الجمهورية الجزائرية الديمقراطيةالشعبية
Democratic and Popular Algerian Republic

وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research

جامعة سعد دحلب البليدة
Saad Dahlab University , Blida1

كلية التكنولوجيا
Faculty of Technology

قسم الإلكترونيك
Department of Electronics

Master's thesis

Field Electronics

Specialization in Embedded Systems Electronics

Presented by: BENMALLEM Feriel

Intelligent Detection of Apricot Ripeness using YOLOv8: Embedded Application on Raspberry Pi 5 for Precision Agriculture

Proposed by: Mrs. NACEUR Djamila

Academic Year 2024-2025

Acknowledgments

*First of all, I thank **ALLAH**, our creator, for giving us the strength, will, and courage to complete this work.*

*I thank **Mrs. NACEUR Djamila**, my supervisor, who guided me in this work.*

*I thank my parents, my sister **Fella** and my brother **Imad Eddine** for their support.*

*I thank **Mr. ACHROUF Ibrahim** for his support and help.*

I thank the member juries for their expertise and guidance.

I would like to express my heartfelt thanks to all the people who helped me, directly or indirectly, in the completion of this work.

Finally, I would like to express my deepest gratitude to my family who have always supported me and to all those who participated in the completion of this thesis, as well as to all the teachers who contributed to our education.

Dedications

It is with profound gratitude and sincere words,

That I dedicate this modest final year project

My dear parents ***Abderrazak*** *and* ***Latifa***

To my dear sisters ***Fella*** *and* ***Chaima Warda***

To my dear brothers ***Adlen, Imad****, and* ***Oussama***

To my dear grandparents ***Madjid BENMALLEM*** *and* ***Malek BENAZALA***

To all my teachers who have taught me, especially ***Mrs. NACEUR Djamila****,* ***Mr A. NAMANE****, and* ***Mrs I. KAOULA****.*

to ***Mr. ACHROF Ibrahim***

To my colleagues and friends ***CHANANE Yasmine, CHANANE Narimène, BENTOUTATI Loubna, DENDANI Khouloud, LARBI AISSA Abderhmane, BENHOCINE Abderaouf, BENMESBAH sarah.***

الملخص

يقترح هذا البحث نظامًا مدمجًا للكشف الذكي عن نضج المشمش، يعتمد على YOLOv8 و Raspberry Pi 5، بهدف تحديث عملية تقييم جودة الفواكه في الجزائر. من خلال إنشاء ومعالجة مجموعة بيانات مخصصة، وتدريب وتقييم عدة نماذج من YOLOv8، ثم تحسين الأداء للنشر في الوقت الحقيقي، يتيح هذا النظام تصنيفًا دقيقًا وسريعًا للمشمش الناضج وغير الناضج. تُعد هذه الحلول استجابة فعالة لتحديات نقص اليد العاملة والخسائر بعد الحصاد، حيث توفر للمزارعين أداة عملية لتحقيق حصاد مثالي وإدارة مستدامة للبساتين. يوضح هذا العمل الفائدة الحقيقية لتقنيات التعلم العميق والذكاء الاصطناعي المدمج في دعم الزراعة الذكية.

الكلمات المفتاحية: الكشف عن نضج المشمش، Raspberry Pi ،YOLOv8 5، الذكاء الاصطناعي المدمج، الزراعة الذكية.

Résumé

Ce mémoire propose un système embarqué pour la détection intelligente de la maturité des abricots, basé sur YOLOv8 et Raspberry Pi 5, afin de moderniser l'évaluation de la qualité des fruits en Algérie. Grâce à la création et au prétraitement d'un jeu de données personnalisé, à l'entraînement et à l'évaluation de plusieurs modèles YOLOv8, puis à l'optimisation pour un déploiement en temps réel, le système permet une classification précise et rapide des abricots mûrs et non mûrs. Cette solution répond aux défis de la pénurie de main-d'œuvre et des pertes post-récolte, offrant aux agriculteurs un outil pratique pour une récolte optimale et une gestion durable des vergers. Ce travail illustre l'apport concret du deep learning et de l'IA embarquée pour l'agriculture de précision.

Mots-clés : Détection de Maturité de l'Abricot, YOLOv8, Raspberry Pi 5, IA embarquée, agriculture de précision

Abstract

This thesis presents an embedded system for intelligent apricot ripeness detection using YOLOv8 on Raspberry Pi 5, aiming to modernize fruit quality assessment in Algerian agriculture. By creating and preprocessing a custom dataset, training and evaluating several YOLOv8 models, and optimizing for real-time edge deployment, the system enables accurate, fast, and objective classification of ripe and unripe apricots. The solution addresses challenges of labor scarcity and post-harvest losses, offering farmers a practical tool for better harvest timing and sustainable orchard management. This work demonstrates the potential of deep learning and embedded AI to bring tangible benefits to precision agriculture.

Keywords: Detecting apricot ripeness, YOLOv8, Raspberry Pi 5, Embedded AI, Precision Agriculture

LIST OF ACRONYMS AND ABBREVIATIONS

AI: Artificial Intelligence

AP: Average Precision

AUC: Area Under the Curve

BCE Loss: Binary Cross-Entropy Loss

BG: Background

CIoU Loss: Complete Intersection over Union Loss

CNNs: Convolutional Neural Networks

COCO: Common Objects in Context

CPU: Central Processing Unit

CSI: Camera Serial Interface

DFL Loss: Distribution Focal Loss

DL: Deep Learning

DNN: Deep Neural Network

F1-Curve: F1-score vs. Confidence Threshold

F1-Score: Harmonic mean between precision and recall

FN: False Negatives

FLOPs: Floating point operations per second

FP: False Positives

FPS: Frames per Second

GPIO: General Purpose Input/Output

GPU: Graphics Processing Unit

HSV: Hue, Saturation, Value

IDE: Integrated Development Environment

INT8: 8-bit integer quantization

IoU: Intersection over Union

k-NN: k-nearest neighbors

LPDDR4X: Low Power Double Data Rate

4X **MB**: Mega bytes **mAP**: mean Average

Precision **microSD**: Micro Secure Digital

MIPI CSI: Mobile Industry Processor Interface Camera Serial Interface

ML: Machine Learning

NMS: Non-Maximum Suppression

ONNX: Open Neural Network Exchange

OS: Operating System

P: Precision

P-Curve: Precision vs. Confidence Threshold

PIL: Python Imaging Library

PR-Curve: Precision–Recall curve

R: Recall

RAM: Random Access Memory

ReLU: Rectified Linear Unit

R-CNN: Regions with Convolutional Neural Networks

RoIs: Regions of Interest

RPi: Raspberry Pi

RPN: Region Proposal Network

SGD: Stochastic Gradient Descent

SPPF: Spatial Pyramid Pooling Fast

SSD: Single Shot MultiBox Detector

SVMs: Support Vector Machines

TP: True Positives

TPU: Tensor Processing Unit

USD: US Dollar

USB: Universal Serial Bus

VRAM: Video Random Access Memory

YOLO: You Only Look Once

Table of Contents

LIST OF FIGURES

LIST OF TABLES

GENERAL INTRODUCTION

Agriculture plays a central role in the economy and food security of many countries particularly in Algeria, where apricot cultivation holds special importance. However, managing the harvest of this fruit remains a major challenge due to its highly perishable nature, the variability of its ripening stages, and the difficulty of objectively assessing the optimal harvest time. Traditionally, this evaluation relies on manual visual inspection a subjective, time-consuming method that heavily depends on human expertise. These limitations often lead to significant post-harvest losses, inconsistent product quality, and challenges in meeting the demands of both local and international markets.

In light of these challenges, the introduction of intelligent and automated solutions emerges as a promising path to modernize agricultural practices and improve farm profitability. In recent years, the remarkable progress of artificial intelligence particularly deep learning has enabled the development of computer vision systems capable of analyzing images in real time with high precision. Among the most effective architectures, the YOLO (You Only Look Once) family has stood out for its speed and efficiency, with YOLOv8 currently representing the state of the art for object detection in resource-constrained environments.

Within this context, a central question arises: how can we design an embedded, intelligent, and reliable system capable of automatically detecting apricot maturity under real-world conditions, to support farmers in their decision-making and reduce post-harvest losses? This thesis is part of that innovation effort and proposes the development of an intelligent apricot maturity detection system based on the YOLOv8 model, deployed on a Raspberry Pi 5 embedded platform. The proposed approach includes the creation and annotation of a representative dataset, the training and evaluation of the detection model, and its integration into an embedded device for real-time field use. This solution aims to provide farmers with a practical, cost-effective, and high-performance tool capable of optimizing harvest management and improving the quality of fruits intended for market.

To present the entirety of the work in a clear and structured manner, this thesis is organized into three main chapters:

- **Chapter 1**: Agronomic context, state of the art, and scientific foundations, presenting the challenges of apricot cultivation, the limitations of traditional methods, and the theoretical foundations of artificial intelligence and computer vision systems.

- **Chapter 2**: Methodology for developing the detection system, detailing data collection and processing, YOLOv8 model configuration and training, as well as validation steps.
- **Chapter 3**: Results and discussion, presenting the system's performance, its deployment on Raspberry Pi 5, and a critical analysis of the results obtained under real-world conditions.

CHAPTER 1: Agronomic Context, State of the Art, and Scientific Foundations

1.1. INTRODUCTION

Modern agriculture is undergoing structural and environmental transformations that demand urgent adaptation strategies. Critical challenges including declining rural labor availability, increasing quality standards for exports, and the unpredictable consequences of climate change are putting pressure on traditional cultivation systems [3]. These pressures are particularly acute in fruit production, where harvesting precision directly affects commercial success.

In this context, artificial intelligence (AI), machine learning, and computer vision are emerging as essential tools for modernizing agriculture. These technologies offer new ways to monitor crops, optimize harvest timing, and reduce waste through more consistent and scalable decision-making systems [1].

This chapter provides the agronomic and scientific foundations necessary for understanding the rationale behind applying AI to apricot production. It begins with an overview of apricot cultivation in Algeria and its economic significance. It then outlines the structural and environmental limitations of traditional agricultural practices. Finally, it introduces the principles of intelligent agriculture, followed by a review of the theoretical foundations underpinning this thesis namely, AI, deep learning, computer vision, and object detection.

1.2. Apricot Cultivation in Algeria

1.2.1. Economic and Geographic Significance

Apricot cultivation (*Prunus armeniaca*) plays a strategic role in Algeria's fruit sector, especially in the semi-arid and high-plateau zones. More than 20,000 hectares are dedicated to apricot orchards, with national production fluctuating between 250,000 and 300,000 tonnes per year [3]. Key producing regions include Mitidja (Blida, Médéa, Bouira), Batna, and Aïn Oussera, where thermal amplitudes and high altitudes support both flowering and sugar concentration [13].

Apricots are commercially important for both domestic and export markets due to their versatility in processing (fresh consumption, jams, dried fruit, juices) and relatively short cultivation cycle.

However, the sector's full potential is limited by systemic agronomic and logistical barriers [11].

1.2.2. Agronomic Challenges

Apricot trees are among the earliest bloomers in the temperate zone, with flowering occurring between late February and early April depending on cultivar and region. This phenological trait exposes them to late spring frosts, which frequently damage floral buds and reduce yield consistency [11,13].

In addition, apricots exhibit asynchronous ripening. Fruits on the same branch may reach full maturity at different times within a narrow window, complicating optimal harvest timing and increasing the risk of over- or under-ripe yields [13]. Rapid perishability further compounds the issue, as delays between maturity detection and harvest can result in significant post-harvest losses due to bruising, drop, or microbial spoilage [6,11].

1.2.3. Labor and Market Constraints

The harvest process is labor-intensive and relies heavily on skilled seasonal workers who can visually assess ripeness. However, Algeria faces growing labor shortages due to rural depopulation, urban migration, and an aging agricultural workforce [9]. These demographic pressures hinder the ability to conduct timely and repeated harvesting rounds.

Moreover, the global fruit market increasingly demands standardization in size, maturity, and visual quality criteria that are difficult to meet with manual methods alone. Non-uniform harvests often lead to market rejections or price downgrading [7].

1.2.4. Why Intelligent Solutions Are Needed

Given these structural and agronomic constraints, traditional harvesting strategies struggle to maintain efficiency, uniformity, and profitability. Precision agriculture supported by AI and computer vision offers a promising alternative. These technologies can automate ripeness assessment based on visual features such as color gradients and texture, providing fast and objective evaluations of fruit maturity [1,8].

Furthermore, low-cost embedded systems (e.g., Raspberry Pi) enable field-deployable solutions even in resource-limited settings [8]. However, to implement such systems effectively, a deep understanding of apricot phenology is essential for defining detection targets and training AI models on biologically relevant classes (e.g., immature vs. ripe).

Figure 1.1 illustrates these key developmental phases, from fruit set to physiological ripening, which are distinguished by observable changes in color and size.

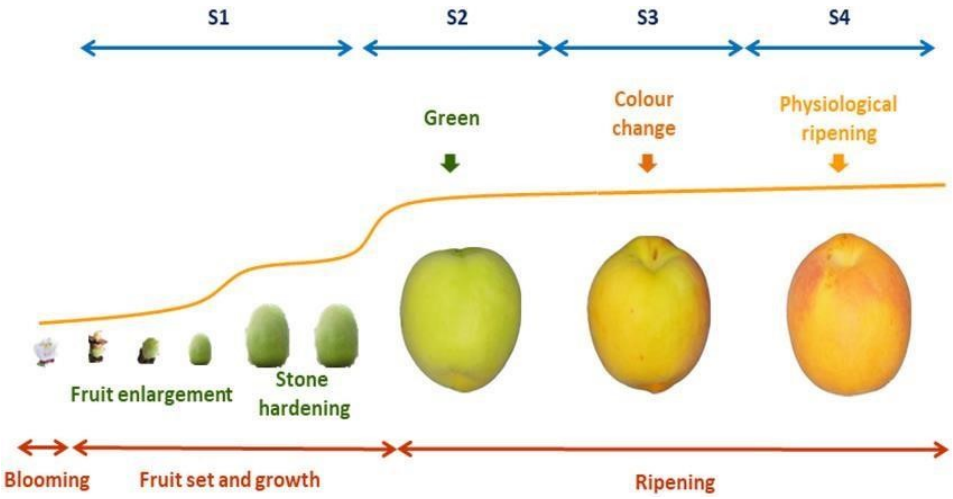


Figure 1.1: Phenological Stages of Apricot Development

These challenges and biological rhythms form the rationale for transitioning to intelligent detection systems, a shift explored further in the next section on the limitations of manual agricultural practices.

1.3. Limitations of Traditional Agriculture

Traditional apricot cultivation in Algeria, while rooted in local expertise and seasonal knowledge, faces critical limitations that constrain its ability to meet modern production, quality, and export demands. This section identifies the principal operational, economic, and technical limitations of manual practices, particularly those impacting productivity, postharvest quality, and market competitiveness.

1.3.1. Perishability and Ripening Challenges

Apricots are non-climacteric fruits with high perishability and a narrow post-harvest window. Their uneven ripening pattern requires multiple harvest passes, increasing labor intensity and the risk of misclassification [11]. Inappropriate harvest timing whether premature or delayed directly compromises flavor, texture, and shelf life. These biological constraints significantly limit storage, transport, and export potential for small-scale producers [11].

1.3.2. Labor Scarcity and Time Sensitivity

Manual harvesting and post-harvest sorting are labor-intensive tasks often concentrated within a short ripening period. Algeria faces increasing rural depopulation, aging farm labor, and

declining interest from younger generations [9]. This chronic shortage leads to unharvested yields, delays, or excessive workloads that affect fruit quality and marketability. The situation is worsened by the need for skilled labor to assess fruit maturity accurately and handle produce with care [6].

1.3.3. Subjectivity in Maturity Assessment

Evaluating apricot maturity based on visual and tactile indicators such as color, firmness, and size remains subjective and prone to inconsistency. Operators' fatigue, varying experience levels, and changing light conditions during fieldwork introduce significant errors [1,6]. Such variability compromises sorting, grading, and even harvest decisions, often resulting in heterogeneous batches that fail export requirements [7].

1.3.4. Post-Harvest Losses and Quality Deterioration

Improper manual handling is a major source of post-harvest losses. Bruising, tearing, or compression damage often occur during harvesting, sorting, or transport. These injuries accelerate fungal growth, reduce shelf life, and lead to batch rejections [11]. In Algeria, postharvest losses in stone fruit chains including apricots are estimated to exceed 20% annually [11], disproportionately affecting small and medium-sized farms with limited access to cold chain infrastructure.

1.3.5. Market Constraints and Export Risk

Export markets impose strict standards regarding appearance, ripeness uniformity, and size. Manual techniques often fail to consistently meet these specifications, increasing the risk of rejection or downgraded classification [7]. The absence of traceable, objective maturity assessment tools also limits producers' ability to meet international certification standards, impeding their access to high-value markets [12].

To synthesize these challenges, Table 1.1 offers a direct comparison of the limitations inherent in manual practices against the potential solutions offered by intelligent, AI-based systems.

Table 1.1: Comparison of Manual vs. Intelligent Agricultural Approaches

Limitation	**Manual Practice**	**Intelligent (AI-based) Alternative**
Maturity Detection	Subjective visual judgment	Objective image-based classification [1,8]
Harvest Scheduling	Labor-driven, timeconstrained	Automated, data-informed timing [8]
Post-Harvest Handling	Inconsistent, damageprone	Precision handling via guided robotics [8]
Labor Dependency	High seasonal labor requirement	Reduced manual input with automation [9]
Market Standard Compliance	Variable and operatordependent	Standardized classification and traceability [7]

These structural and operational limitations reveal the need for scalable, consistent, and intelligent systems capable of automating core tasks in apricot production. The following section explores how artificial intelligence (AI), and more specifically computer vision and deep learning, offer concrete technological responses to these challenges.

1.4. Scientific Foundations of Intelligent Vision Systems

1.4.1. Artificial Intelligence and Agriculture

The agricultural sector is undergoing a profound transformation driven by the integration of advanced technologies, among which artificial intelligence (AI) plays a pivotal role. Faced with growing challenges such as labor shortages, climate variability, soil degradation, and increased demand for food quality and traceability, traditional farming systems are increasingly turning to intelligent automation for sustainable solutions [3].

Artificial intelligence, broadly defined as the capacity of machines to perform tasks that typically require human intelligence, has found significant application in precision agriculture. Through AI-driven systems, farms can monitor crops, optimize resource usage, predict yields, detect diseases, and improve harvesting strategies [1]. These innovations are not only enhancing productivity but also addressing environmental and economic concerns in the agricultural supply chain.

In the context of fruit cultivation, AI has shown particular promise in addressing the complexities of crop management. Tasks such as fruit counting, maturity assessment, disease detection, and quality classification require accurate interpretation of visual data. AI models, particularly those based on machine learning and deep learning, have demonstrated high accuracy in these tasks, even under variable lighting and background conditions typical of realworld orchards [8].

Moreover, AI enables real-time, data-driven decision-making in the field, especially when deployed on embedded platforms such as Raspberry Pi or Jetson Nano[32,33]. These platforms allow lightweight AI models to function independently, reducing reliance on cloud connectivity and enabling timely actions like fruit picking or irrigation control. In resource-constrained environments, this embedded intelligence becomes essential for ensuring both efficiency and scalability.

Figure 1.2 illustrates how AI directly responds to several of the critical challenges faced in agriculture, including environmental uncertainty, manual labor dependence, and quality inconsistency. The synergy between AI capabilities and agricultural needs underscores its growing relevance in field applications

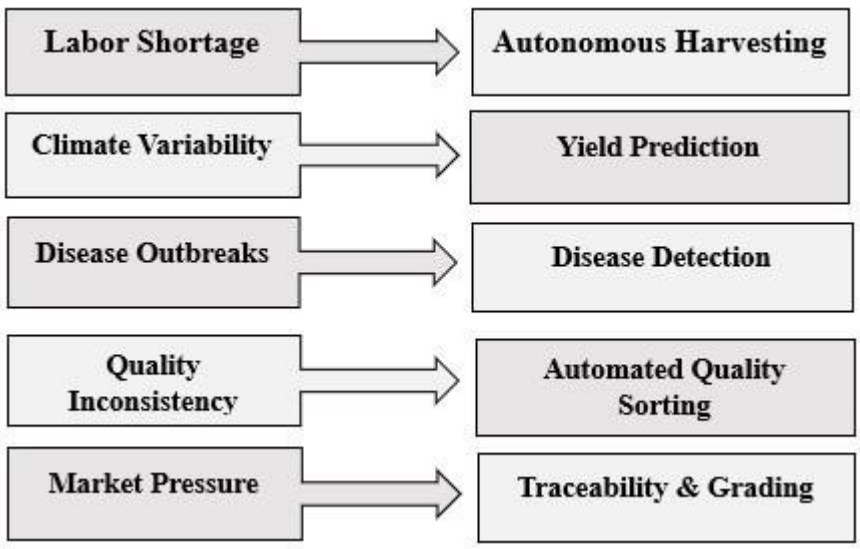


Figure 1.2: Mapping of agricultural challenges to AI-powered solutions

In summary, artificial intelligence serves as a transformative technology for modern agriculture. Its ability to process vast amounts of sensor or image data, adapt to field conditions, and inform decision-making makes it a foundational pillar for the development of intelligent systems such as the one explored in this thesis.

1.4.2. Machine Learning Foundations

Building upon the broader field of artificial intelligence, machine learning (ML) represents a specialized domain that enables systems to learn from data and improve performance without being explicitly programmed [14]. ML is particularly effective in agricultural applications that require adaptive pattern recognition such as image-based fruit detection, classification, and yield estimation [1].

