Disease Detection and Mapping in Olive Groves using UAVs and Deep Learning for Precision Agriculture

Maurício Herche Fófano de Morais

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Electrical and Computer Engineering. Work developed under the Dual Degree Program between the Polytechnic Institute of Bragança (IPB) and the Federal Center for Technological Education of Minas Gerais (CEFET-MG).

Work oriented by:

Prof. Doctor José Luís Sousa de Magalhães Lima

Prof. Doctor Murillo Ferreira dos Santos

Prof. Doctor João Carlos Sampaio Maldonado Costa Mendes

Bragança

2025

Disease Detection and Mapping in Olive Groves using UAVs and Deep Learning for Precision Agriculture

Maurício Herche Fófano de Morais

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Electrical and Computer Engineering. Work developed under the Dual Degree Program between the Polytechnic Institute of Bragança (IPB) and the Federal Center for Technological Education of Minas Gerais (CEFET-MG).

Work oriented by:

Prof. Doctor José Luís Sousa de Magalhães Lima

Prof. Doctor Murillo Ferreira dos Santos

Prof. Doctor João Carlos Sampaio Maldonado Costa Mendes

Bragança

2025

Dedication

I dedicate this work to my parents, Marcio and Diana, who encouraged and supported me throughout my journey and helped make it possible to finish this work.

Acknowledgement

The journey from being someone who rarely left their room and comfort zone to becoming a person who travelled alone to another country, far from friends and family, to participate in a dual degree program, was arduous and filled with obstacles and uncertainties. Although it may seen like a solitary path, in the end, the opportunity to be far from home has been invaluable for self development and personal growth. This journey was only possible thanks to the support of many people, to whom i would like to express my deepest gratitude:

To my parents, Marcio and Diana, for all their love, emotional and financial support, and for encouraging me to seize this opportunity.

To my advisors, José Lima, Murillo Ferreira, and João Mendes, for their guidance, support, and for providing me with opportunities throughout this journey, including the change to travel and visit a lot of countries as a result of my work.

To my friends, Iury Rodrigues, Iago Rangel, Rian Marques, and Bruno Resende, for being by my side during the many years spent at CEFET, and to my friends Gabriel, Matheus, Arielle, and Renato for all of their support and for the moments we shared together at Bragança.

I am extremely grateful to CEFET-MG, IPB, and CeDRI for all the opportunities and facilities provided to me during this dual degree journey. I would also like to thank Eng. José Batista for allowing me to use his olive grove as the test field for this dissertation, as well as Mr. Guido Berger, for helping me with the collection of all the data necessary to conduct this work.

Finally, I extend my sincere tranks to everyone who, in some way or another, contributed to the completion off this work. I would also like to thank the National Council for Scientific and Technological Development (CNPq) for all the support provided to this project.

Abstract

This dissertation presents the implementation and validation of a cost-effective, Unmanned Aerial Vehicle (UAV) based system for automated detection and spatial mapping of olive knot disease in olive groves. Addressing the need for accessible and efficient plant disease monitoring in Precision Agriculture (PA), the proposed methodology leverages existing UAV imagery capabilities with lightweight Deep Learning (DL) models, specifically the You Only Look Once (YOLO) object detection architecture, to enable scalable and accurate detection. The academic contributions presented in this work have resulted in two peer-reviewed publications related to the dissertation topic, as detailed at the end of this document. An annotated dataset of UAV-acquired images was compiled, and several state-of-the-art YOLO object detection models were trained and evaluated under identical conditions. The best-performing model achieved a strong $F1$-score, demonstrating good results in detecting olive knot disease and accurately mapping its spatial distribution within the plantation. The workflow integrates spatial cross-referencing of detections with UAV flight path data and proximity analysis, enabling the assignment of disease detections to individual trees. An interactive map interface, developed using the *Folium* Python library, provides visualization of the disease distribution and supports practical grove management. The experimental results indicate that cost-effective UAVs and lightweight DL models can be effectively combined for plant disease detection and spatial analysis, offering a robust and scalable approach for real-world agricultural applications. Limitations regarding early symptom detection and image quality are discussed, and directions for future work are proposed.

Keywords: Unmanned Aerial Vehicle, Disease Detection, Olive Knot, YOLO, Computer Vision, Deep Learning.

Resumo

Esta dissertação apresenta a implementação e a validação de um sistema economicamente viável baseado em Veículo Aéreo Não Tripulado (VANT) para detecção automatizada e mapeamento espacial da doença nó da oliveira em olivais. Atendendo à necessidade de monitoramento eficiente e acessível de doenças em plantas no âmbito de agricultura de precisão, a metodologia proposta utiliza as capacidades existentes de imagens de VANT em conjunto com modelos computacionalmente leves de aprendizado profundo, utilizando especificamente a arquitetura de detecção de objetos YOLO, permitindo uma detecção escalável e precisa. As contribuições acadêmicas apresentadas neste trabalho resultaram em duas publicações científicas revisadas por pares relacionadas ao tema da dissertação, conforme detalhado no final deste documento. Foi compilado um conjunto de dados anotados a partir de imagens obtidas por VANT, e diversos modelos YOLO de detecção de objetos, considerados estado da arte, foram treinados e avaliados sob as mesmas condições. O modelo de melhor desempenho obteve um $F1$-score elevado, demonstrando bons resultados na detecção da doença e no mapeamento de sua distribuição espacial dentro da plantação. O fluxo de trabalho utilizado integra o referenciamento cruzado das detecções com dados da trajetória do VANT e análise de proximidade, possibilitando a atribuição das detecções de doença a árvores individuais. Um mapa interativo, desenvolvido com a biblioteca Python *Folium*, fornece a visualização da distribuição da doença e apoia a tomada de decisões no olival. Os resultados experimentais indicam que VANTs com boa relação custo-benefício e modelos computacionalmente leves de aprendizado profundo como o YOLO podem ser combinados de forma eficaz para detecção de doenças em plantas e análise espacial, oferecendo uma abordagem robusta e escalável para aplicações agrícolas reais. As limitações

relacionadas à detecção de sintomas iniciais e à qualidade das imagens são discutidas, e direções para trabalhos futuros são sugeridas.

Palavras-chave: Veículo Aéreo Não Tripulado, Detecção de Doenças, Nó da Oliveira, YOLO, Visão Computacional, Aprendizado Profundo.

Contents

List of Tables

List of Figures

Acronyms

AI Artificial Inteligence

CNN Convolutional Neural Network

CV Computer Vision

DL Deep Learning

FLOP Floating-Point Operations per Second

FN False Negative

FP False Positive

GPS Global Positioning System

GPU Graphics Processing Unit

IDE Integrated Development Environment

IoU Intersection over Union

mAP Mean Average Precision

ML Machine Learning

MSE Mean Squared Error

P Precision

PA Precision Agriculture

R Recall

R-CNN Region-based Convolutional Neural Network

RGB Red-Green-Blue

SSD Single Shot Multibox Detector

TP True Positive

UAV Unmanned Aerial Vehicle

YOLO You Only Look Once

Chapter 1

Introduction

Precision Agriculture (PA) has revolutionized modern farming by integrating advanced technologies such as Global Positioning System (GPS), sensors, Unmanned Aerial Vehicles (UAVs), and robotics to enhance productivity, sustainability, and address global food security challenges [1]. By leveraging these technologies, farmers can make data-driven decisions that minimize resource waste and environmental impact while maximizing crop yields [2]. PA provides precise insights into soil health, crop conditions, and ecological factors, paving the way for more efficient and sustainable agricultural practices [3].

In the context of PA, UAVs combined with Computer Vision (CV) have emerged as powerful tools, particularly for the detection and monitoring of plant diseases. High-resolution cameras and advanced sensors on UAVs can capture detailed aerial imagery of large agricultural areas in a fraction of the time required for manual inspections [4]. CV algorithms process this imagery to identify patterns, anomalies, or symptoms indicative of diseases, such as discolouration, wilting, or lesions on leaves and stems [5].

For instance, UAVs equipped with Red-Green-Blue (RGB) and multispectral cameras have been effectively used for precise disease identification and monitoring [2], [6]. By integrating Machine Learning (ML) models, such as Deep Learning (DL) architectures, these systems can accurately identify specific diseases, even in complex and heterogeneous environments [7]. The combination of UAVs and CV not only enhances the scalability and efficiency of disease detection but also reduces reliance on labour-intensive methods,

contributing to more sustainable and precise crop management practices [8].

Olive farming is deeply rooted in Portuguese agricultural tradition, with olive trees believed to have been introduced over 3000 years ago [9]. Today, Portugal stands as one of the world's leading producers of high-quality olive oil, ranking sixth globally and preparing to become the third-largest producer by 2026 [10], [11]. The country has witnessed a remarkable evolution in its olive sector, with production multiplying fivefold since the early 21^{st} century [12], [13]. This growth is primarily attributed to modernized practices, including intensive and super-intensive practices [14]. These advancements have enabled Portugal to achieve self-sufficiency in olive oil production since 2014 and become a net exporter, producing 150% of its domestic needs [15].

Despite the successes in Portugal's olive sector, significant challenges remain, particularly from diseases like olive knot [16]. This bacterial disease, caused by *Pseudomonas savastanoi pv. savastanoi*, has a substantial economic impact on olive production. Yield losses can reach up to 20% or more in severely affected orchards, necessitating increased labour for pruning and removal of diseased material, higher costs for chemical or biological controls, and potential replanting expenses [17]. Additionally, olive knot affects not only the quantity but also the quality of olives, leading to off-flavour fruit and reduced market competitiveness [18]–[20].

The olive sector contributes significantly to Portugal's Economy, generating over €500 million annually from exports and supporting rural communities through job creation [21]. Approximately 98% of Portuguese olive oil is classified as Virgin or Extra Virgin, reflecting its exceptional quality [13]. Effective management of diseases like olive knot is crucial for sustaining productivity and ensuring continued growth in the sector. By integrating PA technologies, such as UAVs and CV, into disease management strategies, Portugal can enhance its position in the global olive oil market while maintaining high-quality production standards.

This context underscores the importance of leveraging advanced technologies to address pressing challenges in olive cultivation. Therefore, this dissertation aims to explore innovative solutions that can further support the sustainability and competitiveness of

Portugal's olive industry.

Furthermore, this work is under and significantly contributes to the international cooperation Project 442696/2023-0, titled "*Study of Cooperative and Autonomous Inspection in Plantations*", also funded by CNPq. The project aims to develop an innovative cooperative system involving robots and other autonomous agents to inspect fields using computer vision techniques, sensor integration, and intelligent cooperation.

1.1 Objectives

The objective of the proposed work is to develop and validate a cost-effective, UAV-based system capable of detecting olive knot disease from aerial imagery and accurately pinpointing the geographic locations of infected trees within an olive grove. The system is designed to leverage lightweight DL models and affordable UAV platforms, making it accessible and practical for real-world deployment in resource-constrained agricultural environments.

This being the general objective, the work will be divided into the following specific tasks:

- Study the state-of-the-art DL frameworks for their effectiveness in plant disease detection;
- Create a novel, annotated image dataset of olive knot disease, collected using a cost-effective UAV platform;
- Collect GPS data for each olive tree within the study area to enable spatial referencing of disease occurrences;
- Train and compare several object detection models to identify the most suitable approach for the task;
- Develop a method to associate disease detections with spatial locations in the field;

- Implement a visualization strategy to present detection and location results in an accessible format;
- Test and validate the system's performance using collected data and field observations.

1.2 Document Structure

This work is structured into seven chapters, beginning with a comprehensive introduction to the theme, followed by a presentation of the motivations that justify the development of this study and the objectives it aims to achieve.

The second chapter provides an overview of key concepts essential for understanding this work. Furthermore, it reviews pertinent related research.

The third chapter presents a study of the tools, frameworks, and devices employed during the development of this work.

The fourth chapter introduces the methodology used in the development of this work, giving a detailed description of the activity plan used to develop the work.

The fifth chapter presents a detailed description of all the stages of the project development, which includes illustrative schemes of the system's workflow and the tools used to conduct the work.

The sixth chapter presents the application of the methodology developed, along with a detailed discussion of the experimental results obtained.

In the seventh and final chapter, the conclusions drawn from the study are presented, alongside an analysis of the results in relation to the initial objectives. This chapter also outlines potential improvements and proposes future research directions.

Chapter 2

State of the Art

This chapter provides a literature review that contextualizes the primary concepts explored in this work. It highlights several projects and scientific papers where UAVs have been employed in disease detection, often in conjunction with CV technologies. Additionally, it examines the frameworks that have been utilized in such tasks, emphasizing their core characteristics and capabilities.

2.1 Theoretical Background

To support the development of automated disease detection systems, it is necessary to consider a few theoretical foundations and technological approaches. This section outlines key background concepts relevant to image analysis, neural networks, and the use of computational methods in agriculture.

2.1.1 Deep Learning

DL is a subset of ML that leverages artificial neural networks with multiple layers, also known as deep neural networks, to automatically learn hierarchical representations from large volumes of data [22]. Inspired by the structure and functioning of the human brain, these neural networks consist of interconnected computational units or neurons,

organized into distinct layers: an input layer that receives raw data, multiple hidden layers that progressively extract increasingly abstract features, and an output layer responsible for producing the final predictions [23]. The term deep refers explicitly to the presence of multiple hidden layers, which enable the model to capture complex patterns and relationships within data that simpler models cannot [22].

The remarkable rise and success of DL in recent years can be attributed to several key factors: the availability of massive datasets required to train sophisticated models, significant advancements in computational hardware (Graphics Processing Units (GPUs)) enabling efficient parallel processing, and the development of accessible software frameworks, like TensorFlow [24] and PyTorch [25]. These factors have collectively facilitated rapid progress, allowing DL algorithms to surpass traditional ML approaches in various tasks, including image recognition, speech recognition, and autonomous systems [23].

Unlike traditional ML methods, DL learning models automatically discover relevant features directly from raw input data through iterative training processes involving forward propagation, which calculates predictions and back propagation, which adjusts the model's parameters based on prediction errors [26]. To better demonstrate these processes, Figure 2.1 shows a more detailed explanation of how these processes work on a Neural Network.

This capability allows deep neural networks to effectively handle unstructured data such as images, audio, and text without explicit feature engineering.

Central to this iterative optimization is gradient descent, an algorithm that plays a pivotal role in minimizing the loss function of a model. The gradient descent operates by iteratively adjusting model parameters, such as weights and biases, in the direction of the steepest descent of the loss function. The steepest descent is mathematically represented by the derivative (or gradient) of the loss function with respect to each parameter. The gradient provides crucial information about how the loss changes in response to slight variations in the parameters, enabling the algorithm to determine the optimal direction for adjustment. By moving against the gradient, essentially following the negative derivative, the algorithm seeks to find the minimum value of the loss function, thereby improving

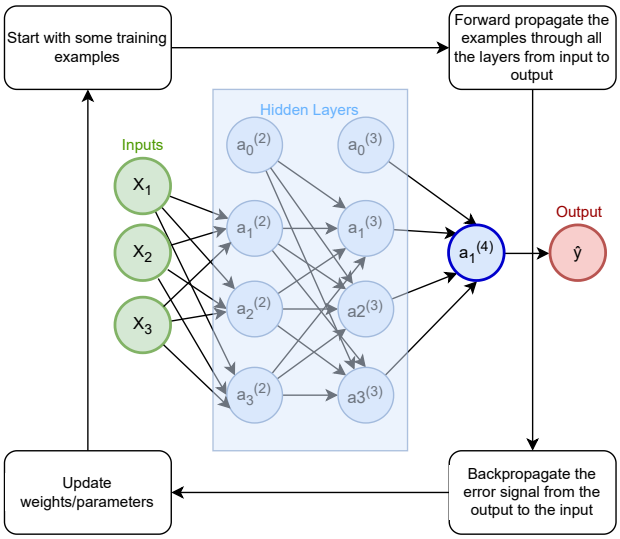


Figure 2.1: Demonstration of Forward and Backward propagation processes. Source: Adapted from [26].

model predictions [27].

The process is analogous to descending a mountain: starting from an initial position, one follows the steepest downward slope repeatedly until reaching a valley or minimum. This method ensures that DL models converge toward optimal solutions, enabling them to generalize well across new data.

Within CV specifically, DL has revolutionized traditional methods by significantly improving accuracy and enabling more sophisticated tasks such as object detection, segmentation, and image generation. The following sections of this work explore Convolutional Neural Networks (CNNs) and their application in object detection frameworks, such as You Only Look Once (YOLO).

2.1.2 Convolutional Neural Networks (CNNs)

CNNs, a specialized class of DL models, have emerged as one of the most effective architectures for analysing visual data, such as images and videos. Inspired by biological processes, specifically the structure and functioning of the animal visual cortex, CNNs

leverage spatial locality and hierarchical feature extraction to efficiently recognize patterns within complex visual inputs [28].

Unlike traditional fully connected neural networks, CNNs utilize convolution operations, local connectivity, shared weights, and pooling layers to reduce computational complexity and improve generalization capabilities significantly. A CNN architecture typically consists of several interconnected layers, each performing distinct functions to progressively extract meaningful features from the raw input data [29].

Convolutional layers form the core component of CNNs. They apply a set of learnable filters, also known as Kernels, across small localized regions of the input image. These filters perform convolution operations, which are mathematical computations involving dot products between filter weights and corresponding input pixels, to generate feature maps [30]. Each feature map highlights specific visual patterns such as edges, textures, or shapes [28].

In addition, CNNs employ shared weights, which means that the same filter parameters are applied to all spatial positions within a given convolutional layer. This approach drastically reduces the number of parameters required, allowing CNNs to learn relevant features from large datasets efficiently [29].

The convolutional operation depends on three key variables: the input (image or feature map), the kernel (filter), and the output (feature map). The kernel is a small matrix of weights that slides over the input matrix (image) to produce the output matrix. The convolutional operation for a 2D input X and a kernel K can be mathematically defined by Equation 2.1:

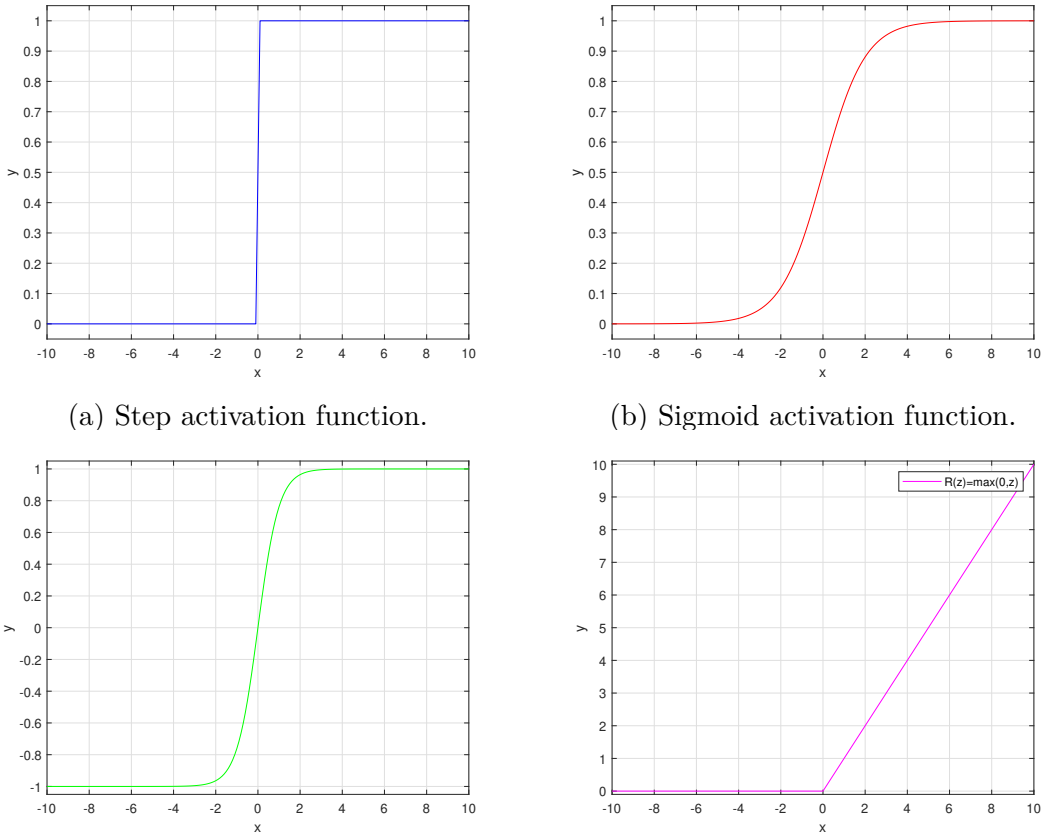
$$Y(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n) \cdot X(i+m, j+n), \tag{2.1}$$

where $Y(i,j)$ is the output feature map, $K(m,n)$ are the kernel weights of size $M \times N$ and $X(i+m, j+n)$ are the corresponding input values.

Following the extraction of basic features through convolutional operations, activation functions introduce non-linearity into the network to better capture complex patterns.

The most common activation function used in CNNs is the Rectified Linear Unit (ReLU), which maps negative values to zero and maintains positive values unchanged. Non-linear activations enable CNNs to model complex patterns and relationships within data by selectively activating neurons based on learned features [29].

