend on the task at hand.

To further evaluate the performance of Train6 and Train3, the same tests using the same image frames as with YOLOv7 in Section 5.1.1.

Table 5.8 shows the results for the detection of each image frame described in Section 5.1.1 for the Train6 model using an inference image size of 1088x1088 pixels and a confidence threshold of 42.3%. This confidence threshold was chosen based on where the F1 value is highest, as described previously (visualised in Figure B.7(a)). This table represents the number of Apples that were detected correctly (TP), the number of detections that were miss-classified as Apples (FP), the number of Apples present in each image left to be detected (FN), the number of Double-Detections, which represents the number of fruit that was detected twice, and finally a comparison between the detected fruit and the total amount of fruit present in the Frame, manually counted. This model produced an overall accuracy of 75.7% between the detections and manually counted fruit. The model performed best in Frame4 and Frame6, which are frames with most favourable lighting and fruit being perceptible through the trees. This model also performed better in Frame2 than in Frame3, indicating that the preprocess of changing hue, saturation and value to highlight the red apples might help identify apples. However, this difference is not significant. In the case of Frame10, the model showed an accuracy 82.8% in detecting fruit from a smartphone image, which indicates that this model might perform well in videos recorded from smartphones when directly pointing at trees.

Table 5.8: Performance of the model Train6 in image frames extracted from videos with an inference image size of 1088x640 pixels and a confidence threshold of 42.3%.

Image	TP	FP	FN	Double-Detection	Detected/Human Count (%)
Frame1	112	0	39	2	112/151 (74.2%)
Frame2	101	0	36	0	101/137 (73.7%)
Frame3	99	0	38	1	99/137 (72.3%)
Frame4	70	0	12	0	70/82 (85.4%)
Frame5	54	1	20	0	54/74 (73%)
Frame6	62	0	11	1	62/73 (84.9%)
Frame7	132	0	38	0	132/170 (77.6%)
Frame8	91	0	35	1	91/126 (72.2%)
Frame9	30	0	15	0	30/45 (66.7%)
Frame10	24	0	5	0	24/29 (82.8%)
Total	775	1	249	5	775/1024 (75.7%)

Table 5.9 shows the results for the same experiment performed for the Train6 model, as

described previously. Compared with Train6, Train3 performed relatively better overall, with a 2.6% increase in performance. The Train3 model had one less Double-Detection and one more False Positive than the Train6 model. However, it could also detect more fruit in every frame except in Frame7 and Frame9. These results show that the Train3 model is more fit for the task than the Train6 for almost any image configuration except when there is a 45º angle between the camera and the trees. Contrary to Train6, this model performed better without the enhanced hue, saturation and value to highlight the apples. Both Train3 and Train6 models performed best with the Frame6 using a linear lens, indicating that this type of configuration might be more favourable for fruit counting.

Table 5.9: Performance of the model Train3 in image frames extracted from videos with an inference image size of 1088x640 pixels and a confidence threshold of 41.8%.

Image	TP	FP	FN	Double-Detection	Detected/Human Count (%)
Frame1	121	0	30	2	121/151 (80.1%)
Frame2	103	0	34	1	103/137 (75.2%)
Frame3	108	0	29	1	108/137 (78.8%)
Frame4	71	0	11	0	71/82 (86.6%)
Frame5	56	0	18	0	56/74 (75.7%)
Frame6	67	0	6	0	67/73 (91.8%)
Frame7	119	0	51	0	119/170 (70.0%)
Frame8	106	2	20	0	106/126 (84.1%)
Frame9	29	0	16	0	29/45 (64.4%)
Frame10	25	0	4	0	25/29 (86.2%)
Total	805	2	219	4	805/1024 (78.6%)

To further evaluate the performance of the Train3 model, another test was performed with YOLOv7 in Section 5.1.1. A low confidence threshold of 5% was chosen to determine if the model could achieve close to 100% detection accuracy without drastically increasing the number of False Positives.

Table 5.10 shows the results for this evaluation. There is also a column labelled FPA (False Positives Apple), which corresponds to apples also detected in the image. However, these Apples were placed on the ground or in trees in rows behind where the camera was recording. This value was not considered for intended fruit counting since they were not in the relevant trees. Overall, with a confidence threshold of 5%, the model was able to achieve 95% accuracy throughout the different Frames. The number of False Positives increased by 19, totalling 21, and Double-detections increased by 22, totalling 26. The number of False Positive Apples was 99, meaning the model detected 99 apples that were, in fact, apples but should not be considered for the counting of fruit. After this test, it is possible to conclude that the model can detect almost every single fruit, even if occluded, but at risk of identifying unintended apples, as it will provide an overcounting. It is important to note that the over-counting of fruit was more significant in Frame4, Frame5 and Frame6, while in the other Frames, it was not noticeable.

The final test performed was to count the number of fruits in the video with the help

Table 5.10: Performance of the model Train3 in image frames extracted from videos with an inference image size of 1088x640 pixels and a confidence threshold of 5%.

Image	TP	FPA	FP	FN	Double-Detection	Detected/Human Count (%)
Frame1	141	1	1	10	5	141/151 (93.4%)
Frame2	127	0	5	10	5	127/137 (92.7%)
Frame3	123	0	2	14	2	123/137 (89.8%)
Frame4	81	36	3	1	3	81/82 (98.8%)
Frame5	73	38	3	1	2	73/74 (98.6%)
Frame6	73	12	1	0	2	73/73 (100.0%)
Frame7	164	7	0	6	3	164/170 (96.5%)
Frame8	119	0	4	7	4	119/126 (94.4%)
Frame9	45	0	0	0	0	45/45 (100.0%)
Frame10	29	5	2	0	0	29/29 (100.0%)
Total	975	99	21	49	26	975/1024 (95.2%)

of tracking. YOLOv8 has two native trackers implemented, the Bot-SORT and ByteTrack. The videos chosen for this evaluation were Video1, Video2, and Video3, as with YOLOv7 presented in the final evaluation in Section 5.1.1. The evaluation was performed using the Train3 model as it presented better results than Train6. The video was analysed with an image size of 1088x640 pixels since YOLOv8 modifies the image size to maintain the aspect ratio. The confidence threshold chosen was 41.8% as it is when the model obtains the highest F1 Score as seen previously (visualised in Figure B.3(a)). As stated in Section 5.1.1, Video1, Video2, and Video3 present an estimation of 9328, 7750, and 12056 of apples in each video, respectively.

The Train3 model, in combination with the Bot-SORT tracker, presents a fruit counting of 8028, 9821, and 8317 for Video1, Video2, and Video3, respectively. These values correspond to an accuracy of 86.1%, 126.7%, and 70% accuracy for Video1, Video2, and Video3, respectively. These results are coherent with the evaluation of the model with Frame images. However, in Video2, the model produced an overcounting of fruit. Section 5.2.2 presents a qualitative evaluation of the model after the analysis of each video.

The Train3 model, in combination with the ByteTrack tracker, presents a fruit counting of 6698, 10620 and 6188 for Video1, Video2, and Video3, respectively. This corresponds to an accuracy of 71.8%, 137.0%, and 51.3% for Video1, Video2, and Video3. Combined with this tracker, the model presented an undercounting of fruit in Video1 and Video3 but an overcounting in Video2 compared with the Bot-SORT tracker. Section 5.2.2 presents qualitative results of these trackers after video analysis.

5.2.2 YOLOv8 Qualitative Results

After analysis of each image frame of the evaluation of Train3 and Train6 detection models on Image Frames extracted from videos performed in Section 5.2.1, it was possible to acquire more information. Similar to what was discussed in Section 5.1.2, both models were to detect most of the apples present in the image. However, several apples were

almost completely occluded, making detecting those apples extremely challenging. After analysis of each image frame, it was possible to conclude that the results in Tables 5.8 and 5.9 were coherent with the data visualised in the image frames. With this in mind, the Train3 model showed a better performance in detecting more apples that were occluded than Train6. At the same time, both models were able to detect every visible apple. The only exception to this was in Frame7 and Frame9, where the Train6 model was able to detect more fruit. This indicates that Train6 might be more beneficial when counting fruit in videos recorded with a 45º angle between the camera and the trees. After this, the result image frames from the evaluation of Train3 were analysed with a confidence threshold of 5%. It was possible to conclude that the model detected almost every apple in the image. The ones that were not detected were almost completely occluded apples, which are detectable by a person through context and environment perception. However, they were not visible enough for a model to classify it as an apple. Even though this shows the model's capability to achieve high accuracy, it has drawbacks. The model identified many apples that it was not supposed to. In Images like Frame4, Frame5, Frame6, and Frame7, so many apples from trees behind the target trees and the ground were visible, making the model detect them more easily and produce a fruit overcount.

Regarding the evaluation of the model Train3 in combination with the Bot-SORT tracker in detecting and counting the amount of fruit present in Video1, it was possible to conclude that both the model and tracker were performing exceptionally well. All the visible apples were being detected and tracked throughout the entire time the fruit was in the video. Fruit that suffered occlusions throughout its trajectory was being re-tracked effectively by Bot-SORT without classifying it with a different ID, producing more accurate counting. Fruits not detected at all were entirely occluded by leaves and other fruit, making it very challenging for the model to detect them. The Bot-SORT showed no evidence of ID switching, an issue frequently present in SORT and in specific situations for DeepSORT.