Machine learning can be defined as the development of algorithms that automatically identify patterns and relationships in datasets, and then use these patterns to make predictions or decisions [14]. It is categorized into four primary types:

- **Supervised Learning:** The model is trained on labeled data, making it suitable for tasks like fruit classification, where each image is annotated as "ripe" or "unripe" [15].
- **Unsupervised Learning:** The algorithm finds hidden structures in unlabeled data, useful for clustering different crop types or identifying outliers [15].
- **Semi-supervised Learning:** Combines a small amount of labeled data with a large amount of unlabeled data, reducing annotation effort in large-scale agricultural datasets [16].
- **Reinforcement Learning:** The system learns through interactions with the environment, receiving feedback via rewards or penalties; though promising, this method is less common in static vision tasks like fruit detection [17].

ML algorithms, such as support vector machines (SVMs), decision trees, and k-nearest neighbors (k-NN), have been widely used in agricultural studies [18]. However, their effectiveness is often constrained when dealing with complex and high-dimensional data such as raw images. For tasks like apricot maturity detection which require nuanced visual interpretation traditional ML may struggle to generalize under real-world variability in lighting, occlusion, and fruit morphology [1,18].

Nevertheless, ML serves as a foundation for more advanced techniques, providing essential groundwork for deep learning models. Figure 1.3 illustrates the relationship between AI, ML, and deep learning, emphasizing the nested structure and their respective scopes.

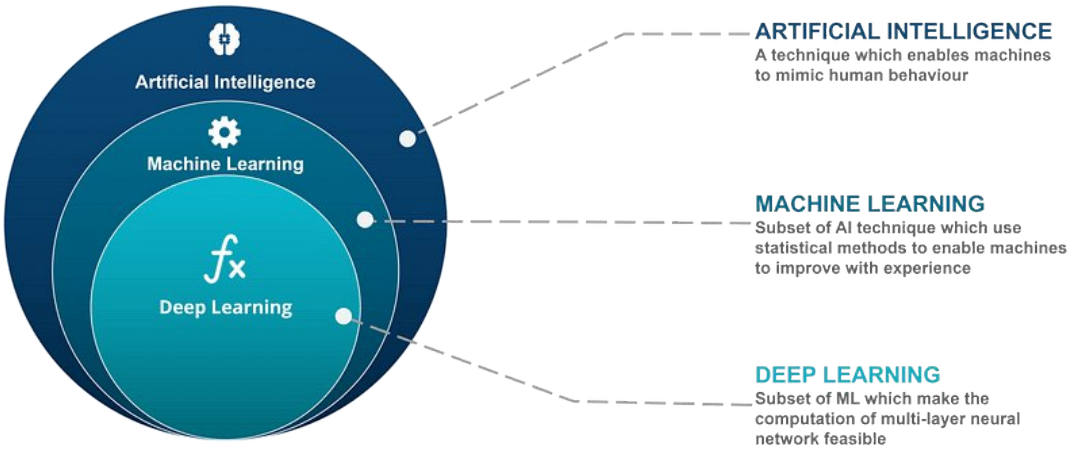


Figure 1.3: Relationship between AI, Machine Learning, and Deep Learning [19]

In conclusion, machine learning plays a vital role in the digital transformation of agriculture. While its limitations in complex visual analysis have driven the field toward deep learning, its principles remain foundational for the intelligent systems addressed in this work.

1.4.3. Deep Learning

Deep learning (DL) is a subfield of machine learning that focuses on artificial neural networks with multiple layers known as deep neural networks capable of automatically extracting hierarchical features from raw data. Unlike traditional machine learning algorithms that often rely on handcrafted features, deep learning models learn these features directly from data, enabling higher accuracy in complex pattern recognition tasks [24].

In the context of image analysis, deep learning particularly convolutional neural networks (CNNs) has revolutionized fields such as medical diagnostics, autonomous driving, and precision agriculture [1]. These models have demonstrated exceptional performance in tasks such as object detection, classification, and segmentation, owing to their ability to process highdimensional data like images with minimal manual intervention.

The core strength of deep learning lies in its ability to generalize across varied conditions, such as changes in lighting, background, or occlusion. This capability is critical in agricultural environments, where natural variability can challenge conventional algorithms. For example,

in fruit maturity detection, CNNs can learn discriminative features that distinguish ripe from unripe apricots even when visual cues are subtle and data is noisy [1].

Despite its strengths, deep learning also presents several limitations:

- **Data dependency:** Requires large amounts of annotated data, which can be costly and time-consuming to collect in agricultural settings [6,21].
- **Computational demands:** Training deep neural networks requires significant computational resources, including GPUs or TPUs, which may not always be accessible to field researchers or small-scale producers [22].
- **Interpretability:** Deep learning models are often viewed as "black boxes," making it difficult to interpret the decision-making process a critical barrier for applications requiring explainability, such as plant disease diagnosis or agricultural certification [23].
- **Overfitting risks:** With small datasets, there is a risk of overfitting, where the model performs well on training data but poorly on unseen data [19].

These challenges are gradually being addressed through techniques such as data augmentation, transfer learning, model compression, and the development of explainable AI frameworks [1,22].

Figure 1.4 provides an overview of a basic deep neural network (DNN) architecture, illustrating how deep learning models process input data through multiple interconnected layers to extract hierarchical features and make predict

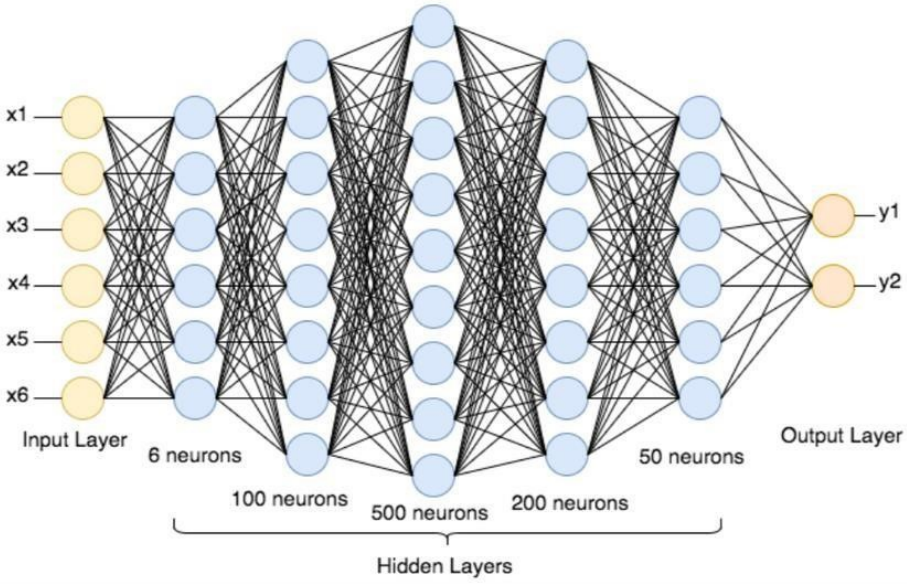


Figure 1.4: Basic architecture of a deep neural network (DNN) [10]

In summary, deep learning offers a powerful solution for fruit maturity detection, enabling robust performance in unstructured and variable environments. Its ability to learn complex visual patterns from raw image data makes it particularly well-suited for the task at hand. The next section focuses on convolutional neural networks, the foundational architecture powering most modern deep learning applications in computer vision.

1.4.4. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) represent the backbone of most deep learning models applied to image processing tasks, including those in agriculture. Emerging directly from the foundations of deep learning, CNNs are particularly suited for structured grid-like data such as images [24]. Their layered architecture enables them to automatically learn spatial hierarchies of features, making them highly effective for visual pattern recognition tasks where manual feature engineering is insufficient.

CNNs are composed of several types of layers, each playing a specific role in processing the input image:

- **Convolutional layers** apply filters (kernels) that scan the input image and detect local features such as edges, textures, or color gradients.
- **Activation functions**, typically the Rectified Linear Unit (ReLU), introduce nonlinearity into the network, allowing it to model complex patterns [16].
- **Pooling layers** (such as max pooling) reduce the spatial dimensions of the data, improving computational efficiency and controlling overfitting by focusing on the most prominent features.
- **Fully connected layers** near the output stage integrate all extracted features to make the final classification or regression decision [16,24].

One of the main strengths of CNNs is their ability to extract relevant features from raw image data without manual intervention. This is particularly advantageous in agriculture, where images captured in the field are subject to non-uniform lighting, occlusions by leaves or branches, and complex backgrounds. CNNs can generalize well under such conditions by learning translationinvariant and spatially local patterns [22,1].

In the context of apricot maturity detection, CNNs form the core of object detection models such as YOLOv8. These models not only classify fruit as ripe or unripe but also localize them within the frame, enabling real-time monitoring or robotic harvesting. By integrating CNNs within YOLO's end-to-end architecture, the system benefits from high-speed inference and robust detection performance, even in noisy or variable environments [12,21].

The basic architecture of a Convolutional Neural Network (CNN), illustrating the sequential flow from input to output layers, is presented in Figure 1.5.

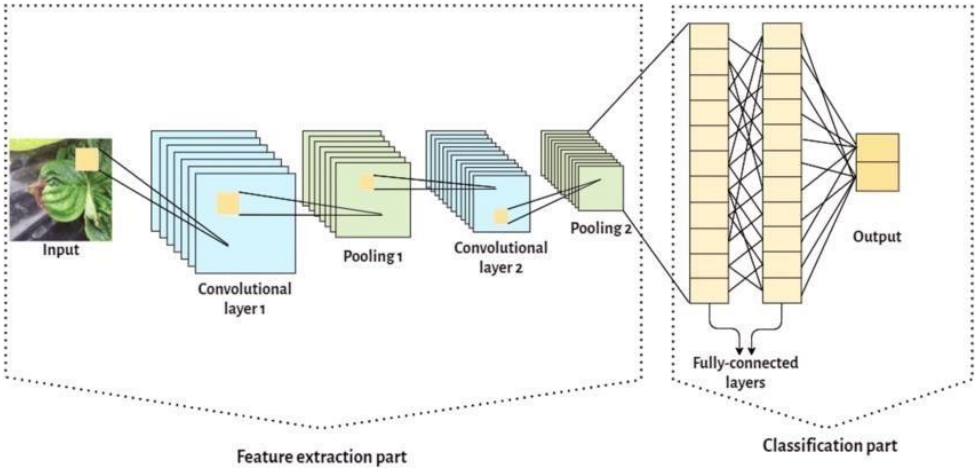


Figure 1.5: Basic architecture of a Convolutional Neural Network (CNN) [10]

The flexibility and scalability of CNNs have led to their adoption in nearly all state-of-the-art computer vision systems in agriculture, from disease diagnosis to yield prediction and maturity detection. Their use in this project lays the groundwork for the object detection system developed in subsequent chapters.

The next section examines how CNNs are integrated into object detection frameworks, with a particular focus on the evolution and comparative performance of the YOLO family.

1.5. Object Detection in Computer Vision

Computer vision, a prominent subfield of artificial intelligence, seeks to replicate and enhance human visual perception by enabling machines to interpret, analyze, and act upon visual data [1,24]. It underpins a wide range of applications from medical diagnostics and autonomous navigation to precision agriculture where automated visual understanding is essential for effective decision-making and operational efficiency.

Among its core tasks, three stand out: image classification, object detection, and image segmentation. Classification involves assigning a single label to an entire image; object detection identifies and localizes multiple objects within a scene using bounding boxes; segmentation performs the most detailed analysis by labeling each pixel according to object class [25]. Clarifying these distinctions is critical in this thesis, which focuses on the detection and spatial localization of ripe and unripe apricots in complex orchard environments a challenge that aligns most directly with object detection techniques.

To clarify these distinctions, Figure 1.6 visually compares image classification, object detection, semantic segmentation, and instance segmentation.

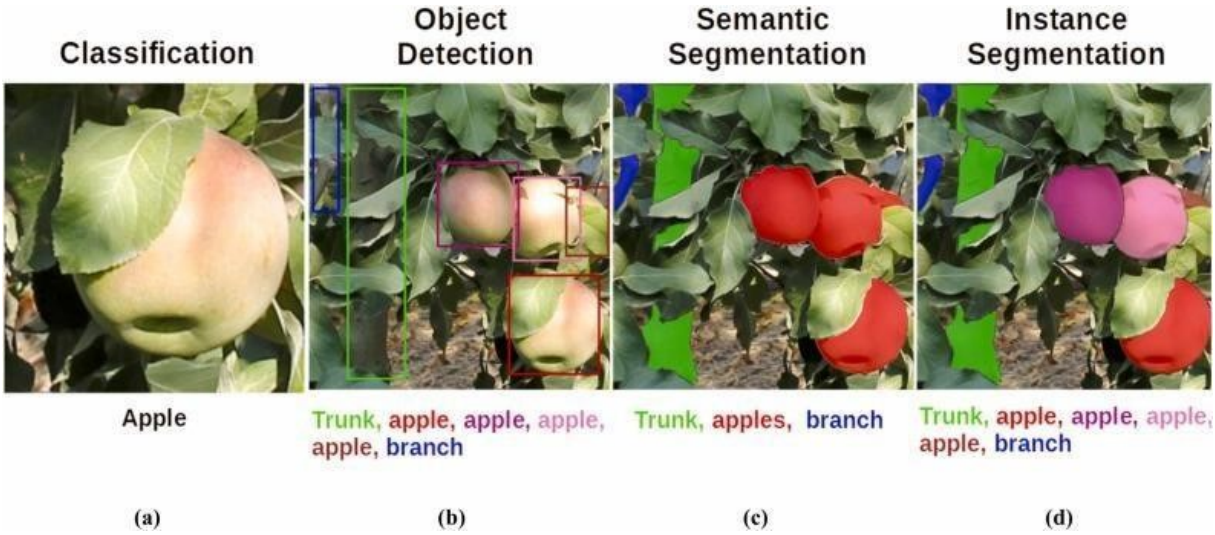


Figure 1.6: Visual Comparison of Image Classification, Object Detection, Semantic Segmentation, and Instance Segmentation [25].

(a) Classification assigns a single label to the image.

(b) Object detection identifies multiple apples with bounding boxes.

(c) Semantic segmentation labels each pixel by class (e.g., apple, leaf, trunk).

(d) Instance segmentation distinguishes between individual fruit instances.

The next subsection delves into the fundamentals of object detection, outlining its principles and its essential role in enabling real-time, automated fruit maturity assessment under agricultural field conditions.

1.5.1. Fundamentals of Object Detection

At its core, object detection involves two key operations: locating objects of interest within an image and assigning a category label to each instance. This dual-task output bounding box regression and class prediction is typically achieved through deep learning models, most notably Convolutional Neural Networks (CNNs), which extract hierarchical spatial features from the input image [1,8]. These features are then passed through specialized detection heads that generate candidate object regions, refine box coordinates, and classify each object according to predefined categories.

In agricultural applications, this pipeline must operate under far more challenging conditions than in controlled environments. Field images often contain heterogeneous lighting, partial occlusions from foliage, and dense clustering of similar-looking fruits, all of which increase the likelihood of missed or incorrect detections [18,21]. Furthermore, the physical variability among fruits such as differences in shape, size, or color gradients across ripening stages demands that detection models be both precise and adaptable.

Despite these complexities, the practical impact of robust object detection in agriculture is substantial. For fruit maturity assessment, it allows automated systems to reliably distinguish between ripe and unripe fruits within the same scene. In yield estimation, it supports accurate fruit counting over time and across plots. In robotic harvesting, it provides the necessary localization data to guide end-effectors toward target fruits while avoiding non-target regions like leaves or stems [22].

Given these requirements, object detection architectures must strike a balance between speed, accuracy, and computational efficiency especially for embedded systems such as Raspberry Pi– based platforms used in precision agriculture. The following subsections examine the evolution of deep learning–based object detection algorithms and evaluate their strengths and limitations in this context.

1.5.2. Overview of Major Object Detection Algorithms

Modern object detection has evolved through distinct algorithmic paradigms each reflecting tradeoffs between computational cost, accuracy, and inference speed. In the context of real-time agricultural applications such as fruit maturity detection, understanding the architecture and practical implications of these models is critical. This section presents a comparative overview of the most influential object detection algorithms to date: R-CNN, Fast R-CNN,

Faster RCNN, SSD, and the YOLO family. Each is evaluated for its detection strategy, performance metrics, and suitability for embedded systems in agricultural environments.

a. Two-Stage Detectors: Region-Based CNN Approaches

- **R-CNN (Regions with Convolutional Neural Networks)** introduced the idea of applying deep learning to object detection by first generating region proposals using selective search, then classifying each with a CNN [26]. While accurate, the model is computationally expensive due to its multi-step pipeline and repeated CNN execution per region. Its high latency (~47 seconds per image on CPU) renders it unusable for real-time or embedded tasks.
- **Fast R-CNN** addressed this bottleneck by applying the CNN to the entire image once and extracting features from regions of interest (RoIs) using a specialized pooling layer [26]. This improved both training and inference speed, but still relied on external proposal generation, limiting its practical deployment.
- **Faster R-CNN** resolved the final bottleneck by integrating a Region Proposal Network (RPN) into the architecture [26]. This allows for end-to-end training and faster detection (5–17 FPS), while preserving accuracy. However, the two-stage nature still incurs higher computational cost, limiting use on lightweight hardware such as Raspberry Pi or similar edge devices.

b. One-Stage Detectors: SSD and YOLO Paradigms

- **SSD (Single Shot MultiBox Detector)** eliminates the proposal step by predicting object locations and classes directly from multiple feature maps in a single pass [4]. It uses multiscale feature extraction to enhance detection across object sizes and achieves respectable mAP scores (72–78%) at real-time speeds (~30–60 FPS). SSD's relatively moderate model size and single-shot architecture make it more suitable for resourceconstrained environments.
- **YOLO (You Only Look Once)** redefines object detection as a single regression problem, simultaneously predicting bounding boxes, class probabilities, and confidence scores for all regions of interest in a single global image pass [5]. This global context integration boosts speed and enables rapid inference, achieving over 150 FPS on modern GPUs, with mAP scores up to ~90% in recent versions. YOLO's compact architectures, anchor-free strategies, and progressive optimizations have made it a widely preferred option for embedded and mobile deployment scenarios.

To provide a comprehensive overview of the key distinctions and performance metrics among various object detection models, Table 1.2 presents a comparative analysis.

Table 1.2: Comparative Analysis of Major Object Detection Models

Algorithm	**Architecture**	**mAP (IoU=0.5)**	**Speed (FPS)**	**Model Size (MB)**	**Embedded Suitability**	**Strengths**	**Limitations**
R-CNN	Two-stage	58–66%	< 1	~500–600 MB [26]	Very low	Historically accurate (ImageNet/CNNbased)	Very slow; ~2000 RoIs per image; not suitable for real-time
Fast R-CNN	Two-stage	65–70%	~7–10	~300–350 MB [26]	Low	Faster than RCNN; single forward pass per image	Still depends on selective search for region proposals
Faster R-CNN	Two-stage	70–80%	~10–20	~170–250 MB [27]	Moderate	Integrated RPN; improved speed and accuracy	Higher memory and latency; not optimal for edge devices
SSD	One-stage	72–78%	~30–60	~100–120 MB [4]	Good	Multi-scale detection; fast; moderate size	Weaker performance on small or overlapping objects

YOLO (family)	One-stage	65–90%	~45–220	~10–80 MB [5,28]	Excellent	Global image processing; realtime speed; scalable	Early versions weak on small/close objects; improved in later versions[28]

Note: Values are approximate and based on standard benchmarks using high-end GPUs (e.g., NVIDIA Tesla K40, GTX 1080 Ti, or Tesla V100) as reported in . Actual performance will vary depending on implementation, dataset, input resolution, and hardware (GPU, CPU, edge TPU,..etc).

c. Scientific Synthesis: Why One-Stage Detectors Excel in Real-Time Agricultural Systems

From an engineering and agronomic perspective, single-stage object detectors especially SSD and YOLO are the most appropriate for deployment in embedded systems operating in agricultural environments. Unlike two-stage models that process candidate regions sequentially, one-stage models treat detection as a unified task. This significantly reduces latency and computational demand while supporting robust performance under variable lighting, partial occlusion, and noise [1,22].

Real-time detection is vital in fruit harvesting systems, where robotic end-effectors must respond instantly to visual inputs. YOLO and SSD meet this requirement by achieving high frame rates with compact architectures often under 100 MB in model size which is ideal for devices like Raspberry Pi or NVIDIA Jetson platforms [27]. Moreover, YOLO's strategy of global image interpretation in a single forward pass allows it to retain contextual cues, making it particularly effective in natural orchard scenes.

Although early versions of YOLO and SSD suffered from reduced accuracy on small or overlapping objects, recent research has introduced techniques such as feature pyramid networks, spatial attention, and multi-scale anchor-free decoding to mitigate these issues [28]. These ongoing advancements are steadily improving the robustness and generalization of single-stage detectors, making them increasingly suitable for tasks such as fruit maturity estimation, yield mapping, and autonomous harvesting.

In conclusion, the architectural simplicity, real-time speed, and low hardware footprint of single stage detectors explain their dominance in practical precision agriculture applications, particularly in embedded systems.