Other commonly used activation functions are the Step Function, Sigmoid Function, and the Tanh function, with different types of functions being variations of those [31]. Figure 2.2 shows the graphics of those activation functions:



(a) Step activation function.

(b) Sigmoid activation function.

(c) Tanh activation function.

(d) ReLU activation function.

Figure 2.2: Commonly used activation functions.

Subsequently, pooling layers are critical components within CNNs, positioned typically after convolutional and activation layers. Their primary purpose is to reduce the spatial dimensions of feature maps, thereby reducing computational complexity, memory usage, and the risk of overfitting, while simultaneously enhancing the network's ability to

generalize and recognize patterns irrespective of minor positional variations in input data [28], [32].

Among the pooling layers, there is a critical concept known as stride, which defines the step size at which a filter moves across the input during convolution or pooling operations. Stride determines how much the filter shifts horizontally and vertically after each cycle, directly influencing the spatial dimensions of the output feature map [33].

A stride of 1 means that the filter will shift one pixel at a time, while larger strides will skip pixels, leading to downsampling and reduced computational complexity. Figure 2.3 shows an example of the stride convolution with a 3×3 filter, resulting in a downsampled input of 3×3 [33]:

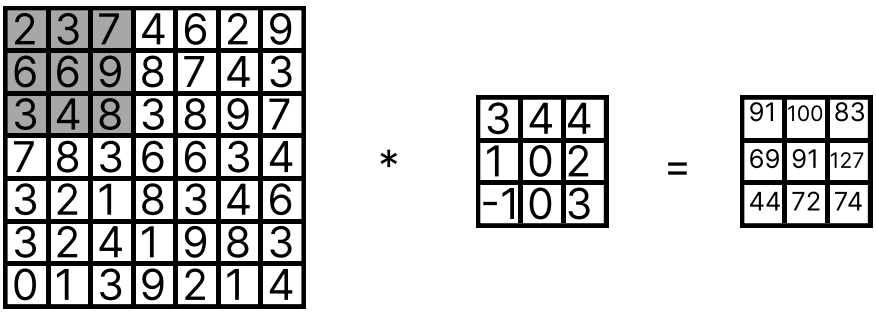


Figure 2.3: Stride convolution example.

While larger strides improve efficiency and introduce translation invariance, they may result in a loss of fine-grained information. Properly selecting stride values is essential for balancing resolution, efficiency, and feature extraction [33].

Pooling operations function by summarizing local regions of feature maps through downsampling. A pooling layer applies a filter across the feature map using a predefined stride, aggregating information within each window into a single representative value. The reduction in spatial dimensions directly lowers the number of parameters required in subsequent layers, thus significantly improving computational efficiency and speeding up training and inference processes [32].

The two most common pooling methods are max pooling and average pooling:

- **Max Pooling**: selects the maximum value within each pooling region;

- **Average pooling**: computes the average value of activations within each pooling region.

After multiple convolutional and pooling stages extract increasingly abstract representations from input data, CNN architectures typically include one or more fully connected layers in their final stages. These layers interpret high-level features extracted by the preceding layers and produce predictions or classifications based on learned representations [29].

The output layer generates predictions based on learned representations. For that purpose, this layer commonly employs activation functions such as softmax to convert output into probability distributions over predefined classes [29]. As an example, Figure 2.4 shows a layer-by-layer construction of a simple CNN, with Convolutional Layers, Pooling Layers, and Fully Connected Layers:

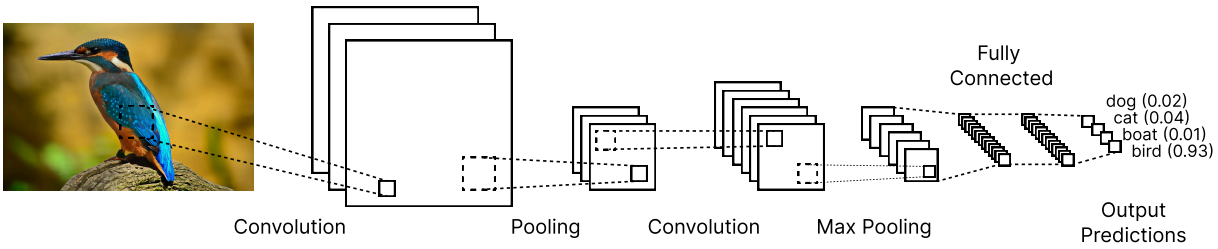


Figure 2.4: CNN example.

Beyond the architecture itself, several training configuration choices, known as hyperparameters, play a crucial role in the performance and generalization of CNNs. Hyperparameters are configuration settings that control how a neural network learns and processes information during training. Unlike model parameters, which are learned from data, hyperparameters are manually set before training begins.

The learning rate is one of the most critical hyperparameters in training a neural network, as it determines the step size for updating the model's parameters during gradient descent. A high learning rate allows the model to converge quickly to the minimum of the loss function, but risks overshooting the optimal solution. In contrast, a low learning rate ensures more precise updates, but can significantly slow down training. Finding the

right balance is essential, and techniques like adaptive optimizers (e.g., Adam Optimizer) are often employed to adjust the learning rate during training dynamically [34].

The batch size refers to the number of samples processed before updating the model's weights. Smaller batch sizes lead to faster updates and higher variance in gradient estimation, which can help escape local minima but may result in noisier convergence. Conversely, larger batch sizes provide more stable gradient estimates and better utilization of hardware resources, but require more memory and may slow down each iteration [35].

The number of epochs defines how many times the entire dataset is passed through the network during training. Training for too few epochs can lead to underfitting, where the model fails to capture complex patterns in the data. On the other hand, too many epochs may result in overfitting, where the model performs well on training data but poorly on unseen data. Techniques like early stopping are often used to monitor validation performance and halt training when improvements no longer appear, preventing unnecessary computation and overfitting [36].

Dropout is a widely used regularization technique that helps prevent overfitting by randomly deactivating a fraction of neurons during each training iteration. By doing so, dropout forces the network to rely on multiple pathways for feature extraction rather than over-relying on specific neurons or features. Dropout rates typically range from 20% to 50%, with higher rates being more aggressive in regularization [37].

Weight decay, also known as L2 regularization, penalizes large weight values by adding their squared magnitude to the loss function during optimization. This discourages overly complex models with large weights that may overfit the training data. A hyperparameter controls the strength of weight decay, often denoted as λ, which determines how much regularization is applied [38].

Finally, the loss function serves as a measure of how well or poorly a neural network's predictions match the true labels. The choice of loss function depends on the task at hand: the Mean Squared Error (MSE) is commonly used for regression problems, and cross-entropy loss is standard for classification tasks. The loss function guides the optimization

process by providing feedback on how to adjust model parameters to minimize prediction errors.

2.1.3 Transfer Learning

Transfer learning is a ML technique that leverages knowledge gained from one task to improve performance on a different but related task [39]. It is based on the principle that it is more practical to build upon prior knowledge than to start learning entirely from scratch, similarly to how humans naturally apply existing knowledge to solve different situations or problems.

In transfer learning, a model trained on a source task is adapted for a target task. The process typically involves reusing the learned weights of the pre-trained model and fine-tuning them for the new task. As an example, in CV, the initial layers of a neural network might learn general features like edges, lines, textures, while later layers specialize in task-specific patterns. Transfer learning allows the reuse of these general features, requiring only the final layers to be retrained for the target task [40]. This method saves computational effort and improves generalization by building upon previously learned representations.

2.1.4 Object Detection

Object detection can be defined as a fundamental task within CV and Artificial Inteligence (AI), which involves identifying and localizing instances of objects belonging to specific semantic categories within digital images or videos. It combines two primary subtasks, the object localization, which determines the precise spatial location of objects by delineating them with bounding boxes, and object classification, which assigns each detected object to a predefined category or class [41].

Object detection techniques typically leverage ML methods, especially deep learning models such as CNNs, which learn visual features from large annotated datasets. These models consist of interconnected layers that extract increasingly abstract representations

from input images, enabling accurate identification and localization of multiple objects simultaneously [42].

Notable architectures include two-stage detectors like Region-based Convolutional Neural Network (R-CNN), which separate localization and classification tasks for higher accuracy, and single-stage detectors such as YOLO, which integrate these tasks for faster performance. Figure 2.5 shows an image that has been processed by a model trained for object detection:

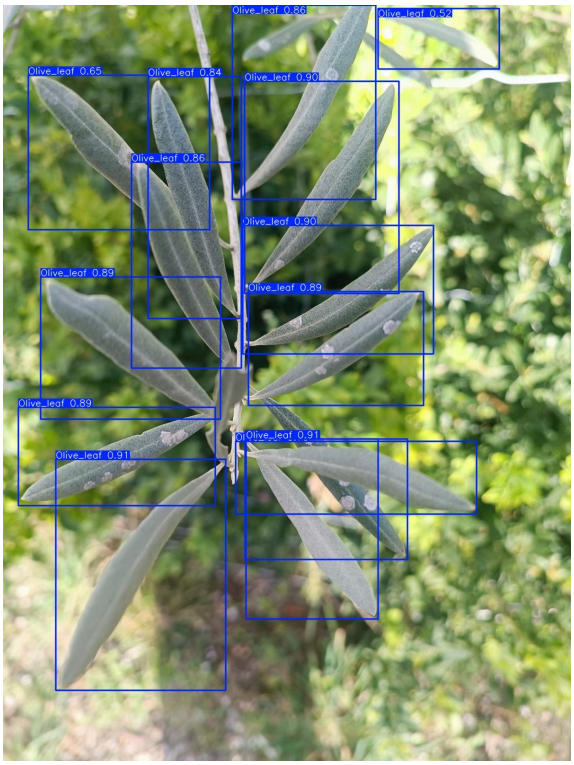


Figure 2.5: Object detection example.
Source: [43].

While object detection represents a significant advancement in CV, it is only one of several key tasks that contribute to understanding visual data. To clarify these distinctions, the following segment compares object detection with classification and segmentation, highlighting their unique methodologies, outputs, and applications.

Classification is a CV task that involves assigning one or more predefined labels or

categories to an entire image or specific regions within an image, based solely on visual content. The primary goal of classification is to recognize and categorize images into distinct classes without explicitly identifying the precise spatial locations of objects [44]. Figure 2.6 shows an image that has been processed by a model trained for object classification:

Figure 2.6: Object classification example.
Source: [45].

Segmentation provides the most detailed spatial information by assigning class labels at the pixel level. The goal of segmentation is to partition an image into meaningful regions or segments corresponding to different semantic categories or individual object instances. Segmentation can be categorized into two main types: semantic segmentation and instance segmentation. Semantic segmentation assigns each pixel to a specific class category without distinguishing between different instances of the same class [46]. Instance segmentation goes a step further by differentiating individual instances of objects within the same category [47]. Figure 2.7 shows an image that has been processed by a model

trained for instance segmentation:

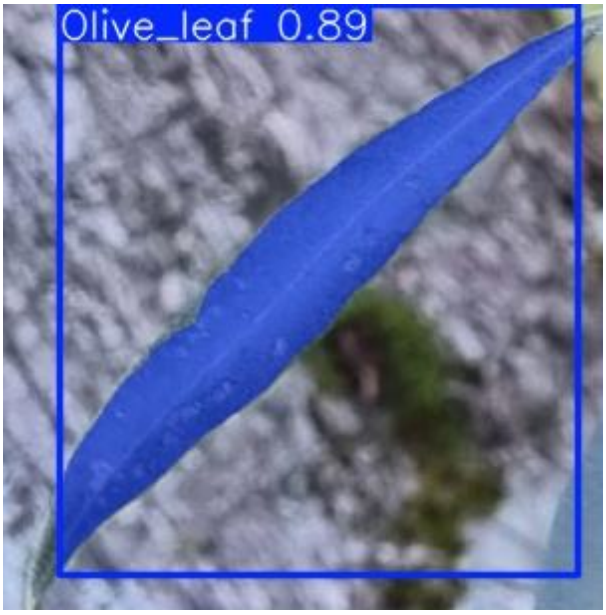


Figure 2.7: Object segmentation example.
Source: [43].

For a more comprehensive overview, Table 2.1 summarizes the main differences between classification, detection, and segmentation.

In the context of these tasks, CNNs have proven particularly effective for detection and related challenges.

As the size and resolution of images used in data problems continues to grow, CNNs have become indispensable for addressing challenges in tasks such as detection, segmentation, and classification. Larger images inherently contain more detailed information, which is critical for identifying subtle patterns or small objects within complex scenes. CNNs are particularly well-suited for these tasks due to their ability to automatically extract hierarchical features directly from raw input data, progressively capturing low-level details (e.g., edges and textures) and high-level abstractions (e.g., shapes or objects).

Moreover, as stated in Section 2.1.2, the use of filters and shared weights allows CNNs to efficiently process large images without requiring an impractical number of parameters and will enable them to recognize features regardless of their position within the

Task	Definition	Output	Architectures	Applications
Classification	Assigns labels or categories to an entire image or specific regions within an image based on visual content.	Class labels	CNN-based models (e.g., AlexNet, VGGNet, ResNet [48]–[50])	Image tagging, face recognition, medical diagnosis
Detection	Identify and localize multiple objects within an image by assigning class labels and delineating their locations with bounding boxes.	Class labels and Bounding Boxes	CNN-based models (e.g., R-CNN, YOLO, SSD [51]–[53])	Autonomous driving, surveillance, augmented reality
Segmentation	Divide an image into distinct segments by assigning class labels at the pixel level, outlining boundaries and regions.	Pixel masks	Semantic segmentation, instance segmentation (e.g., Mask R-CNN [46])	Medical imaging analysis, autonomous vehicles.

Table 2.1: Comparison between Classification, Detection and Segmentation.

image. This scalability and efficiency make CNNs particularly effective for handling high-resolution images in modern datasets. By leveraging deeper architectures with multiple convolutional layers, CNNs can progressively capture increasingly abstract features, enabling robust performance even on complex tasks with large-scale inputs. As such, CNNs remain foundational tools in CV, capable of balancing computational demands with the need for precision in large-scale image analysis.

2.1.5 Single-Stage vs Double-Stage Frameworks

Single-stage object detection frameworks, such as YOLO and Single Shot Multibox Detector (SSD), are designed to perform object localization and classification in a single forward pass through the network. These frameworks divide the input image into a grid and predict bounding boxes and class probabilities directly, without the need for a

separate region proposal step. This unified approach makes single-stage models highly efficient, enabling real-time performance and reduced computational overhead [52], [53]. In contrast to this unified approach, double-stage object detection frameworks, such as Faster R-CNN, operate in two distinct steps: first, they generate region proposals that identify potential object locations, and second, they refine these proposals by classifying them and adjusting their bounding boxes. This two-step process allows double-stage networks to achieve higher accuracy and precision, particularly for tasks requiring fine-grained detection or handling complex scenes [54], [55]. Figure 2.8 shows a demonstration of how a single-stage detector works compared to a double-stage detector:

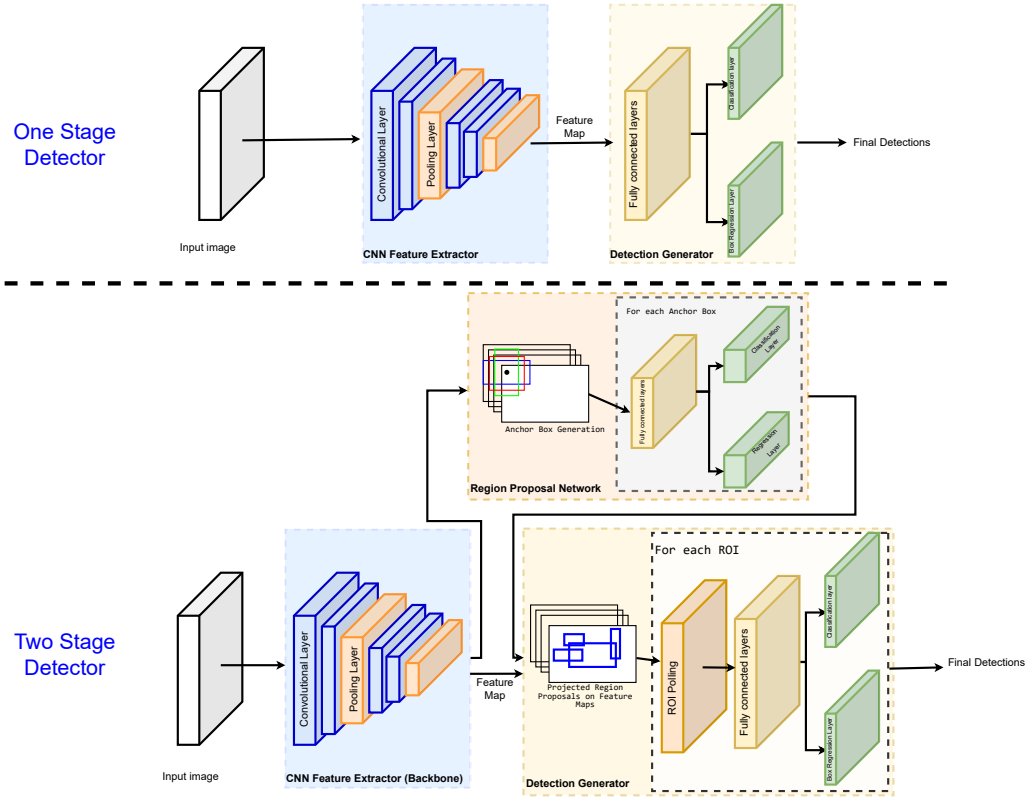


Figure 2.8: Single-stage x Double-stage comparison.
Source: Adapted from [55].

Single-stage frameworks are particularly advantageous for embedded systems due to their lightweight architecture and computational efficiency, making them ideal for

resource-constrained environments. Embedded systems often have limited processing power and memory, and single-stage frameworks like YOLO are specifically designed to meet these constraints while maintaining real-time performance. For tasks like disease detection, where the goal is to deploy models on portable devices, the simplicity and speed of single-stage frameworks ensure that the system can operate effectively without compromising usability or responsiveness. Additionally, their ability to balance efficiency with reasonable accuracy makes them a practical choice for scenarios where computational resources are limited but real-time decision-making is essential.

Driven by these practical considerations, recent advancements in single-stage object detection frameworks have closed the performance gap with double-stage networks, making them highly competitive in terms of accuracy while retaining their inherent speed advantage. Innovations such as anchor-free designs [56], feature pyramid networks [57], and advanced loss functions like focal loss [58] have enabled single-stage models to handle complex tasks with precision comparable to double-stage frameworks.

Modern iterations of YOLO incorporate efficient backbone architectures and multi-scale feature extraction techniques that enhance detection accuracy across a wide range of object sizes [59]. These improvements have rendered double-stage frameworks increasingly unnecessary for tasks that do not demand a very high precision. In applications based on embedded systems, single-stage frameworks now provide state-of-the-art solutions. Their ability to achieve high performance without the added complexity of multi-stage processing has made them the preferred choice for practical applications that prioritize speed and resource efficiency over marginal gains in accuracy.

Works [55], [60]–[63] demonstrate that the most recent iterations of single-stage frameworks exhibit competitive performance compared to double-stage models across various scenarios, in some cases even surpassing them. This progress highlights that the advancements in single-stage frameworks are now sufficient to replace double-stage models in many applications, particularly those where computational efficiency and real-time performance are prioritized over achieving greater precision.

2.1.6 Historical Context and Evolution of YOLO

The You Only Look Once (YOLO) algorithm has played a transformative role in the field of object detection, evolving from a groundbreaking innovation in 2016 to one of the most widely used frameworks in CV today. Introduced by Joseph Redmon and his collaborators [52], YOLO was designed to address the limitations of earlier object detection models, which were often computationally expensive and unsuitable for real-time applications.

The first version, YOLOv1, introduced a grid-based approach that, while revolutionary for its speed, struggled with accurately detecting small objects [52]. Subsequent versions brought incremental yet significant improvements: YOLOv2 added anchor boxes, batch normalization, multi-scale training, and the Darknet-19 backbone [64]. YOLOv3 further enhanced detection accuracy and robustness by introducing the deeper Darknet-53 backbone with residual connections and multi-scale predictions [65]. YOLOv4 incorporated CSPDarknet53, Spatial Pyramid Pooling, and advanced data augmentation techniques such as Mosaic and Self-Adversarial Training [66].

The evolution continued with YOLOv5, which transitioned to PyTorch, making implementation and scaling easier. It featured CSPNet and PANet for feature aggregation and introduced anchor-free detection variants [67]. YOLOv6 targeted industrial and edge applications with a CSPDarknet backbone and decoupled heads for classification and regression, optimizing both speed and accuracy [68]. YOLOv7 emphasized efficiency and scalability through a novel layer aggregation and re-parameterized convolutions [69].

A significant leap occurred with YOLOv8, released by Ultralytics in 2023, which adopted an anchor-free detection approach. This eliminated the need for predefined anchor boxes by predicting object centers directly, simplifying the detection process and reducing computational overhead [70]. YOLOv8 also introduced architectural refinements such as enhanced feature concatenation, reducing parameter count and tensor size without sacrificing accuracy. It incorporated advanced training techniques like mosaic augmentation to improve robustness under diverse conditions.