Using the ByteTrack tracker in Video1 provided information for the low count compared to Bot-SORT. It was possible to understand that it was much more difficult to assign a tracking ID to partially occluded fruit. While with Bot-SORT, partially occluded detections were more often tracked, even for only a few frames. Assigning fewer detections with tracks provided a lower fruit count. However, the tracker was also performing exceptionally well in tracking all detections assigned without any ID-switching.

Regarding the evaluation of the model Train3 in combination with the Bot-SORT tracker for fruit counting in Video2, it was possible to conclude that the model and tracker were detecting and tracking all the visible and partially occluded fruit in the target trees. The over-counting was due to the many apples present in the ground or in the tree row behind being detected and tracked. It was then possible to conclude that the model and tracker performed the task as intended with no False Positive detections. To solve the overcounting problem in this video, an increase in the confidence threshold might be beneficial. The tracker again performed no ID-switching, achieving excellent results in tracking detections.

In comparison, ByteTrack applied to the same model performed an even greater over-count. The same problems that arose with Bot-SORT were also verified with this tracker. However, this time, the tracker occasionally performed ID-switching in the apples detected on the trees behind the target trees. Since these detections are smaller and with a lower confidence score, the tracker had more difficulty re-identifying the same detections, assigning new IDs to previously tracked fruit. This was the main observable reason for the over-counting. Besides this, as with Video1, it was seen that the tracker had more difficulty assigning tracks to a partially occluded fruit than Bot-SORT.

Regarding the evaluation of the model Train3 in combination with the Bot-SORT tracker for fruit counting in Video3, it was possible to conclude that the model was performing similarly to Video1. The model detected and correctly tracked every visible and partially occluded fruit in the video without any ID-switching. The lower fruit count can be explained because this video is recorded with a 45º angle to the right side. This implied that leaves and other apples were occluding the right side of the trees, and certain trees could have sections occluded by other trees to their side.

In comparison, using the ByteTrack tracker showed the same issues as in Video1. The tracker was tracking all identified detections correctly and without any ID-switching. However, many apples were visible and partially occluded, with no track assigned, and consequently were not tracked. This resulted in a fruit undercount in comparison with Bot-SORT.

In conclusion, based on the analysis executed, the Bot-SORT provided a more stable tracker for this task and obtained better results. However, it was also understood that specific videos might perform better with higher or lower confidence thresholds than using where the F1 Score is highest for the test set.

5.3 Georeferencing

This section provides examples of the methodology results used for the georeferencing of fruit. GoPro cameras revealed the main challenge in obtaining accurate data for this task. However, finding the intersection of a fruit's straight lines still proved to be a useful solution for this task. To evaluate this method, the Exp26 model was used in a video not used before. This video was used due to its telemetry data. Most videos provided irregular and incoherent data, such as the camera pointing directly at the ground when, in reality, the camera's direction was close to parallel to the ground, and also significant changes in Altitude measurements. Often, the camera registered drastic differences in altitude in few seconds. Considering this, the video used had the telemetry data more normalised. The video used a wide lens and was recorded at 10 km/h in a moving vehicle. The video was also recorded under normal lighting conditions, where apples were visible and recognised, but some shadow was present.

While testing the algorithm for calculating the approximation of the intersection of the straight lines, it was verified that for every frame, an intersection of at least two lines

does not happen. Although there is always an approximation of the point closest to the intersection, this showed that some of the results might be affected. Figures 5.1 and 5.2 represent a front and top view, respectively, of the georeferencing of fruit for seven consecutive frames in the video mentioned. An entire row of apple trees contains several thousands of apples, so visualising an entire tree row is complex. Considering this, small splits of trees are evaluated throughout an entire video to evaluate the accuracy of the method. Comparing Figure 5.1 with the same section in the video, it was possible to conclude that the method could locate fruit accurately with some occasional deviations. In the case of fruit with IDs 522 and 725, although they are very close in the video, contrary to Figure 5.1, the 725 is slightly higher. Fruits 731 and 730 are also represented too much to the right side, and in Figure 5.2, they are also represented excessively behind the tree line. This was because these apples were only detected in the last couple of frames. This implies fewer straight lines to calculate the intersection from, producing more inaccurate results. However, after analysing the following frames, it was concluded that this issue was resolved as more detections for this fruit were acquired. Fruits 454 and 721, as represented in Figure 5.2, are further in the front than the other fruit present in the tree. This is because these fruits are detected for a few frames, leading to an inaccurate approximation of the intersection point.

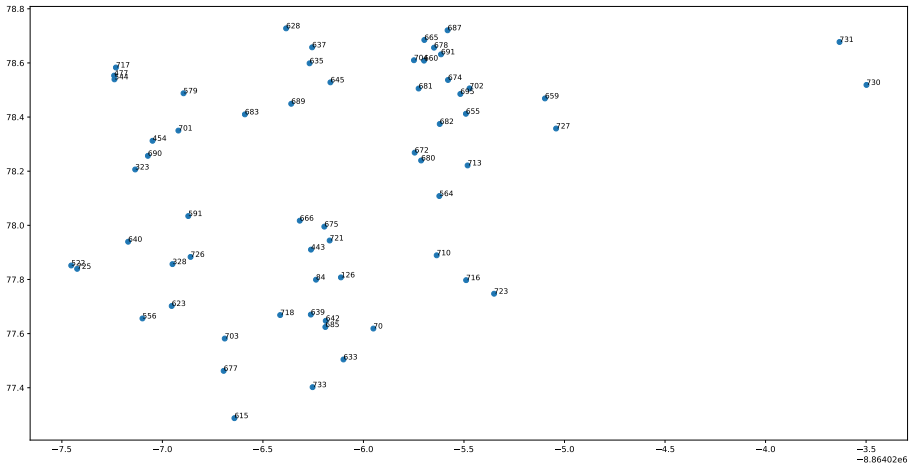


Figure 5.1: Georeferencing of fruit for seven consecutive frames represented in a front view.

These outliers only happen if, in reality, a fruit was only detected for a few frames, which produces an inaccurate approximation of the intersection. After analysis of several sequential frames, it was possible to conclude that these outliers happen occasionally. Nevertheless, this only happens in fruit visible for a small set of frames where the intersection of straight lines is inaccurate.

Although this method presented a practical solution, visualisation is still challenging.

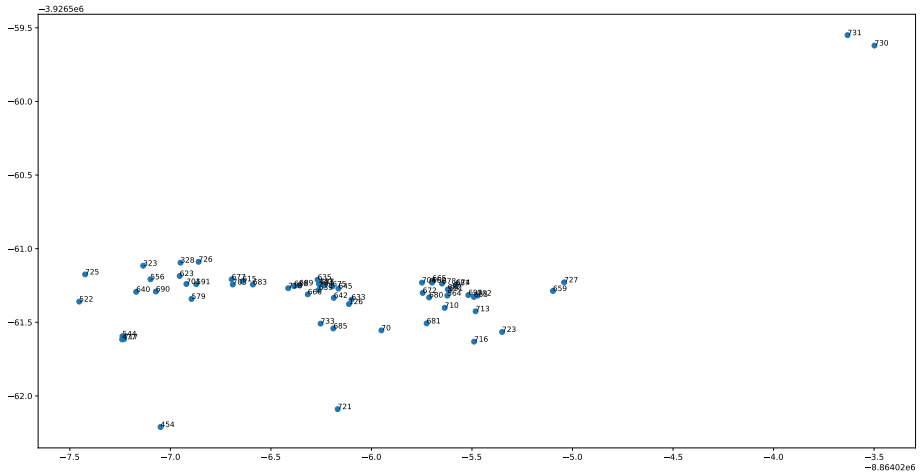


Figure 5.2: Gereferencing of fruit for seven consecutive frames represented in a top view.

An apple orchard can have more than one hundred trees per row, corresponding to thousands of apples across several meters. Thus, difficulting the process of visualisation and analysis of georeferenced fruit.

5.4 Discussion

YOLOv7 and YOLOv8, in combination with tracking systems, provided an effective method for counting fruit in video. The most challenging task was to train a model capable of counting fruit that could perform effectively in various scenarios. All the models were tested in an extensive testing set with new and unseen images, where the models were to maintain high metrics. Besides this, the most effective models were also tested in 10 different image frames with different configurations and from different orchards. In this case, all the models showed above 70% accuracy in detection. However, it is essential to note that fruit counting was measured against fruit counted by a person, including fruit almost entirely occluded and only identified by human perception. However, when the confidence threshold was lower, the models were able to detect with around 90% accuracy, with the drawback of detecting fruit that was not present in the target tree.

After training the Exp-Tree model and evaluating the test set, an additional test was made on a short video to see if the model detected some trees. The resulting video showed that the model rarely would detect a tree accurately. This evaluation demonstrated that detecting trees is a more complex task, as obtaining this data is more difficult, but also due to the complex features of trees.