1.5.3. Evolution and Comparative Analysis of the YOLO Family

Since its initial release, the YOLO (You Only Look Once) series has undergone significant architectural evolution, consistently pushing the boundaries of real-time object detection. Across eight major versions, YOLO has shifted from a coarse grid-based detector to a highly optimized, anchor-free framework deployable on lightweight embedded hardware. This section provides a focused, version-by-version comparison of the YOLO family, highlighting its architectural refinements, empirical performance, and embedded applicability.

a. Technical Progression from YOLOv1 to YOLOv8

- **YOLOv1 (2016):** Introduced the core principle of single-stage detection, enabling high speed but limited in accuracy particularly on small objects due to its coarse output grid [37].
- **YOLOv2 (2017):** Incorporated anchor boxes, batch normalization, and a new backbone (Darknet-19), improving object localization and generalization[37].
- **YOLOv3 (2018):** Adopted a multi-scale feature strategy using Darknet-53, enabling better detection across object sizes but at the cost of increased model size [37].
- **YOLOv4 (2020):** Integrated CSPDarknet-53, spatial pyramid pooling, and novel augmentations (Mosaic, DropBlock), pushing accuracy and speed in parallel [37].
- **YOLOv5 (2020):** Developed independently by Ultralytics, it introduced a PyTorchnative implementation with modular scaling options (n, s, m, l, x), boosting accessibility and deployment flexibility. It quickly became the de facto standard in real-time detection, especially in research and embedded applications [37,38].
- **YOLOv6 and YOLOv7 (2022):** Focused on reparameterization, task alignment, and efficient training strategies. YOLOv7, in particular, offered competitive mAP with better runtime efficiency for medium-sized models [38].
- **YOLOv8 (2023):** Marked a significant shift with an anchor-free detection head, a decoupled classification-regression architecture, and enhanced export capabilities. It supports multiple formats (ONNX, TensorRT, CoreML) and features robust performance in both detection and segmentation tasks. Its small-footprint variants (e.g., YOLOv8n) are particularly suited for Raspberry Pi–class systems, balancing accuracy and inference speed for real-time field use [38,39].

b. Community Versions and Experimental Forks

While community-driven extensions such as YOLOv9, YOLOv10, and YOLOv11 have emerged with claims of performance boosts, these remain largely experimental and lack stable release pipelines, peer-reviewed validation, or consistent hardware compatibility. As of 2025, YOLOv8 remains the most robust, well-documented, and production-ready version for embedded agricultural systems [38,39].

To further contextualize the evolution and demonstrate the advantages of YOLOv8 for embedded systems, Table 1.3 provides a comparative summary of key YOLO versions from v1 to v8.

Table 1.3: Comparative Summary of YOLO Versions (v1–v8)

Version	**Year**	**Key Innovations**	**mAP (IoU=0.5)**	**Speed (FPS)**	**Model Size**	**Embedded Suitability**
YOLOv1	2016	Single-stage, grid prediction	~63%	~45	~60 MB	Low
YOLOv2	2017	Anchors, BN, Darknet-19	~76%	~40	~190 MB	Medium
YOLOv3	2018	Multi-scale, Darknet-53	~78–80%	~30	~250 MB	Medium
YOLOv4	2020	CSPNet, SPP, Mosaic aug.	~84%	~35–50	~244 MB	Good
YOLOv5	2020	PyTorch, modular scaling	~90% (COCO val)	~100	~7.5 MB (nano)	Excellent
YOLOv6/7	2022	Task-aligned, reparam. blocks	~52–56% (mAP@.5:.95)	~30–60	~70–150 MB	Good

YOLOv8	2023	Anchor-free, decoupled head	~53% (mAP@.5:.95)	~100–120	~3.2 MB (nano)	Excellent

Note: Metrics are based on standard COCO benchmarks. Performance on agricultural datasets may vary depending on training, preprocessing, and deployment platform. Detailed evaluations are provided in Chapter 3.

c. Selection Rationale for YOLOv8

For this project, YOLOv8n was selected as the optimal model due to its compact size, anchorfree architecture, and high frame-rate compatibility with the Raspberry Pi 5. Its validated performance, toolchain support (Ultralytics ecosystem), and smooth integration with embedded frameworks make it a stable and efficient choice for real-time maturity detection in field conditions.

1.5.4. YOLOv8 Architecture

The YOLOv8 architecture represents the latest evolution in single-stage object detectors, designed to maximize inference speed and detection accuracy while remaining deployable on resourceconstrained platforms such as the Raspberry Pi. Its modular organization and anchorfree detection strategy make it particularly well-suited for real-time applications in embedded agricultural systems [37].

a. Modular Structure: Backbone, Neck, and Head

YOLOv8 is structured into three main functional components, each optimized for a specific stage of the detection pipeline:

- **Backbone**: Responsible for initial feature extraction, the backbone processes input images through a series of convolutional and C2f (Cross-Stage Partial Fusion) blocks. This enables the model to capture both low-level details and high-level semantic patterns across multiple scales [38].
- **Neck**: The neck module aggregates and refines features from different stages of the backbone using upsampling, concatenation, and additional C2f layers. This multi-scale

fusion enhances the model's ability to detect objects of varying sizes and improves robustness under challenging field conditions [37].

- **Head**: The detection head outputs bounding boxes, class probabilities, and confidence scores. In YOLOv8, the head employs an anchor-free approach, directly predicting object locations and classes without relying on predefined anchor boxes. This simplifies training and improves generalization, especially in variable environments [39].

The specific internal components and their arrangement within the Backbone, Neck, and Head modules of the YOLOv8 architecture are visually detailed in Figure 1.7.

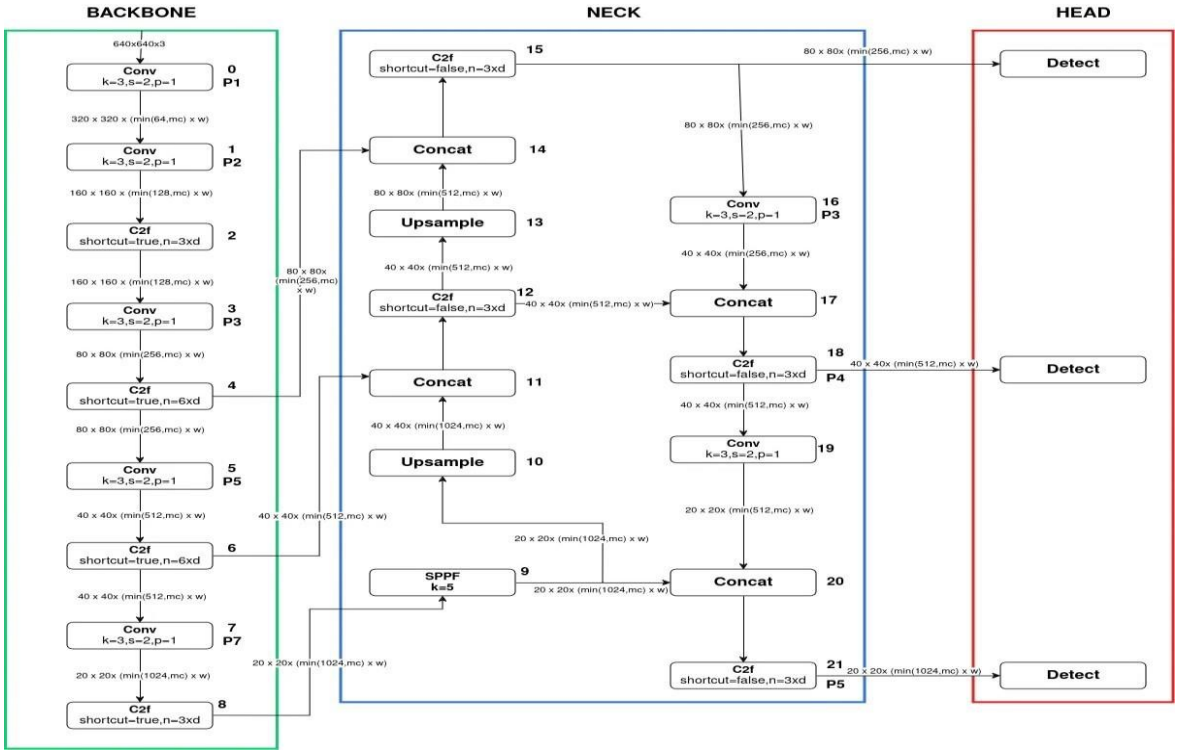


Figure 1.7: Structural Overview of the YOLOv8 Architecture

This diagram illustrates the data flow within YOLOv8, from initial feature extraction (Backbone), through multi-scale feature fusion (Neck), to final object prediction (Head). Key modules include C2f blocks, upsampling layers, and an anchor-free detection head, all optimized for efficient, realtime inference on embedded hardware.

To further summarize the functions and specific elements within each of these components, Table 1.4 provides a detailed overview of the YOLOv8 architectural components.

Table 1.4: Summary of YOLOv8 Architectural Components

Block	**Element / Module**	**Main Function**

Backbone	Conv, C2f, SPPF	Multi-level feature extraction; SPPF (Spatial Pyramid Pooling Fast) for aggregation
P1 to P5	Hierarchical layers	Capturing scale diversity
Neck	Upsample, Concat, C2f	Multi-scale feature fusion, semantic enrichment, parameter efficiency
Head	Detect (Anchor-Free)	Direct prediction of bounding boxes and classes; improved speed and flexibility

b. Key Advantages of YOLOv8

- **High Precision and Robustness**: YOLOv8 achieves strong accuracy across diverse datasets and remains resilient to noise, occlusion, and lighting variation key challenges in agricultural imagery [37,39].
- **Modular, Scalable Design**: The architecture is available in multiple sizes nano, small, medium, large, and extra-large allowing adaptation to different hardware constraints and real-time requirements [36].
- **Native Segmentation Support**: YOLOv8 supports both object detection and segmentation within a unified model, increasing its versatility for precision agriculture use cases [38].
- **Easy Export and Integration**: The model is compatible with ONNX, TorchScript, and CoreML, and is natively supported by Ultralytics tools. This facilitates rapid deployment and cross-platform experimentation on devices such as the Raspberry Pi or Jetson Nano [36].
- **Anchor-Free Detection**: The removal of anchor boxes reduces configuration complexity, accelerates training, and improves model generalization especially for realworld agricultural data with variable object scales and occlusion [39].

By combining architectural efficiency with state-of-the-art detection strategies, YOLOv8 provides a strong foundation for real-time, embedded vision systems in agriculture. Its design directly addresses the speed, simplicity, and deployability required for robust operation on platforms like the Raspberry Pi 5, as detailed in the subsequent chapters.

1.5.5. Performance Evaluation Metrics for Object Detection

The evaluation of object detection models requires a multifaceted approach that accounts not only for classification accuracy but also for spatial localization, computational efficiency, and suitability for the target deployment environment. In precision agriculture, where object detection systems operate under variable lighting, partial occlusion, and tight time constraints, a robust and comprehensive set of performance metrics is essential. This section presents the main quantitative indicators used to evaluate object detection models, with particular emphasis on their relevance for embedded and real-time applications in the agricultural domain [2,8,22].

a. Precision, Recall, and F1-Score

At the foundation of detection performance lie the concepts of precision and recall, which originate from classical information retrieval and binary classification:[36]

- **Precision (P)** measures the proportion of predicted bounding boxes that are correct:

$$\textbf{precision} = \frac{\textbf{True Positives (TP)}}{\textbf{True Positives (TP)} + \textbf{False Positives (FP)}}$$

A high precision score indicates that most detected objects are relevant and correctly localized. This is crucial in agricultural contexts where false detections may misguide harvesting robots or inflate maturity estimates.

- **Recall (R)** measures the proportion of actual objects that were correctly detected:

$$\textbf{recall} = \frac{\textbf{True Positives (TP)}}{\textbf{True Positives (TP)} + \textbf{False Negatives (FN)}}$$

High recall ensures that most target objects (e.g., ripe fruits) are detected, even under partial occlusion or poor visibility.

- **F1-Score** provides a harmonic mean between precision and recall, offering a balanced indicator when both metrics are important:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

This score becomes particularly relevant when a system must optimize both detection reliability and completeness such as when monitoring fruit maturity progression over time.

b. Intersection over Union (IoU)

To determine whether a predicted bounding box corresponds to a true object, object detection models rely on Intersection over Union (IoU). This metric quantifies the overlap between a predicted box Bp and a ground truth box Bgt as defined by [36]:

$$\text{IoU} = \frac{|Bp \cup Bgt|}{|Bp \cap Bgt|}$$

An IoU threshold (IoU $\geq$ 0.5) is used to define a true positive meaning the predicted box overlaps sufficiently with the actual object. If the IoU is below the threshold, the detection is counted as a false positive, even if the class label is correct [22,28].

As illustrated in Figure 1.8, IoU quantifies the overlap between predicted and ground truth bounding boxes.

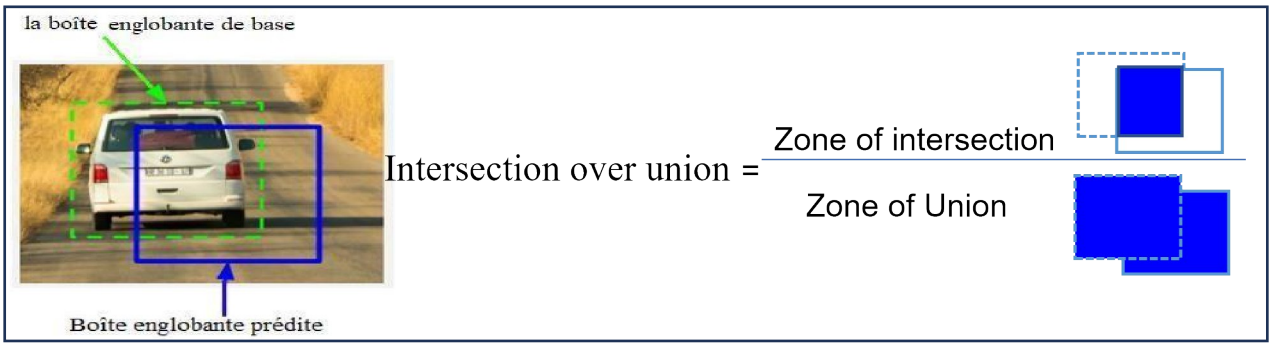


Figure 1.8: Illustration of Intersection over Union (IoU) between a predicted bounding box and the ground truth.

IoU is calculated as the ratio of the area of overlap to the area of union between the two boxes. Further illustrating the practical implications of this metric, Figure 1.9 provides visual examples of scenarios with high versus low IoU scores.

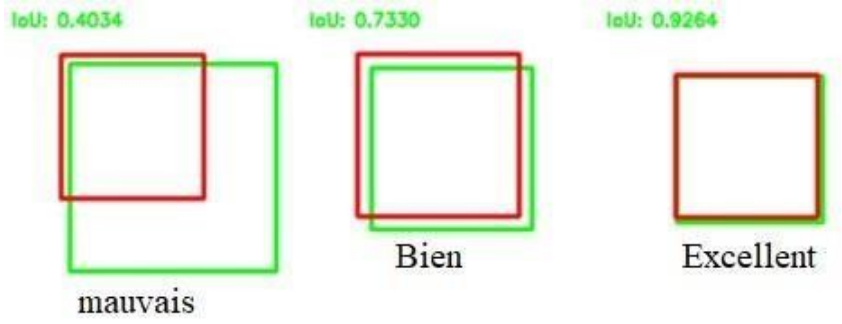


Figure 1.9: Visual examples of high vs low IoU scores.

- A high IoU indicates that the model has successfully localized the object.
- A low IoU reflects poor localization or a wrong prediction.

IoU is critical for agricultural scenarios where fruits may be tightly clustered or partially occluded. Precise localization affects not only detection accuracy but also physical interactions such as robotic picking.

c. mean Average Precision (mAP)

The most comprehensive and widely used metric in object detection is mean Average Precision (mAP). It aggregates the model's precision-recall performance across all classes and across multiple IoU thresholds.

For a given class, the average precision (AP) is computed by integrating the area under the precision-recall curve. The mAP is then calculated as the mean of these APs over all classes [36].

Figure 1.10 illustrates an example of a precision-recall curve, highlighting how the area under this curve corresponds to the Average Precision (AP) used in mAP calculation.

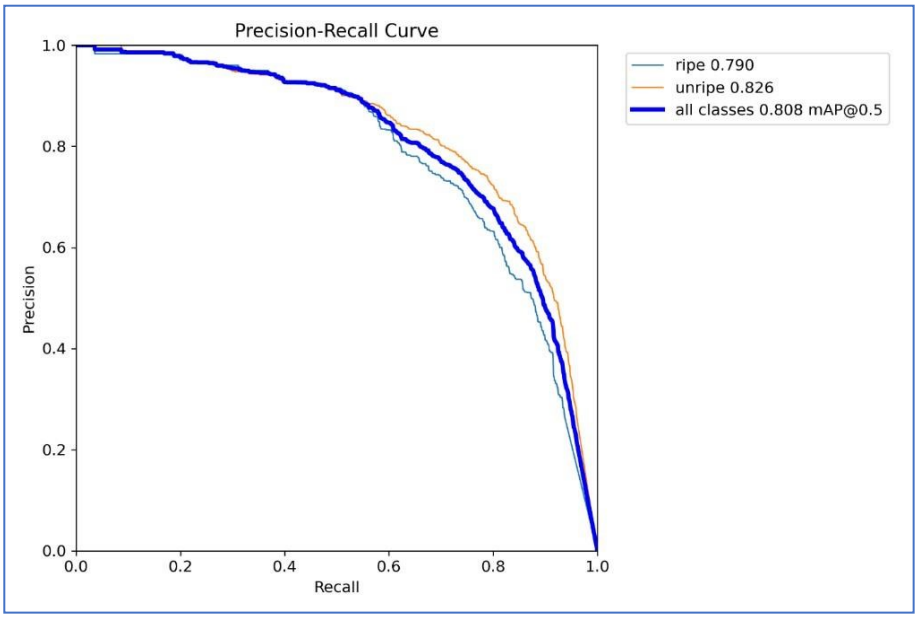


Figure 1.10: Example of a precision-recall curve for a single object class.

The area under the curve corresponds to the Average Precision (AP) used in mAP calculation.

- Modern benchmarks such as COCO use a more stringent standard:

mAP@[0.5:0.95]=mean AP over IoU thresholds from 0.5 to 0.95 (step 0.05)

This multi-threshold evaluation penalizes models that fail to achieve precise localization. It is particularly relevant in agricultural tasks where overlapping fruits, complex backgrounds, and variable scales challenge the model's spatial accuracy [2,22].

d. Inference Speed (FPS)

Frames per Second (FPS) quantifies how many images a detection system can process in one second. It reflects real-time responsiveness a critical factor in robotic harvesting, where decisions must be made instantaneously based on live camera input.

Lightweight, single-stage models like YOLO are typically optimized for high FPS (often exceeding 150 FPS on GPUs), whereas two-stage models such as Faster R-CNN operate at lower speeds (10–20 FPS) [5,26,27]. On embedded devices like Raspberry Pi or Jetson Nano, achievable FPS is much lower due to limited hardware resources, making speed a decisive performance constraint [27].

e. Model Size and Computational Complexity

In addition to accuracy and speed, model size typically expressed in megabytes (MB) determines whether a neural network can be deployed on a resource-constrained device. Smaller models are easier to load, update, and execute in low-power environments. For example, YOLO variants often range from 5 MB to 80 MB, while two-stage models with heavy backbones can exceed 200 MB [28].

Model size correlates with:

- **RAM and storage requirements**
- **Inference latency**
- **Power consumption**, which is a critical factor for solar- or battery-powered agricultural deployments

Some studies also consider FLOPs (floating point operations per second) as a complementary indicator of computational load [27].

f. Importance of These Metrics in Agricultural AI

In agricultural field conditions, system performance cannot be evaluated based on accuracy alone. Models must also meet constraints of speed, robustness, and hardware compatibility. A fruit detection system with high precision but low recall may miss harvestable fruits; one with high recall but low precision may trigger unnecessary harvesting actions. Likewise, models too large to run on local edge devices introduce latency, reliance on internet connectivity, or excessive energy costs.

Consequently, performance evaluation must be multi-dimensional. The metrics presented here provide a rigorous framework for selecting, comparing, and optimizing models for embedded AI applications in precision agriculture. They serve not only as benchmarks but also as practical indicators of how well a detection model will perform in the real-world constraints of orchard environments.

1.6. Embedded System Constraints and Deployment Challenges

The practical deployment of deep learning–based object detection models in agriculture does not end with achieving high performance in experimental settings. Instead, it must account for the realities of running these models on embedded systems deployed directly in the field. These

platforms such as the Raspberry Pi, Jetson Nano, or Google Coral offer a compact, low-cost, and energy-efficient alternative to cloud-based solutions, enabling AI applications to function locally and autonomously [2,22,27].

This localized, embedded approach is particularly relevant in agriculture, where connectivity is often intermittent, latency constraints are critical for real-time decision-making, and privacy or energy constraints make cloud inference impractical [2,22,30]. Applications such as robotic harvesting, fruit counting, and maturity monitoring increasingly rely on embedded systems to ensure timely and context-sensitive actions without dependence on external servers [18,22].

However, the deployment of object detection models on such platforms is constrained by multiple factors: limited computational resources (CPU, GPU), restricted memory, thermal management, and energy efficiency requirements. These constraints impose significant tradeoffs in model selection, preprocessing, and runtime behavior [27,30,31].

Furthermore, agricultural environments introduce unique operational challenges. Variability in natural lighting, occlusions by foliage, weather conditions, and camera positioning can significantly degrade detection accuracy even for well-trained models. The deployment system must therefore balance model robustness, speed, and hardware constraints to ensure reliable performance in these uncontrolled environments [2,22,28,31]

1.6.1. Embedded Platforms for On-Device AI in Agriculture

The shift from laboratory AI models to real-world agricultural deployment depends critically on the availability of embedded computing platforms that are both efficient and field-resilient.

In environments where power, mobility, and budget are constrained, centralized or cloud-based processing becomes impractical due to bandwidth limitations, latency concerns, or system fragility [2,22,30]. Embedded platforms such as the Raspberry Pi, Jetson Nano, and Google Coral provide the necessary balance of portability, energy efficiency, and inference capability for real-time AI applications in precision agriculture.