YOLOv9 brought further advancements with the introduction of Progressive Gradient Information (PGI) and GEL modules, which improved gradient flow and feature aggregation [71]. YOLOv10 was notable for eliminating the need for Non-Maximum Suppression (NMS) through an NMS-free training approach and dual label assignments [72].

YOLO11, introduced in late 2024, represents a major step forward, incorporating advanced architectural elements such as the C3k2 block for improved feature propagation, SPPF for efficient multi-scale feature aggregation, and the C2PSA mechanism for enhanced spatial attention [56]. This version expanded YOLO's capabilities to include instance segmentation, pose estimation, and oriented object detection, making it highly versatile for complex computer vision tasks. Evaluations on benchmark datasets demonstrate that YOLO11 achieves superior accuracy and real-time inference speeds compared to its predecessors, with a lightweight design suitable for deployment on resource-constrained devices [59].

Finally, YOLO12, the latest version as of this dissertation, builds upon YOLO11 with further architectural and functional refinements. It introduces Dynamic Convolution Blocks that adapt kernel weights to input features for improved detection across varying object scales and orientations. It also incorporates Multi-Scale Transformer Layers for better contextual understanding [73]. YOLO12 also features hybrid loss functions to balance classification and localization, and a Unified Detection Framework that allows simultaneous object detection, segmentation, and pose estimation within a single architecture. Benchmark evaluations highlight YOLO12's state-of-the-art inference speed and accuracy, setting a new standard for real-time object detection frameworks [73].

2.1.7 YOLO Architecture

As stated in 2.1.6, YOLO11 was designed to enhance speed, accuracy, and efficiency for real-time applications. This section will describe in greater detail the architecture used behind the YOLO11 framework. Using Figure 2.9 as a base, the YOLO framework is composed of the following:

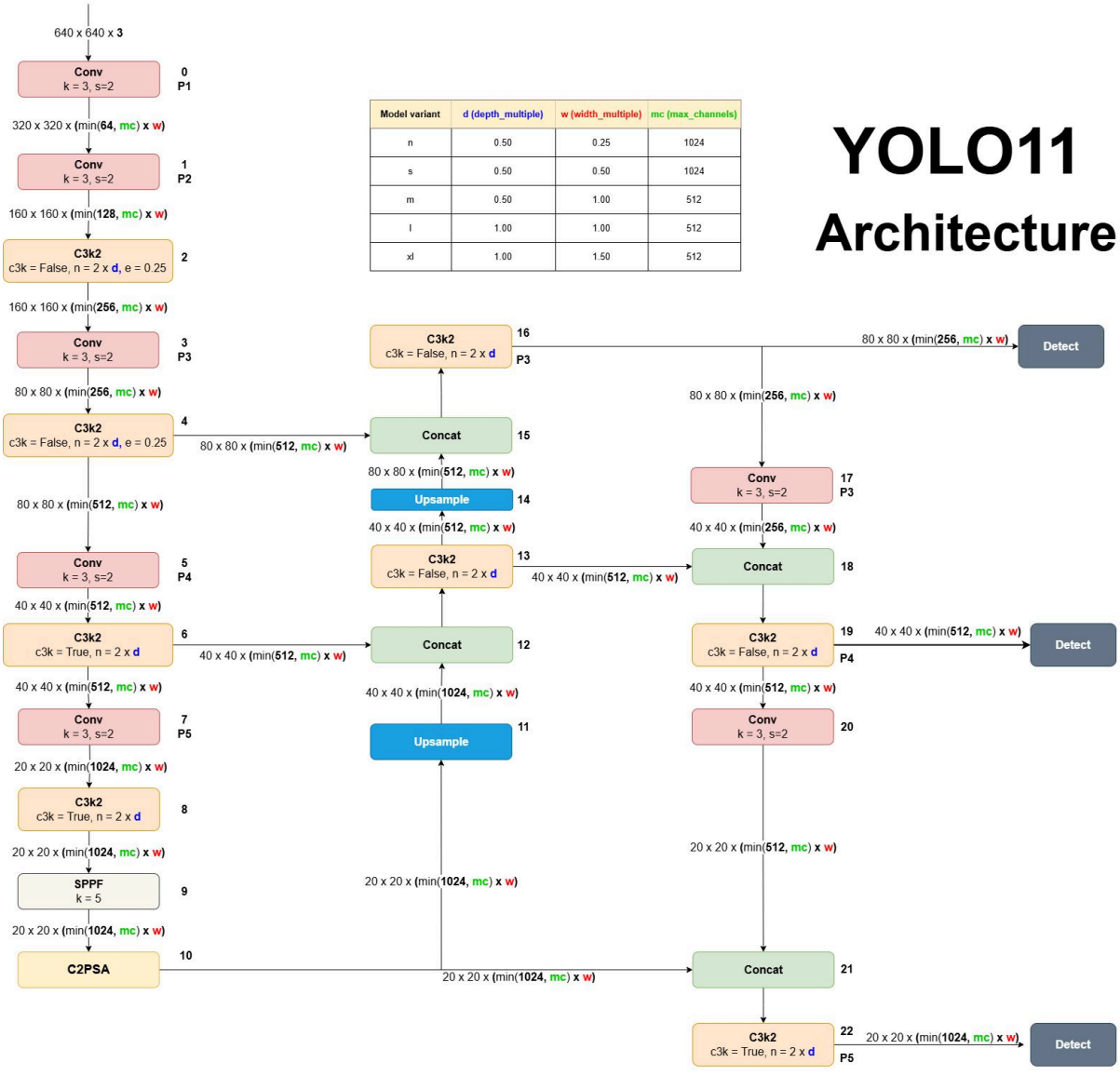


Figure 2.9: YOLOv11 Architecture.
Source: Adapted from [56], [59].

To begin with, the backbone forms the foundation of the YOLO11 architecture, is responsible for initial feature extraction from the input images, and includes the following key components:

- Conv Blocks: These consist of convolutional layers followed by batch normalization and SiLU activation functions. They process input tensors to extract low-level features efficiently;
- C3k2 Block: This block replaces the C2f block used in earlier YOLO versions. It uses two smaller convolutions instead of one large convolution. The block splits feature maps into two branches, one processed through bottleneck layers and the other merged directly with the output;
- Cross Stage Partial: This network split feature maps into two parts, processing one through bottleneck layers while merging the other part with the output. This design improves gradient flow and reduces computation.

After initial feature extraction, the neck component further processes and aggregates these features to facilitate robust multi-scale object detection. Key components include:

- Spatial Pyramid Pooling Fast: This module pools features at various scales using max-pooling operations with different kernel sizes. It captures multi-scale contextual information, which is essential for detecting objects of varying sizes;
- Upsampling Layers: These layers increase spatial resolution to detect smaller objects more effectively;
- C2k2 Block: Similar to its use in the backbone, this block is employed in the neck to improve feature aggregation after upsampling and concatenation.

In addition to these components, YOLO11 incorporates attention mechanisms to further enhance detection accuracy by focusing on the most relevant regions of the image.

Once features have been aggregated and refined, the detection head is responsible for generating the final predictions for object locations and classes at multiple scales:

- Multi-Scale Predictions: YOLO11 outputs predictions from three feature maps (P3, P4, P5 in Figure 2.9), corresponding to different levels of granularity. This ensures effective detection of objects ranging from small to large sizes;

- Concat Operations: Feature maps from different layers are concatenated to combine information across scales, improving detection accuracy.

To accommodate a range of application requirements and computational constraints, YOLO11 and subsequent versions are available in five distinct variants: YOLO11n (Nano), YOLO11s (Small), YOLO11m (Medium),YOLO11l (Large) and YOLO11x (Extra-Large). These variants are tailored to meet different computational requirements and use cases, balancing speed, accuracy, and resource consumption. Table 2.2 shows a detailed comparison of the YOLO11 model variants:

The architectural differences between the YOLO11 variants stem from architectural adjustments in depth, width, and maximum channels:

- Depth Multiplication: Controls the number of layers in the model. Nano has the least depth, while Extra-Large has the most layers for richer feature extraction;

- Width Multiplication: Determines the number of filters in convolutional layers. Larger models have wider layers, enabling more detailed processing;

- Maximum Channels: Sets an upper limit on channel dimensions. Nano models use fewer channels, whereas Extra-Large models utilize significantly more channels to capture intricate details.

To better illustrate these architectural differences, Figure 2.9 shows in greater detail where each architectural difference between the variants affects the framework as a whole.

Another difference between the variants is their resource consumption. This is defined by the Floating-Point Operations per Seconds (FLOPs), in which larger models have higher FLOPs, indicating greater computational complexity.

Model Variant	Key Features	Applications	Performance Metrics
YOLOv11n	Extremely lightweight with minimal parameters, optimized for devices with limited computational resources and fast inference speed.	Ideal for edge devices and real-time applications where speed is critical.	Highest FPS among all variants. Lower detection accuracy compared to larger models. Suitable for tasks requiring rapid processing but less precision.
YOLOv11s	Slightly larger than Nano with improved accuracy, balanced design for speed and precision.	Suitable for general purpose tasks requiring moderate accuracy and speed	Competitive FPS. Higher mAP compared to Nano. Strikes a balance between inference speed and detection accuracy.
YOLOv11m	Designed for general purpose use, offers improved accuracy with moderate requirements.	Appropriate for applications demanding for higher precision without extreme computational cost.	Higher mAP than Small and Nano Slightly lower FPS due to increased complexity. Suitable for more complex real-time tasks.
YOLOv11l	Larger architecture with enhanced feature extraction, focuses on high accuracy while maintaining reasonable inference speed.	Ideal for tasks requiring detailed object detection and segmentation in controlled environments.	High mAP and precision. Slower inference compared to medium. Best suited for applications prioritizing accuracy over speed
YOLOv11x	Maximum accuracy and performance, incorporates the most advanced feature extraction mechanisms.	Recommended for high performance applications such as medical imaging or autonomous systems requiring superior precision.	Highest mAP among all variants. Lowest FPS due to computational intensity. Requires powerful hardware like GPUs.

Table 2.2: Comparison between the YOLO11 model variants.
Source: Adapted from [59].

2.1.8 How YOLO Makes Predictions?

YOLO begins by dividing the input image into a grid of $S \times S$ cells. For example, as illustrated in the Figure 2.10, an 8×8 grid segments the image into smaller regions. Each cell in this grid is responsible for detecting objects whose center falls within that

cell. This means that the central cell of an object is tasked with predicting the bounding box for that object [52].

Figure 2.10: Example of a 8×8 grid image.

Each grid cell predicts B bounding boxes, where each box includes the coordinates of the bounding box relative to the cell and the confidence score, which is a value between 0.0 and 1.0 indicating the likelihood that the box contain an object, while also reflecting the accuracy of the bounding box's position and size [52].

If a cell detects an object with high confidence, YOLO highlights the bounding box with dense lines, making it more prominent. Conversely, if no object is detected or confidence is low, the bounding box is less emphasized, as can be seen in Figure 2.11.

After predicting bounding boxes, YOLO calculates a class probability map for each grid cell. This map assigns probabilities to each class based on the detected objects within the bounding boxes (Figure 2.12). By combining the confidence scores with class probabilities, YOLO identifies which objects are present and their respective categories [52].

The system then refines its predictions by merging overlapping boxes and selecting those with high confidence scores. As seen in Figure 2.13, YOLO used these refined predictions to detect specific features, with the final output displaying the detected objects

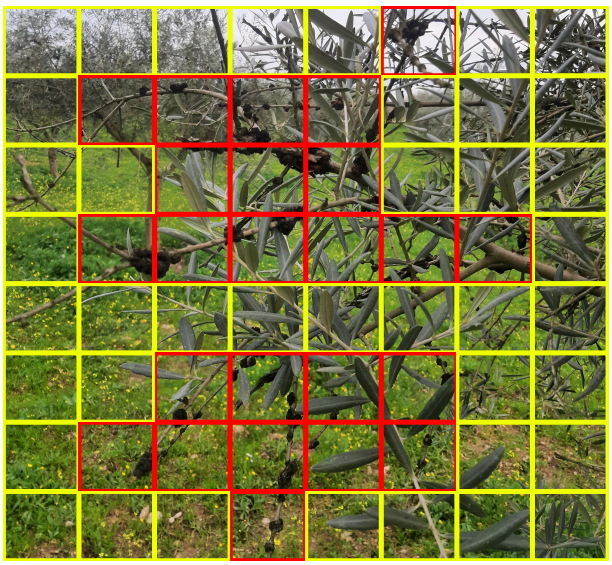


Figure 2.11: Example of a 8×8 grid image, where red grids mean the box has a probability > 0 of containing the desired class, while the yellow grids mean that the box has a probability $= 0$.

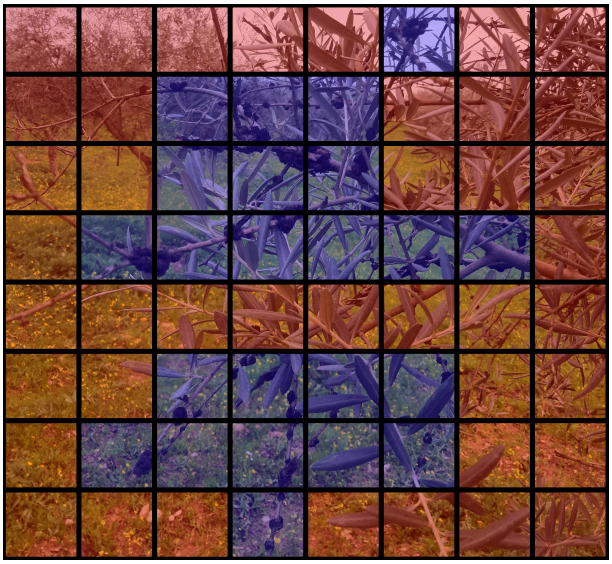


Figure 2.12: Example of a class probability map.

with their corresponding bounding boxes and class labels [52].

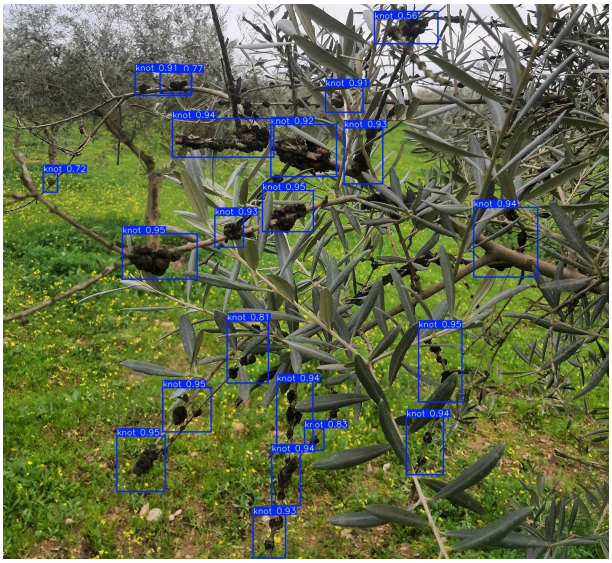


Figure 2.13: Final object detection result.

2.1.9 Evaluation Metrics

In object detection, evaluating the performance of a model requires robust and widely accepted metrics that measure its ability to detect and localize objects accurately. Among these metrics, Precision (P), Recall (R), Mean Average Precision (mAP), Intersection over Union (IoU), and the $F1$ Score are commonly used to provide a comprehensive assessment of a model's effectiveness [74], [75].

Precision (P) quantifies the proportion of correctly identified objects among all predicted instances. It reflects the model's ability to minimize false positives, ensuring that only relevant objects are detected [74]. Mathematically, P is defined by Equation 2.2:

$$P = \frac{TP}{TP + FP}, \tag{2.2}$$

where True Positive (TP) represents the number of correctly detected objects, and False Positive (FP) refers to instances where the model incorrectly classifies non-object regions as objects.

Recall (R) measures the proportion of actual objects in the dataset that the model correctly identifies. It indicates how well the model captures all relevant instances, reducing

missed detections [74]. The formula for R is expressed in Equation 2.3:

$$R = \frac{TP}{TP + FN}, \tag{2.3}$$

where False Negative (FN) represents cases where the model does not detect actual objects.

The Intersection over Union (IoU) is a metric used to evaluate the overlap between the predicted bounding box and the ground truth bounding box. It provides a measure of how accurately the predicted boundary aligns with the actual object boundary [76]. IoU is calculated as shown in Equation 2.4:

$$IoU = \frac{|A \cap B|}{|A \cup B|}, \tag{2.4}$$

where $|A \cap B|$ is the area of overlap between the predicted and ground truth bounding boxes, and $|A \cup B|$ is their combined area. Figure 2.14 shows an example of IoU on an olive tree branch:

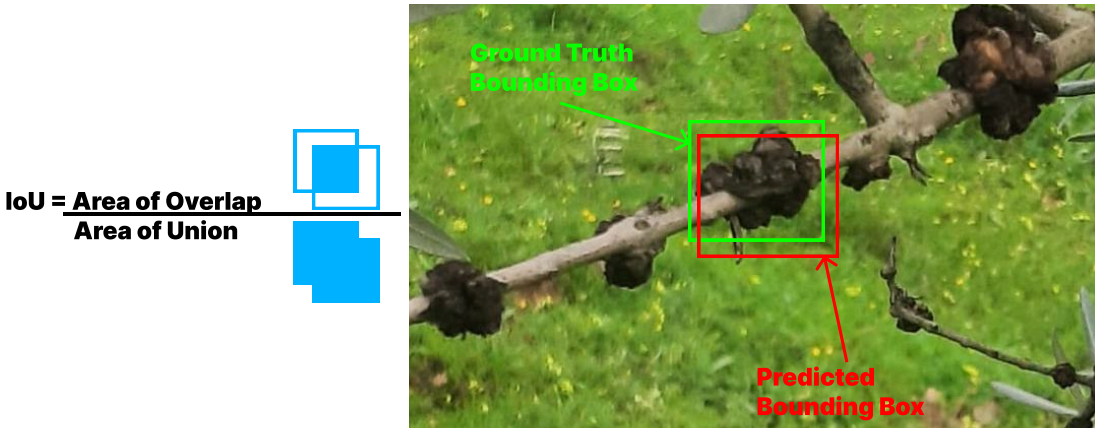


Figure 2.14: Intersection over union example on an olive tree branch.

To accurately determine whether a prediction is classified as a TP or a FP, a defining criterion must be established, the Threshold. In the context of IoU, the threshold serves as the boundary that determines whether a predicted bounding box sufficiently overlaps with the ground truth box to be considered correct.

For example, if the IoU between a predicted box and its corresponding ground truth box exceeds a predefined threshold (e.g., 0.5), the prediction is classified as a TP. In contrast, if the IoU falls below this threshold, even if the confidence score is high, the prediction is considered a FP due to insufficient overlap.

Thresholds directly influence metrics such as P and R. A higher IoU threshold demands stricter alignment between boxes, improving localization accuracy, but potentially discarding valid detections with slightly misaligned boundaries. On the other hand, lower thresholds allow for more flexibility in overlap evaluation but may tolerate imprecise bounding boxes, increasing the incidence of FPs [76].

The Mean Average Precision (mAP) combines P and R to provide a more comprehensive evaluation metric. The Average Precision (AP) quantifies the area under the precision-recall curve, which measures performance across different confidence thresholds [77]. It is mathematically expressed by Equation 2.5:

$$AP = \int_0^1 P(R)dR, \tag{2.5}$$

and mAP is computed as the mean of AP values across all classes, and is mathematically expressed by Equation 2.6:

$$mAP = \frac{\sum_{i=1}^{N} AP_i}{N}, \tag{2.6}$$

where N represents the number of object classes.

In object detection, two specific variants of mAP are commonly used:

- mAP50: This evaluates mAP at a fixed threshold of 0.5, assessing how well predicted bounding boxes align with ground truth;

- mAP50 − 95: This averages mAP values across multiple IoU thresholds ranging from 0.5 to 0.95 in increments of 0.05, providing a more detailed evaluation under varying conditions [77].

Finally, the $F1$ Score is a metric that combines P and R into a single value by calculating their harmonic mean. It balances these two metrics, penalizing models that perform well on one while neglecting the other [74]. The $F1$ Score is defined by Equation 2.7:

$$\mathrm{F1} = \frac{2PR}{P+R}. \tag{2.7}$$

This score ranges from 0 to 1, where higher values indicate better overall performance by achieving a balance between P and R.

2.1.10 The Olive Knot Disease

The olive knot disease, caused by *Pseudomonas savastanoi pv. savastanoi*, is a significant threat to olive cultivation worldwide [16], [21]. It manifests as woody galls or knots on branches, trunks, and occasionally leaves and fruits [17]. These knots weaken the tree by disrupting nutrient flow, leading to branch dieback, defoliation, reduced yields, and even tree death in severe cases [78]. Infected olives often result in substandard products, such as olive oil with undesirable taste and odour [79].

The bacteria primarily enter trees through wounds caused by pruning, frost damage, hail injury, or harvesting activities. Wet conditions further exacerbate the spread of infection as knots release bacteria that can infect neighbouring trees [80]. The disease cycle involves the bacteria surviving in knots and as epiphytes on leaves and twigs, where they can be spread by water, wind, insects, and human activities [18].