Exp26 and Train3 were the models that performed best overall for YOLOv7 and YOLOv8, respectively. These models were also used for testing and tracking to count fruit in video. Overall, the YOLOv8 performed better in all tests. Surprisingly, it is a

smaller method and obtained faster inference times compared to YOLOv7. Although the models still showed a lower accuracy than what was visualised, it is essential to note that the results were compared to estimations, making it challenging to have accurate metrics. However, observing the result videos made it possible to conclude that all visible and partially occluded fruit were being detected and tracked effectively. The only fruit almost entirely occluded that solely appeared for a few frames in the entire video would occasionally not be counted.

Considering all the different trackers used in this study, different conclusions were drawn. The SORT tracker, which is the simplest tracker used, performed relatively well. However, the challenges that SORT induces could be visualised. It does not keep track of occluded fruit as effectively as other state-of-the-art tracking systems. On the other hand, DeepSORT proved effective in resolving this issue. Every fruit being tracked throughout its trajectory, if it became occluded for a few frames and then visible again, the track assigned would be the same. However, the DeepSORT also provided an issue with ID-switching. This issue was only present in videos with deformations in the image frame. This was the case for the videos with wide lenses and videos where the wide lenses were stabilised. Since stabilisation performs the inverse effect artificially, some deformations were still present. The DeepSORT tracker extracts feature points for tracking. In this scenario, as the side of the image suffers some deformations, the tracking would extract different feature points and assign a different ID for the same fruit. It is important to note that this ID switching often did not necessarily mean assigning a new ID but also assigning an ID already given to another different fruit. However, the fruit counting did not increase drastically.

Bot-SORT and ByteTrack trackers performed extremely effectively in resolving these issues and tracking fruit through occlusions and deformations of the image. However, Bot-SORT performs slightly better than ByteTrack, as it would more easily assign detections made by the model.

It is imperative that through each video in which counting is performed, some experimentation of the confidence threshold to use must be performed. The confidence threshold was used as a guideline based on the F1 Score value. However, each different video may have a lower or higher confidence threshold that obtains the best results. As seen in Video2, the confidence threshold for the models where F1 Score was highest was confirmed to be too low, detecting fruit that belonged to trees behind the target trees. This implies that a higher confidence threshold should be used for this specific video. The same can be applied to Video1 and Video3, in which a lower confidence threshold would have provided results closer to the estimations. However, the resulting video in each case must be analysed to ensure there are no False Positives. Analysing each video and each configuration, it was also concluded that using an angle towards the trees might make fruit counting challenging. With a 45º angle for filming, even though, in some cases, the fruits behind the tree were clearly detected, many fruits on the opposite side of the camera were occluded and not counted.

A simple evaluation of the image zones where the model would more easily identify fruit was also performed. It was shown that when directly pointing at the trees, the number of detections was well distributed, with a slightly higher number of detections in the centre since, on the sides, the whole tree may not be represented. When the camera is positioned at a 45º angle with trees, it was seen that the model would detect more apples in the far zones of the image. However, in these zones, the right side of the trees is more occluded.

Considering the georeferencing of the fruit task, the method used provided accurate results. However, the importance of accurate telemetry data also became evident. The correct calibration of cameras with access to telemetry will imply a more accurate georeference of fruit. Through analysis of the different frames and localisation of fruit, it was possible to conclude that the method worked effectively. However, fruit must be detected and tracked several times across different frames so that the approximation of the intersection point between all the straight lines of that fruit is as accurate as possible to the actual location of the fruit. In cases where this fails to happen, outliers may appear where the location of the fruit does not seem accurate. These outliers can then be considered as missed detections (detections that do not represent fruit) or fruit that is almost completely occluded, only appearing for a few frames in the video.

In summary, the usage of YOLOv8 with Bot-SORT and YOLOv7 provided a good solution for this task, both in detecting and tracking fruit. However, DeepSORT posed a challenge in images susceptible to deformations, while the YOLOv8 with both trackers provided overall faster inference times.

6

Conclusion

6.1 Conclusion

Counting and georeferencing fruit in an orchard using Deep Learning (DL) methods is essential before fruit harvesting. This allows for a faster and more accurate estimation of fruit production. Georeferencing of fruit gives producers a better insight into the density distribution of their orchards. These two tasks help producers achieve better management and control over production. Helping decrease resource waste and improve efficiency. This study was an entry point in evaluating fast and reliable systems with as minimum investments as possible.

Firstly, a trip to a Portuguese apple orchard was made to acquire data for the execution of this study. Several videos with different devices and configurations were recorded. Some of this data was used for the training, while other was used for testing.

Secondly, several datasets were built to test the performance of the model. However, the main goal was to build the largest dataset with as much variety of data as possible. For this, labelled images from Portuguese orchards' videos and publicly available Apple datasets were combined into a single dataset.

Thirdly, the YOLOv7 and YOLOv8 models were trained with different training parameters to better evaluate the model's implications during training. The trained models were then evaluated in different tests to have a broader view of their performance and how well they were fit for the task. In parallel, the georeference of fruit method was performed and then evaluated. From these evaluations, it was concluded the Exp26 model from YOLOv7 and Train3 from YOLOv8 had the best performance. However, the YOLOv8 with the Bot-SORT tracker provided the most accurate and consistent results of all models and trackers evaluated.

In summary, this study comprised an extensive evaluation of object detection models and tracking systems for counting fruit in videos, as well as the evaluation of a methodology for using this data and georeference fruit. Although the results provided by the model for the counting fruit in video differ from the estimation of the producers, it was possible to conclude that the models were detecting and tracking all visible and partially occluded

fruit in the video. The georeferencing solution provided accurate results with some occasional errors due to miss-detections. Solving this problem using the least squares optimization algorithm provided a fast solution for the number of apples present in a video. However, due to the complexity and number of fruit in an orchard visualisation was challenging.

6.2 Challenges

During the execution of this study, several challenges arose. The training of object detection models needs a vast amount of labelled images. However, labelling such images becomes challenging since identifying and labelling images, including thousands of fruits, is arduous and demanding. Thus, using public datasets and data augmentation methods was essential for this study.

The biggest challenge that was highlighted in this study is the occlusion of fruit. Occlusion can drastically impact the detection and tracking performed in fruit and consequently influence counting. Even though the model can detect and identify partially occluded fruit, when these are entirely occluded, or shadows are affecting their visibility, the detection accuracy might be affected. In an orchard, several factors can result in occlusion, such as a high presence of trees and apples occluding other apples when they are too close to each other. Lighting conditions are another cause for occlusion. If a video is recorded with low sunlight, occluded apples behind leaves might suffer further occlusion through shadows, making it more difficult for the model to detect fruit. Camera motion is another issue that affects counting and georeference of fruit. Suppose a camera during a recording does not have a regular and predictable movement, such as moving side-to-side or up-and-down. In that case, it makes tracking fruit more difficult, leading to more missed detections and inaccurate counting.

Considering the Georeferencing method, it was revealed that it can be effective. However, it was also revealed that with the absence of several detections throughout sequential frames, the intersection point of all straight lines may become inaccurate and challenging. Besides this, accurate telemetry data is also essential for the georeferencing task. If a camera records the altitude, GPS location, and the direction where the camera is pointing, this data is subject to errors and great variety. This can further increase the difficulty in georeferencing fruit. Camera motion may also impact this. If a camera changes direction and moves up and down frequently, georeferencing of fruit is affected.

The evaluation of the model's accuracy in counting fruit in the video was another challenge in the execution of this study. It is complicated to acquire metric precision, as the number of fruit is obtained through estimates. Either through estimations provided by producers with their knowledge of their production or through counting several apple trees and then estimating the value for how many apples are recorded. However, these estimations are almost always prone to some errors. As some trees may have more apples than others, this can produce inaccurate estimations, thus producing less accurate

measurements of the model's performance. However, visualising the resulting videos made it possible to conclude that most apples were being detected and tracked throughout the videos.

6.3 Improvements & Future Work

After the realisation of this study, improvements and future work can still be made to overcome some challenges discussed. The stabilisation of the recording camera is imperative for good results for the model to track detections accurately. Although GoPro cameras have a built-in stabiliser, they may produce some errors. Thus, using external accessories such as a stabiliser can further improve results.

To drastically increase the detection and tracking accuracy, it is necessary to decrease the amount of occlusion. Based on what was observed in the metrics and qualitative results of the models, the models and trackers were accurate in detecting visible and partially occluded apples.

For future work and improvement of the georeferencing of fruit, the size of the bounding boxes obtained can be used to estimate the apple's size. This way, obtaining more information about the density and sizes of the apples in the orchards is possible. This idea moves this task closer to Precision Agriculture using few resources.

Bibliography

[1] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky. "BoT-SORT: Robust associations multi-pedestrian tracking". In: *arXiv preprint arXiv:2206.14651* (2022) (cit. on p. 47).

[2] S. ALBAWI, T. A. MOHAMMED, and S. AL-ZAWI. *Understanding of a Convolutional Neural Network*. 2017. URL: https://ieeexplore.ieee.org/abstract/document/8308186?casa_token=iJYN4JaeJnkAAAAA:h8tIh0H5s5jnDtfB-iTD_rYU73Yx0oKxjButfElLTUFk8c2q_ZawmWW8_u5IbfBQRvnbiQmg4w (cit. on pp. 1, 5).