These devices enable models like YOLO to operate locally on small-scale robots or field monitoring tools. They support tasks such as real-time fruit maturity detection, autonomous harvest triggering, and dynamic orchard monitoring. However, each platform presents distinct trade-offs in terms of speed, memory, cost, and integration complexity. The following

subsections provide an in-depth academic review of these embedded systems, focusing on their suitability for agricultural AI deployments.

a. Raspberry Pi 4 / Raspberry Pi 5

The Raspberry Pi series represents the most accessible embedded computing platform for agricultural research and field deployment. Originally designed for educational purposes, its hardware has evolved to support real-time inference for lightweight AI models. In particular, the Raspberry Pi 5, released in 2023, offers substantial performance improvements over previous generations while retaining its low-cost, low-power design [32].

- **Key Specifications:**

  - **Processor**: Quad-core ARM Cortex-A76 @ 2.4 GHz
  - **RAM**: 4 or 8 GB LPDDR4X
  - **AI Acceleration**: CPU only (Coral USB Accelerator optional)
  - **Power Consumption**: ~5–8 W under load
  - **Cost**: ~35–90 USD (board only)
  - **Camera Support** : 2x MIPI CSI (supports Pi Camera v2.1 and HQ)

Despite its lack of onboard GPU or TPU, the Pi can execute efficient models such as YOLOv8n in real-time at ~5–15 FPS, depending on image resolution and model complexity [27,30]. It runs TensorFlow Lite, ONNX, and PyTorch (CPU version), making it versatile for prototyping and educational use.

In agricultural contexts, the Pi excels in ease of integration with sensors and actuators, and benefits from a vast open-source community. It is especially suitable for fruit maturity detection, environmental sensing, and robot guidance in low-budget settings [22,30].

Figure 1.11 provides a visual representation of the Raspberry Pi 5 board, showcasing its key interfaces including the CSI-connected camera module and GPIO header, essential for embedded applications.

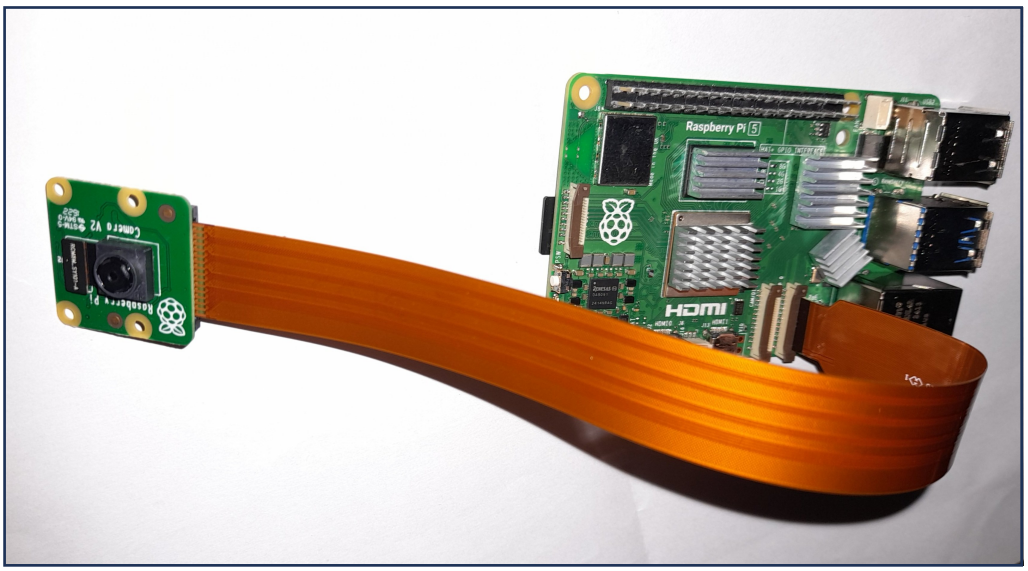


Figure 1.11: Raspberry Pi 5 board with CSI-connected camera module and GPIO header.

- **Strengths**: Affordable, well-supported, low power, easy integration
- **Limitations**: No native GPU or TPU; limited FPS for larger models; heat management may be needed in sunlight [30,31]

b. NVIDIA Jetson Nano

The Jetson Nano, developed by NVIDIA, targets edge AI applications and is specifically designed for accelerated deep learning. It features a 128-core Maxwell GPU and is capable of executing mid-sized object detection models (e.g., YOLOv5s) with reasonable inference speed on low-power systems [27,31,33].

Jetson Nano supports TensorFlow, PyTorch, and TensorRT for optimized inference, enabling 15– 30 FPS on models like YOLOv5s or SSD under controlled conditions. This makes it suitable for faster and more complex models than what the Raspberry Pi can sustain.

However, Jetson Nano presents higher integration complexity and requires careful power and thermal design for field deployment. It is best suited for semi-autonomous ground robots or fixed monitoring stations where real-time object detection is critical and power is available.

- **Strengths**: GPU acceleration, good FPS, optimized AI pipeline.
- **Limitations**: Higher power draw, costlier, less beginner-friendly setup [27,31,33].

c. Google Coral Dev Board and USB Accelerator

The Google Coral platform focuses on ultra-low-power inference through its dedicated Edge TPU coprocessor, which supports only 8-bit quantized TensorFlow Lite models. Available as both a full development board and a plug-and-play USB accelerator, Coral enables real-time inference at extremely low energy costs [30,34].

Coral's performance is excellent when models are properly quantized achieving 30–100+ FPS on supported models. However, its limitation to TensorFlow Lite (INT8) introduces challenges. Not all models convert well to this format without sacrificing accuracy, especially in agriculture where visual noise and subtle color differences are critical to tasks like ripeness detection [30,34].

- **Strengths**: Ultra-efficient, compact, real-time performance
- **Limitations**: Strict model format constraints, conversion complexity, no native support for PyTorch or ONNX [30,34]

d. Synthesis: Comparative Assessment of Embedded Platforms

The comparative review of embedded AI platforms highlights the nuanced trade-offs between processing power, energy efficiency, compatibility, and deployment complexity. While devices like the Jetson Nano and Coral TPU offer specialized acceleration, their integration costs and format constraints limit flexibility in diverse agricultural settings. In contrast, the Raspberry Pi 5 strikes a pragmatic balance between performance and usability supporting lightweight object detection models at low cost, with minimal setup and strong community support.

The following section now builds on this assessment by examining the fundamental hardware constraints that govern real-time AI deployment in the field, and how such limitations directly influence system architecture, model selection, and field robustness.

1.6.2. General Hardware Constraints for Embedded AI in Agriculture

Having identified the Raspberry Pi 5 as the most appropriate embedded platform for this project, it is essential to now explore the specific hardware constraints that inform its practical use in agricultural environments. These constraints not only define the operational limits of AI

systems in the field but also shape critical design choices from the type of object detection model employed to the allowable resolution, frame rate, and system energy budget.

Unlike datacenter-grade hardware, embedded systems must operate with tight restrictions on processing capacity, memory availability, storage throughput, and power supply. These factors impose real-world limitations on model complexity, inference speed, and overall system responsiveness especially under outdoor conditions involving variable light, temperature, and connectivity. The remainder of this section details each of these limitations and explains how they were accounted for in the design and implementation of this work.

a. Processing Power: Lightweight Models for Real-Time Use

Embedded CPUs lack the massive parallelism of GPUs or cloud servers. This severely limits the size and complexity of models that can run at acceptable frame rates. In this project, realtime detection of apricots requires inference within milliseconds per frame. Large models were excluded early due to high computational demands [27,31].

b. Memory and Storage Limitations

Most embedded systems offer between 1 GB and 8 GB of RAM, which must support not just the model, but also the camera feed, detection buffers, and operating system. This limitation restricts the use of high-resolution video, large models, or concurrent processing tasks.

Additionally, microSD-based storage, common on platforms like the Raspberry Pi, offers limited speed and endurance. This affects how much data can be logged or buffered and how quickly models and scripts can be loaded. For this reason, the system uses low-resolution input (e.g., 640×480) and on-the-fly inference only, avoiding large data accumulation or complex pipelines [32].

- **Low Power Consumption for Field Deployment**

Agricultural deployments often lack direct access to stable power. Whether mounted on a mobile robot, placed in an orchard, or operated in a greenhouse, AI systems must function on batteries or solar panels, making energy efficiency a priority [31,33].

The Raspberry Pi 5, with a typical power draw of 5–8 W, meets these constraints while still supporting basic computer vision. Devices with discrete GPUs, like the Jetson Nano (~10 W), or co-processors like the Coral TPU (~2 W), offer alternatives, but introduce new trade-offs in compatibility and setup complexity [30,34].

c. Ease of Camera and Sensor Integration

Precision agriculture applications often require simultaneous access to camera feeds, environmental sensors (e.g., humidity, light), and actuators. Embedded platforms must therefore
provide simple GPIO access, camera compatibility (CSI or USB), and reliable communication interfaces.

The Raspberry Pi was selected in part because it supports native CSI cameras, including the Pi Camera v2.1 used in this project, and offers GPIO headers for future sensor extensions. This ensures tight hardware integration without requiring additional microcontrollers [32,34].

Beyond the Raspberry Pi's specific integration capabilities, understanding the broader landscape of embedded processing units is crucial for optimizing AI deployments. Table 1.5 provides a comparative overview of different embedded processing units, including CPUs, GPUs, and TPUs, detailing their roles in embedded AI.

Table1.5: Comparison of Embedded Processing Units: CPU, GPU, and TPU

Component	**Description**	**Role in Embedded AI**
CPU (Central Processing Unit)	General-purpose processor found in all devices. Excels at sequential tasks.	Suitable for lightweight models (e.g., YOLOv8n), but lacks speed for largescale vision tasks [30,35].

GPU (Graphics Processing Unit)	Optimized for parallel data processing, ideal for large neural networks.	Provides significant acceleration (e.g., Jetson Nano), but increases cost and power usage [33].
TPU (Tensor Processing Unit)	Specialized chip for quantized AI inference, developed by Google.	Enables ultra-fast, low-power inference (e.g., Coral), but only with 8-bit TensorFlow Lite models [34,36].

In this project, the decision to use a CPU-only platform (Raspberry Pi 5) reflects the need for simplicity, compatibility, and field operability, even at the cost of maximum inference speed. By understanding and addressing these constraints, the system design is tailored for robust, field-ready performance in Algerian agriculture.

1.7. CONCLUSION

This chapter laid the conceptual, scientific, and technological foundations of the project, anchoring the issue of apricot cultivation in Algeria within its current challenges. It highlighted the limitations of traditional agricultural practices and the need to integrate intelligent solutions to improve fruit ripeness detection.

On the scientific front, the principles of artificial intelligence, machine learning, deep learning, and computer vision were presented, directly linked to the needs of precision agriculture. A comparative analysis of the main families of object detection algorithms, as well as an in-depth study of the evolution of the YOLO family, have clarified the criteria that will guide the evaluation of models in the subsequent work.

Finally, the comparison of hardware platforms (Raspberry Pi, Jetson Nano, Coral TPU) has made it possible to identify the opportunities and concrete limitations of embedded AI in an agricultural context, taking into account the constraints of performance, cost, and field integration.

CHAPTER 2: Methodology for Detection System Development

2.1. INTRODUCTION

Building on the theoretical and contextual foundations presented in Chapter 1, this chapter details the practical methodology used to design, develop, and deploy an intelligent vision system for apricot maturity detection. The goal is to support more efficient, objective, and scalable decision making in precision agriculture, particularly where manual fruit assessment remains labor intensive and inconsistent.

A structured and reproducible workflow is followed, beginning with the creation of a custom annotated dataset for object detection tasks. This is followed by preprocessing to simulate real world deployment conditions using Raspberry Pi Camera characteristics. The training and comparative evaluation of several YOLOv8 models is then addressed, leading to the selection of a candidate optimized for embedded inference. Finally, the chosen model is deployed on a Raspberry Pi device and tested in real-time conditions to assess its functional behavior.

By translating conceptual insights into applied implementation, this chapter establishes the technical foundation for a lightweight, deployable solution for maturity detection in orchard environments. Performance results and critical evaluation will be presented in Chapter 3.

2.2. Project Workflow and System Pipeline

To contextualize the technical details presented in this chapter, Figure 2.1 summarizes the complete workflow of the intelligent apricot maturity detection system developed in this thesis. This pipeline illustrates the sequential integration of data collection, model development, evaluation, and embedded deployment, culminating in real-world inference and feedback for agronomic decision-making.

➢ **The main stages of the project pipeline are as follows**

- **Data Collection and Annotation**

Acquisition of apricot images under diverse orchard conditions, followed by manual annotation of fruit maturity classes (e.g., ‘ripe’, ‘unripe’) to create a labeled dataset.

- **Model Selection and Training**

Selection of a suitable object detection architecture (YOLOv8), configuration of training parameters, and supervised learning on the annotated dataset to optimize detection and classification performance.

- **Evaluation and Validation**

Quantitative and qualitative assessment of the trained model using independent validation and test sets, including metric analysis (precision, recall, mAP), confusion matrices, and visual inspection of detection outputs.

- **Deployment on Embedded Hardware**

Transfer of the validated model to a Raspberry Pi 5 platform, integration with a real-time camera interface, and adaptation of inference scripts for low-latency, resource-constrained environments.

- **Real-World Inference and Feedback**

Execution of the embedded system in simulated or actual orchard conditions, with live detection outputs supporting agronomic tasks such as assisted harvesting, yield estimation, and maturity monitoring. Feedback from deployment informs further refinement of the system.

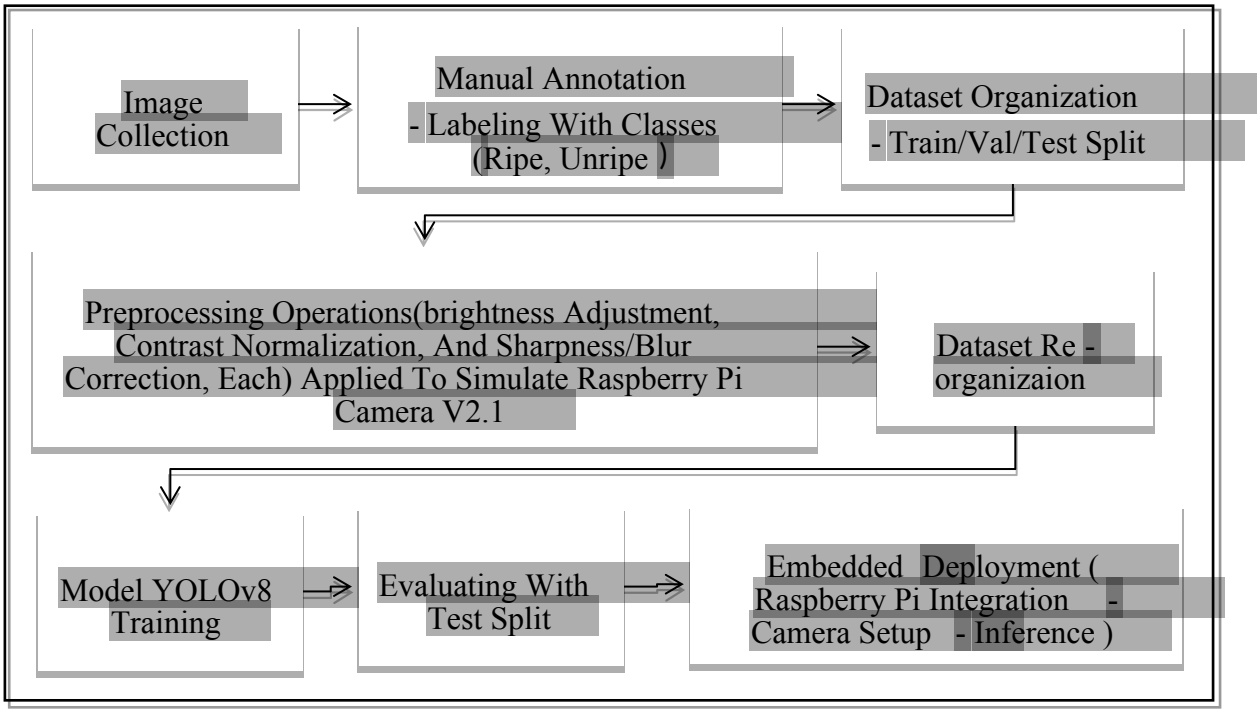


Figure 2.1: System Pipeline for Intelligent Apricot Maturity Detection

Schematic representation of the end-to-end workflow, from image collection and annotation to model training, evaluation, embedded deployment, and real-world inference.

2.3. Dataset Creation

In object detection tasks, the quality, diversity, and contextual relevance of the dataset directly influence the performance and robustness of the trained model. Given the visual variability inherent to agricultural environments such as inconsistent lighting, occlusions, and complex backgrounds the construction of a tailored dataset was a critical first step in this project.

The objective was to build a dataset capable of supporting accurate and generalizable detection of apricot fruits at different maturity stages under conditions approximating real deployment. However, due to the seasonal unavailability of apricots during the initial development period,

field data acquisition was not feasible. As a result, an alternative strategy based on online sources was adopted, acknowledging the practical limitations while striving to ensure scientific rigor through controlled selection and preprocessing.

2.3.1. Image Collection and Sources

A total of 1154 high-resolution images were collected from three main sources:

- **iStock®**: Professionally captured images with strong resolution and color fidelity.
- **Google Images**: Natural and diverse images from orchards, blogs, and agricultural articles.
- **Roboflow Open Datasets**: Public datasets intended for academic use in fruit detection.

These sources were chosen to cover a broad range of visual conditions while maintaining sufficient image quality for reliable object annotation. Although not acquired with the Raspberry Pi Camera v2.1, this image base provided a foundation for domain adaptation via targeted preprocessing (see Section 2.3).

2.3.2. Selection Criteria and Dataset Construction

To maximize generalization and avoid dataset bias, a strict filtering process was applied based on seven technical and contextual criteria, summarized in **Table 2.1**. These criteria align with best practices in object detection dataset design, which emphasize the importance of resolution, viewpoint diversity, lighting variability, and class balance for model robustness [42,43,46]

Table 2.1: Image Selection Criteria for Dataset Construction

Criterion	Description	Visual Example
High Resolution	Clear texture, edges, and contours required for precise bounding box annotation.	

Lighting Diversity	Inclusion of sunlit, shaded, and diffuse-light images to improve model robustness.	
Viewpoint Variation	Top, side, and frontal views to prepare for variable camera positioning.	
Distance Realism	Approximate 30–40 cm fruit to camera distance, similar to Raspberry Pi setup.	
Background Complexity	Presence of leaves, branches, sky, and overlapping elements to mimic orchard scenes.	
Visual Occlusions	Inclusion of partially hidden fruits to simulate real harvesting scenarios.	

Class Representation	Balanced presence of ripe and unripe apricots to avoid class bias during training.	

These criteria ensured that the resulting dataset could capture the full spectrum of visual conditions expected in field deployment. In particular, lighting variation and occlusion were prioritized, as they are among the most frequent sources of error in real-time fruit detection [44]

To illustrate the diversity of visual styles across sources, Figure 2.2 presents examples from each platform. These images highlight the differences in resolution, background noise, and color rendering, all of which were later normalized during the preprocessing phase.

Figure 2.2: Examples of Dataset Images from Different Sources

(a) iStock – high-resolution, professionally composed image

(b) Google Images – orchard scenes with natural lighting and clutter

(c) Roboflow – academic dataset with moderate quality and noise

This image selection phase was not only guided by visual criteria but also by the practical objective of replicating conditions likely to be encountered by the Raspberry Pi Camera. Gi

the domain gap between curated online images and embedded field conditions, this selection strategy served as a preparatory step for the domain adaptation pipeline discussed in Section 2.4.

By constructing a dataset with realistic variability and balanced class distribution, this phase laid the groundwork for training an object detection model capable of generalizing beyond its training set. The next section details how the annotation process was conducted to preserve this variability and prepare the dataset for integration with the YOLOv8 training framework.

2.3.3. Cleaning and Deduplication

Following the initial image collection phase, a dataset cleaning and deduplication step was performed to improve consistency, reduce redundancy, and ensure the quality of inputs used for annotation and training. Image datasets compiled from heterogeneous online sources frequently contain duplicate entries, near-duplicates, or low-value samples that can bias learning and degrade model generalization [42]. Eliminating such redundancies is a widely recognized best practice in object detection workflows [42].

a. Removal of Duplicates

The raw dataset initially comprised 1,154 images. Upon inspection, several files were found to be visually redundant either exact copies or minor variants of the same scene (e.g., cropped, resized, or recompressed versions). To detect and remove these duplicates systematically, a perceptual hashing method was employed using the phash algorithm from the *imagehash* Python library.

Each image was encoded into a compact perceptual hash summarizing its visual content. Pairwise comparisons were conducted using Hamming distance, with a threshold of ≤ 3 used to flag potential duplicates. For each detected group of similar images, one representative was retained while the remaining entries were discarded. This filtering step was automated using a custom Python script, ensuring reproducibility and consistency.

The complete workflow is illustrated in Figure 2.3, which outlines the process of hashing, distance comparison, and duplicate removal.

Figure 2.3: Workflow of Perceptual Hashing and Duplicate Filtering

Visual representation of the image cleaning pipeline. Input images are hashed using phash, compared via Hamming distance, and filtered based on a similarity threshold. Only unique images are retained for further use.

This approach reduced redundancy in the dataset while preserving visual diversity two key conditions for preventing overfitting and promoting robust generalization across varied deployment scenarios.

b. Structured Renaming for Dataset Management

After deduplication, all retained images were renamed following a standardized convention: apricot-1.jpg, apricot-2.jpg, and so on. This step facilitated structured dataset management, particularly during annotation with Roboflow and integration into the YOLOv8 training framework.