The disease is endemic in several olive-growing areas and affects all cultivars, although susceptibility varies among them [20]. As an example, Figure 2.15 shows an olive tree infected by the disease:

Figure 2.15: Example of an infected olive tree.

2.2 Related Works

This section presents a review of works related to this research. Understanding and acknowledging previous advances and findings are essential to contextualize the contribution of this work.

The integration of AI, ML, and CV has significantly advanced the field of agricultural disease detection. These technologies have enabled the development of automated systems capable of identifying plant diseases with high accuracy, thereby reducing reliance on traditional manual inspection methods [81]. Studies utilizing machine learning algorithms, particularly CNNs, have demonstrated their effectiveness in detecting plant

diseases across various crops. For example, in [82], a CNN-based system for tomato leaf disease detection achieved an impressive accuracy of 99.6%. In contrast, a similar approach for potato leaf disease detection reached validation accuracies exceeding 98% [83]. These results underscore the potential of DL models to process complex patterns in image data and classify multiple disease types under diverse environmental conditions. Comparative analyses further highlight the importance of selecting optimal algorithms for specific datasets, as seen in, where K-Nearest Neighbours outperformed CNNs in specific tomato leaf disease detection tasks [84].

In the context of PA, UAVs have emerged as transformative tools, offering scalable solutions for monitoring large agricultural fields [85]. Equipped with high-resolution imaging sensors, UAVs enable the collection of detailed data for applications such as weed density evaluation [86], fruit detection [87], plant phenotyping [88], and disease detection [89]–[91]. For instance, [86] employed a modified U-Net architecture, which demonstrated precise weed density mapping with IoU scores exceeding 93%. Similarly, multi-UAV systems integrated with deep learning models like YOLOv5 have facilitated real-time fruit detection in orchards, achieving mAP scores as high as 86.8% [87]. These systems address challenges such as occlusions and lighting variations while optimizing orchard management practices. UAV imaging has also proven effective for plant phenotyping by extracting traits like canopy cover and vegetation indices with high accuracy [88]. Furthermore, UAV-based vegetation monitoring has been utilized to assess soil erosion in olive orchards, demonstrating how vegetation indices can predict soil loss risk and contribute to sustainable agricultural practices [89].

A recent study proposed a robust, cost-effective drone-based system for multiclass plant disease detection, addressing the challenges of variable image quality, lighting, and disease symptom preservation in field conditions [92]. The authors introduced an improved EfficientNetV2-B4 DL model, enhanced with additional dense layers. They demonstrated its effectiveness using both the PlantVillage public dataset and real-world images captured by a DJI Phantom 3 UAV. Their system achieved good results, with AP, R, and Accuracy values of 99.63%, 99.93%, and 99.99%, respectively. The approach was shown to be

robust to noise, blurring, and other image distortions, despite being in a controlled test environment.

By being a single-stage framework, known for its lightweight, the integration of YOLO models with UAV imagery has further enhanced the capabilities of disease detection systems by enabling real-time object identification on resource-constrained devices, like UAVs and embedded systems. YOLO models are particularly well-suited for agricultural applications due to their speed and accuracy in detecting disease under varying environmental conditions. For example, in [93], YOLOv5 has been employed to detect bacterial spot disease in bell pepper plants with mAP scores reaching 90.7%. Other lightweight variants, like the MGA-YOLO variant developed by [94], have demonstrated high efficiency for apple leaf disease detection on mobile devices, achieving mAP scores of 94% while maintaining a model size of just 10 Mb. Comparative studies on YOLO models for guava and mango disease detection reveal that versions like YOLOv8 outperforms earlier versions like YOLOv7 in terms of accuracy and computational efficiency [95], which highlights the evolution of the framework in these tasks, making it a very usable option to detecting diseases. Additionally, innovative adaptations such as improved YOLO11 algorithms have been used to detect aphid infestations on Lycium barbarum plants (berry plants) with enhanced localization accuracy [96].

A significant advancement arises when combining YOLO models with UAV imagery, specifically for disease detection tasks. This integration leverages the strengths of both technologies: UAVs provide high-resolution field data while YOLO models enable precise identification of diseases within captured images [97]. For instance, [90] researches the optimization of olive disease classification through transfer learning with UAV imagery achieved. This research achieved validation accuracies as high as 99% using the MobileNet-TL architecture. Similarly, [98] made a comparative analysis of YOLO models for melon leaf disease classification, which highlighted the superiority of YOLOv9 in detecting diseases under complex conditions, with mAP scores nearing 97%. These approaches demonstrate the potential of UAV-assisted innovative agriculture systems to monitor crop health efficiently across large plantations.

The application of these technologies has also been explored in specific use cases such as rice leaf disease detection using low-altitude UAV imaging combined with spectral feature analysis [91]. This approach achieved high prediction accuracy for narrow brown leaf spot severity and fungicide efficacy evaluation, demonstrating the practicality of UAV-based systems in addressing critical agricultural challenges.

A recent study further expands on this integration by evaluating various YOLO models for bean leaf disease detection caused by *Coleoptera* pests under natural field conditions [99]. The research compared five YOLO variants, and found that YOLOv9 exhibited the best performance, with a mAP50 score of 57.8% and mAP50-95 score of 51.4%. Enhancements such as integrating different optimizers into the YOLO framework further improved precision and recall metrics across confidence thresholds. This study highlights the adaptability of advanced deep learning techniques when applied to real-world agricultural challenges characterized by complex backgrounds and environmental variability.

Chapter 3

Tools

3.1 Used Tools

This section provides an overview of the software utilized in the system's development. The selection of these tools was influenced by prior familiarity with them and their user-friendly nature.

3.1.1 PyCharm

PyCharm is a widely recognized Integrated Development Environment (IDE) designed explicitly for Python programming and developed by JetBrains. PyCharm is extensively used by developers, researchers, and industry professionals due to its comprehensive suite of features that streamline Python-based project developments across various domains, including web development, data science, and ML. With advanced capabilities such as intelligent code assistance, integrated tools for debugging and testing, and robust code navigation and refactoring options, the IDE enhances productivity by optimizing the development process and ensuring greater accuracy and efficiency.

In addition, PyCharm's compatibility with scientific libraries like NumPy, Matplotlib, and Pandas makes it an ideal choice for data analysis and visualization tasks. It also supports popular web frameworks such as Django and Flask, enabling effective collaboration

on multidisciplinary projects. PyCharm's cross-platform availability ensures accessibility across Windows, macOS, and Linux environments. The Screenshot in Figure 3.1 illustrates the PyCharm Professional Edition Workspace:

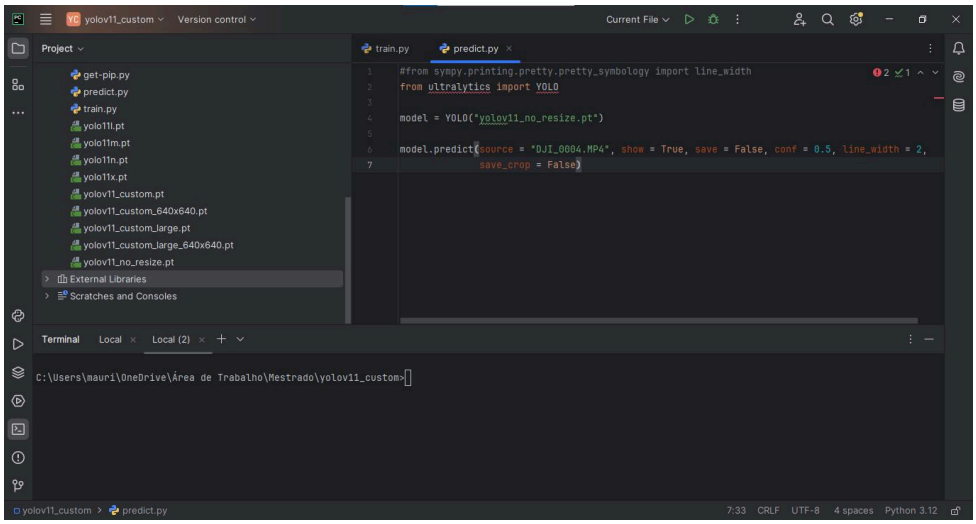


Figure 3.1: PyCharm Professional Edition workspace.

3.1.2 Roboflow

Roboflow is a comprehensive online platform designed to streamline the development of CV models. It offers powerful tools for dataset annotation, augmentation, and management, making it suitable for developers and enterprises across various industries.

The annotation tool in Roboflow is highly efficient, allowing users to label images using bounding boxes and polygons manually or through AI-assisted methods like Label Assist and Smart Polygon. These features enhance accuracy while speeding up the annotation process, making it ideal for tasks such as object detection and segmentation. The screenshot in Figure 3.2 shows the Roboflow annotation tool workspace.

Roboflow also provides advanced augmentation tools that can generate up to 50 variations of each image. These include techniques like random flips, rotations, cropping, exposure adjustments, noise addition, and mosaic transformations. Such augmentations improve dataset diversity and model robustness while reducing training costs by enabling

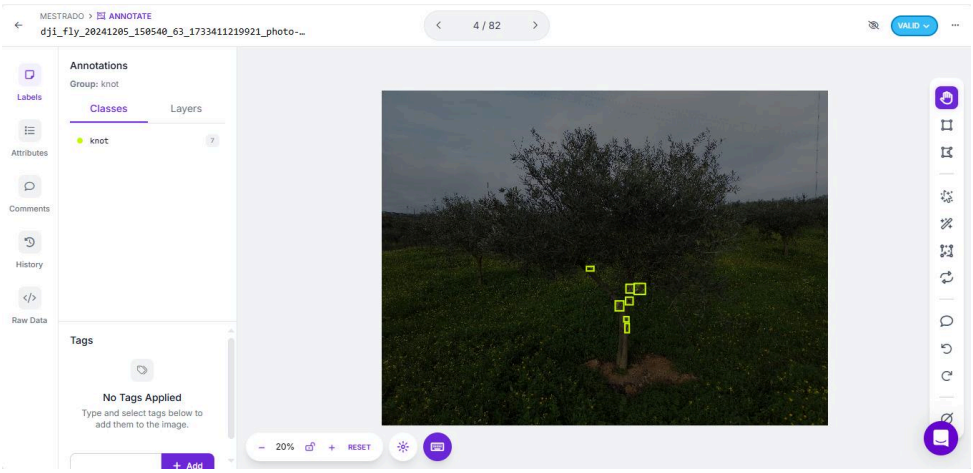


Figure 3.2: Roboflow annotation tool workspace.

offline preprocessing.

A standout feature of Roboflow is its ability to export annotated datasets into a wide range of formats, ensuring compatibility with different frameworks and workflows. Supported formats include CSV, YOLO, COCO, TensorFlow, TFRecord, VOC XML, JSON, DARKNET, and more. This flexibility allows users to seamlessly integrate their datasets into diverse ML pipelines. Table 3.1 shows some of the supported formats by the platform.

Additionally, Roboflow supports preprocessing steps like image resizing and class name sanitization during export. By combining annotation, augmentation, and export functionalities with robust dataset management tools, Roboflow provides a comprehensive solution for CV development [100].

JSON	XML	TXT	CSV	Other
COCO	Pascal VOC	YOLO Darknet	TensorFlow Object Detection	OpenAI Clip Classification
COCO-MMDetection		YOLO v3 Keras	RetinaNet Keras	TensorFlow TFRecord
CreateML		YOLO v4 PyTorch	Multi-Label Classification	Server Benchmark
Florence 2 Object Detection		Scaled-YOLOv4		
OpenAI		YOLOv5 Oriented Bounding Boxes		
		meituan/YOLOv6		
		YOLOv5 PyTorch		
		YOLOv7 PyTorch		
		YOLOv8		
		YOLOv8 Oriented Bounding Boxes		
		YOLOv9		
		YOLOv11		
		YOLOv12		

Table 3.1: Roboflow supported export formats.

3.1.3 AirData

The AirData UAV platform is a comprehensive flight data management and analysis tool that supports a wide variety of drones and flight applications. AirData seamlessly integrates with numerous flight apps, making it compatible with major drone manufacturers such as DJI, Autel, Parrot, Skydio, and Pix4D. Supported apps include DJI Fly, DJI GO, DJI Pilot, Autel Explorer, Pix4D, Drone Sense, and many others. This compatibility ensured that flight logs from different UAVs can be uploaded automatically to the platform without requiring manual intervention.

One of AirData's standout features is its ability to convert raw flight logs (.dat files) into accessible formats such as .csv or .gpx. These formats enhance usability by enabling integration with third-party tools for further analysis. The .csv files include detailed metrics such as altitude, speed, orientation, and GPS coordinates.

AirData also provides detailed insights into each flight through features like 3D flight

visualization and playback options. These tools allow operators to review flights comprehensively and identify issues such as sudden altitude changes or irregularities in the flight path.

3.2 Used Equipments

3.2.1 DJI Mini 3

The UAV system employed in this study was the DJI Mini 3 (Figure 3.3), known for its lightweight and compact design, with a take-off weight of 248g [101]. It is equipped with a 1/1.3-inch CMOS camera capable of capturing high-resolution 48 MP still images and recording 4K videos at up to 30 fps. The UAV has a maximum flight duration of 38 minutes. Furthermore, its downward vision system ensures stable hovering, while the three-axis gimbal delivers smooth image stabilization, making it well-suited for aerial data collection in agricultural settings.

Figure 3.3: DJI Mini 3.

Chapter 4

Methodology

This chapter will outline the definition of the problem and the proposed solution. It will also discuss the key concepts applied during the development phase.

4.1 Definition of the Problem

As stated in Section 2.1.10, the Olive knot disease poses a significant threat to olive plantations worldwide. The disease manifests as irregular growths or knots on branches and trunks, leading to reduced yield and weakened trees. Detecting and managing this disease early is critical to minimizing its impact on agricultural productivity.

Traditional methods for identifying the disease are highly dependent on manual inspection by farmers. This process is labour-intensive, time-consuming, and prone to human error, particularly when dealing with large-scale plantations. Furthermore, the visual characteristics of the disease can often be complex and difficult to distinguish from surrounding vegetation or natural imperfections on the tree surface, making accurate detection challenging.

Environmental factors such as lighting conditions, weather variability, and terrain differences further complicate the detection process. These challenges highlight the need for an efficient, scalable, and accurate method to identify olive knot disease in large plantations. Addressing this problem requires innovative approaches that leverage technology

to overcome the limitations of traditional inspection methods while ensuring reliable detection across diverse environmental conditions.

4.2 Proposed Solution

To tackle the challenges of detecting olive knot disease in large-scale plantations, this dissertation proposed an integrated solution combining UAV imagery with ML techniques. The methodology leverages UAVs equipped with high-resolution cameras to capture aerial images of olive trees, enabling efficient data collection and providing a comprehensive view of the plantation. These images form the basis for training and evaluating the object detection models, specifically the variants of YOLO, under realistic field conditions. The solution further incorporates spatial cross-referencing of detections with precise tree locations, allowing for accurate mapping of disease incidence within the grove. This workflow was designed to achieve balance between detection accuracy and practical deployment in agricultural environments.

4.2.1 Data Acquisition and UAV platform

The UAV platform selected for this study was the DJI Mini 3, which was detailed in Section 3.2. The data acquisition process took place on April 9, 2025, in Mirandela, Portugal (coordinates: 41.514016°N, -7.175772°W). A total of 2050 high-resolution RGB images were captured under varying environmental conditions, including diverse backgrounds, lighting scenarios, and angles. These variations were intentionally incorporated to ensure the dataset accurately represented the complexities of detecting the disease. Figure 4.1 shows an aerial view of the studied olive grove.

As explained in Section 2.1.10, olive knot disease primarily affects branches and trunks. Therefore, a UAV performing a simple overhead scan of the olive groves would not be able to detect the disease effectively. Instead, lateral flights were conducted along the planting rows within the olive groves. During the flights, the UAV was always oriented westward, to ensure consistency along the data collection and later processing of the data. The

Figure 4.1: Aerial view of the olive plantation, situated in Mirandela, Portugal.

image collection process, illustrated in Figure 4.2, began at the green triangle. The UAV performed lateral flights looking westward until it reaches the red circle at the end of the row, then moved to the nearest green circle to begin the next row. This pattern continued until the drone reached the red triangle at the end of the grove, marking the conclusion of the image collection.

Additionally, the flight log data recorded by the UAV will be used to extract approximate coordinates (latitude and longitude) for each detected tree showing signs of olive knot disease. This process involves interpreting flight logs using the AirData software and exporting them into a .csv format for analysis with Python scripts. The trained YOLO model will then predict instances of olive knot disease from recorded UAV video footage. By matching timestamps (in milliseconds) between detections in video frames and corresponding entries in the flight logs, UAV coordinates at detection points can be determined. These coordinates will later be processed and visualized on images or maps to pinpoint infected tree locations within the plantation.

This flight pattern was chosen to simplify the process of identifying approximate coordinates during subsequent analysis.

Figure 4.2: Flight path used to collect the data.

4.2.2 Tree Mapping and Geospatial Cross-Referencing

Each olive tree in the studied area was individually mapped, with its geographical coordinates (latitude and longitude) recorded using *Google Maps*. This mapping was made to enable the spatial identification of each tree within the plantation. A geospatial cross-referencing methodology was implemented to associate disease detections from UAV video with specific trees.

When the object detection model identifies an olive knot in a video frame, the corresponding time stamp of that video is used to get the UAVs coordinates at the moment from the flight log. These coordinates were compared to the mapped tree locations using the Python library *GeoPy* [102], specifically the *geodesic* function, which calculates the distance between two points on the Earth's surface. If the UAVs position at the time of detection is within 1.5 meters of a mapped tree, that tree will then be flagged as potentially infected. This process ensures that each detection is attributed to the nearest tree, enabling the mapping of disease incidence within the olive grove.

4.2.3 Geospatial Visualization of Results

To enhance the interpretability and presentation of the geospatial analysis, the *Folium* Python library [103] was used for visualizing detection results on interactive maps. This was achieved by overlaying the detection points and tree positions on satellite base maps, providing an accessible way to explore the distribution of the disease within the plantation. This approach was also used to facilitate communication of findings with Farmers and support more informed decision-making, regarding disease management and intervention strategies.

4.2.4 Dataset Creation

Given the absence of publicly available datasets tailored to olive knot detection, it was necessary to create a dedicated dataset for training and evaluation. The process began with annotating the collected images using the Roboflow online platform, which facilitated manual labelling by creating bounding boxes around infected areas. Roboflow was chosen for its accessibility, ease of use, and ability to streamline annotation tasks directly through a web browser.

4.2.5 Annotation Methodology and Handling of Adjacent Instances

During the manual annotation process, a consistent methodology was followed to ensure reliable ground truth labels. In instances where multiple olive knot instances appeared in close proximity to one another, the visual distinction between those knots could be challenging, sometimes making two separate knots occur as a single, larger instance. Figure 4.3 shows a visual example of this problem.

To address this, the methodology described by [104] was followed, where individual bounding boxes were used for each distinct olive knot, even when they were very close together. This approach was adopted to improve the accuracy of the IoU metric. If

Figure 4.3: Example of olive knot instances in close proximity.

adjacent knots were annotated as a single instance, and the model predicted them as one or two separate instances, the IoU might fall below the defined threshold, leading to misclassification of the prediction. By using a consistent approach during annotation, the ground truth labels can better reflect the distribution of the disease while also reducing ambiguity during model assessment.

4.2.6 Dataset Preprocessing

The foundation of any robust ML pipeline lies in the quality and consistency of its data. In this work, dataset preprocessing was treated as a critical stage, directly shaping the reliability and accuracy of the entire detection system. Beyond simple collection and annotation, a rigorous curation process was implemented to maximize the dataset's informative value and minimize sources of noise and bias.

Images that could potentially impact model performance, such as those with very low resolution, significant occlusion, or poor lighting, were identified and, where appropriate, cropped out of the dataset. This careful filtering helps to mitigate the influence of ambiguous or low-quality samples from introducing noise and degrading the evaluation metrics.

Such diligence is essential, as mismatches between annotation and model detection, especially in borderline cases, can significantly distort the assessment of model effectiveness. Nevertheless, even after this filtering process, the dataset still retain many characteristics of real-world conditions, which is a good thing, as it ensures that the model is properly exposed to the natural variability and complexity encountered in practical environments.

As some of the cropping procedures may lead to images with a low resolution, all images with resolutions lower than 640×640 pixels were standardized to a minimum resolution of 640×640 pixels. This step not only prevents the detrimental effects of variable input sizes but also ensures that the model receives higher quality visual information, a best practice known to enhance the performance and stability of DL models in CV tasks [105]. By enforcing these preprocessing steps, the groundwork is laid for meaningful, reproducible, and high-impact model training.

4.2.7 Model Training

The model training is where the core of this work is located, as it is here that the system's ability to detect olive knot disease is created and refined. Leveraging the carefully curated dataset, the training process was designed to rigorously compare the capabilities off multiple state-of-the-art YOLO architectures (YOLOv8, YOLO11 and YOLO12) across all available variants.

The selection of those object detection models for comparative analysis in this research was guided by both their prominence in the current literature and their status as the most recent and advanced iterations of the YOLO series at the time of this study. YOLOv8 was chosen due to its widespread adoption and recognition as a benchmark in recent academic works, as a brief survey of the literature reveals that the vast majority of contemporary object detection studies utilize YOLOv8 as their primary model, reflecting its reputation as the previous state-of-the-art in this domain. This prevalence in the research community makes YOLOv8 an essential baseline for evaluating subsequent advancements.