[3] N. H. Ali and G. M. Hassan. "Kalman filter tracking". In: *International Journal of Computer Applications* 89.9 (2014) (cit. on p. 9).

[4] P. Antoniadis. *Differences between epoch, batch, and Mini-batch*. 2022-11. URL: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch (cit. on p. 18).

[5] Appledetector. *Appledetector Dataset*. Open Source Dataset. visited on 2023-02-01. 2022-06. URL: https://universe.roboflow.com/appledetector-cqnh0/appledetector (cit. on p. 29).

[6] Bachelorarbeit. *Apples Dataset*. Open Source Dataset. 2023-06. URL: https://universe.roboflow.com/bachelorarbeit/apples-fk4us (cit. on p. 33).

[7] N. Barla. *The Complete Guide to Object Tracking [+V7 tutorial]*. URL: https://www.v7labs.com/blog/object-tracking-guide (cit. on p. 4).

[8] A. Bathija and G. Sharma. "Visual object detection and tracking using Yolo and sort". In: *International Journal of Engineering Research Technology* 8.11 (2019) (cit. on p. 5).

[9] A. Bewley et al. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003 (cit. on pp. 44, 45).

[10] G. Bishop, G. Welch, et al. "An introduction to the kalman filter". In: *Proc of SIGGRAPH, Course* 8.27599-23175 (2001), p. 41 (cit. on p. 8).

[11] J. Brownlee. *Understand the impact of learning rate on neural network performance*. 2020-09. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (cit. on p. 18).

[12] N. Buhl. *Yolo models for object detection explained [yolov8 updated]*. 2023-02. URL: https://encord.com/blog/yolo-object-detection-guide/#:~:text=YOLOv8%20is%20the%20latest%20iteration,loss%20function%20for%20increased%20accuracy. (cit. on pp. 14, 15, 35).

[13] S. Challa. *Fundamentals of Object Tracking*. Cambridge University Press, 2011 (cit. on p. 4).

[14] Y.-R. Chen, K. Chao, and M. S. Kim. "Machine vision technology for agricultural applications". In: *Computers and Electronics in Agriculture* 36.2 (2002), pp. 173–191. ISSN: 0168-1699. DOI: https://doi.org/10.1016/S0168-1699(02)00100-X. URL: https://www.sciencedirect.com/science/article/pii/S016816990200100X (cit. on p. 1).

[15] J. Cohen. *Computer Vision for tracking*. 2021-08. URL: https://thinkautonomous.medium.com/computer-vision-for-tracking-8220759eee85 (cit. on p. 11).

[16] *Convolutional Neural Networks (cnns) explained*. URL: https://deeplizard.com/learn/video/YRhxdVk_sIs (cit. on pp. 7, 8).

[17] Emre. *datasaet3 Dataset*. Open Source Dataset. visited on 2023-02-01. 2022-10. URL: https://universe.roboflow.com/emre-zifto/datasaet3 (cit. on p. 29).

[18] M. M. Faisal. *Yolov7 object tracking using PYTORCH, opencv and DeepSORT*. 2023-01. URL: https://github.com/MuhammadMoinFaisal/YOLOv7-DeepSORT-Object-Tracking (cit. on p. 45).

[19] FTs. *Apple Dataset*. Open Source Dataset. 2022-08. URL: https://universe.roboflow.com/fts/apple-emogu (cit. on p. 32).

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 17).

[21] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on pp. 17, 22).

[22] K. Itakura et al. "Automatic pear and apple detection by videos using deep learning and a Kalman filter". In: *OSA Continuum* 4.5 (2021-05), pp. 1688–1695. DOI: 10.1364/OSAC.424583. URL: https://opg.optica.org/osac/abstract.cfm?URI=osac-4-5-1688 (cit. on pp. 17–19).

[23] R. Khandelwal. *Hungarian algorithm*. 2022-03. URL: https://arshren.medium.com/hungarian-algorithm-6cde8c4065a3 (cit. on p. 11).

[24] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 22).

[25] A. Koirala et al. *Deep learning for real-time fruit detection and Orchard Fruit Load Estimation: Benchmarking of 'mangoyolo' - precision agriculture*. 2019-02. URL: https://link.springer.com/article/10.1007/s11119-019-09642-0 (cit. on p. 19).

[26] M. Li et al. "Agricultural greenhouses detection in high-resolution satellite images based on convolutional neural networks: Comparison of faster R-CNN, YOLO v3 and SSD". In: *Sensors* 20.17 (2020), p. 4938 (cit. on p. 4).

[27] T. Liu. *apple yolo5 2 Dataset*. Open Source Dataset. visited on 2023-02-01. 2022-03. URL: https://universe.roboflow.com/tianhao-liu/apple_yolo5_2 (cit. on p. 29).

[28] T. Liu. *apple yolo5 Dataset*. Open Source Dataset. visited on 2023-02-01. 2022-03. URL: https://universe.roboflow.com/tianhao-liu/apple_yolo5 (cit. on p. 29).

[29] X. Liu et al. "Robust Fruit Counting: Combining Deep Learning, Tracking, and Structure from Motion". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1045–1052. DOI: 10.1109/IROS.2018.8594239 (cit. on pp. 23–25).

[30] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[31] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110 (cit. on p. 24).

[32] Manzanas. *DS-PG2-Apples Dataset*. Open Source Dataset. 2023-03. URL: https://universe.roboflow.com/manzanas/ds-pg2-apples (cit. on p. 33).

[33] Manzanas. *DS-PG7-MinneApple Dataset*. Open Source Dataset. 2023-03. URL: https://universe.roboflow.com/manzanas/ds-pg7-minneapple (cit. on p. 33).

[34] J. Marks. *Computer vision in agriculture 2023*. 2023-01. URL: https://voxel51.com/blog/how-computer-vision-is-changing-agriculture-in-2023/ (cit. on pp. 1, 2).

[35] A. Mehra. *Understanding yolov8 architecture, applications & features*. 2023-06. URL: https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/#:~:text=A%20modified%20version%20of%20the,flow%20between%20the%20different%20layers. (cit. on p. 14).

[36] Mehul. *Object tracking in videos: Introduction and common techniques*. 2020-10. URL: https://aidetic.in/blog/2020/10/05/object-tracking-in-videos-introduction-and-common-techniques/ (cit. on p. 5).

[37] R. J. Meinhold and N. D. Singpurwalla. "Understanding the Kalman filter". In: *The American Statistician* 37.2 (1983), pp. 123–127 (cit. on p. 8).

[38] K. Moore, N. Landman, and J. Khim. *Hungarian maximum matching algorithm*. URL: https://brilliant.org/wiki/hungarian-matching/ (cit. on p. 11).

[39] A. Nagpal. *L1 and L2 regularization methods*. 2017-10. URL: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c (cit. on p. 18).

[40] K. O'Shea and R. Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015) (cit. on pp. 6–8).

[41] *Object detection guide*. URL: https://www.fritz.ai/object-detection/ (cit. on p. 4).

[42] M. Rahnemoonfar and C. Sheppard. "Deep count: fruit counting based on deep simulated learning". In: *Sensors* 17.4 (2017), p. 905 (cit. on pp. 21–23).

[43] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017-07 (cit. on p. 17).

[44] M. M. Rizwan. *Yolov7 architecture explanation*. URL: https://www.plugger.ai/blog/yolov7-architecture-explanation (cit. on pp. 12–14).

[45] RizwanMunawar. *Yolov7-object-tracking: Yolov7 object tracking using pytorch, opencv and sort tracking*. URL: https://github.com/RizwanMunawar/yolov7-object-tracking (cit. on p. 44).

[46] F. L. de la Rosa et al. "Geometric transformation-based data augmentation on defect classification of segmented images of semiconductor materials using a ResNet50 convolutional neural network". In: *Expert Systems with Applications* 206 (2022), p. 117731. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.117731. URL: https://www.sciencedirect.com/science/article/pii/S0957417422010120 (cit. on p. 10).

[47] Sadasd. *fullindoorsapple Dataset*. Open Source Dataset. visited on 2023-02-01. 2022-10. URL: https://universe.roboflow.com/sadasd/fullindoorsapple (cit. on p. 29).

[48] Sanyam. *Understanding multiple object tracking using DeepSORT*. 2022-11. URL: https://learnopencv.com/understanding-multiple-object-tracking-using-deepsort/ (cit. on p. 45).

[49] A. Segunpta. *Assignment problem: Meaning, methods and variations: Operations research*. 2017-03. URL: https://www.engineeringenotes.com/project-management-2/operations-research/assignment-problem-meaning-methods-and-variations-operations-research/15652 (cit. on p. 10).

[50] D. Shah. *The Essential Guide to data augmentation in Deep Learning*. URL: https://www.v7labs.com/blog/data-augmentation-guide (cit. on p. 10).