Although the final numeric sequence included some discontinuities due to further removals in subsequent phases, this had no effect on dataset integrity or functionality. The renaming strategy ensured traceability, simplified preprocessing, and improved compatibility across the project pipeline.

2.3.4. Manual Annotation with Roboflow

Following the cleaning and restructuring of the dataset, a manual annotation phase was carried out to define ground truth labels for object detection. This step involved localizing individual apricots in each image and assigning them to one of two predefined classes. All annotations were performed using the Roboflow web platform, selected for its compatibility with YOLOv8, intuitive bounding box tools, and integrated dataset export features.

Each image was reviewed individually, and rectangular bounding boxes were manually drawn around visible apricots. Two object classes were used, defined as follows :

- **Ripe**: Fully developed fruits exhibiting dominant orange to reddish hues.
- **Unripe**: Immature or partially developed fruits, typically green, pale yellow, or lacking full pigmentation.

This binary classification schema was established to support multiple downstream agricultural objectives (e.g., thinning, selective harvesting), and aligns with practical visual cues used by expert fruit pickers. The classification relied on color-based visual maturity, which is widely adopted in fruit detection literature for early-stage AI deployment [20,12].

During the annotation process, several quality control measures were applied to ensure label accuracy and class purity:

- Only fruits that were clearly identifiable and unobstructed were labeled.
- Ambiguous or partially visible apricots were excluded if their maturity stage could not be reliably determined.
- A subset of images initially included in the dataset was discarded upon inspection, as they were found to contain non-apricot fruits (e.g., small peaches, plums) or low-resolution scenes that impaired precise labeling.
- Consistency in class assignment was maintained across annotators through reference visual guidelines established during preliminary annotation rounds.

These precautions reflect best practices in manual annotation for object detection, where label noise and visual ambiguity are known to degrade model performance [43]. The resulting dataset was therefore refined not only for technical correctness but also for biological relevance in the context of apricot maturity detection.

At the conclusion of this phase, the dataset consisted of 823 annotated images, each containing one or more bounding boxes labeled as ripe or unripe. This labeled dataset serves as the foundational ground truth for all subsequent training, validation, and deployment stages.

To illustrate the annotation process, Figure 2.4 presents representative screenshots from the Roboflow platform, highlighting the visual interface and examples of annotated images with both classes clearly distinguished.

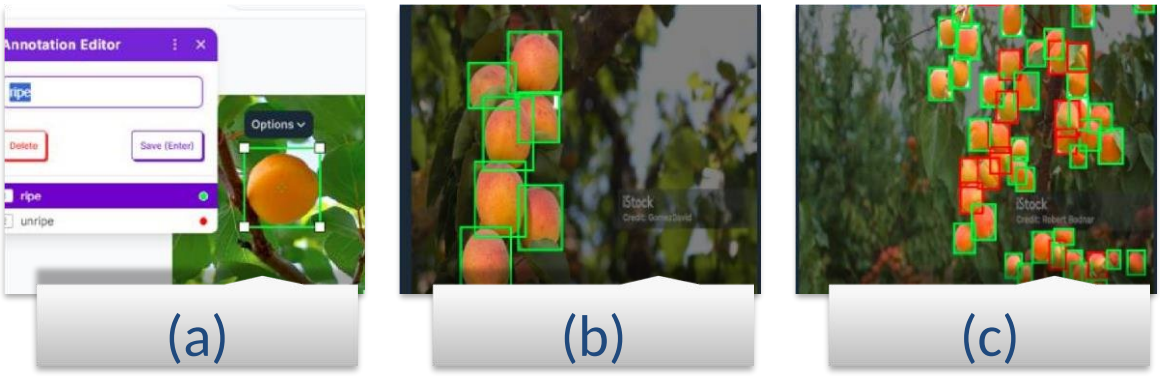


Figure 2.4: Manual Annotation of Apricots Using Roboflow

(a) Roboflow interface showing bounding box editing and class assignment.

(b) Example of an image labeled exclusively with ripe apricots.

(c) Densely populated scene annotated with both ripe (green) and unripe (red) bounding boxes.

2.3.5. Dataset Structuring and Splitting for YOLOv8

To prepare the annotated dataset for training with the YOLOv8 object detection framework, a standardized directory structure and data partitioning protocol were implemented. YOLOv8 requires a specific format that clearly separates training, validation, and test subsets, with corresponding label files stored in parallel directories. This structure enables automatic association between images and annotations during model training and evaluation.

The dataset was organized as follows:

- Images/train/ and labels/train/
- images/valid/ and labels/valid/
- images/test/ and labels/test/

This hierarchical format ensures compatibility with the Ultralytics training pipeline and promotes reproducibility in both local and cloud-based environments.

To support robust model generalization and reliable performance evaluation, the dataset was partitioned using an 80/10/10 split a widely accepted standard in object detection literature [42,36].
The subsets serve the following functions:

- **Training set (80%)**: used to iteratively update model weights,
- **Validation set (10%)**: used during training to monitor performance and tune internal parameters,
- **Test set (10%)**: held out entirely during training and reserved for final model evaluation.

The partitioning process was carried out using the Roboflow platform, which provides automated splitting while maintaining class distribution balance across subsets. This is particularly important in binary classification tasks, where imbalanced class representation can lead to biased learning and skewed performance metrics [43].

After splitting, the final dataset included:

- 658 images for training,
- 83 for validation,
- 82 for testing.

This distribution is visualized in Figure 2.5, which illustrates the proportion of images allocated to each subset.

Figure 2.5: Dataset Split Used for YOLOv8 Training

Distribution of the 823 annotated images across training (80%), validation (10%), and test (10%) subsets, as performed by the Roboflow platform.

Proper dataset partitioning is essential for detecting overfitting, which occurs when a model memorizes training data including noise or irrelevant patterns instead of learning generalizable features. By evaluating model performance on independent subsets, this approach helps ensure that the trained model can perform reliably on unseen data, a prerequisite for deployment in realworld agricultural scenarios.

2.3.6. Dataset Format and YOLOv8 Compatibility

To ensure seamless integration with the YOLOv8 training framework, the finalized dataset was exported from Roboflow in full compliance with the official Ultralytics YOLOv8 format [36]. This step was critical to guarantee structural compatibility, reduce preprocessing errors, and support reproducibility across development environments.

Prior to export, all images were uniformly resized to 640×640 pixels a resolution natively supported by YOLOv8. This resizing ensured consistent input dimensions, reduced computational overhead, and simplified subsequent data augmentation. Such standardization is widely recommended in real-time object detection pipelines to facilitate batch processing and memoryefficient training [42].

The exported dataset adopted a hierarchical directory structure consistent with YOLOv8 expectations, dividing the images and corresponding annotation files into separate subfolders for each data split:

- images/train/, labels/train/
- images/valid/, labels/valid/
- images/test/, labels/test/

This organization enables the YOLOv8 engine to automatically locate image-label pairs and ensures a reproducible workflow for training, validation, and inference.

A core component of the YOLO dataset format is the data.yaml configuration file, which defines:

- The relative paths to the image folders for each split,
- The number of object classes (nc), and
- The class names used in annotation ('ripe' and 'unripe').

This configuration file serves as the entry point for model initialization and training, allowing YOLOv8 to interpret dataset structure without additional manual setup.

Figure 2.6 illustrates an example of this data.yaml configuration file from the apricot ripeness dataset.

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 2
names: ['ripe', 'unripe']
```

Figure 2.6: data.yaml Configuration File of my dataset

Defines image directory paths and class names for YOLOv8 model initialization.

Each image in the dataset is paired with a .txt annotation file in YOLO format, where each row encodes a single bounding box using normalized coordinates:

<class_id> <x_center> <y_center> <width> <height>

- **class_id:** Integer class index (0 for ripe, 1 for unripe),
- **x_center, y_center:** Center coordinates of the bounding box (normalized to [0, 1]),
- **width, height:** Box dimensions, also normalized.

This compact format enables fast parsing and GPU-accelerated training, and is widely adopted in object detection research due to its simplicity and efficiency [36].

A sample YOLOv8 annotation file, demonstrating the normalized bounding box coordinates and class IDs, is shown in Figure 2.7.

```
1 0.34609375 0.509375 0.20703125 0.26328125
1 0.4625 0.31484375 0.1640625 0.22890625
1 0.70078125 0.503125 0.1828125 0.2328125
0 0.24375 0.3921875 0.1734375 0.2421875
0 0.2375 0.68515625 0.2046875 0.27109375
1 0.628125 0.2203125 0.1734375 0.2375
0 0.4703125 0.55546875 0.16640625 0.253125
1 0.5859375 0.803125 0.17734375 0.25390625
0 0.56171875 0.459375 0.12578125 0.24140625
0 0.36015625 0.2984375 0.1265625 0.19140625
1 0.7390625 0.75 0.16796875 0.20234375
1 0.49296875 0.1734375 0.125 0.18203125
0 0.18125 0.284375 0.16328125 0.22890625
1 0.0515625 0.29140625 0.10078125 0.190625
1 0.1921875 0.078125 0.15 0.15390625
0 0.49765625 0.89453125 0.140625 0.1984375
1 0.80859375 0.49140625 0.13671875 0.21796875
```

Figure 2.7: Sample YOLOv8 Annotation File (.txt) from my dataset

Each row represents one object with its class ID and normalized bounding box parameters.

This export step finalized the dataset preparation phase and served as a critical bridge between manual annotation and model training, ensuring that the dataset could be directly consumed by the YOLOv8 framework with no further modification.

2.4. Preprocessing Strategy and Applied Techniques

The performance of deep learning models in real-world scenarios is highly sensitive to discrepancies between training data and deployment environments a phenomenon commonly referred to as domain shift [41]. In the context of this project, while the initial dataset consisted of high-resolution images sourced from curated online repositories, the target deployment platform the Raspberry Pi Camera v2.1 produces images of lower quality, with reduced contrast, variable brightness, and mild lens-induced distortions. Without adequate adaptation, this mismatch could significantly degrade the model's generalization and reliability during inference in agricultural settings.

To bridge this domain gap, a targeted preprocessing strategy was developed to simulate the visual characteristics of Raspberry Pi imagery while maintaining the annotation accuracy of the original dataset. The strategy was guided by two primary objectives :

- **Minimize domain shift** by transforming high-quality training images to visually resemble Raspberry Pi captures in terms of contrast, lighting, and sharpness;
- **Enhance model robustness** by exposing the detection model to more realistic and variable input conditions, increasing its tolerance to environmental noise and camera limitations [42].

Unlike generic augmentation pipelines, the approach adopted here was data-driven and image specific. A reference set of Raspberry Pi images was captured under various field conditions direct sunlight, shadows, and occlusion. For each high-resolution image in the dataset, a visually similar reference image was identified based on two quantitative indicators:

- **Brightness**, measured via the Value (V) channel of the HSV color space,
- **Contrast**, estimated by the standard deviation of grayscale pixel intensities.

Each high-resolution image was then transformed to approximate the visual profile of its reference Raspberry Pi counterpart. This pairing strategy ensured that modifications were not arbitrary but informed by actual sensor output under real use conditions.

The preprocessing operations are summarized in Table 2.2, which details the transformations applied, their intended effects, the metrics guiding them, and the tools used for implementation.

Table 2.2: Summary of Preprocessing Operations Applied to Simulate Raspberry Pi Camera v2.1 Output

Transformation	**Purpose**	**Implementation Details**
Brightness Adjustment	Simulate variable lighting conditions observed in the field	HSV Value was computed and adjusted using ImageEnhance.Brightness based on matched Raspberry Pi frames
Contrast Normalization	Reflect reduced contrast typical of Raspberry Pi sensor output	Grayscale standard deviation was used as a contrast proxy; adjustments applied via ImageEnhance.Contrast

Sharpness and Blur	Mimic optical softness and lens imperfections	Laplacian variance measured sharpness; filters (OpenCV Gaussian blur, PIL sharpen) applied accordingly
File Management	Ensure traceability and reproducibility	Transformed images were renamed using a _rpi suffix and stored in separate versioned folders

This scientifically grounded preprocessing pipeline allowed the YOLOv8 model to learn not only from idealized images but also from input approximating its final deployment conditions. In doing so, the model becomes better equipped to handle environmental variability and sensor limitations inherent in real-world agricultural settings.

2.4.1. Preprocessing Pipeline: Methodology and Implementation

To operationalize the preprocessing strategy described earlier, a structured and reproducible pipeline was implemented using Python in a controlled Google Colab environment. The objective was to systematically transform high-resolution training images to resemble the visual profile of Raspberry Pi Camera v2.1 output. The pipeline was composed of three sequential stages: brightness adjustment, contrast normalization, and sharpness/blur correction, each informed by measurable characteristics extracted from real Raspberry Pi reference frames.

The rationale behind the ordering of these transformations reflects interdependencies between visual features. Specifically, brightness directly influences perceived contrast an overly bright or dark image compresses the intensity range, distorting subsequent contrast adjustment. Likewise, both lighting and contrast affect the detection and estimation of sharpness. Therefore, the sequence adopted (brightness → contrast → sharpness/blur) ensures visual stabilization before edge-based operations.

Figure 2.8 provides a schematic overview of the complete preprocessing workflow, from original high-resolution inputs to the final RPi-style outputs used for YOLOv8 training.

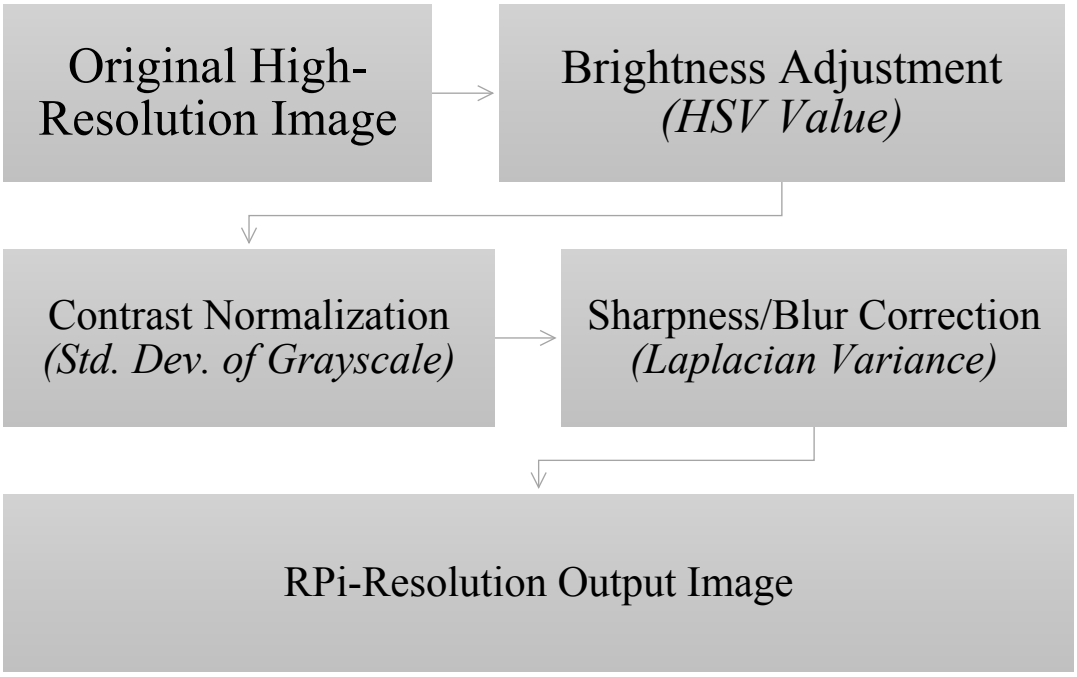


Figure 2.8: Structured preprocessing pipeline simulating Raspberry Pi Camera v2.1 output

The pipeline consists of sequential transformations applied in three stages: brightness correction, contrast normalization, and sharpness/blur adjustment. Each stage is guided by reference metrics and produces a versioned output.

a. Brightness Adjustment

➢ **Metric:**

Mean luminance was computed from the HSV Value channel to quantify image brightness.

➢ **Method:**

Each high-resolution image was paired with its closest Raspberry Pi reference based on luminance similarity. The ratio between target and source brightness guided the adjustment factor.

➢ **Implementation:**

```
enhancer = ImageEnhance.Brightness(image)
adjusted_image = enhancer.enhance(brightness_factor)
```

➢ **Tools:**

- OpenCV (cv2.cvtColor) for RGB to HSV conversion
- NumPy for calculating brightness metrics
- Pillow (PIL) for image adjustment (ImageEnhance.Brightness)

b. Contrast Normalization

- **Metric:**
  Contrast was evaluated as the standard deviation of grayscale pixel values, reflecting intensity dispersion.
- **Method :**
  Images were matched to Raspberry Pi references by contrast level. The adjustment factor was calculated as the ratio of target to original standard deviation.
- **Implementation:**

```
enhancer = ImageEnhance.Contrast(image)

adjusted_image = enhancer.enhance(contrast_factor)
```

- **Tools :**

- NumPy for contrast computation
- Pillow (PIL) for contrast enhancement(ImageEnhance.Contrast)

c. Sharpness and Blur Correction

➢ **Metric :**

• Sharpness was quantified using the Laplacian variance, which estimates the presence of edges and texture detail.

➢ **Method :**

- If Laplacian variance was lower than the reference, sharpening was applied to enhance edge clarity.
- If variance was higher, a light Gaussian blur was applied to replicate the slight softness typical of embedded optics.

➢ **Implementation:**

- **Sharpening** sharpened_image=image.filter(ImageFilter.SHARP)
- **Gaussian blur** blurred_image=cv2.GaussianBlur(image, (3,3),0)

➢ **Tools :**

- OpenCV for computing Laplacian variance and applying blur
- Pillow (PIL) for sharpening (ImageFilter.SHARPEN)

To ensure traceability and reproducibility, each preprocessing step produced a new dataset version stored in a dedicated folder. Images were renamed with a _rpi suffix, preserving compatibility with YOLO annotation files and allowing partial re-use of intermediate outputs. The following naming convention was adopted:

- brightness-adjusted/
- brightness-contrast/
- brightness-contrast-blur-sharp/

This folder hierarchy allowed modular validation of each transformation stage and supported controlled experimentation during model training. To illustrate the visual impact of each transformation, Figure 2.9 presents a step-by-step progression of a sample image across the pipeline.

Figure 2.9: Sequential preprocessing transformations applied to a sample image

(a) Original high-resolution image

(b) After brightness adjustment (global luminance corrected)

(c) After contrast normalization (enhanced object-background separation)

(d) After blur/sharpening correction (simulated embedded optics softness)

Each stage in Figure 2.9 reflects targeted improvements based on objective visual properties. Brightness normalization (b) reduced underexposure in shaded regions; contrast enhancement (c) sharpened the distinction between apricots and foliage without increasing noise; final sharpness correction (d) produced a more natural, field-like image texture consistent with Raspberry Pi optics.

This modular and scientifically grounded preprocessing pipeline ensured that the training data not only matched the operational camera conditions, but also preserved label accuracy and modelreadiness for real-world inference.

2.4.2. Folder Structuring and Image Renaming

To preserve the integrity of the original dataset and ensure rigorous traceability at every stage of the processing pipeline, all images resulting from the preprocessing operations were renamed systematically. The adopted convention was to retain each image's original filename and append the suffix (_rpi), indicating that the image had been visually adapted to simulate Raspberry Pi camera conditions. For example:

apricot-001.jpg → apricot-001_rpi.jpg

The transformed images were stored in separate, versioned folders corresponding to each preprocessing stage (e.g., brightness-adjusted/, brightness-contrast/, final-rpi-style/). This hierarchical structure offers several advantages:

- Faithful reproducibility of the processing pipeline,
- Simplified navigation for debugging or partial reuse,
- Clear separation between original data and preprocessed variants, in line with best practices in dataset management for computer vision applications [43].

2.4.3. Annotation Duplication for Preprocessed Images

Since the applied transformations (brightness, contrast, sharpness) did not alter the position or shape of the annotated objects, it was not necessary to reanalyze or manually relabel the modified images.

The bounding boxes and object classes (ripe / unripe apricots) remained valid after each preprocessing step.

To ensure full compatibility with the YOLOv8 format, the corresponding .txt annotation files were duplicated and renamed to match the transformed images. For example:

apricot-025.txt → apricot-025_rpi.txt

Each duplicated file retained the original YOLOv8 annotation structure (class ID, normalized center coordinates, width, and height). This approach ensured seamless integration into the training pipeline while maintaining spatial consistency and annotation accuracy [36].

2.4.4. Final Dataset Composition

To enhance the robustness and generalization capacity of the YOLOv8 model under variable visual conditions, the final dataset was constructed by merging the original high-resolution images with the Raspberry Pi–style preprocessed versions. This approach offers several benefits:

- Significant increase in data volume without introducing false positives,
- Diversification of lighting and texture conditions,
- Improved model resilience under real-world embedded inference scenarios.

Table 2.3 provides a detailed summary of the final dataset composition, outlining the number of images and associated annotations for both original and preprocessed versions

Table 2.3: Summary of Final Dataset Composition and Annotation Count

Image Type	**Number of Images**	**Associated Annotations**
Original high-resolution images	823	823 .txt files
Preprocessed images (_rpi)	823	823 .txt files
Total	**1,646**	**1,646 files**

This enriched dataset, balanced across the “ripe” and “unripe” classes, was used as-is for training the model in a Kaggle environment, following the input structure required by YOLOv8 [36].

The resulting hierarchical structure of the final dataset, prepared for YOLOv8 training, is visually represented in Figure 2.10.