YOLO11 was selected because it represented the latest major release from Ultralytics,

the maintainers of the YOLO framework, at the outset of this dissertation in September 2024. As documented in both the official Ultralytics documentation [59] and independent analyses, YOLO11 introduced significant architectural enhancements that improved accuracy, efficiency, and versatility across a range of computer vision tasks, making itself the state-of-the-art framework at that time.

During the course of this research, YOLO12 was released in February 2025, dethroning YOLO11 and establishing itself as the new state-of-the-art according to ultralytics and the broader CV community [73], [106]. YOLO12 incorporates further innovations, leading to superior performance in both accuracy and inference speed. Including YOLO12 in this study was therefore imperative to ensure the relevance and completeness of the comparative evaluation, as omitting the latest model would have limited the contemporary significance and impact of the findings.

By evaluating these tree models, this research provides a up-to-date assessment of the current capabilities of those architectures for olive knot disease detection.

To ensure scientific rigour and fair comparison, all models were trained under standardized conditions, with hyperparameters carefully controlled across experiments. Where hardware constraints necessitated adjustments, such as VRAM limitations, equivalent variants were treated consistently to preserve comparability and allow training on the available hardware. This approach eliminates potentially negative factors and allows for clear attribution of performance differences to model architecture rather than training setup.

Transfer learning was employed as a strategic choice, initializing models with weights pre-trained on large-scale datasets. This not only accelerates training and reduces computational demands but also allows the models to be trained with more generalized visual features, which enhances their ability to specialize in the olive knot detection with the available data.

Cross-validation was integrated into the training pipeline to rigorously assess generalization and reduce the risk of overfitting. This was done by iteratively partitioning the data into training, validation, and test subsets, allowing the model's robustness to be

systematically evaluated on unseen data, ensuring that the reported performance metrics are both reliable and indicative of real-world applicability.

Figure 4.4 encapsulates this methodology, illustrating the integration of data collection, data preprocessing, data annotation, model training, evaluation, and spatial mapping, supporting the proposed solution:

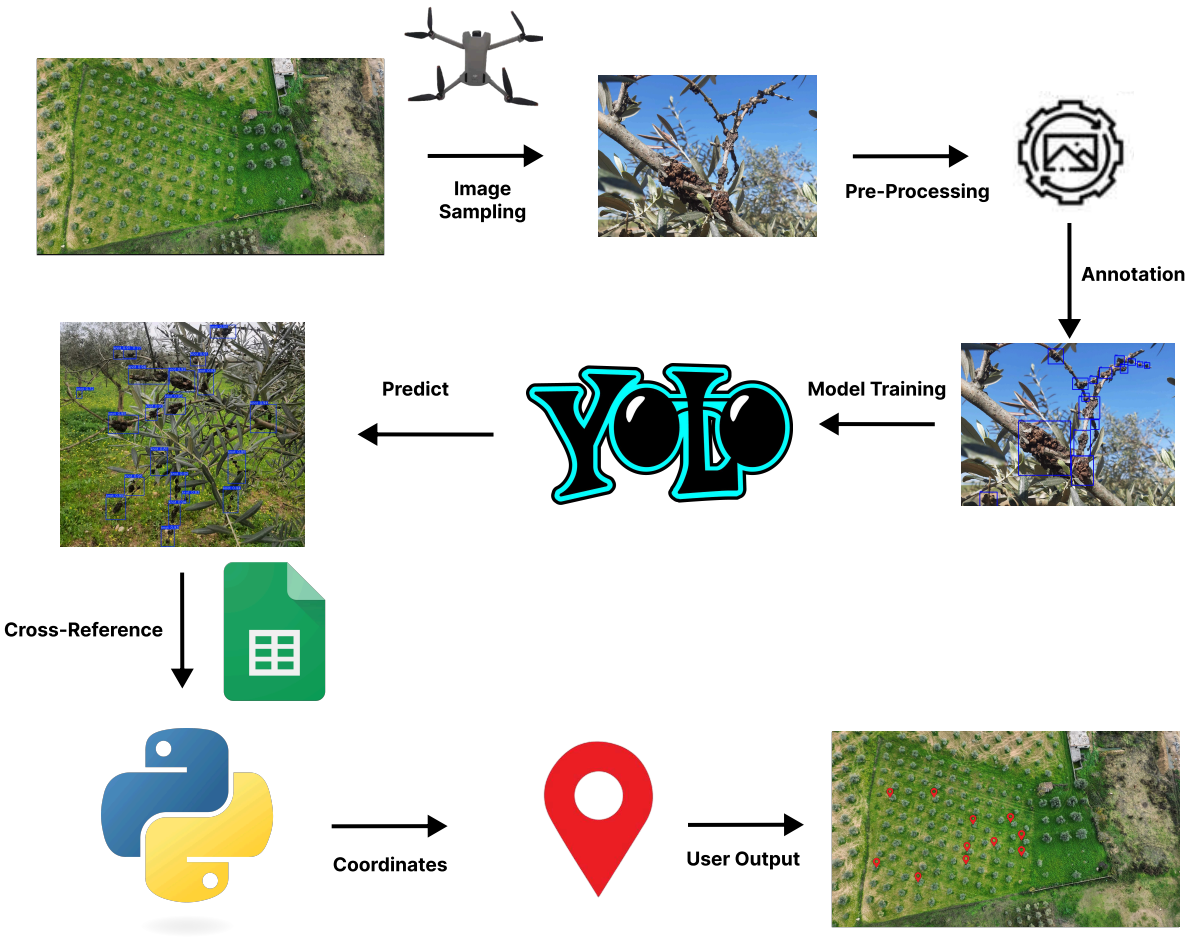


Figure 4.4: Overall system architecture.

Chapter 5

Development

This chapter presents the development of the proposed work, detailing the data collection, the experimental environment configuration, the dataset split, the hyperparameter settings, the methods used for cross-referencing the detections with the flight path data, and the methods used to verify the proximity of the detections to nearby olive trees.

5.1 Data Collection

The data collection process was conducted over five separate UAV flights. This division was primarily due to the need for battery replacements, which is a common operational constraint in UAV-based fieldwork. Additionally, an unexpected technical issue during one of the flights required a restart of the UAV, resulting in an extract flight and corresponding flight log.

Figure 5.1 presents, in a satellite image, the combined flight path of the UAV throughout the data collection campaign. In this figure, green markers indicate the starting points where the UAV began collecting images, while red markers represent the endpoints of each specific flight. The flight paths are colour-coded in blue, red, green, purple, and orange, corresponding to the first, second, third, fourth, and fifth flights, respectively. Images collected from the first, third, fourth, and fifth (blue, green, purple, and orange) that contained visible olive knot instances were manually annotated and incorporated into the

dataset. In contrast, no images from the red flight path (second flight) were included in the dataset. As a result, all images from this flight remain unseen to the models and will be exclusively used for testing the spatial mapping algorithm:

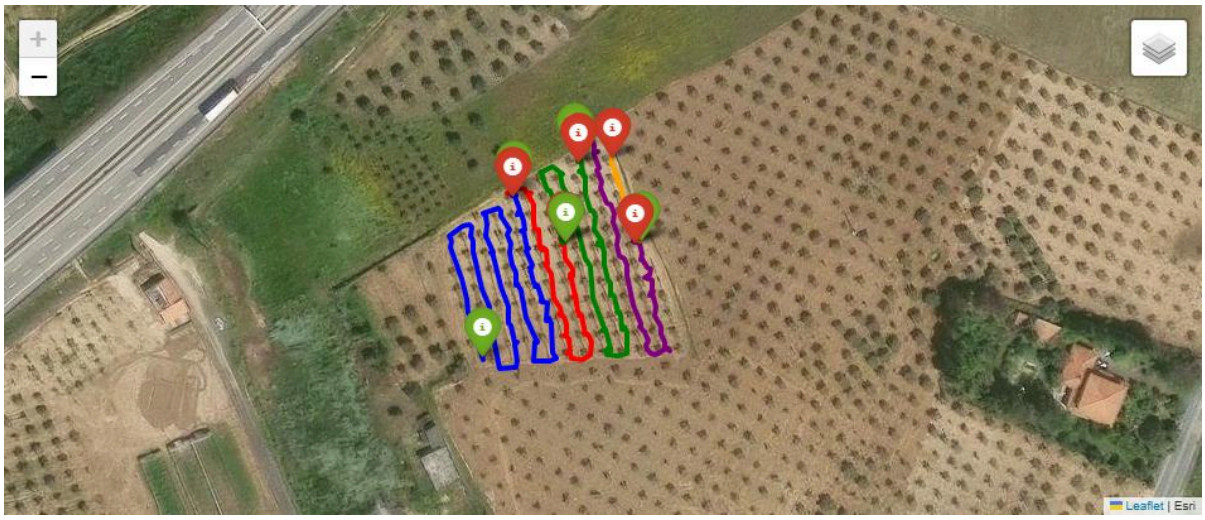


Figure 5.1: Combined flight paths of the all the UAV flights.

All flights were executed following the methodology outlined in 4.2.1, with the UAV performing lateral passes along the planting rows within the olive trees. It is important to note that all UAV flights were conducted manually, with an operator directly controlling the UAV throughout the data collection process. No autonomous flight modes or pre-programmed flight paths were used.

During the data collection process, a digital zoom was employed when hovering over each olive tree. This allowed for a closer inspection of each tree, at the cost of the loss of some image resolution, as digital zoom achieves magnification by cropping and enlarging the original image [107].

Furthermore, all UAV flights were conducted manually, with an operator directly controlling the drone throughout the data collection. No autonomous flight modes or pre-programmed flight paths were used; instead, each maneuver and inspection was performed in real time by a human operator. This manual approach provided greater flexibility in positioning the UAV for optimal image capture and allowed for immediate adjustments in response to field conditions.

5.1.1 Dataset Split

The dataset was divided to support cross-validation. Specifically, the data was split into three subsets: 70% for training, 20% for validation, and 10% for testing. The dataset was split using random sampling to ensure that each subgroup is representative of the overall data distribution. The complete dataset comprises 2050 images, which were distributed according to the aforementioned proportions.

5.2 Experimental Environment

The experimental environment designed for this work was built to support efficient training and evaluation of deep learning models. All experiments were constructed using the PyTorch deep learning framework. Development was carried out using the PyCharm development environment.

The experiments were executed on a workstation, with its configuration described by Table 5.1. The use of a dedicated GPU significantly accelerated model training and inference.

Table 5.1: Experimental environment configuration.

Environment Configuration	**Parameter**
Operating system	Windows 10
CPU	AMD EPYC 7443 CPU@ 2.85GHz
GPU	GeForce RTX 4090 (24GB)
Development environment	PyCharm 2024.3
Language	Python 3.11.9
Framework	Ultralytics YOLO 8.3.108
Cuda version	CUDA 11.8

A virtual environment was created to manage dependencies and ensure reproducibility, isolating the project packages from system-wide installations. PyTorch and all required libraries were installed within this environment, with GPU support enabled through the CUDA 11.8 toolkit version.

5.3 Hyperparameters Settings

To ensure a fair and meaningful comparison between the YOLO variants, all models, with the exception of the Extra-Large variants, were trained with the hyperparameters defined by Table 5.2. This configuration eliminated potential confounding factors and allowed the performance difference to be attributed to the model architectures themselves rather than variations in training setup. The selected hyperparameters were chosen based on their proven effectiveness in related works and best practices reported in the literature.

Model Version and Variant	**Hyperparameter**	**Parameter**
YOLOv8, YOLO11 and YOLO12 (nano, small, medium, large)	Epochs	300
	Batch Size	**16**
	Optimizer	AdamW
	Learning Rate	0.002
	Patience	20
	Weight Decay	0.00048
	Input Image Size	640
YOLOv8, YOLO11 and YOLO12 (extra-large)	Epochs	300
	Batch Size	**12**
	Optimizer	AdamW
	Learning Rate	0.002
	Patience	20
	Weight Decay	0.00048
	Input Image Size	640

Table 5.2: Hyperparameters.

The optimizer is responsible for updating the model's weights during training based on the computed gradients. The optimizer selected for all variants was AdamW, a modern adaptive optimization algorithm widely recognized in the literature for its effectiveness in DL tasks.

The patience parameter, set to 20, is used as part of an early stopping strategy. During training, if a monitored metric (such as validation loss or accuracy) does not improve for 20 consecutive epochs, the training process is stopped. This prevents unnecessary computation and mitigates the risk of overfitting by stopping when the model ceases to make progress.

The input image size was standardized to 640 × 640 pixels for all model variants, following common practice in the literature for object detection tasks using the YOLO architecture. This input size offers a balanced compromise, as it is sufficiently large to present important visual details necessary for detecting smaller objects, like olive knots, while remaining computationally efficient for training and inference on modern GPUs. Setting a standard for input size also ensures consistency across experiments, making fair and meaningful comparison between different models easier.

The remaining selected values for epochs, batch size, learning rate, and weight decay follow the recommended defaults provided by the YOLO framework and are widely adopted in related literature. The number of epochs were set to 300, a value that usually allows sufficient training cycles for the model to properly converge without too much risk off overfitting, especially in datasets of moderate size. The batch size was chosen as 16, as it is a value in line with common practice for training YOLO models while also balancing the need for efficient GPU utilization. For the extra-large models, that value was reduced to 12 due to VRAM limitations, with this value being the highest possible for training in the available hardware. The learning rate and weight decay were also set according to common recommendations for the AdamW optimizer in object detection tasks.

5.4 Flight Path Cross-Referencing

One major challenge in cross-referencing UAV detections with flight path data is the mismatch between the number of video frames and the number of recorded UAV position instances in the flight log. For example, a typical video may contain approximately 16000 frames, while the corresponding flight log contains only 6000 position entries. This discrepancy arises because the flight log records data at a lower frequency, in this case, every 0.1 seconds, whereas the model makes predictions on every individual video frame.

To address this, it is necessary to align the two datasets so that each video frame can be associated with a corresponding UAV coordinate. Simply reducing the number of video frames to match the flight log would risk losing valuable information about the

disease. Instead, the approach chosen was to increase the number of flight path instances to match the number of video frames through interpolation.

The *NumPy* library's *interp* function was used for this purpose. This function performs one-dimensional linear interpolation, estimating the UAVs position at any given video frame timestamp based on the available flight log data. The general equation for the linear interpolation used by *interp* can be seen below:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}, \tag{5.1}$$

where x is the target timestamp (frame time), x_0, y_0 and x_1, y_1 are the known data points surrounding x, and y is the interpolated value (latitude or longitude).

To generate the target timestamps for interpolation, the function *arange* from the *NumPy* library was used, producing a sequence based on the number of video frames. Each frame index was multiplied by the frame interval, calculated:

$$\frac{1000}{fps}, \tag{5.2}$$

where *fps* is the video frame rate (26.06 frames in this work).

Table 5.3 provides an overview of the number of frames in each video compared to the number of instances in each corresponding flight path. The 3^{rd} flight had a problem in its recording, so the number of video frames in that video was much lower than the other videos.

Flight	**Video Frames**	**Flight Log Entries**
1^{st} flight (blue path)	24540	13990
2^{nd} flight (red path)	16015	6147
3^{rd} flight (green path)	647	6210
4^{th} flight (purple path)	18868	3899
5^{th} flight (orange path)	17172	951
Total:	77242	31197

Table 5.3: Video frames x Flight log entries.

It is important to note that, due to the UAVs inspection methodology, the UAV often

remains stationary at a given coordinate for several consecutive frames before moving quickly to the next tree. This means that interpolating the flight log data will most likely not introduce significant errors, as the UAVs position is relatively stable during these inspection periods. This approach ensures that no detections are missed due to temporal misalignment. The resulting spatial analysis accurately reflects the UAVs inspection path and the locations of detected olive knots.

5.5 Spatial Cross-Referencing of Detections and Tree Locations

The verification of the proximity of detected olive knot instances to nearby olive trees was performed through a geospatial cross-referencing methodology based on [108].

Each olive tree in the study area was mapped with its geographical coordinates, represented as pairs of latitude and longitude values (ϕ_i, λ_i) for the i^{th} tree, collectively denoted as $(\phi_i, \lambda_i)_{i=1}^{N}$.

The UAV video footage was analysed frame-by-frame by the detection model, producing a set of predicted detections. Each detection was associated with a timestamp corresponding to a video frame index t_j where j indexes the frames.

The interpolation used to estimate the UAVs position at each video frame yielded UAV coordinates (ϕ_j, λ_j) for each frame t_j. To determine whether a detected olive knot corresponds to a specific tree, the geodesic distance d (the locally shortest arc between two points on a surface) between the detection coordinate (ϕ_j, λ_j) and each tree coordinate (ϕ_i, λ_i) was calculated using the geodesic function from the *GeoPy* library [102], which can be seen below:

$$d = geodesic((\phi_j, \lambda_j), (\phi_i, \lambda_i)). \tag{5.3}$$

A detection was considered to be in proximity to a tree if the distance d was less than or equal to a predefined radius r of 2 meters ($d \leq r$).

Formally, for each tree i, a boolean indicator P_i was defined by:

$$P_i = \bigvee_j (d_{ij} \leq r), \tag{5.4}$$

where d_{ij} is the geodesic distance between tree i and detection j, and $\bigvee$ denotes the logical OR over all detections j.

This indicates that tree i is flagged as infected if any detection falls within the predefined proximity radius.

Figure 5.2 shows all the mapped olive trees as blue circles using the *Folium* library. Figure 5.3 shows the mapped olive trees as well as the flight path followed by the UAV in red. The aerial view of the olive groves shows that the UAV consistently flew closest to the olive trees on the left (west) side. This positioning ensures that there is no risk of a detection mistakenly identifying two trees as infected due to their proximity, as the UAV was never close enough to more than one tree at a time in that area:

Figure 5.2: Olive trees locations.

To enhance the visualization and interpretation of the spatial analysis, the *Folium* library was also used to generate an interactive map of the olive grove. Whenever a tree was flagged as infected, based on the proximity analysis, its corresponding circle on the map was coloured red. Additionally, when hovering the mouse over a red circle, a

Figure 5.3: Olive trees locations and UAV flight path.

pop-up appears displaying that the tree is infected along with its latitude and longitude coordinates, allowing users to identify and locate infected trees within the plantation easily.

Chapter 6

Results

This chapter shows the results obtained from the training of the different models and the related analysis. Additionally, it includes the outcomes of the predictions made by the best-performing model, as well as the visualization of these results using the map interface. The methods and results of cross-referencing tree locations with disease detections are also discussed in this chapter.

6.1 Model Evaluation and Comparison

To rigorously assess the performance of object detection architectures for olive knot detection, a set of experiments was conducted using all available variants of the YOLOv8, YOLO11 and YOLO12 models. All models were trained under identical conditions, following the conditions described in Chapter 4.

Table 6.1 summarized the performance metrics for each variant of YOLOv8, YOLO11, and YOLO12, including precision, recall, $F1$ Score, $mAP@50$, and $mAP@50-95$. These metrics were obtained by evaluating each trained model on the designated test dataset split, ensuring an unbiased assessment of their generalization capabilities. Several key observations can be made from these results:

Firstly, it is notable that the small, medium, and large variants of each YOLO version often outperform the extra-large variant, particularly in terms of $F1$ Score and mAP.

Model	**Precision**	**Recall**	**F1 Score**	$mAP@50$	$mAP@50-95$
YOLOv8n	0.752	0.704	0.727	0.768	0.422
YOLOv8s	0.771	0.698	0.732	0.781	0.435
YOLOv8m	0.787	0.756	0.771	0.817	0.494
YOLOv8l	0.847	0.771	0.807	0.845	0.508
YOLOv8x	0.779	0.708	0.742	0.783	0.435
YOLO11n	0.806	0.736	0.769	0.818	0.499
YOLO11s	0.821	0.761	0.789	0.839	0.506
YOLO11m	0.805	0.751	0.777	0.825	0.486
YOLO11l	0.831	**0.804**	0.817	0.853	0.525
YOLO11x	0.799	0.763	0.781	0.821	0.473
YOLO12n	0.813	0.756	0.784	0.837	0.504
YOLO12s	0.832	0.755	0.792	0.834	0.513
YOLO12m	**0.848**	0.801	**0.824**	**0.861**	**0.555**
YOLO12l	0.802	0.776	0.789	0.832	0.488
YOLO12x	0.833	0.749	0.789	0.839	0.499

Table 6.1: Test validation results of each model.

For example, YOLO12m achieves the highest $mAP@50$ (0.861) and $F1$ Score (0.824), surpassing the corresponding extra-large model, which achieves an $F1$ Score of 0.789 and $mAP@50$ of 0.839. Similarly, in both YOLOv8 and YOLO11, the large variant outperforms the extra-large in both $F1$ Score and mAP metrics. The best metrics among all models are highlighted in bold, with the YOLO12m being the best in all metrics, except recall, where it loses to the YOLO11l by a mere 0.003.