[51] D. Simon. "Kalman filtering". In: *Embedded systems programming* 14.6 (2001), pp. 72–79 (cit. on pp. 8, 9).

[52] J. Solawetz. *What is Yolov8? the ultimate guide*. 2023-01. URL: https://blog.roboflow.com/whats-new-in-yolov8/ (cit. on pp. 2, 14, 15).

[53] S. Srinivasan. *Kalman filter: An algorithm for making sense from the insights of various sensors fused together.* 2018-04. URL: https://towardsdatascience.com/kalman-filter-an-algorithm-for-making-sense-from-the-insights-of-various-sensors-fused-together-ddf67597f35e (cit. on p. 9).

[54] C. Szegedy, S. Ioffe, and V. Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261 (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261 (cit. on p. 22).

[55] J. Traa. "Least-squares intersection of lines". In: *University of Illinois Urbana-Champaign (UIUC)* (2013) (cit. on p. 50).

[56] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *arXiv preprint arXiv:2207.02696* (2022) (cit. on pp. 2, 12–14, 44).

[57] Z. Wang, K. Walsh, and A. Koirala. "Mango fruit load estimation using a video based MangoYOLO—Kalman filter—hungarian algorithm method". In: *Sensors* 19.12 (2019), p. 2742 (cit. on pp. 19–21).

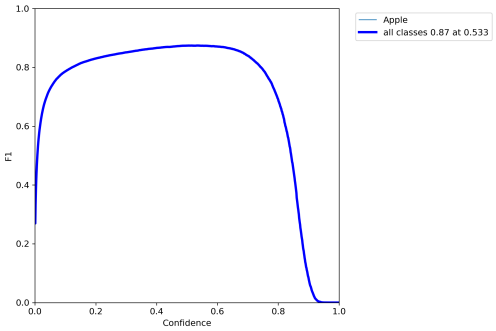
[58] Wangwang. *Apple Dataset*. Open Source Dataset. 2022-05. URL: https://universe.roboflow.com/wangwang/apple-shybf (cit. on p. 32).

[59] C. Wolfe. *Why 0.9? towards better momentum strategies in deep learning.* 2021-02. URL: https://towardsdatascience.com/why-0-9-towards-better-momentum-strategies-in-deep-learning-827408503650 (cit. on p. 18).

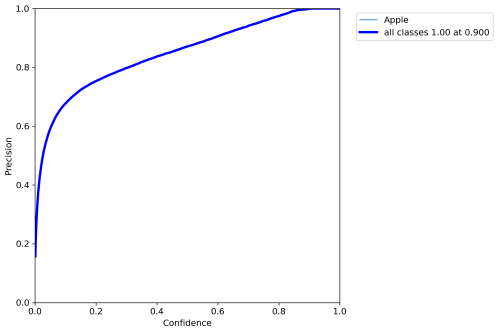
[60] Y. Zhang et al. "Bytetrack: Multi-object tracking by associating every detection box". In: *European Conference on Computer Vision*. Springer. 2022, pp. 1–21 (cit. on p. 46).

A

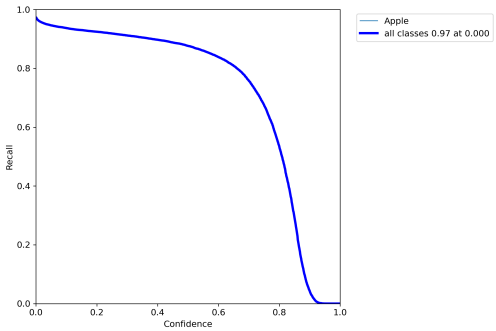
YOLOv7 Metric Graphs



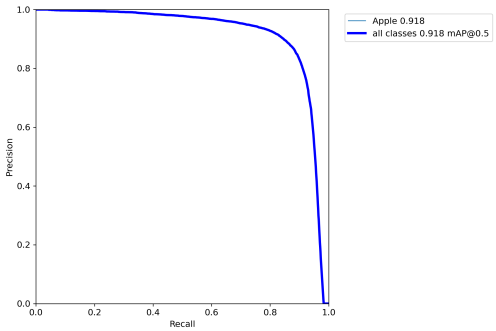
(a) F1 Curve



(b) Precision Curve

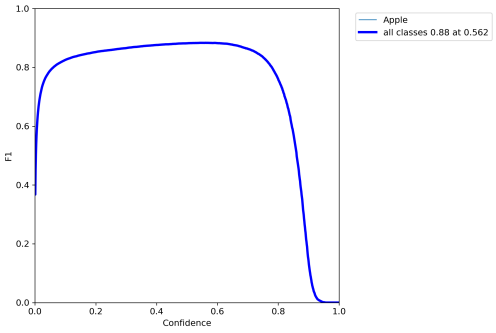


(c) Recall Curve

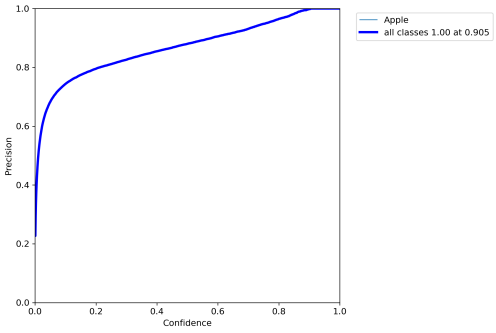


(d) Precision-Recall Curve

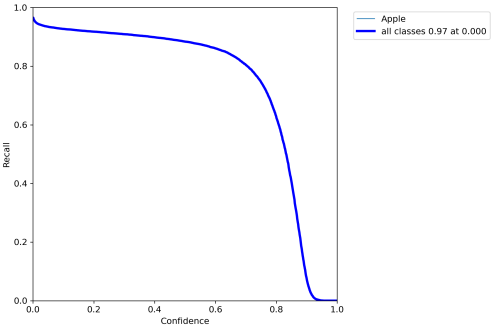
Figure A.1: Metric Graphs and Curves of model Exp25 for 1088x1088 pixels



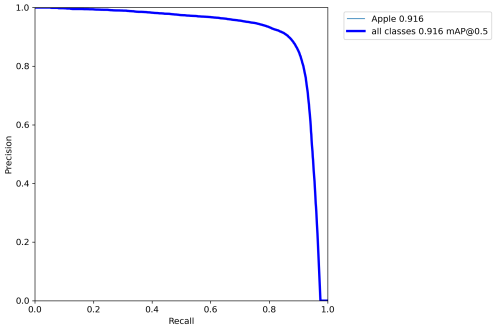
(a) F1 Curve



(b) Precision Curve

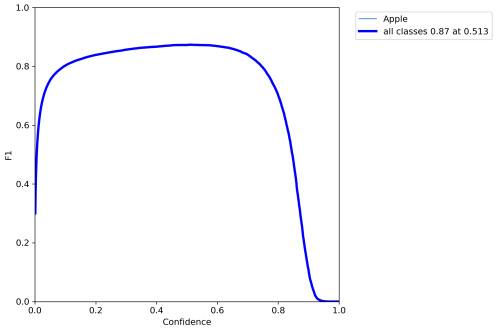


(c) Recall Curve

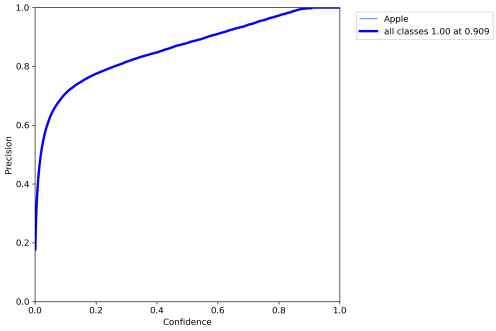


(d) Precision-Recall Curve

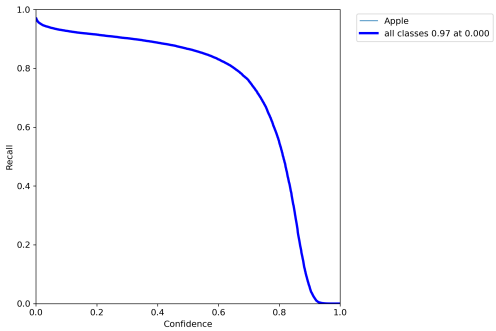
Figure A.2: Metric Graphs and Curves of model Exp26 for 1088x1088 pixels