```
dataset final/ ├── data.yaml
├── train/
│   ├── images/
```

Figure 2.10: Final Dataset Directory Structure.

This figure illustrates the final organization of the training data folder, detailing the hierarchical arrangement, sample file naming conventions indicating transformation stages with '_rpi', and the integration of images with their corresponding annotation files.

2.5. YOLOv8 Model Training

Following the construction of a robust and augmented dataset (Section 2.3), this phase focused on training several YOLOv8 model variants to classify apricots into 'ripe' and 'unripe' categories. The training was carried out in a controlled environment, with a consistent experimental setup designed to explore the trade-offs between model accuracy, inference speed, and embedded deployment constraints. This section presents the architecture selection rationale, the training configurations, and the technical considerations guiding these choices.

2.5.1. Model Configuration and Training Parameters

a. Selection of YOLOv8 Architectures

Six YOLOv8 variants were selected for training: YOLOv8n, YOLOv8s, and YOLOv8m, each trained for two durations (50 and 100 epochs). This comparative approach was designed to evaluate the trade-off between detection capacity, inference speed, and suitability for embedded deployment.

Heavier models such as YOLOv8l and YOLOv8x were intentionally excluded for two main reasons:

- **Hardware constraints**: Larger architectures require significantly more memory and computational power, which limits their applicability for deployment on resourceconstrained

devices like the Raspberry Pi 5. Recent work in embedded AI recommends lightweight models for low-power environments [31].

- **Dataset size**: With a modest training set of 1,646 annotated images, deeper architectures are at higher risk of overfitting, without guaranteed gains in detection accuracy [43].

The selected variants represent a meaningful architectural spectrum.

Table 2.4 provides a concise overview of the YOLOv8 model variants chosen for training, detailing their technical descriptions and target usage scenarios.

Table 2.4: Overview of YOLOv8 Model Variants Selected for Training

Model	**Technical Description**	**Target Usage**
YOLOv8n	Ultra-lightweight, fast, designed for edge devices	Real-time detection on Raspberry Pi
YOLOv8s	Balanced compromise between accuracy and speed	Near real-time deployment
YOLOv8m	Higher capacity, trained for comparative benchmarking	GPU reference for accuracy comparison

Each architecture was trained for both 50 and 100 epochs to assess how extended training impacts accuracy, convergence, and the risk of overfitting.

b. Common Training Parameters

To ensure consistent comparisons, all models were trained on the final merged dataset (original plus Raspberry Pi–style images), organized in the standard YOLOv8 format (train/, valid/, test/) and referenced by a data.yaml configuration file.

The following parameters were applied uniformly across all training sessions:

- **Input resolution**: 640 × 640 pixels

This is the default size recommended for YOLOv8 [36]. It provides a good compromise between detection precision and computational efficiency. Higher resolutions could improve small object detection but would increase memory usage and training time.

- **Batch size**: 16

This value ensures stable gradient updates while remaining within the memory limits of the Kaggle GPU environment. It offers an effective balance for medium-sized datasets without exhausting available VRAM.

- **Number of epochs**: 50 and 100
  An epoch corresponds to a full pass through the training set. The dual-duration strategy enables observation of performance evolution while mitigating overfitting risks, which are common with small datasets [46].

- **Optimizer**: Stochastic Gradient Descent (SGD)
  The default optimizer in YOLOv8, SGD is widely used in object detection tasks for its stability and convergence properties.

- **Learning rate**: Automatically managed by YOLOv8
  Adaptive learning rate control helps ensure robust convergence without the need for manual tuning.

- **Loss functions**:

- **CIoU Loss (Complete Intersection over Union)**
  Applied to bounding box regression, CIoU considers overlap area, center distance, and aspect ratio alignment. It improves object localization compared to simple IoUbased loss functions.

- **Binary Cross-Entropy (BCE) Loss**
  Used for classification, BCE is appropriate for binary tasks such as distinguishing ripe vs. unripe apricots. It penalizes incorrect or low-confidence predictions.

These standardized parameters provide a stable experimental baseline, limiting the variables to model architecture and training duration. This strategy enables objective comparison of YOLOv8 variants under realistic conditions while taking into account the practical constraints of precision agriculture and embedded hardware deployment.

2.5.2. Training Environment and Execution Setup

All training sessions were conducted in a cloud-based environment using Kaggle Notebooks equipped with NVIDIA Tesla T4 GPUs. This configuration ensured adequate computational resources for training all six YOLOv8 variants under consistent conditions. The selected platform also facilitated reproducibility, version control, and efficient file management.

The table 2.5 summarizes the training infrastructure and workflow adopted throughout the model development phase:

Table 2.5: Training Environment and Execution Configuration

Component	**Description**
Platform	Kaggle Notebooks (cloud-based, GPU-enabled)
GPU	NVIDIA Tesla T4 (16 GB VRAM)
Python Environment	Python 3.11, preconfigured with Ultralytics, OpenCV, NumPy
YOLO Library	ultralytics (official YOLOv8 implementation)
Notebook Structure	One notebook per model variant (YOLOv8n/s/m – 50 & 100 epochs)
Dataset Format	YOLOv8-compatible structure (train/, valid/, test/, with data.yaml)
Training Command Format	model.train(data='data.yaml', epochs=XX, imgsz=640, batch=16, name='…')
Training Duration (50 epochs)	Approximately 20–30 minutes depending on model size
Output Artifacts	best.pt, results.png, confusion_matrix.png, metric curves, logs
Checkpointing	Automatically handled by Ultralytics API

This standardized environment enabled direct comparison across different model architectures and epoch durations, with no variability in computing conditions or dataset paths.

2.5.3. Generated Artifacts and Training Metrics

Each training session produced a comprehensive set of evaluation and configuration files, stored under the directory runs/train/. These outputs were essential for monitoring learning behavior, detecting anomalies (such as overfitting or class imbalance), and selecting the most performant model for downstream inference and deployment.

Table 2.6 provides a comprehensive overview of the various output files and folders generated by the YOLOv8 training sessions, detailing their contents.

Table 2.6: Output Files Generated by YOLOv8 Training Sessions

File / Folder	**Description**
best.pt	Final weights from the epoch with highest validation performance
results.png	Summary plot of loss curves, mAP, precision, and recall over all epochs
F1_curve.png	F1-score plotted against confidence thresholds
P_curve.png	Precision vs. confidence curve
R_curve.png	Recall vs. confidence curve
PR_curve.png	Precision–Recall curve across all thresholds
confusion_matrix.png	Matrix showing correct and incorrect class predictions
results.csv	Tabular logs of all epoch-wise metrics (precision, recall, mAP, losses)
opt.yaml, args.yaml	Configuration files for experiment reproducibility

These outputs supported several critical analysis steps:

- **results.png** allowed visual inspection of performance progression over time, enabling early identification of saturation or instability.
- **confusion_matrix.png** was used to diagnose class confusion, especially between visually similar classes like 'ripe' and 'unripe'.
- **results.csv** provided numerical insight into precision, recall, and mean Average Precision (mAP) at multiple thresholds (0.5 and 0.5:0.95), supporting objective comparisons across models.

2.5.4. Interpretation of Epoch-Wise Training Curves

Figure 2.11 presents the evolution of key training and validation metrics across epochs, as generated by YOLOv8 in the results.png file. This composite figure provides critical insight into the model's learning dynamics over time, highlighting trends in convergence, generalization, and class discrimination performance. The top panels display training losses and performance metrics, while the bottom panels show corresponding validation metrics.

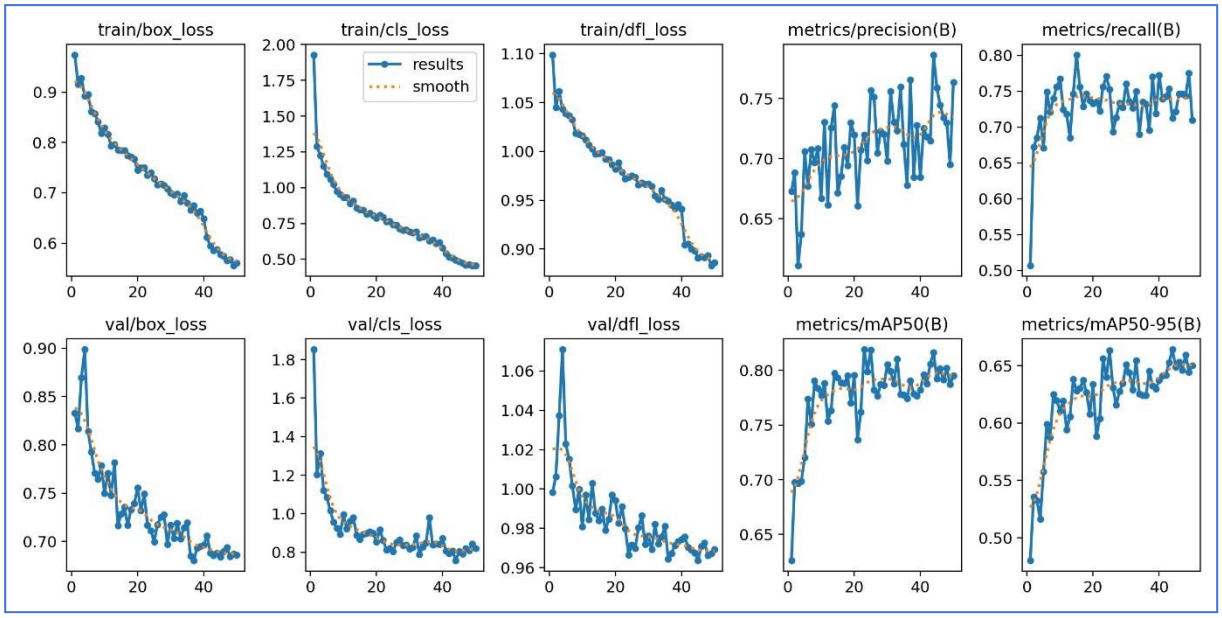


Figure 2.11: Training Performance Curves Generated by YOLOv8 (results.png)

Each curve serves a specific diagnostic function:

- **Box Loss (box_loss)**:

  Represents the localization error in bounding box regression. A consistent downward trend across both training and validation curves indicates effective spatial learning and convergence. If the validation curve diverges from the training loss, it may signal overfitting or poor generalization.

- **Class Loss (cls_loss)**:

  Measures the classification error in predicting the correct class label (ripe or unripe). A decreasing curve reflects improved class discrimination. Close alignment between training and validation curves suggests stable learning behavior.

- **Distribution Focal Loss (dfl_loss)**:

  Used for refining bounding box precision, especially regarding anchor-free center and width predictions. Lower DFL values correlate with improved spatial accuracy and model confidence.

- **Precision (per epoch)**:

  Indicates the proportion of correct detections among all predicted bounding boxes. An upward trend shows that the model becomes more selective and reduces false positives, which is critical for embedded deployment where misclassifications can trigger incorrect robotic actions.

- **Recall (per epoch)**:

  Reflects the model's ability to detect relevant ground truth objects. Rising recall indicates increasing sensitivity to true apricots in the dataset, minimizing missed detections important for tasks such as maturity monitoring and thinning.

- **mAP50 and mAP50–95**:

  These are aggregate indicators of detection quality across classes and thresholds.

- **mAP@0.5** evaluates detection performance at a moderate IoU threshold (50% overlap) and is commonly used for deployment decisions.

- **mAP@0.5:0.95** averages performance across ten stricter IoU thresholds (0.50 to 0.95), offering a more rigorous view of overall detection precision and spatial accuracy.

  A stable upward progression in both curves confirms that the model improves in terms of localization and classification over time.

  The orange dotted lines represent smoothed versions of each curve, allowing clearer visualization of global trends by reducing noise due to individual batch variations.

  These training curves play a decisive role in:

- Diagnosing training quality (e.g., underfitting, overfitting, instability);
- Selecting the optimal epoch for model export (best.pt);
- Justifying confidence in the model's reliability for real-time use in agricultural environments.

2.5.5. Interpretation of Post-Training Metric Curves

The set of post-training metric curves provides a multi-dimensional diagnostic view of model behavior across varying confidence thresholds. Each curve highlights a specific performance indicator F1-score, precision, recall, or the precision–recall relationship and helps assess how well the model balances false positives and false negatives. This information is crucial not only for selecting the most robust version of the model (best.pt), but also for determining a confidence threshold suited to real-world deployment conditions, particularly in agricultural contexts involving maturity classification.

The following figures (Figure 2.12 through Figure 2.15) visually represent these key post-training metric curves, providing a comprehensive diagnostic view of the YOLOv8 model's performance.

a. F1-Curve: F1-Score vs. Confidence Threshold

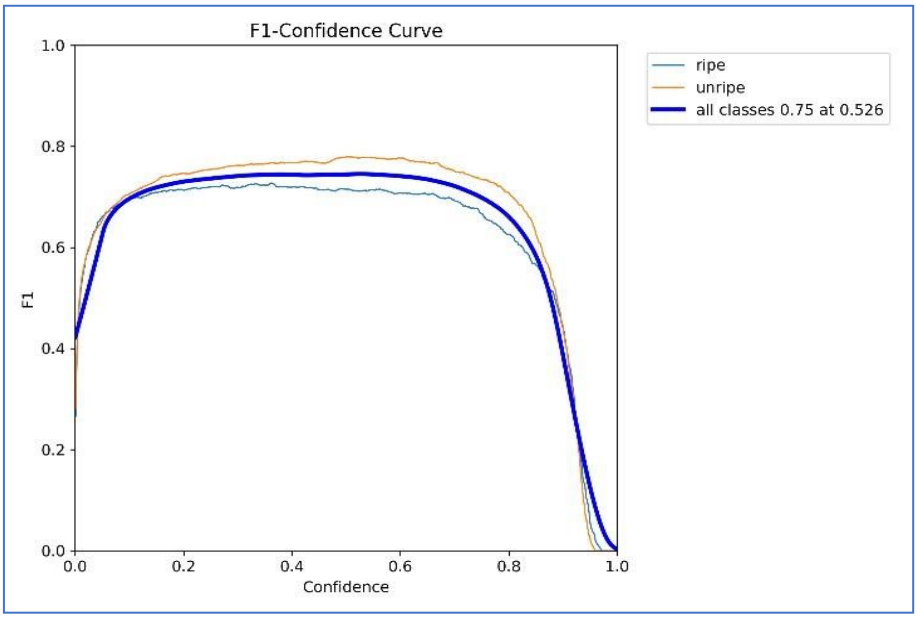


Figure 2.12: YOLOv8 Training Curves: F1-score vs. Confidence Threshold

The F1-curve plots the harmonic mean of precision and recall as the confidence threshold varies from 0 to 1. It measures the model’s ability to maintain a balanced trade-off between sensitivity (recall) and specificity (precision).

- A broad and elevated peak indicates stable performance across multiple thresholds, signaling robustness in practical deployment.
- A sharp or unstable peak suggests sensitivity to threshold tuning, which may reduce consistency in real-time applications.

- The optimal threshold for deployment often corresponds to the maximum point of the F1-curve, where the model best balances false positives and false negatives.

  In fruit maturity detection, this balance is vital: misclassifying an unripe apricot as ripe (or vice versa) may lead to unnecessary interventions or missed harvesting opportunities.

b. PR-Curve: Precision vs. Recall

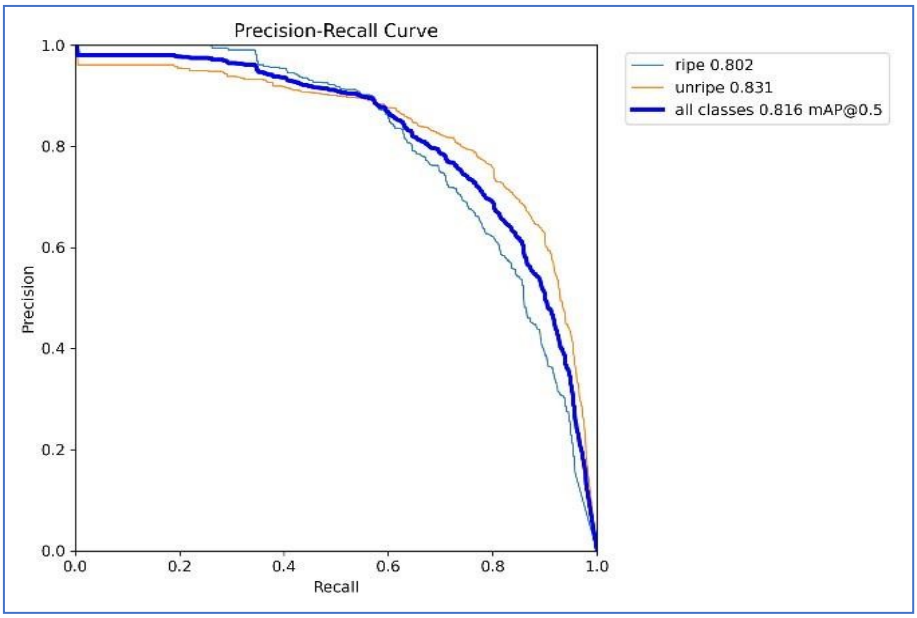


Figure 2.13: YOLOv8 Training Curves: Precision–Recall Curve