This outcome is somewhat unexpected, as larger models, such as the extra-large variants, are typically anticipated to deliver superior performance due to their greater capacity and representational power. However, the results suggest otherwise in this work, and there are a few likely reasons for this outcome. The dataset used for training consisted of 2050 images, which, while adequate for a specialized application, may not provide the diversity or sheer volume of diversity required for the largest models to achieve their full potential. Larger models generally benefit from extensive and varied data to generalize well, and in their absence, their increased complexity can become a liability. Another important consideration is the tendency of larger models to overfit when trained on relatively small or homogeneous datasets. Given the objective of comparing model architectures under equal

conditions, the available dataset may inherently favour models that are less data-hungry and better suited to smaller, less diverse datasets. In this context, the medium and large variants appear to strike the optimal balance across evaluation metrics.

The $F1$-confidence curve for each model (Figures 6.1, 6.2, and 6.3) shows how the choice of detection confidence threshold affects the trade-off between precision and recall. Typically, the $F1$ score increases with confidence, peaks at an optimal value, and then declines as true positives are missed. Models with broader peaks are more robust to threshold selection. The precision-recall curve (Figures 6.4, 6.5, and 6.6) indicates overall detection performance, with curves closer to the upper right representing better results.

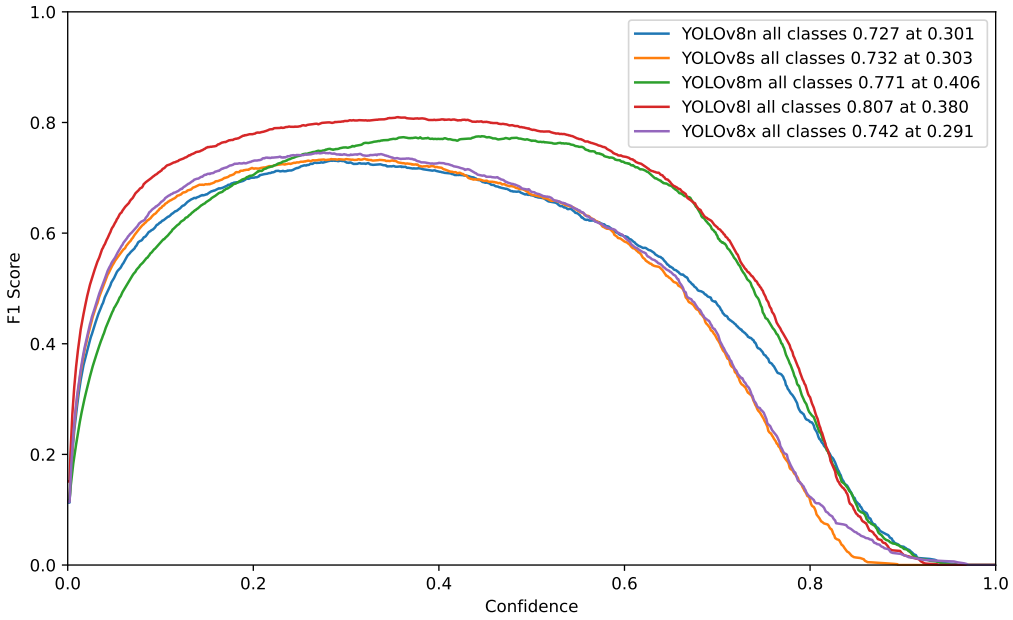


Figure 6.1: YOLOv8 variants F1-Confidence curve comparison.

Figure 6.1 presents the $F1$-confidence curves for the YOLOv8 variants. Here, the YOLOv8m and YOLOv8l stand out. According to the caption, e.g. YOLOv8m all classes 0.771 at 0.406, indicates that YOLOv8m achieved its peak $F1$ score (0.771) at a confidence threshold of 0.406. YOLOv8l, on the other hand, reached a higher peak $F1$ score (0.807) at a slightly lower confidence threshold (0.380). Notably, the YOLOv8l curve exhibits a broader peak, suggesting greater robustness and less sensitivity to the exact choice of confidence threshold. The remaining variants performed considerably worse and are not

further discussed here.

Figure 6.2 shows the $F1$-confidence curves for the YOLO11 variants. In this case, the performance of the models is closer, with the small, medium, and large variants all achieving similar peak $F1$ scores. The YOLO11l model attains the best peak, with 0.817 at 0.328, followed by YOLO11s, with 0.789 at 0.324 and YOLO11m with 0.777 at 0.366. Similarly to the YOLOv8l, the YOLO11l displays a broader peak, indicating robustness to the choice of confidence threshold.

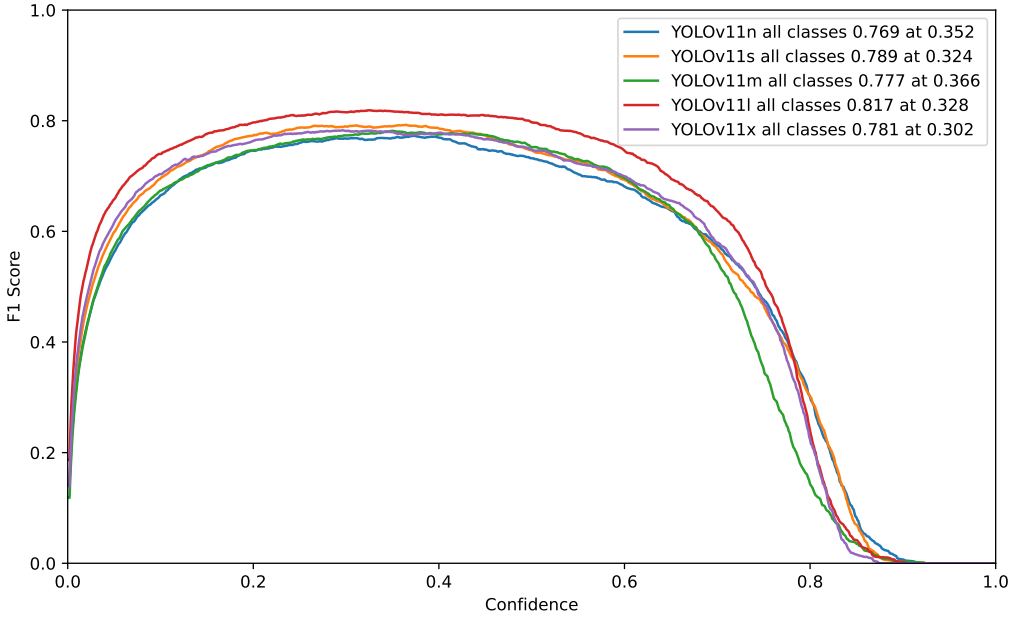


Figure 6.2: YOLO11 variants F1-Confidence curve comparison.

Then, in Figure 6.3, we have the $F1$-confidence curves for the YOLO12 variants. The difference between the variants are even smaller, reflecting a close competition among all models. The YOLO12m achieves the highest $F1$ score, with a 0.824 at 0.366. Although all YOLO12 variants achieved their peak $F1$ scores at confidence thresholds below 0.4, all their curves remain broad and stable, only losing performance after 0.6 confidence. This indicates that these models are robust to threshold selection and maintain a considerably high performance across a wide range of confidence values.

Figure 6.4 shows the precision-recall curves for the YOLOv8 variants. YOLOv8l achieves the highest $mAP@50$, with 0.845, indicating the largest area under the curve

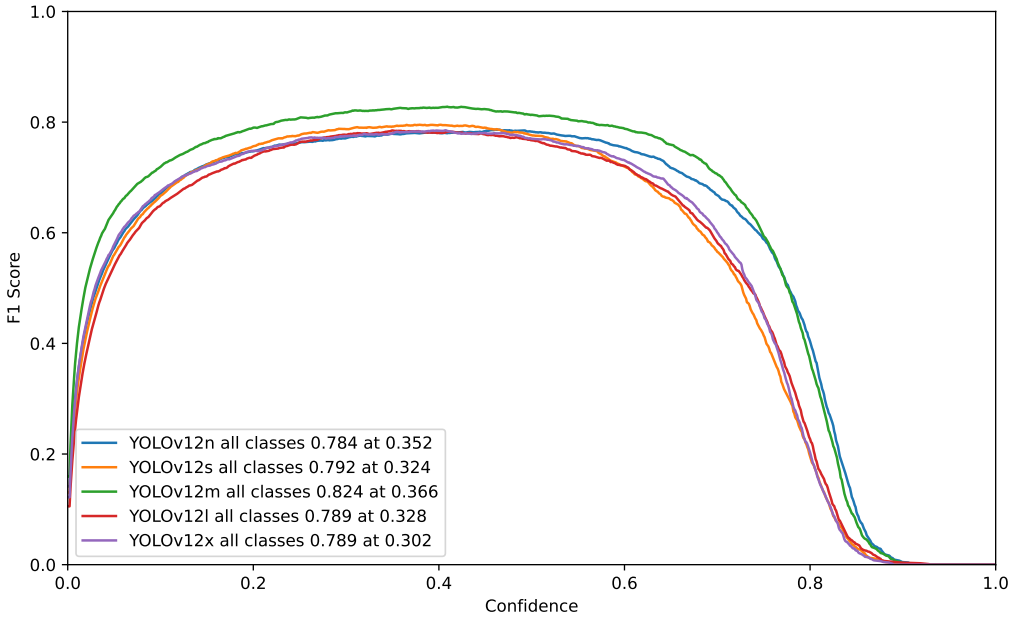


Figure 6.3: YOLO12 variants F1-Confidence curve comparison.

and thus the best average detection performance among YOLOv8 models. YOLOv8m follows as the second best performer with a $mAP@50$ 0.817, showing consistency with the $F1$-confidence results.

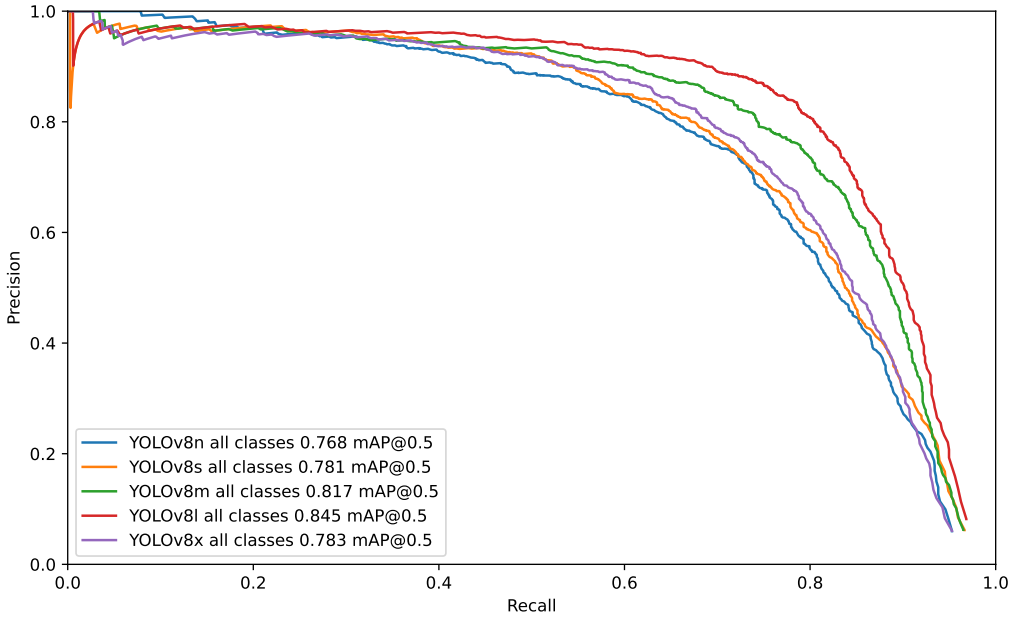


Figure 6.4: YOLOv8 variants precision-recall curve comparison.

Figure 6.5 presents the precision-recall curves for the YOLO11 variants. Here, the models are even closer in performance, but YOLO11l stands out with the best $mAP@50$ of 0.852, followed by YOLO11s. The curves confirm the trend seen in the $F1$-confidence analysis, with the large variant providing better results.

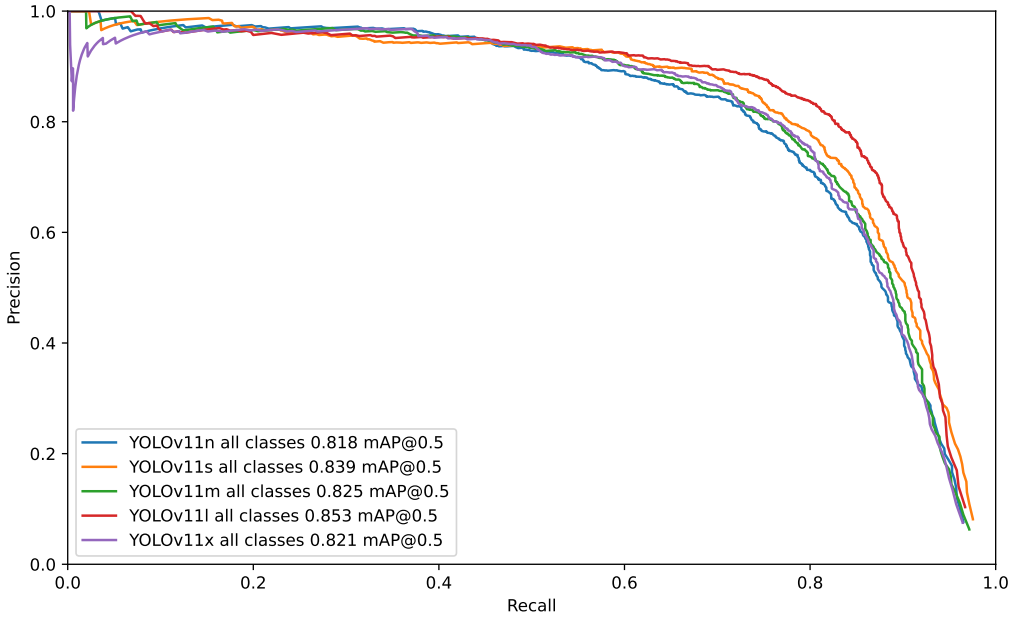


Figure 6.5: YOLO11 variants precision-recall curve comparison.

Finally, Figure 6.6 shows the same curves but for the YOLO12 variants. While most variants are closely matched, the YOLO12m model achieves a notably higher $mAP@50$ of 0.861, positioning its curve the closest to the upper right corner and confirming its superior performance.

In summary, the detailed analyses of the $F1$-confidence and precision-recall curves reinforce the earlier quantitative findings. For this dataset and detection task, the medium and large YOLO variants, particularly the YOLO12m variant, provide the most reliable and robust detection performance. The broad peaks and high mAP values indicate that these models are not only accurate but also stable across a wide range of confidence thresholds, which makes them well-suited for practical applications in olive knot detection.

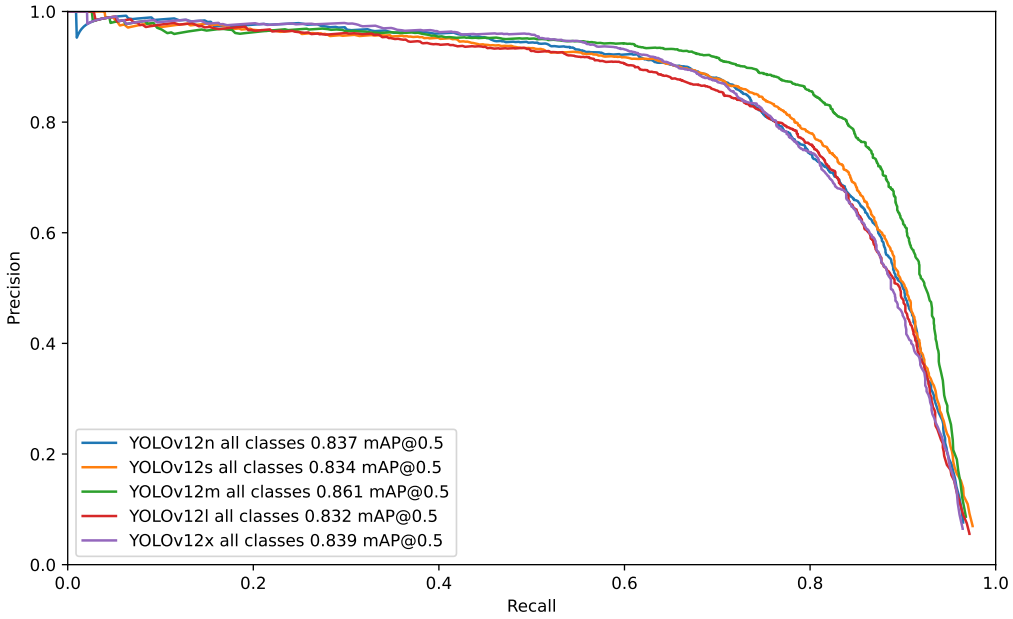


Figure 6.6: YOLO12 variants precision-recall curve comparison.

6.2 Detection Results and Spatial Mapping

After identifying the best-performing model from the YOLO variants, this section presents the results of applying that model to the flight videos, while focusing on the spatial mapping of detections within the olive grove.

6.2.1 Description of the Detection and Mapping Process

The detection process began by running the selected model on the flight videos, resulting in a set of predicted olive knot instances. A confidence threshold of 0.5 was applied, and the *stream* parameter in the prediction mode was set to *True*. This approach was adopted to prevent out-of-memory errors, which can occur because the default prediction mode returns a Python list of *Results* objects, storing all predictions in RAM. Since the input videos are lengthy, this led to memory exhaustion on the workstation. By enabling the *stream* parameter, the model instead returns a generator of *Results* objects, which is a memory-efficient construct that yields one result at a time, only as needed, rather than processing and storing the entire sequence in memory simultaneously.

It is important to note that this detection and mapping process was applied exclusively to the second video, corresponding to the red flight path in Figure 5.1, as this video was reserved as a completely unseen set of data for testing this algorithm.

To accurately localize these detections within the grove, each frame's timestamp was cross-referenced with the interpolated UAV flight path data, as described in 4. A geodesic proximity method was employed to map the detections to individual olive trees. For every detection, the geodesic distance between the detection cross-referenced coordinates and each mapped tree's coordinates was calculated. A detection was considered to correspond to a tree if it occurred within a 2-meter radius of that tree. Trees flagged as infected were thus with at least one detection within this proximity threshold.

6.2.2 Spatial Visualization and Analysis

Figure 6.7 presents the ground truth map generated from on-site visual inspections during the UAV data collection phase. In this map, trees marked in orange correspond to those confirmed as infected with olive knot disease. In contrast, trees marked in blue were either not infected or exhibited mild symptoms that were barely visible even upon close human inspection. As such, these subtle cases are unlikely to be detectable by UAV imagery or the detection model:

Figure 6.7: Second flight ground truth.

Figure 6.8 displays the spatial distribution of infected trees as predicted by the best-performing model. In this visualization, the model's detections are mapped in the olive grove, allowing direct comparison with the ground truth. The results demonstrate that the system is capable of accurately detecting and localizing most instances of olive knot disease, with the majority of predicted infected trees aligning closely with those identified during field inspection:

Figure 6.8: Model predicted spatial distribution.

However, there are a few notable discrepancies. In two cases, the model failed to detect infection in trees that were visually confirmed as infected; this can be categorized as false negatives. Conversely, there is one instance where the model predicted infection in a tree that was not identified as infected during visual inspection, categorizing this as a false positive. The trees with associated problems are shown on the map in Figure 6.9, where green boxes indicate the problematic detections:

Figure 6.9: Highlighted problematic detections.

Figure 6.10 illustrates one of the trees where olive knot instances should have been detected but were missed, likely due to the low resolution of the image for that particular tree. In blue, there are a few bounding boxes illustrating instances of knots that should have been detected. Figure 6.11 shows the tree in which the model predicted the presence of a knot where there should not have been one. This false positive was caused by the model misidentifying a partially covered trunk under poor lighting conditions as an olive knot.

These examples highlight both the strengths and current limitations of the approach. At the same time, the geodesic proximity method and UAV flight pattern enable robust spatial attribution of detections and the system demonstrates strong performance in most cases, detection accuracy can still be affected by factors such as image quality, resolution, occlusion, subtlety of symptoms, and challenging visual conditions, leading to some degree of misclassification.

In all the spatial mapping analyses presented thus far, the manually collected coordinates for each olive tree were used as the basis for mapping detections and ground truth. However, in scenarios where the olive grove is extensive, reaching kilometres of extension, manually recording the position of every tree can become an extremely labour-intensive

Figure 6.10: False negative example.

Figure 6.11: False positive example.

and time-consuming task. To address this, this work is also capable of providing the geographic coordinates for each detection directly.