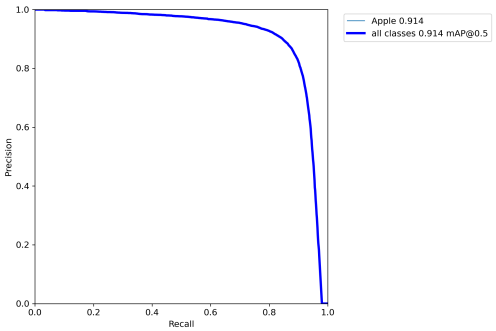
(a) F1 Curve



(b) Precision Curve

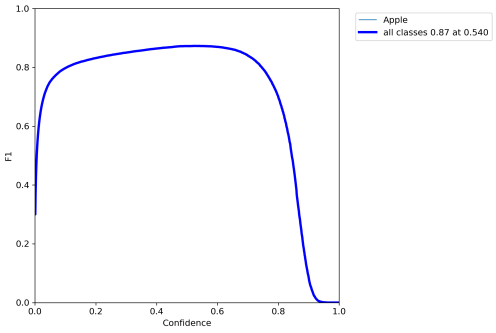


(c) Recall Curve

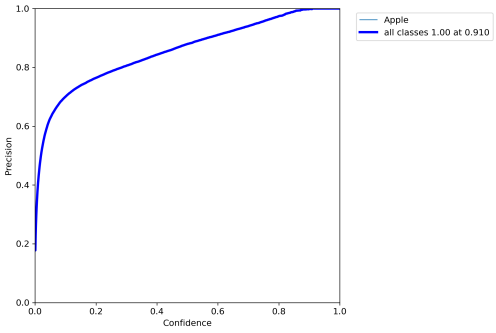


(d) Precision-Recall Curve

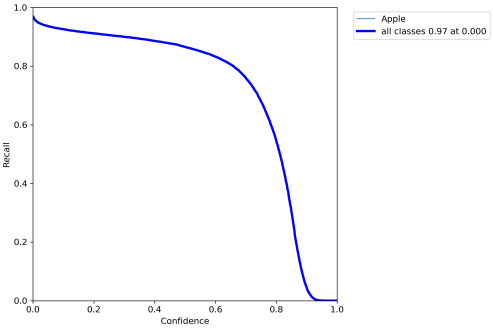
Figure A.3: Metric Graphs and Curves of model Exp43 for 1088x1088 pixels



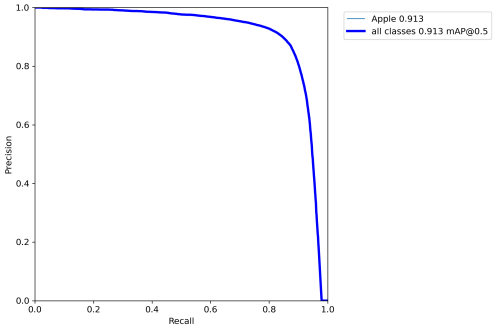
(a) F1 Curve



(b) Precision Curve

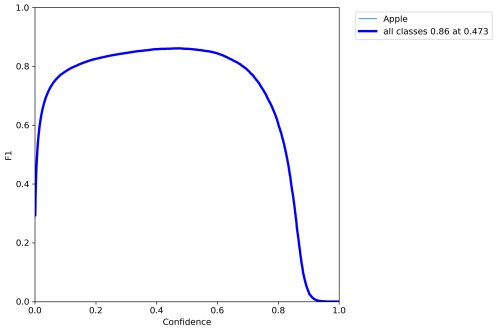


(c) Recall Curve

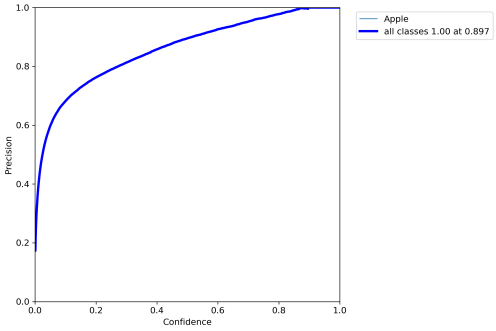


(d) Precision-Recall Curve

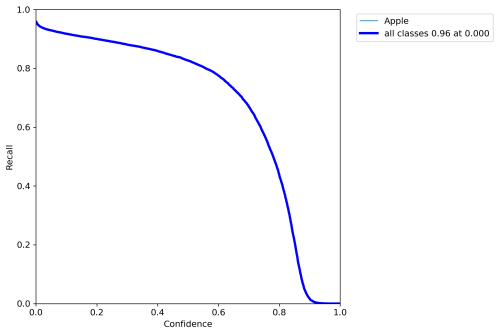
Figure A.4: Metric Graphs and Curves of model Evolve for 1088x1088 pixels



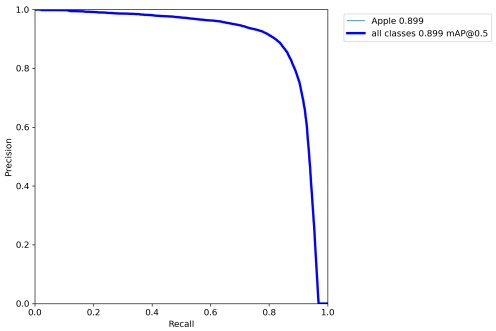
(a) F1 Curve



(b) Precision Curve

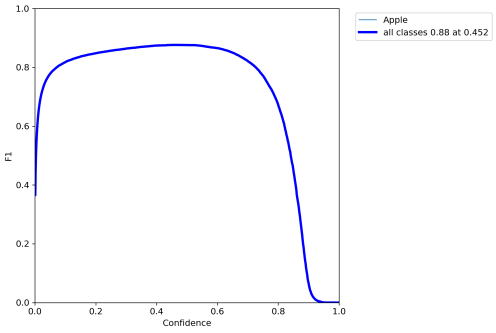


(c) Recall Curve

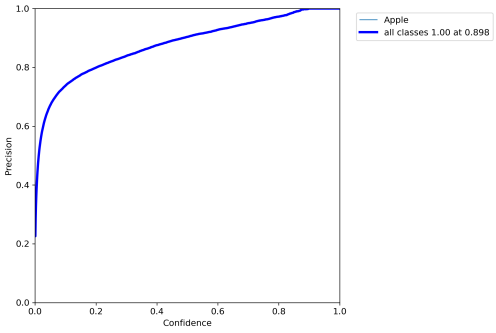


(d) Precision-Recall Curve

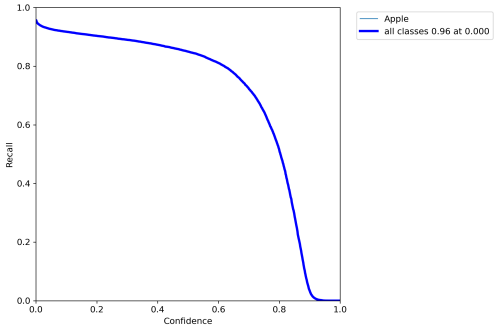
Figure A.5: Metric Graphs and Curves of model Exp25 for 640x640 pixels



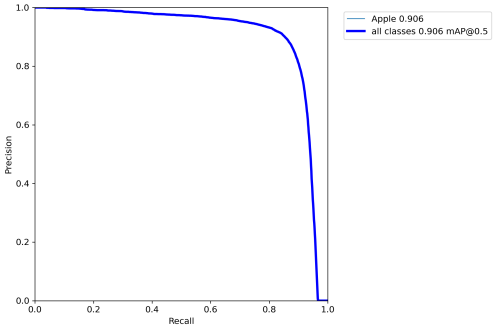
(a) F1 Curve



(b) Precision Curve

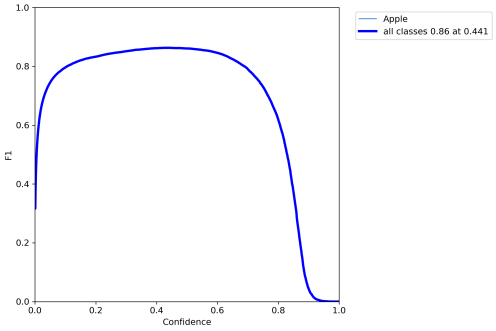


(c) Recall Curve

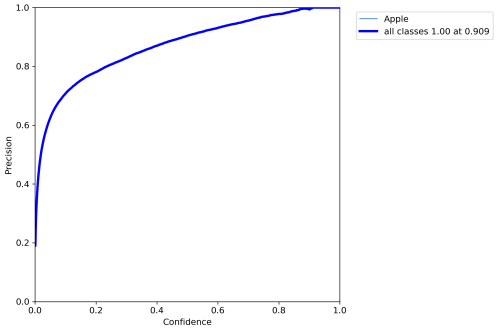


(d) Precision-Recall Curve

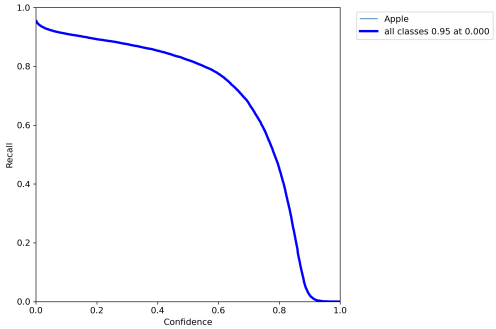
Figure A.6: Metric Graphs and Curves of model Exp26 for 640x640 pixels