The precision–recall (PR) curve provides a threshold-independent view of how precision evolves as recall increases. It is particularly useful when evaluating models on imbalanced datasets or tasks with significant consequences for misclassification.

- A PR-curve that remains close to the top-right corner reflects a strong and consistent model: most target fruits are detected accurately and with minimal error.
- The area under the curve (AUC) serves as a compact summary of global detection quality. AUC values closer to 1 indicate excellent precision–recall trade-offs.

  For maturity detection, this curve helps ensure that improving recall (e.g., detecting more fruits) does not come at the cost of an unacceptable rise in false positives

c. P-Curve: Precision vs. Confidence Threshold

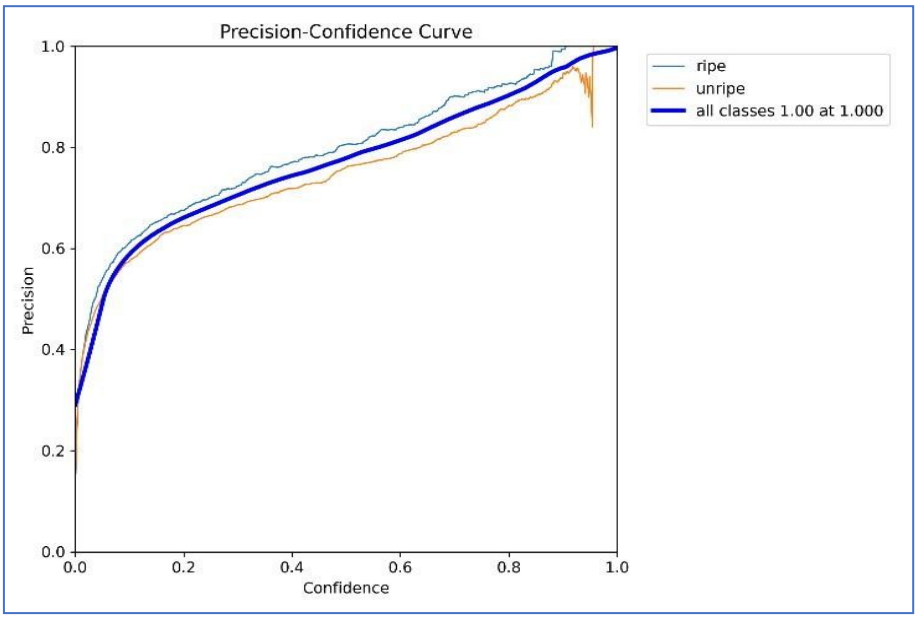


Figure 2.14: YOLOv8 Training Curves: Precision vs. Confidence Threshold

The P-curve illustrates how the model's precision varies as the confidence threshold increases. It reflects the reliability of the confidence scores produced by the model.

- A flat and high curve suggests the model remains precise even when only highconfidence detections are retained.
- A steep decline at higher thresholds indicates overconfidence in incorrect predictions, which may result in unnecessary rejections or misinterpretations.

In embedded systems, such as Raspberry Pi–based deployments, where decisions may be automated, confidence reliability becomes a critical factor. The P-curve thus supports the calibration of filtering mechanisms based on confidence.

d. R-Curve Recall vs. Confidence Threshold

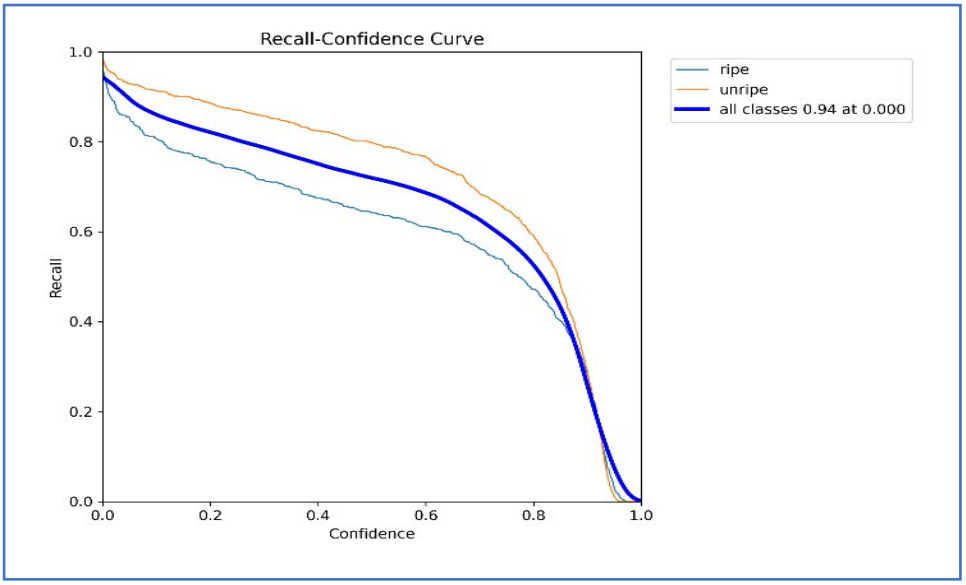


Figure 2.15: YOLOv8 Training Curves: Recall vs. Confidence Threshold

The R-curve shows how recall decreases as the confidence threshold increases. This curve directly reveals how many correct detections are lost when low-confidence predictions are filtered out.

- A **gradual slope** indicates a tolerant model that maintains detection capability even at stricter confidence levels.
- A **sudden drop** suggests that the model may be too conservative, potentially missing relevant detections—an issue in use cases requiring comprehensive fruit monitoring.

In the context of maturity detection, especially when both ripe and unripe fruits must be considered for thinning or forecasting, maintaining recall is essential to avoid blind spots.

e. Diagnostic Role in Selecting the Final Model (best.pt)

Although YOLOv8 automatically selects the best.pt model based on the highest validation mAP@0.5, the metric curves provide critical complementary information for validating that choice and fine-tuning deployment parameters.

- The F1-curve pinpoints the optimal balance point between recall and precision.
- The PR-curve evaluates whether this balance is maintained across the full spectrum of recall.
- The P- and R-curves help assess the model's sensitivity to threshold changes and its confidence calibration.

A well-performing best.pt model suitable for embedded, real-time deployment is typically characterized by:

- A high and well-centered F1-curve peak;
- A PR-curve with a large area under the curve
- And stable P- and R-curves indicating reliable confidence estimation and detection coverage.

These elements ensure that the selected model performs consistently in variable conditions both visually and operationally making it suitable for maturity detection tasks in real agricultural environments.

2.6. Evaluation of the best.pt Model on the Independent Test Set

At the end of the training phase, the YOLOv8 framework automatically selects the model that achieved the highest mean Average Precision at an IoU threshold of 0.5 (mAP@0.5) on the validation set. This model is exported as the best.pt file. However, this internal selection while indicative does not guarantee the model's actual ability to generalize to unseen data.

To obtain an impartial assessment of the model's robustness, an independent evaluation was conducted on the test set, a subset of annotated images excluded from both training and validation (see Section 2.3.). This protocol follows established best practices in machine learning to prevent performance overestimation [53].

2.6.1. Global Metrics and Diagnostic Curves

The evaluation was performed using the model.val() function from the Ultralytics YOLOv8 API, applied to the standardized structure of the test folder. This process automatically generates a complete set of results both quantitative metrics and performance curves which are stored in the runs/val/ directory.

The main global metrics calculated on the full test set are as follows:

- **Precision**
- **Recall**
- **mAP@0.5**
- **mAP@0.5:0.95**

These values are also illustrated in Section 2.5.4.

In addition to these scalar indicators, four diagnostic curves illustrated in Section 2.5.5.were generated to visualize the model's behavior across varying confidence thresholds:

- **P-curve**
- **R-curve**
- **F1-curve**
- **PR-curve**

2.6.2. Confusion Matrix Analysis

In addition to the diagnostic curves, a confusion matrix was generated to assess class-wise performance. For each class, it indicates the number of true positives (TP), false positives (FP), and false negatives (FN). This representation allows for the identification of systematic errors, such as frequent misclassifications between the "ripe" and "unripe" classes.

The normalized confusion matrix, shown in Figure 2.16, expresses these results as percentages, facilitating comparison between classes despite potential imbalance. Diagonal values represent correct classifications, while off-diagonal elements reflect confusions.

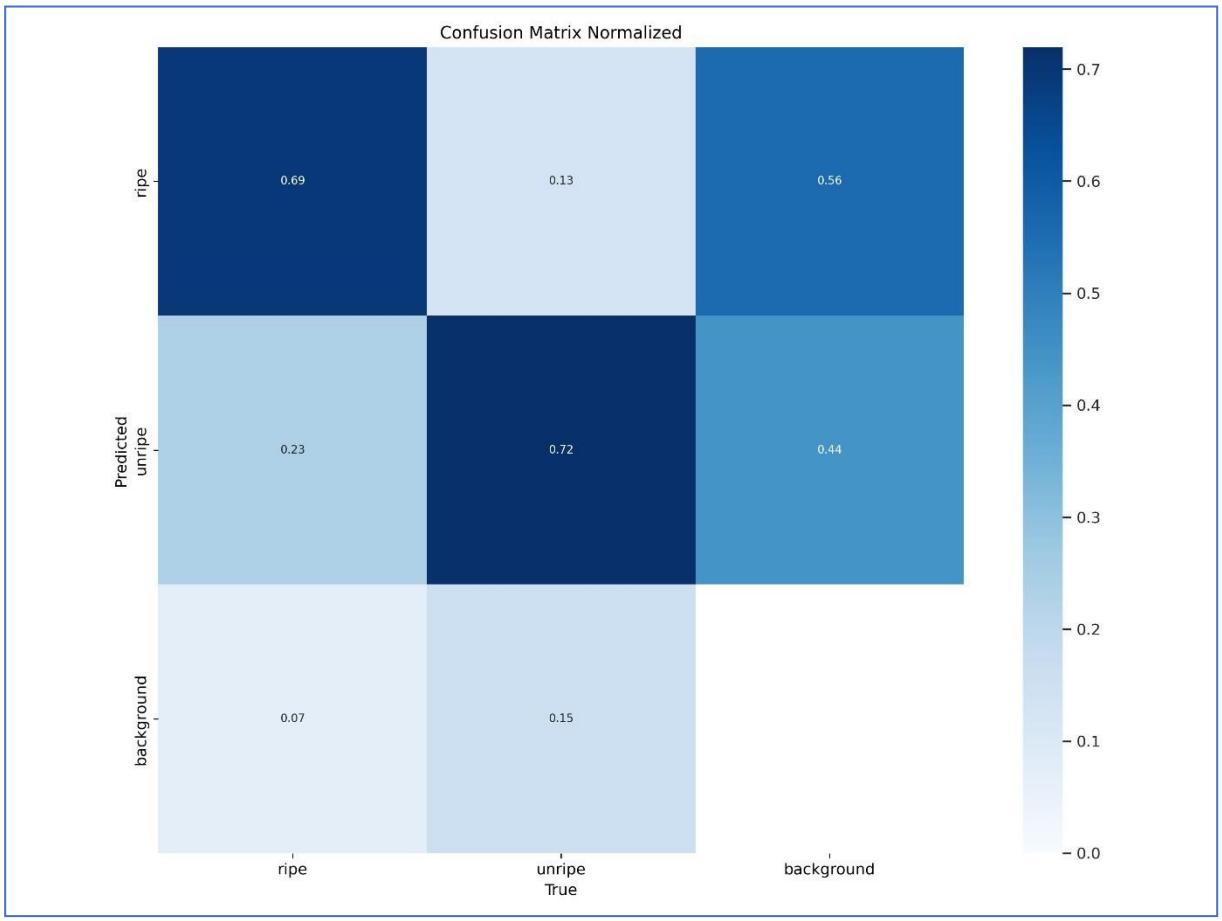


Figure 2.16: Normalized Confusion Matrix for Test Set Evaluation

Each cell expresses the proportion of predictions for a given class. Class confusions are visually easy to detect.

This matrix provides a fine-grained analysis of the model's sensitivity to each target class and will be used in Chapter 3 to explain the performance differences observed across tested variants.

2.6.3. Qualitative Test Batch Visualizations

Finally, the evaluation process generates visualizations of predictions on the test images, with bounding boxes overlaid (val_batch*.jpg). These images provide visual insight into:

- The accuracy of object localization
- The correctness of class labels
- The presence of duplicates (false positives) or missing detections (false negatives)

These qualitative results are illustrated in Figure 2.17. They are especially useful for assessing the model's robustness under varied lighting, complex backgrounds, or partial fruit visibility—common challenges in real-world agricultural settings.

Figure 2.17: Qualitative Visualization of Predictions on the Test Set

This is an Examples of images automatically annotated by the model across different batches (val_batch), which are particularly useful for visually assessing the accuracy of object localization, the correctness of class labels, and for identifying visible errors or edge cases like false positives and missing detections.

The evaluation of the best.pt model on the independent test set provides a comprehensive diagnostic of its generalization capacity. The combination of global metrics, diagnostic curves, confusion matrix, and visual results enables a robust validation of model behavior before any embedded deployment. These findings will be integrated into the detailed comparative analysis in Chapter 3, which aims to identify the most reliable and deployment-ready YOLOv8 variant for maturity detection in real orchard conditions.

2.6.4. Offline Inference on a YouTube Video (Kaggle)

To emulate a realistic use case, a representative video showing apricots under natural conditions was downloaded from YouTube and processed in an offline environment using Kaggle notebooks. The inference script was implemented using the official Ultralytics YOLOv8 API, and two model variants were tested:

- **YOLOv8n-100**: selected for its lightweight architecture suitable for embedded use;
- **YOLOv8m-50**: used as a higher-capacity benchmark to assess potential gains in detection accuracy.

During execution, each video frame was processed individually. The following elements were displayed or recorded:

- Bounding boxes with class label ('ripe') and confidence score,
- Real-time Frames Per Second (FPS) to measure inference speed,
- Annotated output video saved for further qualitative inspection.

This experiment provided a visual benchmark for detection quality and temporal consistency, helping to identify potential issues such as frame-to-frame instability, misdetections under motion blur, or inconsistent classification.

Figure 2.18 illustrates the comprehensive workflow for this offline video inference experiment, summarizing the experimental pipeline for testing YOLOv8 models on pre-recorded apricot videos.

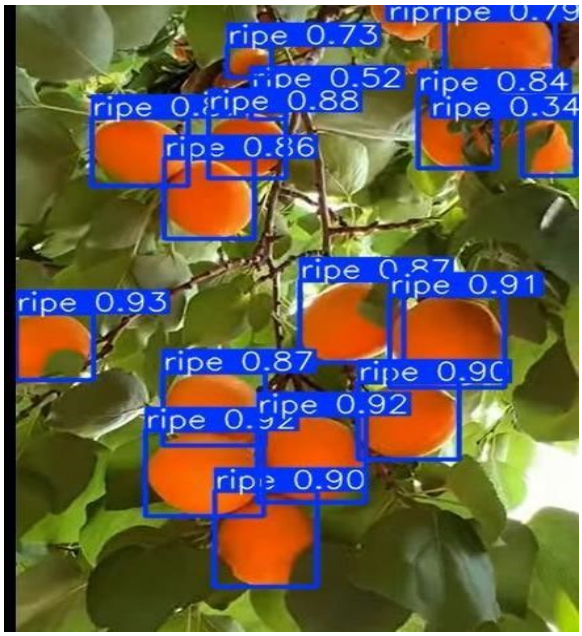


Figure 2.18: Workflow for Offline Video Inference with yolov8m50 (Kaggle)

Summary of the experimental pipeline for testing YOLOv8 models on pre-recorded apricot videos.

2.6.5. Real-Time Inference via Webcam (Local Development Environment)

To validate practical deployment on low-cost hardware, the best.pt model was deployed locally on a consumer PC using Visual Studio Code, a 720p webcam, and the YOLOv8 API. This setup approximates the visual input conditions of embedded vision systems such as the Raspberry Pi Camera Module v2.1.

The real-time test used six ripe apricots placed at varying distances, angles, and lighting conditions to replicate diverse field scenarios. For consistency with the intended embedded deployment, only the YOLOv8n-100 model was tested in this configuration, chosen for its optimal trade-off between accuracy and execution speed.

Due to practical limitations (e.g., lack of access to mixed maturity samples during the test), only the 'ripe' class was retained during inference. The 'unripe' class was filtered out using a postinference script. This constraint was accepted as a temporary adaptation for controlled testing while preserving the relevance of maturity detection validation.

The interface displayed:

- Annotated bounding boxes with predicted class and confidence score,
- A live FPS counter indicating model responsiveness,
- Terminal logs with bounding box coordinates, dimensions, aspect ratios, and confidence levels.

Figure 2.19 visually illustrates this real-time inference setup, showing the testing configuration, the model used, and the outputs monitored during the webcam validation.

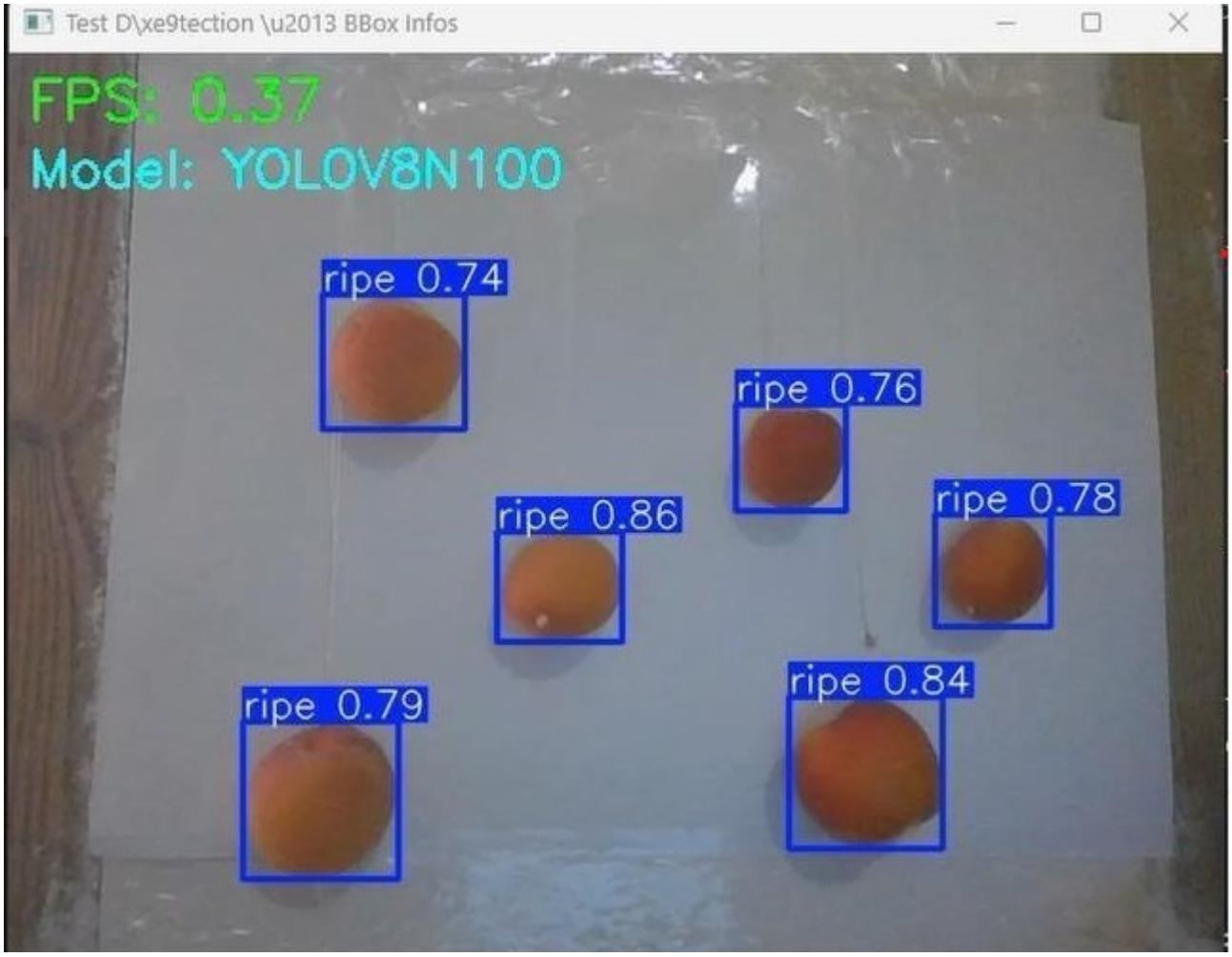


Figure 2.19: Real-Time Inference Setup with Local Webcam

These outputs enabled a functional and visual assessment of the model's behavior in dynamic, uncontrolled visual contexts. They also provided preliminary validation of the system's ability to operate on edge devices where power, latency, and accuracy must be balanced.

In summary These two experiments served complementary purposes:

- The offline video test evaluated model robustness on longer video sequences under natural but reproducible conditions.
- The real-time webcam test simulated the embedded execution pipeline and confirmed the model's ability to handle real-time inference, providing critical insight for future deployment on Raspberry Pi–based hardware.

The results from both tests were saved as annotated videos and inference logs, and will be used in Chapter 3 to support the final evaluation of the selected model's suitability for embedded maturity detection in orchard environments.

2.7. Embedded Deployment on Raspberry Pi 5 (8GB)

To bridge the gap between model training and real-world usage, the trained YOLOv8n-100 model was deployed on a Raspberry Pi 5 (8GB) using its native camera interface. This step directly addresses the agronomic challenges outlined in Chapter 1, which emphasized the need for lightweight, low-cost, and autonomous systems capable of operating under the constraints of Algerian orchard environments. The Raspberry Pi platform was selected for its affordability, community support, and proven suitability for embedded vision tasks.

This deployment validates the model's compatibility with real-time maturity detection objectives such as assisted harvesting, yield estimation, and Thinning without requiring expensive or resource-intensive infrastructure.

2.7.1. System Preparation

The deployment environment was built on a Raspberry Pi 5 running Raspberry Pi OS 64-bit (Bookworm) with Python 3.11. All tests were conducted with the Raspberry Pi Camera Module V2.1, interfaced via the Picamera2 library.

Table 2.7 outlines the essential tools and packages installed, along with their purpose, for setting up the Raspberry Pi 5 environment for YOLOv8 deployment.

Table 2.7: Raspberry Pi 5 Environment Setup for YOLOv8 Deployment

Step	**Tool/Package**	**Purpose**
1	pip install ultralytics	For YOLOv8 model loading and inference
2	sudo apt install libcamera-dev	Required for Picamera2
3	sudo apt install python3-picamera2	Python interface for Raspberry Pi Camera V2
4	pip install opencv-python	Real-time video processing