By filtering duplicate or near-duplicate detection coordinates, it is possible to generate a spatial map of detected infections even in the absence of a manually collected coordinate inventory. Figure 6.12 illustrates this approach. While the resulting coordinates may be slightly less precise compared to the geodesic method, this strategy still enables spatial mapping of olive knot detections, making it feasible to monitor extensive groves:

Overall, these spatial mapping results highlight the effectiveness of the proposed methodology in identifying and localizing olive knot disease within the grove, while also

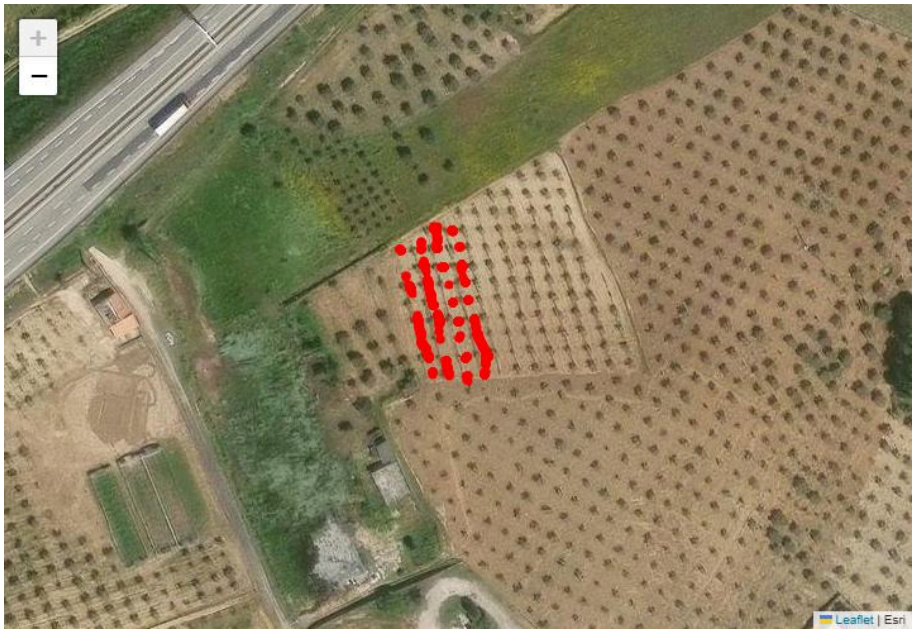


Figure 6.12: Spatial mapping of detections using automatically generated coordinates.

illustrating areas for further refinement, particularly in reducing false negatives and false positives through improved data quality or model tuning.

6.2.3 Visualization and Interactive Map Interface

The interactive map generated using the *Folium* library provides a user-friendly interface for exploring the spatial distribution of olive knot disease within the plantation. All olive trees are represented as markers on the map, with infected trees distinctly highlighted in red to facilitate quick identification. Figure 6.13 illustrates this map.

A key feature of the map is the implementation of tooltips that appear when the user hovers the mouse over any tree marker. These tooltips display information like the precise geographic coordinates of the tree and its infection status. This interactive element enhances the usability of the map by allowing users to obtain detailed information about individual trees without having to cluster on the visual display.

Additional functionalities, such as zooming and panning, enable users to navigate the map efficiently and focus on areas of interest. This map supports decision-making processes by providing precise and accessible spatial information about the incidence of

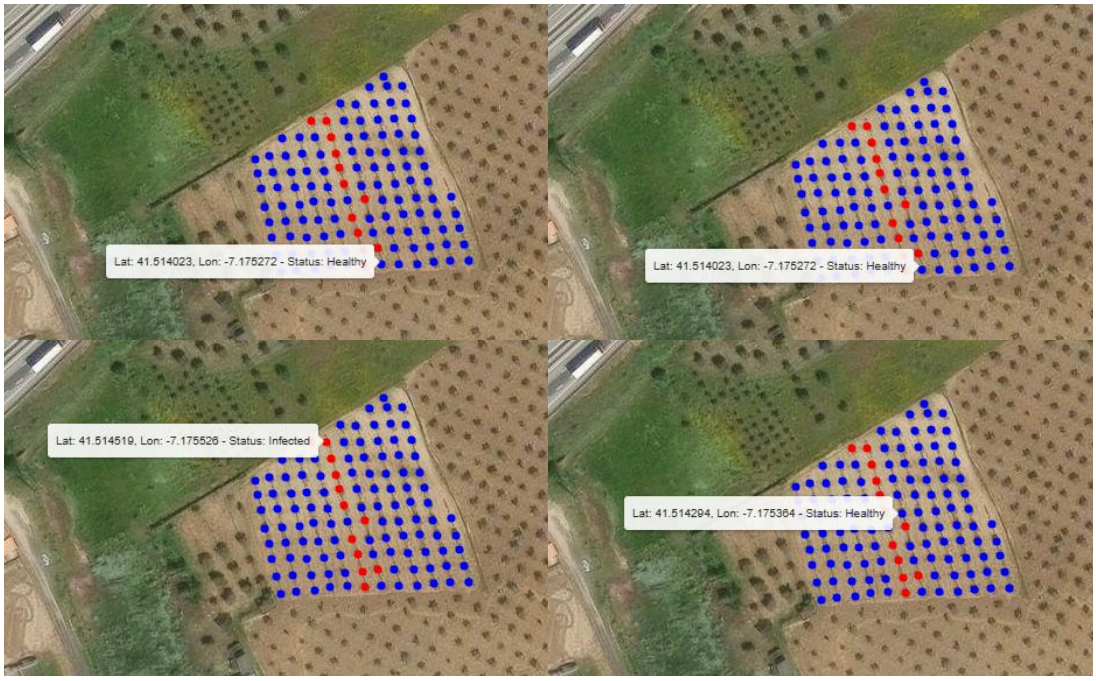


Figure 6.13: Interactive map generated using the *Folium* library.

the disease.

Overall, this visualization tool serves as an effective means to communicate the results of the detection system and supports target disease management interventions.

6.3 Discussion and Interpretation

The results obtained from this study provide a thorough evaluation of the proposed methodology for detecting and mapping olive knot disease through UAV imagery and advanced object detection models. Several important insights and lessons can be drawn from the experimental outcomes.

6.3.1 Inferences and Insights

The comparative analysis of YOLOv8, YOLO11, and YOLO12 variants demonstrates that medium and large model architectures consistently outperform their extra-large counterparts in both $F1$ Score and mAP. This finding suggests that, for the dataset size and

diversity available in this study, increasing model complexity beyond a certain point does not yield additional benefits and may, in fact, hinder generalization. The $F1$-Confidence and precision-recall curves further confirm that the best-performing models achieve a robust balance between precision and recall, with optimal thresholds identified around 0.5 to 0.6. These results validate the effectiveness of the chosen approach for olive knot detection, particularly in scenarios where practical constraints limit data collection.

Spatial mapping of detections revealed a strong correspondence between model predictions and ground-truth infection status, as verified by on-site visual inspections. The integration of UAV flight path data, geodesic proximity analysis, and interactive mapping enabled precise localization of infected trees within the grove. Notably, the spatial distribution of detected infections often reflected the areas where the UAV maintained optimal proximity and image quality, underscoring the importance of careful flight planning in maximizing detection performance.

These findings directly address the primary objective of developing and validating a cost-effective UAV based system for olive knot disease detection and spatial mapping. The successful use of lightweight DL models on imagery collected from an affordable UAV platform demonstrates the feasibility of applying such technology in resource-constrained agricultural contexts.

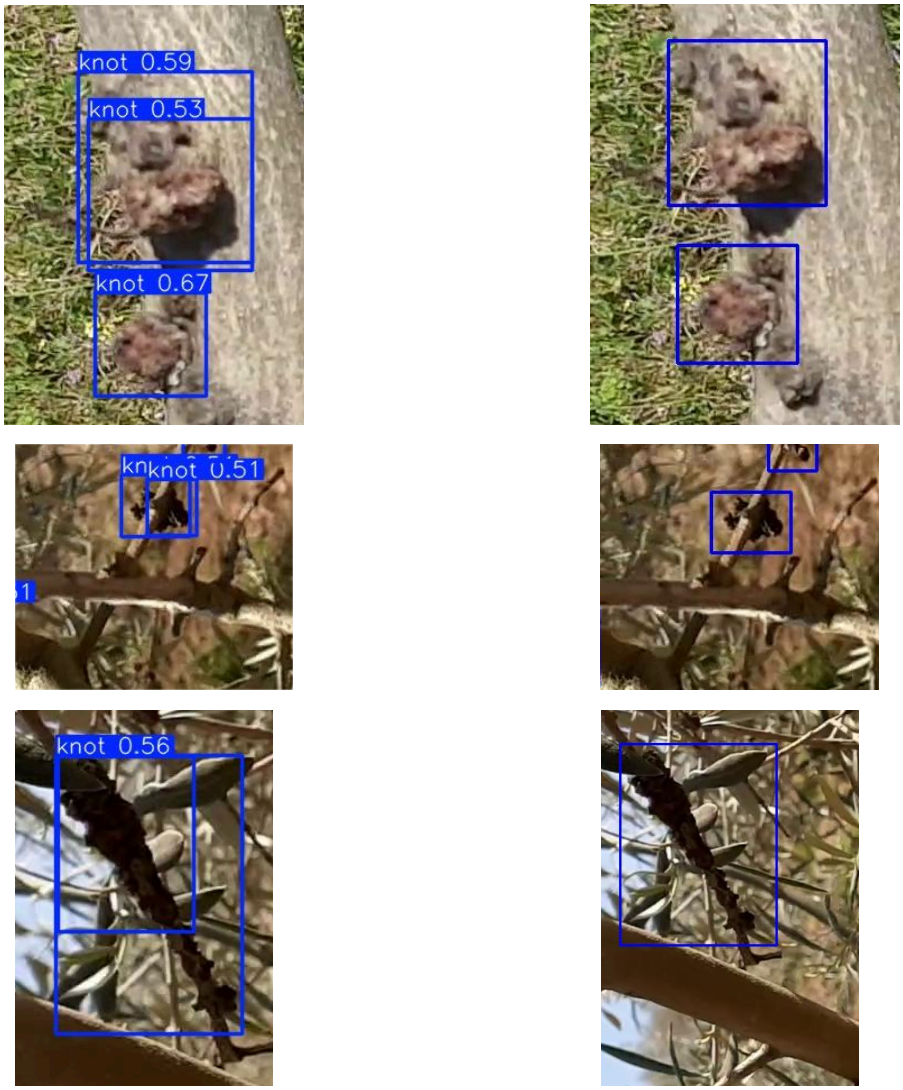
6.3.2 Limitations and Improvements

Despite the overall success of the system, several limitations were encountered. One significant challenge was the impact of image resolution and quality on detection accuracy. In cases where the UAV employed digital zoom or when lighting conditions were suboptimal, the model occasionally failed to detect knots that were present or misclassified non-infected features as knots. The reliance on manual annotation and the inherent subjectivity of visual ground truth also introduce some uncertainty into the evaluation.

Another limitation is related to the scalability of manual tree mapping. While precise coordinates for each tree were available in this study, collecting such data for larger groves

would be impractical. Although the alternative approach of mapping detections directly from UAV-derived offers a feasible solution, it may sacrifice some positional accuracy.

Furthermore, the system's ability to detect very early or subtle symptoms remains limited by the resolution of UAV imagery and the inherent difficulty of visually identifying olive knot symptoms, as they often appear under dense foliage. Manual annotation and tree mapping, while effective for small groves, are not scalable to larger operations without automation. These limitations point to clear directions for future work.



(a) Instances with abnormal bounding boxes. (b) Corresponding ground truth.

Figure 6.14: Side by side comparison of abnormal predictions vs the corresponding ground truth.

A particularly relevant limitation affecting the evaluation metrics arises from annotation and prediction mismatches. In some cases, the ground truth annotations define multiple adjacent olive knots as separate instances, while the model predicts them as a single instance (or vice-versa). These types of scenarios are illustrated in Figure 6.14, which presents images from the validation dataset. When such mischaracterizations occur, the IoU between predicted and ground truth bounding boxes can be negatively affected, as the area of overlap is much smaller than it should be. Consequently, this negatively affects metrics such as precision, recall, and mAP, making it more challenging to accurately assess the model's true detection capabilities. Addressing this issue, potentially through improved annotation protocols or the adoption of techniques, remains an important path for future improvement.

6.3.3 Exceeding Objectives

Several aspects of the system performed beyond initial expectations. The spatial-cross referencing methodology, combining interpolated flight path data and geodesic proximity, proved highly effective for attributing detections to individual trees, even in an area that contains trees relatively close by. The interactive map interface provided an intuitive and informative tool for exploring detection results, supporting both research analysis and practical disease management.

The creation of a novel, annotated dataset using a cost-effective UAV platform was achieved, and the dataset proved to be sufficient for training and evaluating multiple object detection models. The study also highlighted the importance of dataset diversity and image quality for maximizing model performance, thus fulfilling the objective of dataset creation while identifying areas for future improvement.

In summary, the results demonstrate that all primary objectives of this work were addressed. The system was successfully implemented and validated using cost-effective UAV technology and lightweight DL models, with robust spatial mapping and practical visualization tools. The methodology was thoroughly tested and shown to be effective in

real-world conditions, supporting the practical deployment of automated disease detection in olive groves. Nonetheless, the identified limitations highlight important areas for future research and system enhancement.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This dissertation developed and validated a cost-effective, UAV-based system for the automated detection and spatial mapping of olive knot disease in olive groves. Motivated by the need for accessible, scalable, and efficient disease monitoring solutions in agriculture, the work leveraged lightweight deep learning models and affordable UAV platforms to address the challenges of plant disease detection in real-world, resource-constrained environments.

The main objectives were successfully achieved. A novel, annotated dataset of olive knot disease was created using imagery collected from a cost-effective UAV, and several state-of-the-art object detection models (YOLOv8, YOLO11, YOLO12) were trained and rigorously evaluated under identical conditions. Among all tested models, the YOLO12m variant achieved the highest overall performance, with a Precision of 0.848, recall of 0.801, $F1$ Score of 0.824 at a confidence threshold of 0.366, $mAP@50$ of 0.861, and $mAP@50-95$ of 0.555 on the test dataset. These results indicate that the YOLO12m model is highly effective in detecting olive knot disease, providing a robust balance between minimizing false positives and false negatives.

From the ground-truth data collected in the field, 13 olive trees were confirmed to be

infected. The YOLO12m model correctly detected infection in 11 of these trees, missing only 2 infected trees (false negatives), and produced just one false positive, where a healthy tree was incorrectly flagged as infected. This result demonstrates a high detection rate of 84% and a very low false positive rate, emphasizing the practical effectiveness of the proposed system.

The methodology for spatial cross-referencing, which consisted of combining interpolated UAV flight path data and geodesic proximity analysis, proved to be highly effective for attributing detections to individual trees, even in complex field conditions. The interactive map interface developed using the *Folium* library enabled intuitive visualization and analysis of disease incidence, supporting both research and practical grove management. The system was able to detect and localize olive knot disease in most cases, with strong correspondence between model predictions and ground-truth visual inspections.

These findings are in line with those reported in related works, which highlight the potential of UAVs and DL for PA and disease monitoring. However, the research also revealed important limitations. The detection of early or subtle symptoms remains a challenge, primarily due to the resolution limits of the utilized UAV and the inherent difficulty of visually identifying olive knot disease. Image quality, environmental conditions, and the scalability of manual tree mapping also present practical constraints. Similar studies share these limitations in the field and point to clear directions for future research.

The process of developing and evaluating this system provided valuable insights into the interplay between data quality, model complexity, and real-world applicability. The work demonstrates that it is possible to combine affordability, accuracy, and usability in a disease detection system, paving the way for broader adoption in agricultural practice. The strong performance of the YOLO12m model, as evidenced by its metrics, confirms the viability of this approach for practical deployment in olive groves.

7.2 Future Works

For future work, several avenues remain open for further development and refinement. Integrating higher-resolution or multispectral sensors could substantially improve the detection of early-stage symptoms, addressing one of the main limitations identified in this study. The development of automated methods for mapping tree positions would enhance scalability and reduce the need for manual intervention, making the approach more practical for large-scale plantations.

Expanding the dataset to include a greater diversity of conditions, disease stages, and even different diseases affecting olive trees is another important direction. This, combined with the application of more advanced data augmentation techniques, could improve both the robustness and the accuracy of the detection models. Training models capable of identifying multiple disease classes would increase the system's versatility and value to farmers.

Field validation in collaboration with farmers and agronomists will be essential for refining the system and maximizing its real-world impact. Deploying the proposed model on UAVs operating in real-time environments will allow for a comprehensive assessment of performance beyond offline analyses, providing insights into practical feasibility and operational challenges.

Another promising direction is the use of segmentation techniques, both in annotation and model prediction. By applying instance or semantic segmentation, it would be possible to distinguish individual olive knots at the pixel level, rather than relying solely on bounding boxes. This approach could help prevent situations where the model predicts a single olive knot when, in reality, multiple knots are present, or when bounding boxes overlap ambiguously.

Finally, the development of UAV platforms from scratch, based on open-source software and hardware, could enable on-site inference, real-time prediction, and immediate visualization of results. Such a system would allow the UAV to fly along olive groves autonomously, make detections in real-time, save georeferenced results, and present them

through a more customizable and accessible visualization tool. This would significantly enhance the autonomy, scalability, and accessibility of disease monitoring in olive production.

7.3 Academic Contributions

The following are the main contributions resulting from my research work during my master's degree. In total, two papers were developed and submitted to a specialized conference, all of which were accepted and presented:

1. **Morais, M. H. F.**, Mendes, J., Santos, M. F., Fernandes, F. M., Lima, J. L., and Pereira, A. I., "*A YOLO-based approach for Detection of Olive Knot disease through UAV and Computer Vision Technologies*", 26th International Carpathian Control Conference - ICCC. Slovak Republic, 2025.

2. Fernandes, F. M., Santos, M. F., **Morais, M. H. F.**, Lima, J. L., Pereira, A. I., and Mercorelli, P., "*Advancing Sustainability and Productivity: The Role of Precision Agriculture in Vineyards and Olive Groves*", 26th International Carpathian Control Conference - ICCC. Slovak Republic, 2025.

Bibliography

[1] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine Learning in Agriculture: A Review," *Sensors*, vol. 18, no. 8, 2018, ISSN: 1424-8220. DOI: 10.3390/s18082674. [Online]. Available: https://www.mdpi.com/1424-8220/18/8/2674.

[2] L. Benos, A. C. Tagarakis, G. Dolias, R. Berruto, D. Kateris, and D. Bochtis, "Machine learning in agriculture: A comprehensive updated review," *Sensors*, vol. 21, no. 11, p. 3758, 2021.

[3] G. Reina, "Robotics and AI for precision agriculture," *Robotics*, vol. 13, no. 4, 2024, ISSN: 2218-6581. DOI: 10.3390/robotics13040064. [Online]. Available: https://www.mdpi.com/2218-6581/13/4/64.

[4] T. B. Shahi, C.-Y. Xu, A. Neupane, and W. Guo, "Recent advances in crop disease detection using UAV and deep learning techniques," *Remote Sensing*, vol. 15, no. 9, 2023, ISSN: 2072-4292. DOI: 10.3390/rs15092450. [Online]. Available: https://www.mdpi.com/2072-4292/15/9/2450.

[5] A. Bouguettaya, H. Zarzour, A. Kechida, and A. M. Taberkit, "Recent advances on UAV and deep learning for early crop diseases identification: A short review," in *2021 International conference on information technology (ICIT)*, IEEE, 2021, pp. 334–339.

[6] X. Zhang, L. Han, Y. Dong, *et al.*, "A deep learning-based approach for automated yellow rust disease detection from high-resolution hyperspectral UAV images," *Remote Sensing*, vol. 11, no. 13, p. 1554, 2019.

[7] A. Bouguettaya, H. Zarzour, A. Kechida, and A. M. Taberkit, "Deep learning techniques to classify agricultural crops through UAV imagery: A review," *Neural computing and applications*, vol. 34, no. 12, pp. 9511–9536, 2022.

[8] E. Karunathilake, A. T. Le, S. Heo, Y. S. Chung, and S. Mansoor, "The path to smart farming: Innovations and opportunities in precision agriculture," *Agriculture*, vol. 13, no. 8, p. 1593, 2023.

[9] F. L. Santos, F. L. Mondragão-Rodrigues, A. Cordeiro, and C. Peres, "Following olive footprints in portugal.," *Scripta Horticulturae*, 2012.

[10] TPN/Lusa, *Portugal in the olive oil 'premier league'*, 2024. [Online]. Available: https://www.theportugalnews.com/news/2024-09-11/portugal-in-the-olive-oil-premier-league/92006 (visited on 03/17/2025).

[11] Lusa, *Portugal: Country could soon be among top 3 olive oil producers*, 2024. [Online]. Available: https://www.aman-alliance.org/Home/ContentDetail/80227 (visited on 03/17/2025).

[12] Olive Oil World Congress, *Portugal produces the highest percentage of olive oil in the world*, 2025. [Online]. Available: https://www.oliveoilworldcongress.com/portugal-is-the-country-that-produces-the-highest-percentage-of-olive-oil-in-the-world (visited on 03/17/2025).

[13] AICEP, *Portuguese olive oil is the best in the world*, 2024. [Online]. Available: https://www.portugalglobal.pt/en/news/2024/july/portuguese-olive-oil-is-the-best-in-the-world/ (visited on 03/17/2025).

[14] Paolo de Andreis, *Record yields for portugal in the 2021/22 crop year*, 2022. [Online]. Available: https://www.oliveoiltimes.com/production/record-yields-for-portugal/105753 (visited on 03/17/2025).

[15] S. Soloway, "The viability of traditional portuguese olive groves in the alentejo region under a sustainable development framework," *School for International Training*, 2022.