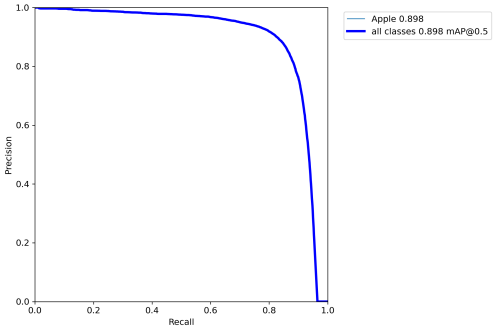
(a) F1 Curve



(b) Precision Curve

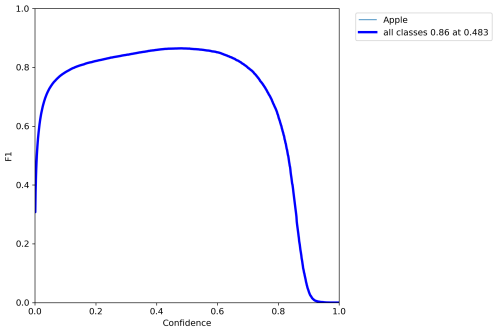


(c) Recall Curve

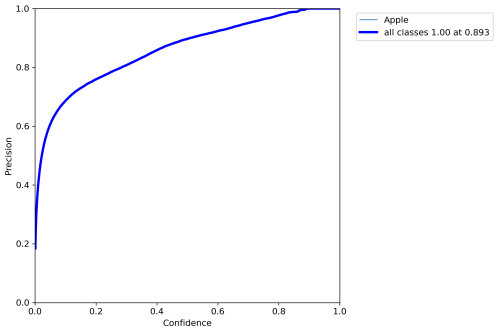


(d) Precision-Recall Curve

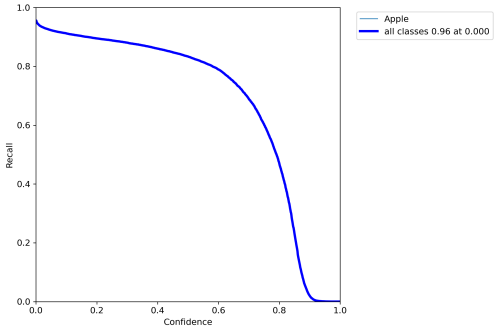
Figure A.7: Metric Graphs and Curves of model Exp43 for 640x640 pixels



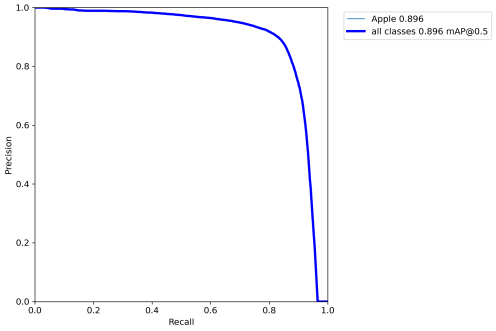
(a) F1 Curve



(b) Precision Curve



(c) Recall Curve

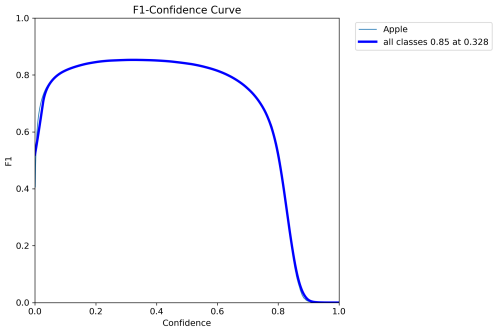


(d) Precision-Recall Curve

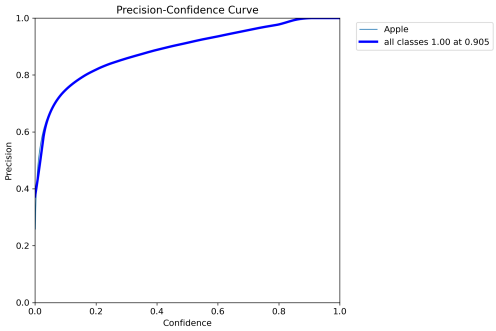
Figure A.8: Metric Graphs and Curves of model Evolve for 640x640 pixels

B

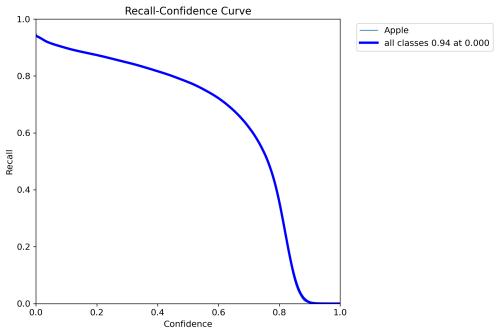
YOLOv8 Metric Graphs



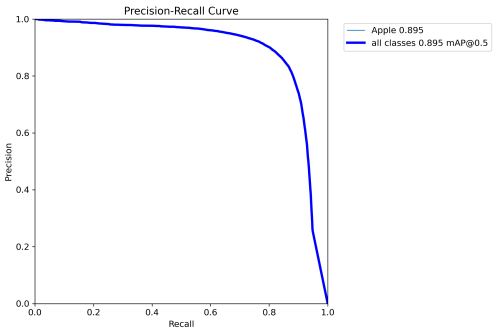
(a) F1 Curve



(b) Precision Curve



(c) Recall Curve



(d) Precision-Recall Curve

Figure B.1: Metric Graphs and Curves of model train3 for 640x640 pixels images

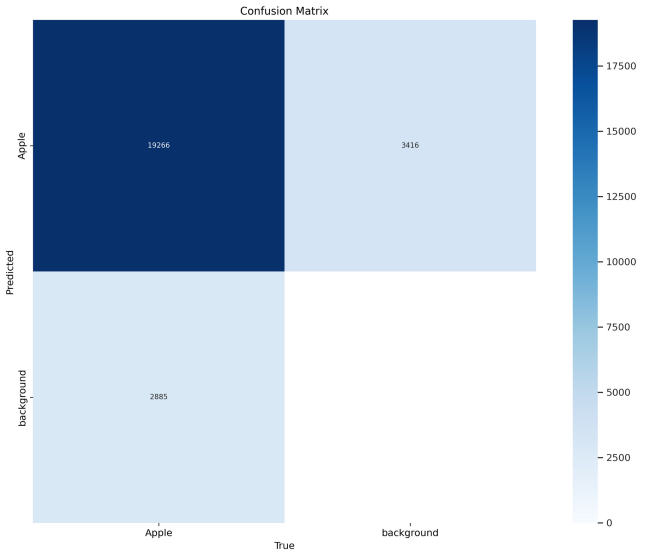
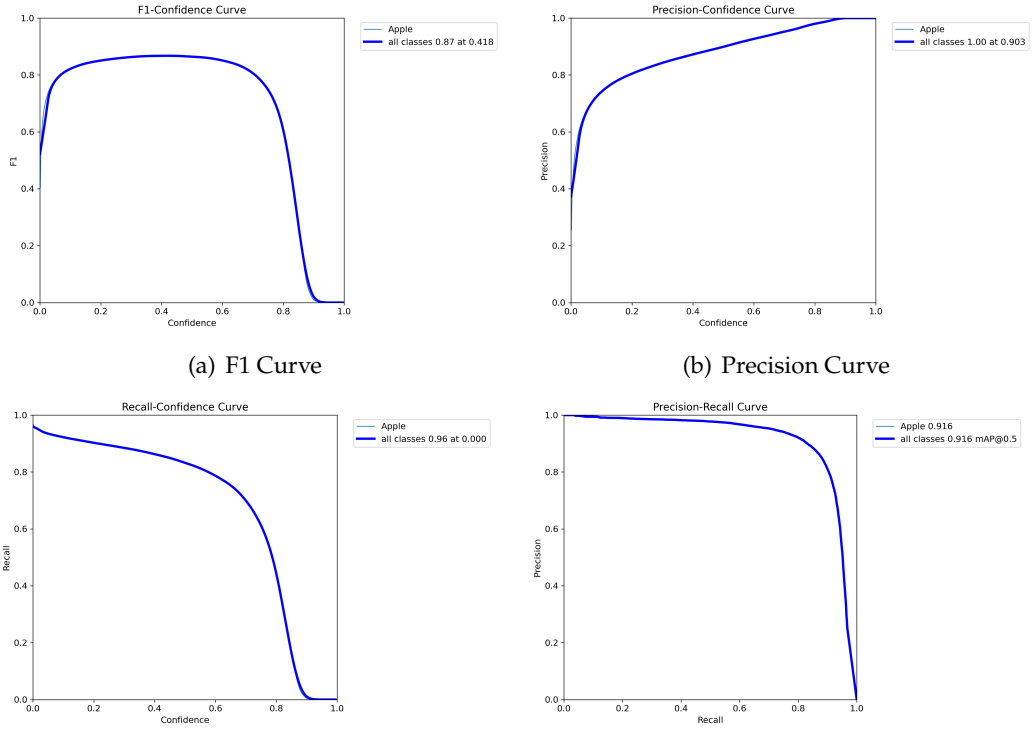


Figure B.2: Confusion Matrix of model3 for 640x640 pixels images



(a) F1 Curve

(b) Precision Curve

(c) Recall Curve

(d) Precision-Recall Curve

Figure B.3: Metric Graphs and Curves of model train3 for 1088x1088 pixels images

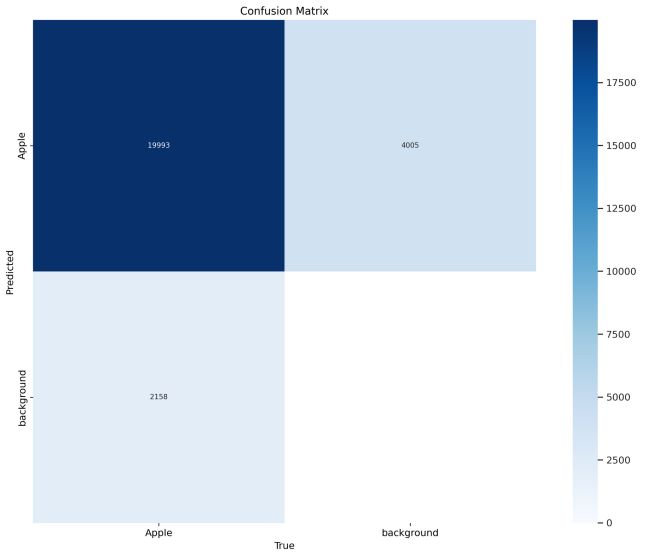
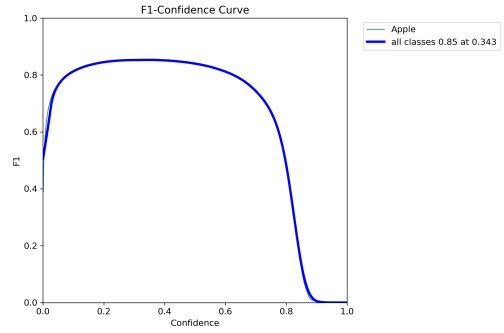
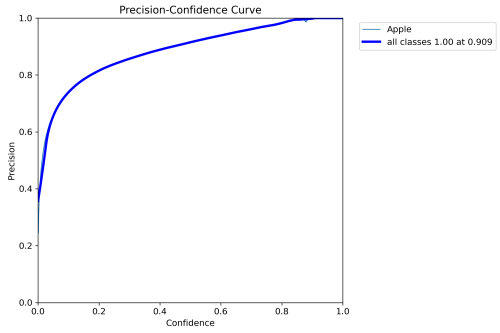


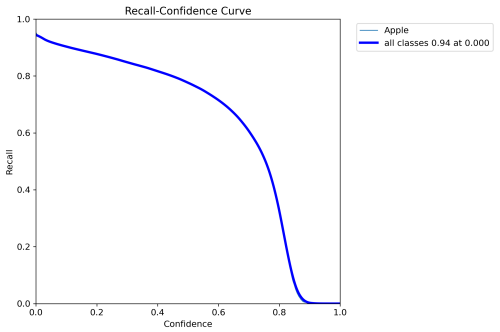
Figure B.4: Confusion Matrix of model3 for 1088x1088 pixels images



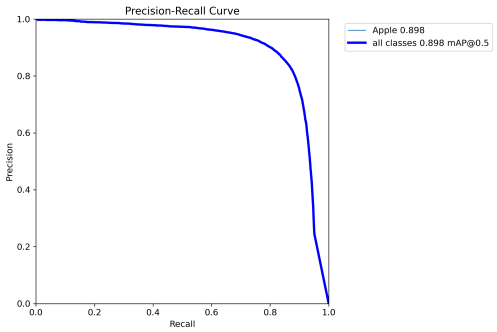
(a) F1 Curve



(b) Precision Curve



(c) Recall Curve



(d) Precision-Recall Curve

Figure B.5: Metric Graphs and Curves of model train6 for 640x640 pixels images

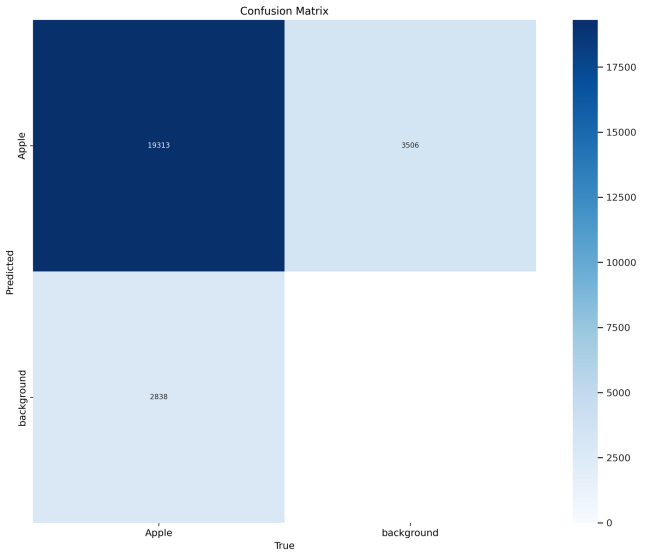
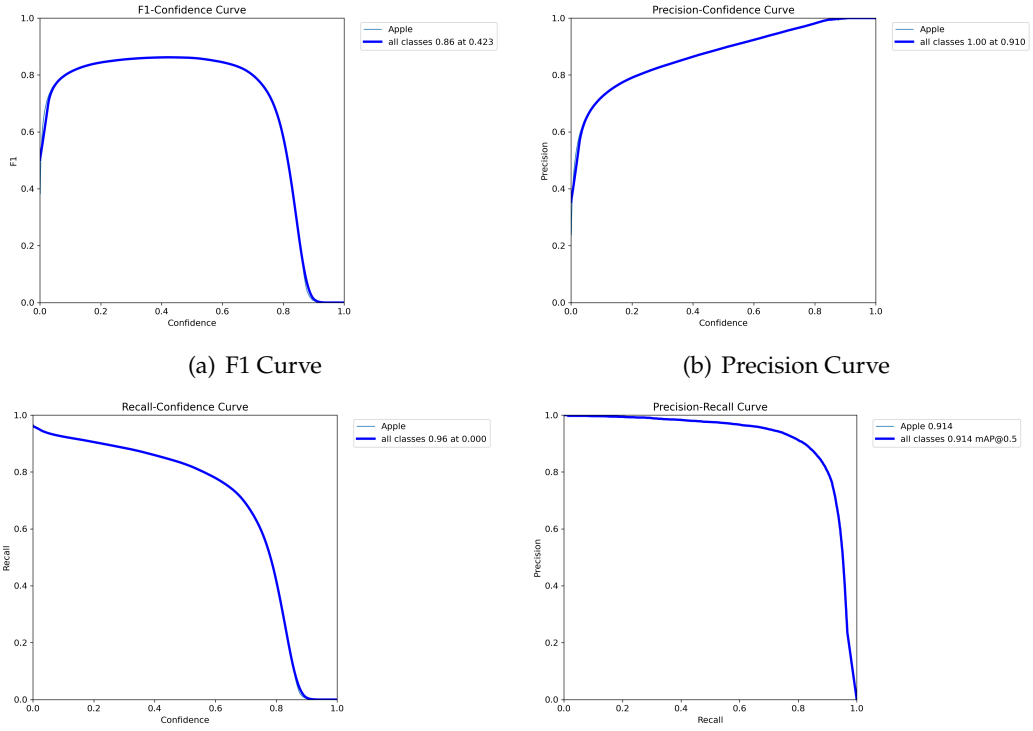


Figure B.6: Confusion Matrix of model3 for 640x640 pixels images



(a) F1 Curve

(b) Precision Curve

(c) Recall Curve

(d) Precision-Recall Curve

Figure B.7: Metric Graphs and Curves of model train6 for 1088x1088 pixels images

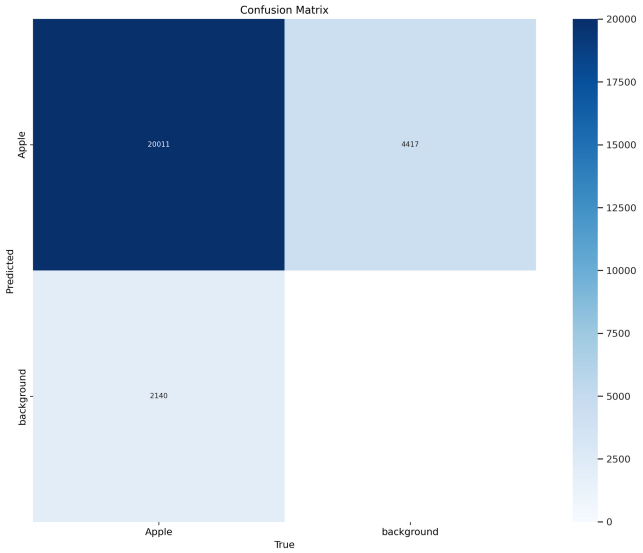


Figure B.8: Confusion Matrix of model6 for 1088x1088 pixels images

C

YOLOv7 Frames Results

Figure C.1: Frame1 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.2: Frame2 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.3: Frame3 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.4: Frame4 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.5: Frame5 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.6: Frame6 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.7: Frame7 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.8: Frame8 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.9: Frame9 Result of Exp25 model with 56.2% confidence threshold from table 5.2.

Figure C.10: Frame10 Result of Exp25 model with 56.2% confidence threshold from table 5.2.