[16] European Innovation Partnership for Agricultural Productivity and Sustainability (EIP-AGRI). "Pests and diseases of the olive tree." (2019), [Online]. Available: https://ec.europa.eu/eip/agriculture/sites/default/files/eip-agri_fg33_pests_and_diseases_olive_tree_starting_paper_2019_en.pdf (visited on 03/17/2025).

[17] Farmonaut. "Organic vs. chemical control: Managing olive knot disease caused by pseudomonas savastanoi in olive trees." (2024), [Online]. Available: https://farmonaut.com/precision-farming/organic-vs-chemical-control-managing-olive-knot-disease-caused-by-pseudomonas-savastanoi-in-olive-trees/ (visited on 03/17/2025).

[18] University of California Integrated Pest Management (UC IPM). "Olive knot." (2011), [Online]. Available: https://ipm.ucanr.edu/home-and-landscape/olive-knot/pest-notes/ (visited on 03/17/2025).

[19] R. Buonaurio, C. Moretti, D. P. da Silva, C. Cortese, C. Ramos, and V. Venturi, "The olive knot disease as a model to study the role of interspecies bacterial communities in plant disease," *Frontiers in plant science*, vol. 6, p. 434, 2015.

[20] M. Chliyeh, A. Ouazzani Touhami, K. Selmaoui, R. Benkirane, and A. Douira, "Inventory and world geographical distribution of the olive tree (Olea europaea L.) diseases caused by viruses, bacteria and phytoplasma," *International Journal of Environment, Agriculture and Biotechnology*, vol. 2, pp. 1410–1440, Jan. 2017. DOI: 10.22161/ijeab/2.3.51.

[21] The Portugal News, *Olive sector: "an economic crown jewel"*, 2023. [Online]. Available: https://www.theportugalnews.com/news/2023-12-06/olive-sector-an-economic-crown-jewel/83926.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] M. Voets, "Deep learning: From data extraction to large-scale analysis," M.S. thesis, The Arctic University of Norway, 2018.

[24] T. Developers, "Tensorflow," *Zenodo*, 2022.

[25] S. Imambi, K. B. Prakash, and G. Kanagachidambaresan, "Pytorch," *Programming with TensorFlow: solution for edge computing applications*, pp. 87–104, 2021.

[26] M. Cilimkovic, "Neural networks and back propagation algorithm," *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*, vol. 15, no. 1, p. 18, 2015.

[27] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.04747.

[28] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[29] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[30] N. Ketkar, J. Moolayil, N. Ketkar, and J. Moolayil, "Convolutional neural networks," *Deep learning with Python: learn best practices of deep learning models with PyTorch*, pp. 197–242, 2021.

[31] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, *Activation functions in deep learning: A comprehensive survey and benchmark*, 2022. arXiv: 2109.14545 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2109.14545.

[32] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485*, 2020.

[33] R. Riad, O. Teboul, D. Grangier, and N. Zeghidour, "Learning strides in convolutional neural networks," *arXiv preprint arXiv:2202.01653*, 2022.

[34] Y. Wu, L. Liu, J. Bae, *et al.*, *Demystifying learning rate policies for high accuracy training of deep neural networks*, 2019. arXiv: 1908.06477 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1908.06477.

[35] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT express*, vol. 6, no. 4, pp. 312–315, 2020.

[36] J. Brownlee, "What is the difference between a batch and an epoch in a neural network," *Machine learning mastery*, vol. 20, no. 1, pp. 1–15, 2018.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.

[39] F. Zhuang, Z. Qi, K. Duan, *et al.*, *A comprehensive survey on transfer learning*, 2020. arXiv: 1911.02685 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1911.02685.

[40] A. Hosna, E. Merry, J. Gyalmo, Z. Alom, Z. Aung, and M. A. Azim, "Transfer learning: A friendly introduction," *Journal of Big Data*, vol. 9, no. 1, p. 102, 2022.

[41] D. Hall, F. Dayoub, J. Skinner, *et al.*, "Probabilistic object detection: Definition and evaluation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 1031–1040.

[42] C. H. Kang and S. Y. Kim, "Real-time object detection and segmentation technology: An analysis of the yolo algorithm," *JMST Advances*, vol. 5, no. 2, pp. 69–76, 2023.

[43] J. Mendes, J. Lima, N. Rodrigues, and A. I. Pereira, "Automated preprocessing of olive leaf images for cultivar classification using yolo11," in *Proceedings of the 3rd International Conference on Optimization, Learning Algorithms and Applications (OL2A 2025)*, ser. Communications in Computer and Information Science, To appear, Instituto Politécnico de Bragança, CeDRI and CIMO, LA SusTEC, Sestri Levante, Italy: Springer, 2025.

[44] Y. Du, N. Pan, Z. Xu, F. Deng, Y. Shen, and H. Kang, "Pavement distress detection and classification based on yolo network," *International Journal of Pavement Engineering*, vol. 22, no. 13, pp. 1659–1672, 2021.

[45] J. Mendes, J. Lima, L. Costa, N. Rodrigues, and A. Pereira, "Deep learning networks for olive cultivar identification: A comprehensive analysis of convolutional neural networks," *Smart Agricultural Technology*, vol. 8, p. 100 470, May 2024. DOI: 10.1016/j.atech.2024.100470.

[46] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[47] E. Mohamed, A. Shaker, A. El-Sallab, and M. Hadhoud, "Insta-yolo: Real-time instance segmentation," *arXiv preprint arXiv:2102.06777*, 2021.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[51] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[52] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[53] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[54] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[55] S. Bengamra, O. Mzoughi, A. Bigand, and E. Zagrouba, "A comprehensive survey on object detection in visual art: Taxonomy and challenge," *Multimedia Tools and Applications*, vol. 83, no. 5, pp. 14 637–14 670, 2024.

[56] R. Khanam and M. Hussain, *Yolov11: An overview of the key architectural enhancements*, 2024. arXiv: `2410.17725 [cs.CV]`. [Online]. Available: `https://arxiv.org/abs/2410.17725`.

[57] G. Yang, J. Lei, Z. Zhu, S. Cheng, Z. Feng, and R. Liang, "Afpn: Asymptotic feature pyramid network for object detection," in *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2023, pp. 2184–2189.

[58] L. Tao, M. Dong, and C. Xu, "Dual focal loss for calibration," in *International Conference on Machine Learning*, PMLR, 2023, pp. 33 833–33 849.

[59] G. Jocher and J. Qiu, *Ultralytics YOLO11*, https://github.com/ultralytics/ultralytics, version 11.0.0, 2024. (visited on 03/18/2025).

[60] A. Lei, Y. Mei, D. Ma, Z. Liu, W. Tao, and F. Huang, "Two-stage transient stability assessment using ensemble learning and cost sensitivity," *Frontiers in Energy Research*, vol. 12, p. 1 491 846, 2024.

[61] G. Sun, Y. Hua, G. Hu, and N. Robertson, "Efficient one-stage video object detection by exploiting temporal consistency," in *European Conference on Computer Vision*, Springer, 2022, pp. 1–16.

[62] H. A. Setyawan, A. Bustamam, and R. A. Buyung, "Analysis performance of one-stage and two stage object detection method for car damage detection.," *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 7, 2024.

[63] B. Karbouj, G. A. Topalian-Rivas, and J. Krüger, "Comparative performance evaluation of one-stage and two-stage object detectors for screw head detection and classification in disassembly processes," *Procedia CIRP*, vol. 122, pp. 527–532, 2024.

[64] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[65] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[66] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[67] R. Khanam and M. Hussain, "What is yolov5: A deep look into the internal features of the popular object detector," *arXiv preprint arXiv:2407.20892*, 2024.

[68] C. Li, L. Li, H. Jiang, *et al.*, *Yolov6: A single-stage object detection framework for industrial applications*, 2022. arXiv: 2209.02976 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2209.02976.

[69] 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*, 2022. arXiv: 2207.02696 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2207.02696.

[70] R. Varghese and M. Sambath, "Yolov8: A novel object detection algorithm with enhanced performance and robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, IEEE, 2024, pp. 1–6.

[71] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, *Yolov9: Learning what you want to learn using programmable gradient information*, 2024. arXiv: 2402.13616 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2402.13616.

[72] A. Wang, H. Chen, L. Liu, *et al.*, *Yolov10: Real-time end-to-end object detection*, 2024. arXiv: 2405.14458 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2405.14458.

[73] Y. Tian, Q. Ye, and D. Doermann, "Yolov12: Attention-centric real-time object detectors," *arXiv preprint arXiv:2502.12524*, 2025.

[74] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[75] J. Pustejovsky and A. Stubbs, *Natural Language Annotation for Machine Learning: A guide to corpus-building for applications*. " O'Reilly Media, Inc.", 2012.

[76] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2019, pp. 658–666.

[77] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[78] J. M. Quesada, R. Penyalver, J. Pérez-Panadés, C. I. Salcedo, E. A. Carbonell, and M. M. López, "Dissemination of pseudomonas savastanoi pv. savastanoi populations and subsequent appearance of olive knot disease," *Plant Pathology*, vol. 59, no. 2, pp. 262–269, 2010.

[79] S. Krid, A. Rhouma, J. M. Quesada, R. Penyalver, and A. Gargouri, "Delineation of Pseudomonas savastanoi pv. savastanoi strains isolated in Tunisia by random-amplified polymorphic DNA analysis," *Journal of applied microbiology*, vol. 106, no. 3, pp. 886–894, 2009.

[80] C. A. Quesada, O. L. Phillips, M. Schwarz, *et al.*, "Basin-wide variations in Amazon forest structure and function are mediated by both soils and climate," *Biogeosciences*, vol. 9, no. 6, pp. 2203–2246, 2012.

[81] S. S. Harakannanavar, J. M. Rudagi, V. I. Puranikmath, A. Siddiqua, and R. Pramodhini, "Plant leaf disease detection using computer vision and machine learning algorithms," *Global Transitions Proceedings*, vol. 3, no. 1, pp. 305–310, 2022.

[82] S. Agnihotri, J. Gupta, N. Garg, and P. Khatri, "Comparative analysis of tomato leaf disease detection using machine learning," in *2023 6th International conference on information systems and computer networks (ISCON)*, IEEE, 2023, pp. 1–5.

[83] M. I. Tarik, S. Akter, A. Al Mamun, and A. Sattar, "Potato disease detection using machine learning," in *2021 Third international conference on intelligent communication technologies and virtual mobile networks (ICICV)*, IEEE, 2021, pp. 800–803.

[84] L. Kouadio, M. El Jarroudi, Z. Belabess, *et al.*, "A review on uav-based applications for plant disease detection and monitoring," *Remote Sensing*, vol. 15, no. 17, p. 4273, 2023.

[85] P. Velusamy, S. Rajendran, R. K. Mahendran, S. Naseer, M. Shafiq, and J.-G. Choi, "Unmanned aerial vehicles (uav) in precision agriculture: Applications and challenges," *Energies*, vol. 15, no. 1, p. 217, 2021.

[86] K. Zou, X. Chen, F. Zhang, H. Zhou, and C. Zhang, "A field weed density evaluation method based on uav imaging and modified u-net," *Remote Sensing*, vol. 13, no. 2, p. 310, 2021.

[87] O. Melnychenko, L. Scislo, O. Savenko, A. Sachenko, and P. Radiuk, "Intelligent integrated system for fruit detection using multi-uav imaging and deep learning," *Sensors*, vol. 24, no. 6, p. 1913, 2024.

[88] F. T. Teshome, H. K. Bayabil, G. Hoogenboom, B. Schaffer, A. Singh, and Y. Ampatzidis, "Unmanned aerial vehicle (uav) imaging and machine learning applications for plant phenotyping," *Computers and Electronics in Agriculture*, vol. 212, p. 108 064, 2023.

[89] A. Beniaich, M. L. Silva, D. V. Guimarães, *et al.*, "Uav-based vegetation monitoring for assessing the impact of soil loss in olive orchards in brazil," *Geoderma Regional*, vol. 30, e00543, 2022.

[90] E. M. Raouhi, M. Lachgar, H. Hrimech, and A. Kartit, "Optimizing olive disease classification through transfer learning with unmanned aerial vehicle imagery.," *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 14, no. 1, 2024.

[91] C. Gu, T. Cheng, N. Cai, *et al.*, "Assessing narrow brown leaf spot severity and fungicide efficacy in rice using low altitude uav imaging," *Ecological Informatics*, vol. 77, p. 102 208, 2023.

[92] W. Albattah, A. Javed, M. Nawaz, M. Masood, and S. Albahli, "Artificial intelligence-based drone system for multiclass plant disease detection using an improved efficient convolutional neural network," *Frontiers in Plant Science*, vol. 13, p. 808 380, 2022.

[93] M. P. Mathew and T. Y. Mahesh, "Leaf-based disease detection in bell pepper plant using yolo v5," *Signal, Image and Video Processing*, pp. 1–7, 2022.

[94] Y. Wang, Y. Wang, and J. Zhao, "Mga-yolo: A lightweight one-stage network for apple leaf disease detection," *Frontiers in plant science*, vol. 13, p. 927 424, 2022.

[95] K. U. Shetty, R. J. Kutty, K. Donthi, A. Patil, and N. Subramanyam, "Plant disease detection for guava and mango using yolo and faster r-cnn," in *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, IEEE, vol. 2, 2024, pp. 1–6.

[96] X. Jia, R. Lv, Y. Zhao, and Y. Zhai, "Identification of lycium barbarum aphid disease based on improved yolov11," in *2024 4th International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, IEEE, 2024, pp. 576–579.

[97] C. Chen, Z. Zheng, T. Xu, *et al.*, "Yolo-based uav technology: A review of the research and its applications," *Drones*, vol. 7, no. 3, p. 190, 2023.

[98] S. N. A. M. Robi, N. A. Mashudi, N. Ahmad, M. A. M. Izhar, H. M. Kaidi, and N. M. Noor, "Comparative analysis of yolo models for melon leaf disease classification in uav-assisted smart agriculture," in *2024 5th International Conference on Smart Sensors and Application (ICSSA)*, IEEE, 2024, pp. 1–5.

[99] D.-C. Rodríguez-Lira, D.-M. Córdova-Esparza, J. M. Álvarez-Alvarado, J.-A. Romero-González, J. Terven, and J. Rodríguez-Reséndiz, "Comparative analysis of yolo models for bean leaf disease detection in natural environments," *AgriEngineering*, vol. 6, no. 4, pp. 4585–4603, 2024.

[100] B. Dwyer, J. Nelson, and T. Hansen, *Roboflow (version 1.0)*, 2024. [Online]. Available: `https://roboflow.com` (visited on 03/25/2025).

[101] DJI, *Dji mini 3 specifications*, 2025. [Online]. Available: `https://www.dji.com/pt/mini-3/specs` (visited on 03/25/2025).

[102] G. Contributors, *Geopy: Geocoding toolbox for python*, Documentation available at `https://geopy.readthedocs.io/`, 2024. [Online]. Available: `https://geopy.readthedocs.io/`.

[103] R. Story, *Folium*, version 0.11.0, Dec. 28, 2020. [Online]. Available: `https://python-visualization.github.io/folium/`.

[104] H. Li, X. Pan, K. Yan, F. Tang, and W.-S. Zheng, *Siod: Single instance annotated per category per image for object detection*, 2022. arXiv: `2203.15353 [cs.CV]`. [Online]. Available: `https://arxiv.org/abs/2203.15353`.

[105] C. F. Sabottke and B. M. Spieler, "Impact of image resolution on deep learning performance in endoscopy image analysis," *Diagnostics*, vol. 11, no. 12, p. 2183, 2021. DOI: `10.3390/diagnostics11122183`.

[106] Y. Tian, Q. Ye, and D. Doermann, *Yolov12: Attention-centric real-time object detectors*, 2025. [Online]. Available: `https://github.com/sunsmarterjie/yolov12`.

[107] X. C. Zhang, Q. Chen, R. Ng, and V. Koltun, *Zoom to learn, learn to zoom*, 2019. arXiv: `1905.05169 [cs.CV]`. [Online]. Available: `https://arxiv.org/abs/1905.05169`.

[108] A. I. De Castro, P. Rallo, M. P. Suárez, *et al.*, "High-throughput system for the early quantification of major architectural traits in olive breeding trials using uav images and obia techniques," *Frontiers in Plant Science*, vol. 10, p. 1472, 2019.

Appendix A

Python Code

```
import numpy as np
import pandas as pd
from ultralytics import YOLO
import folium
from geopy.distance import geodesic
import cv2
import csv

def read_and_format_file(file_path):
    # Initialize an empty list to store the matrix
    data_matrix = []

    # Open and read the file
    with open(file_path, 'r') as file:
        # Use csv.reader to handle comma-separated values
        reader = csv.reader(file)

        # Iterate through each row in the file
        for row in reader:
            # Append the row (as a list) to the data matrix
```

```
21              data_matrix.append(row)
22
23      return data_matrix
24
25  def count_frames(video_path):
26      cap = cv2.VideoCapture(video_path)
27      if not cap.isOpened():
28          print("Could not open the video.")
29          return None
30      total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
31      cap.release()
32      return total_frames
33
34  if __name__ == '__main__':
35      # 1. List of olive tree coordinates (lat, lon)
36      coords = []
37      with open('coords.csv', 'r') as csvfile:
38          reader = csv.DictReader(csvfile)
39          for row in reader:
40              lat = float(row['latitude'])
41              lon = float(row['longitude'])
42              coords.extend([lat, lon])
43
44      oliveiras_coords = [(coords[i], coords[i + 1]) for i in range
            (0, len(coords), 2)]
45
46      # 2. Function to check proximity
47      proximity_radius_m = 2  # Radius in meters
48
49      def is_close(coord1, coord2, radius=proximity_radius_m):
```

```
50          return geodesic(coord1, coord2).meters <= radius
51
52      # 3. Parameters and data reading
53      video_path = "datasets/2025-04-09_Dataset_Mirandela/DJI_0092.
            MP4"
54      flight_log_path = "datasets/2025-04-09_Dataset_Mirandela/2-
            Apr-3rd-2025-11-47AM-Flight-Airdata.csv"
55      model_path = "models/Versoes_Modelos/tese_yolo12m_resize2/
            weights/best.pt"
56
57      def count_frames(video_path):
58          cap = cv2.VideoCapture(video_path)
59          if not cap.isOpened():
60              print("Could not open the video.")
61              return None
62          total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
63          cap.release()
64          return total_frames
65
66      frames_video = count_frames(video_path)
67      flight_log = pd.read_csv(flight_log_path)
68      instances_csv = len(flight_log)
69      flight_log_interval_ms = 100
70      fps = 26.06
71      flight_log['timestamp_ms'] = np.arange(instances_csv) *
            flight_log_interval_ms
72
73      timestamps_frames = np.arange(frames_video) * (1000 / fps)
74      lat_interp = np.interp(timestamps_frames, flight_log['
            timestamp_ms'], flight_log['latitude'])
```

```
75      lon_interp = np.interp(timestamps_frames, flight_log['
            timestamp_ms'], flight_log['longitude'])
76
77      # 4. Run YOLO model on the video (stream=True)
78      model = YOLO(model_path)
79      results = model.predict(
80          source=video_path, show=False, save=False, conf=0.5,
                line_width=2,
81          save_crop=False, save_txt=False, show_labels=True,
                show_boxes=True, show_conf=True, stream=True
82      )
83
84      # 5. Collect detection coordinates
85      detections_coords = []
86      for i, result in enumerate(results):
87          if result.boxes is None or len(result.boxes) == 0:
88              continue
89          for box in result.boxes:
90              lat = lat_interp[i]
91              lon = lon_interp[i]
92              detections_coords.append((lat, lon))
93      unique_detections_coords = list(set(detections_coords))
94
95      # Print each unique detection coordinate
96      for coord in unique_detections_coords:
97          print(f"{coord[0]:.6f},{coord[1]:.6f}")
98
99      # 6. Map olive trees with nearby detections
100     oliveira_detection_map = {}
101     for oliveira in oliveiras_coords:
```

```
102         oliveira_detection_map[oliveira] = any(is_close(oliveira,
                det) for det in unique_detections_coords)
103
104     if oliveira_detection_map:
105         m = folium.Map(location=oliveiras_coords[0], zoom_start
                =18, tiles=None)
106         folium.TileLayer(
107             tiles='https://server.arcgisonline.com/ArcGIS/rest/
                    services/World_Imagery/MapServer/tile/{z}/{y}/{x}',
108             attr='Esri',
109             name='Esri Satellite',
110             overlay=False,
111             control=True
112         ).add_to(m)
113         for oliveira, detected in oliveira_detection_map.items():
114             status = 'Infected' if detected else 'Healthy'
115             tooltip_text = f"Lat: {oliveira[0]:.6f}, Lon: {
                    oliveira[1]:.6f} - Status: {status}"
116             folium.CircleMarker(
117                 location=oliveira,
118                 radius=2,
119                 color='red' if detected else 'blue',
120                 fill=True,
121                 fill_opacity=0.8,
122                 tooltip=tooltip_text
123             ).add_to(m)
124         m.save("olive_detections_map.html")
125         print("Map saved as olive_detections_map.html")
126     else:
127         print("No olive trees to plot on the map.")
```

Listing A.1: Main Python Script