5	Thonny IDE	Used for script editing and execution

Each step ensured that the camera feed, model, and postprocessing logic could run in real time with minimal latency.

2.7.2. Model Transfer and Integration

The trained best.pt model was transferred from the development environment to the Raspberry Pi using a USB drive. It was stored under /home/pi/yolo/ for clarity and maintainability. Organizing the project directory by separating models, scripts, and outputs helped streamline future debugging and iteration.

2.7.3. Real-Time Inference Script (Thonny IDE)

The real-time detection script was developed and executed within the Thonny Python IDE. It integrates Picamera2 for live image capture, OpenCV for annotation, and Ultralytics YOLO for inference. The key components of the script include:

- Initialization of the Picamera2 feed (1280×720)
- Loading the best.pt model
- Capturing each frame and passing it to the model
- Overlaying class labels, confidence, and FPS on the video stream
- Logging inference speed and bounding box data to the terminal
- Saving output videos using OpenCV's Video Writer

This setup provides a simple but powerful prototype for real-time maturity detection in embedded

Figure 2.20 schematically represents this complete on-device detection workflow, illustrating the real-time embedded inference pipeline from image acquisition to detection and visualization on the Raspberry Pi 5.

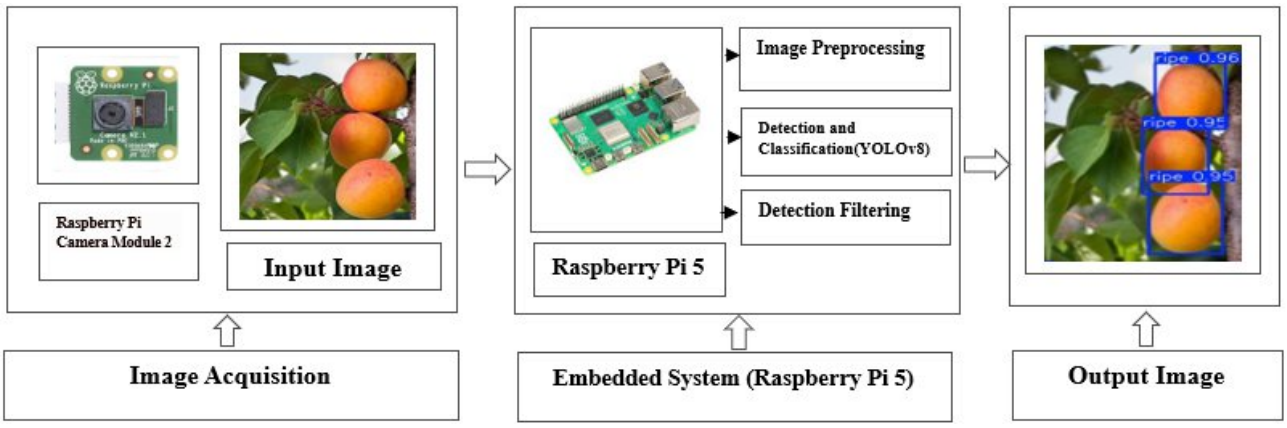


Figure 2.20: Real-Time Embedded Inference Pipeline on Raspberry Pi 5

Schematic representation of the complete on-device detection workflow, from image acquisition using the Pi Camera to inference, optional filtering, and visualization of detected apricots.

2.7.4. Outputs and Diagnostics

During execution, the inference system displayed bounding boxes labeled with the predicted class (ripe or unripe) and confidence scores. An FPS counter was rendered directly on the video stream. Meanwhile, terminal logs showed detailed information on per-frame timing: preprocessing, inference, and postprocessing durations.

These diagnostic outputs are essential for assessing system viability in embedded deployment, especially where speed and stability are critical.

Figure 2.21 provides a visual representation of these real-time inference outputs on the Raspberry Pi 5, showing the detection of apricots, along with displayed bounding boxes, confidence scores and FPS

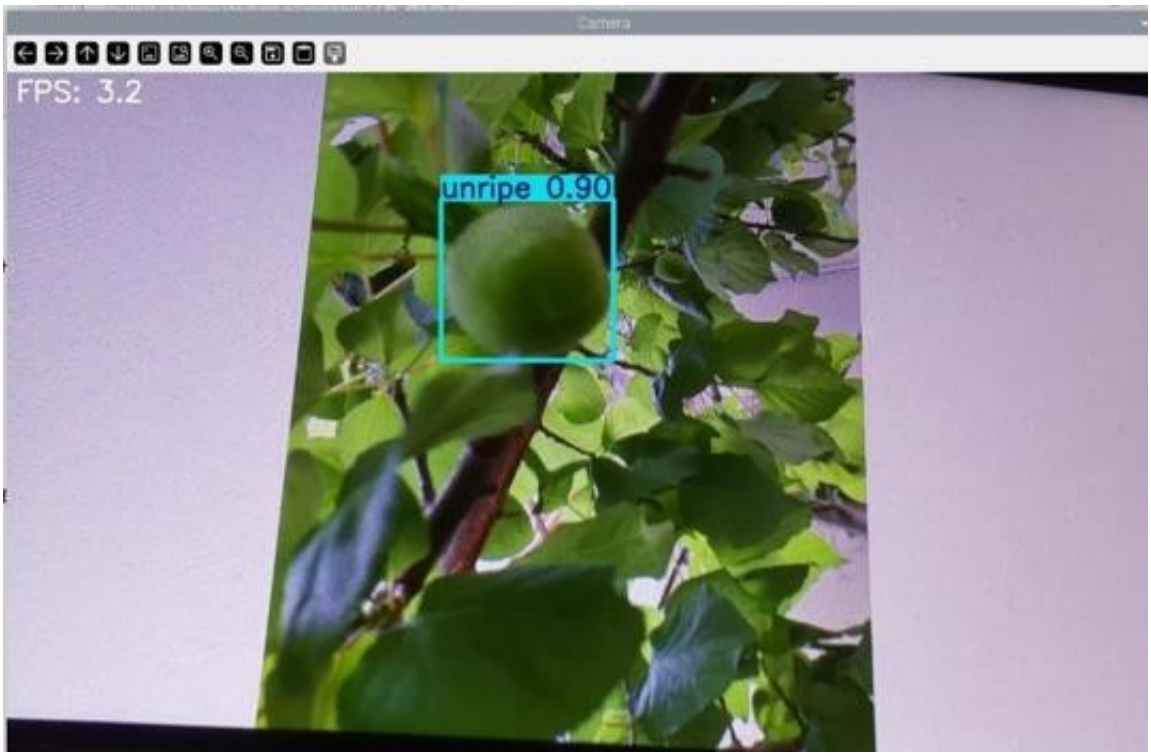


Figure 2.21: Real-Time Inference on Raspberry Pi 5 using Thonny IDE

Visual output of YOLOv8n-100 model detecting ripe and unripe apricots in real time. The script was executed in Thonny with Picamera2 input. Bounding boxes, confidence scores, and FPS are displayed.

2.7.5. Practical Constraints and Future Directions

Due to practical limitations, the real-time tests focused on detecting only one maturity class at a time. The ripe class was tested using fresh apricots under natural lighting, while the unripe class was tested separately using images captured in orchard-like scenes. This separation reflects constraints in available fruit samples not a limitation of the model or platform.

In Chapter 3, we will see the results of these tests after implementing class-based filters and counters. These additions will demonstrate how the deployed model can support all five practical objectives introduced in Section 1.3.3:

- **Assisted harvesting**, by isolating 'ripe' detections for robotic actuation
- **Yield estimation**, by counting detected fruits by class
- **Thinning (Éclaircissage)**, by filtering out unripe detections
- **Maturity monitoring**, by tracking changes in class distribution over time
- **Logistics planning**, through maturity-aware spatial data

The Raspberry Pi deployment thus acts as a functional foundation for future integration with robotic hardware, mobile systems, or smart farming platforms.

2.8. Conclusion

This chapter has detailed the complete technical pipeline developed for intelligent apricot maturity detection, from initial data acquisition and meticulous annotation to model training, evaluation, and embedded deployment. Each methodological step was designed to address the agronomic and operational challenges identified in Chapter 1, ensuring that the resulting system is not only scientifically robust but also adapted to the practical constraints of Algerian orchards.

Through systematic data preprocessing, careful model selection, and rigorous validation, the YOLOv8-based approach demonstrated compatibility with lightweight embedded hardware such as the Raspberry Pi 5. The deployment phase confirmed the feasibility of real-time maturity detection under realistic field conditions, providing a foundation for autonomous, lowcost decision support in fruit production.

The chapter also highlighted the importance of diagnostic outputs and iterative refinement, paving the way for further integration with agronomic workflows such as assisted harvesting,

yield estimation, and maturity monitoring. While certain practical constraints were encountered such as limited access to diverse fruit samples the modularity of the pipeline ensures that future adaptations and scaling are straightforward.

In summary, the technical developments presented in this chapter establish a robust and reproducible framework for intelligent fruit maturity assessment. The next chapter will present a comprehensive evaluation of the system's performance, benchmarking its accuracy, speed, and operational reliability in both controlled and real-world scenarios.

CHAPITRE 3 : Results and discussion

3.1. INTRODUCTION

This chapter is dedicated to the presentation and analysis of the results obtained during the training and testing phases, as well as the practical implementation of the final detection model. It builds upon the methodology defined previously to carry out a structured evaluation of six YOLOv8 variants, differing by architecture size and number of training epochs.

The objective is twofold: first, to understand how training configurations impact detection performance; second, to identify the model that offers the best compromise between accuracy and computational efficiency for embedded deployment. The analysis is conducted in progressive stages: from training behavior interpretation, to test set evaluation, to qualitative and quantitative comparisons. The chapter concludes with the implementation of the selected model on a Raspberry Pi 5 for real-time fruit detection.

3.2. Motivation for Multi-Stage Training

Selecting an effective object detection model involves more than choosing an architecture it requires understanding how model complexity and training duration influence learning behavior, convergence, and generalization. In applied contexts like real-time fruit detection,

these decisions are even more critical due to constraints on computation, latency, and data volume.

This section explores the rationale behind varying both the size and training length of YOLOv8 models. The goal is not only to improve raw accuracy, but to align model behavior with the practical demands of real-world deployment. Each training configuration was guided by early indicators in the learning process, ensuring that experimentation remained grounded in observable performance patterns.

3.2.1. Initial Model Selection: Starting with YOLOv8n-50

The training process began with the YOLOv8n model configured for 50 epochs. This architecture was selected as a starting point due to its minimal computational footprint, making it particularly suited for embedded environments such as the Raspberry Pi 5. Its lightweight nature also aligned with the modest size of the dataset used in this project, reducing the risk of overfitting during initial experimentation.

From the outset, this baseline served not only to establish a performance reference, but also to observe the model's learning dynamics under conservative conditions. The choice of 50 epochs was intended as an initial benchmark to assess whether the model could reach acceptable convergence within a limited number of iterations.

To assess whether the model had reached convergence by epoch 50, we analyzed the training curves automatically generated by YOLOv8, as illustrated in Figure 3.1.

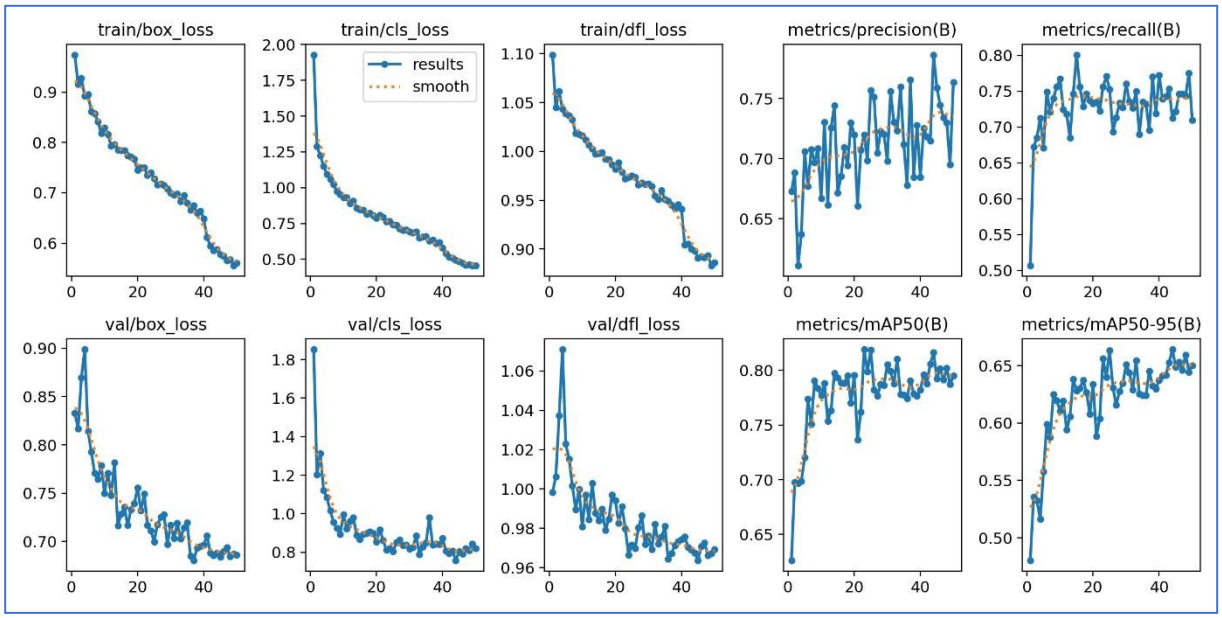


Figure 3.1: training curves of model YOLOv8n avec 50 epoches

The training curves generated for YOLOv8n over 50 epochs show consistent improvement across all major metrics, both for the training and validation phases. The box loss, class loss, and DFL loss display a steady downward trend, indicating that the model continuously refined its object localization and classification capabilities throughout training.

In the upper row, we observe a smooth decline in the training losses (train/box_loss, train/cls_loss, train/dfl_loss), suggesting that the model learned to minimize errors effectively on the training set. The corresponding validation losses (val/box_loss, val/cls_loss, val/dfl_loss) show a similar pattern with no sharp divergence, which indicates that overfitting did not occur during this phase.

The precision and recall metrics, although subject to some fluctuation, generally trend upward. Most notably, both mAP@0.5 and mAP@0.5:0.95 continue to increase until the final epochs, with no indication of saturation or plateauing. This behavior suggests that the model had not yet fully converged by epoch 50, and that further training was likely to result in additional performance gains.

In summary, the visual patterns across all metrics point to a model that was still actively learning and generalizing at the 50th epoch. This justified the decision to extend the training to 100 epochs and to apply the same protocol to other model sizes for consistent evaluation.

3.2.2. Extending Training Duration: Need for 100 Epochs

The initial 50-epoch training of YOLOv8n showed promising trends, but key performance indicators had not yet stabilized. To verify whether the model would continue to improve, the training was extended to 100 epochs. This adjustment aimed to observe the effects of prolonged optimization on detection performance and learning dynamics.
Figure 3.2 visually presents the training curves for the YOLOv8n model after 100 epochs, demonstrating the continued learning and performance improvements.

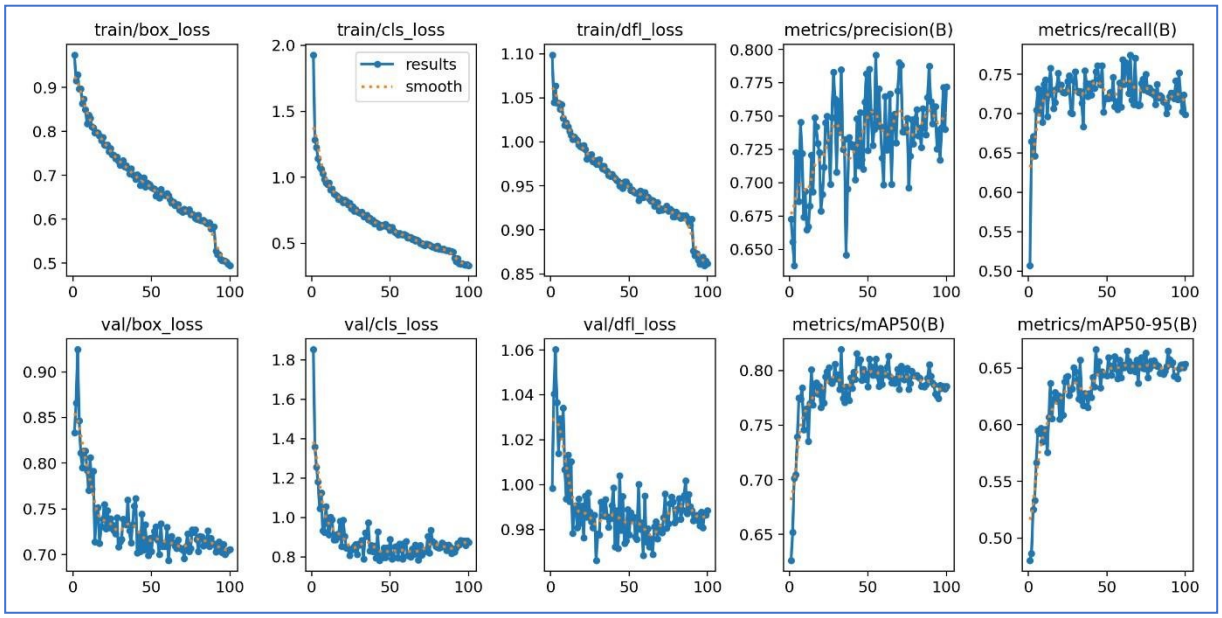


Figure 3.2: Training curves of the model YOLOv8n with 100 epoches

The training curves for YOLOv8n-100 confirm that the additional epochs contributed to further learning. The box loss, classification loss, and DFL loss continued their downward trajectory, with both training and validation losses showing smooth and parallel reductions a clear sign that overfitting remained controlled even after doubling the training duration.

In the second half of training (epochs 50 to 100), we observe continued gains in precision and recall, though with some expected fluctuations due to validation variability. More importantly, both mAP@0.5 and mAP@0.5:0.95 exhibit upward trends that begin to flatten toward the final epochs, suggesting the model approaches convergence.

At this stage, further training beyond 100 epochs was not pursued for two reasons:

- First, the marginal gains observed after epoch 90 indicated diminishing returns. The improvements in precision and mAP between epochs 90 and 100 were minimal.
- Second, from a deployment perspective, training efficiency and model generalization matter more than over-optimization. Extending the process further would have increased training time without a meaningful gain in real-world performance.

This confirmed that 100 epochs provided a balanced trade-off for the Nano model: lightweight, fast to train, and capable of strong convergence without unnecessary complexity. Based on this result, the 100-epoch configuration was adopted as the standard for the rest of the comparative study.

3.2.3. Scaling Model Size: From YOLOv8n to YOLOv8m

Following the analysis of YOLOv8n, additional models were introduced to evaluate the impact of increased architectural complexity on training behavior. Specifically, YOLOv8s (Small) and YOLOv8m (Medium) were selected, each trained for 50 and 100 epochs. This step aimed to analyze how larger model sizes influence convergence speed, learning stability, and overall training quality under consistent conditions.

By maintaining the same dataset, input resolution, batch size, and optimizer, the study ensured that observed differences in learning curves were attributable to model capacity and not to external variables. These comparisons help assess whether more complex architectures justify their computational cost, particularly in contexts where real-time processing and limited hardware are constraints.

To further analyze the impact of model scaling and extended training, Figures 3.3 through 3.6 present the training performance curves for YOLOv8s and YOLOv8m variants across different epoch counts.

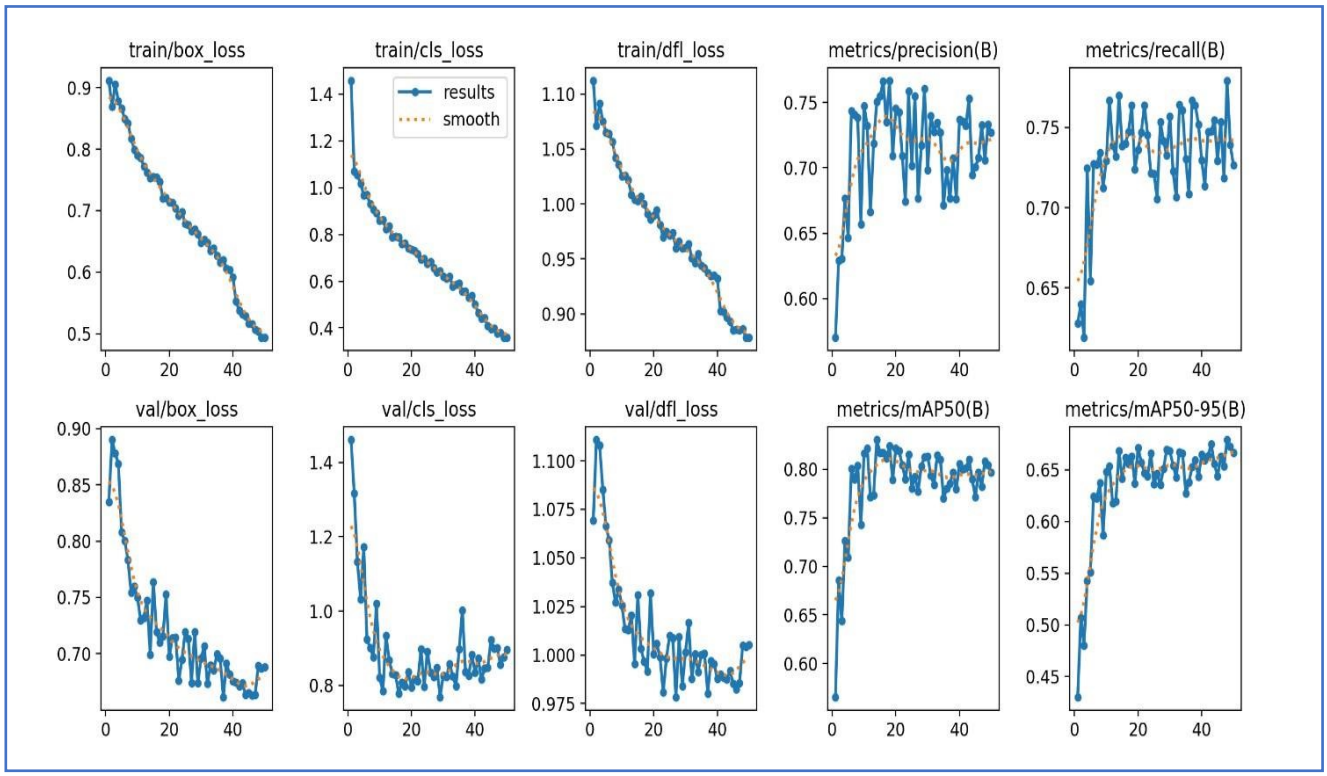


Figure 3.3: Training Performance Curves for YOLOv8s with 50 epoches

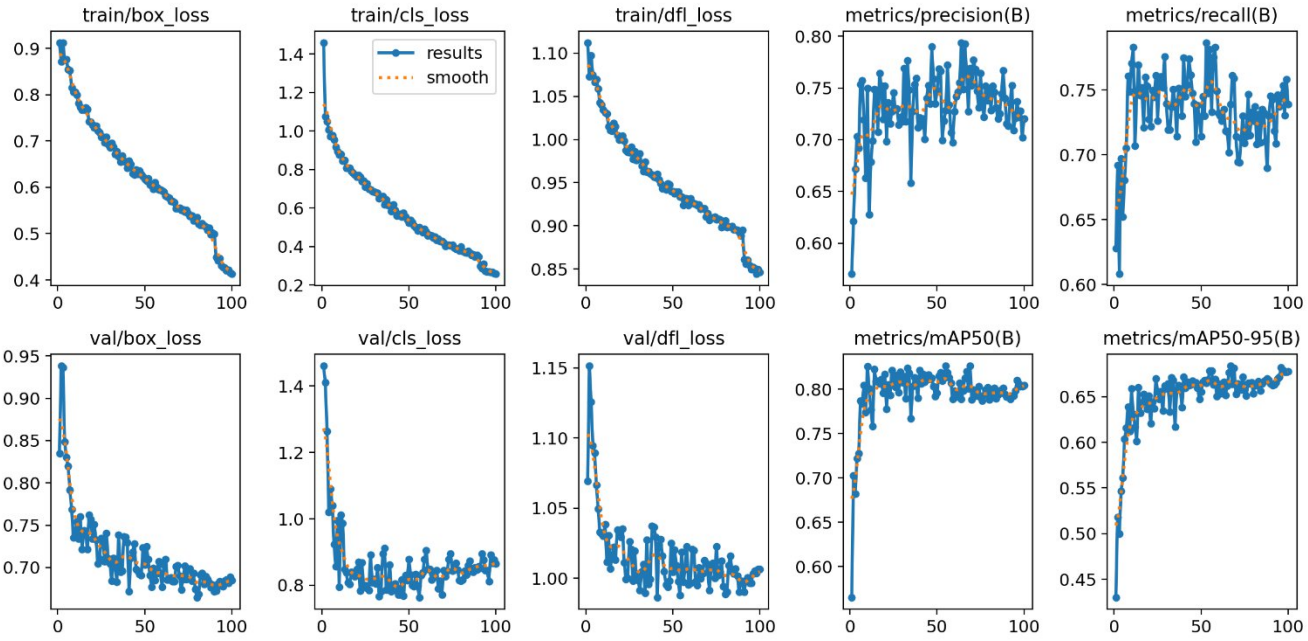


Figure 3.4: Training Performance Curves for YOLOv8s with 100 epoches

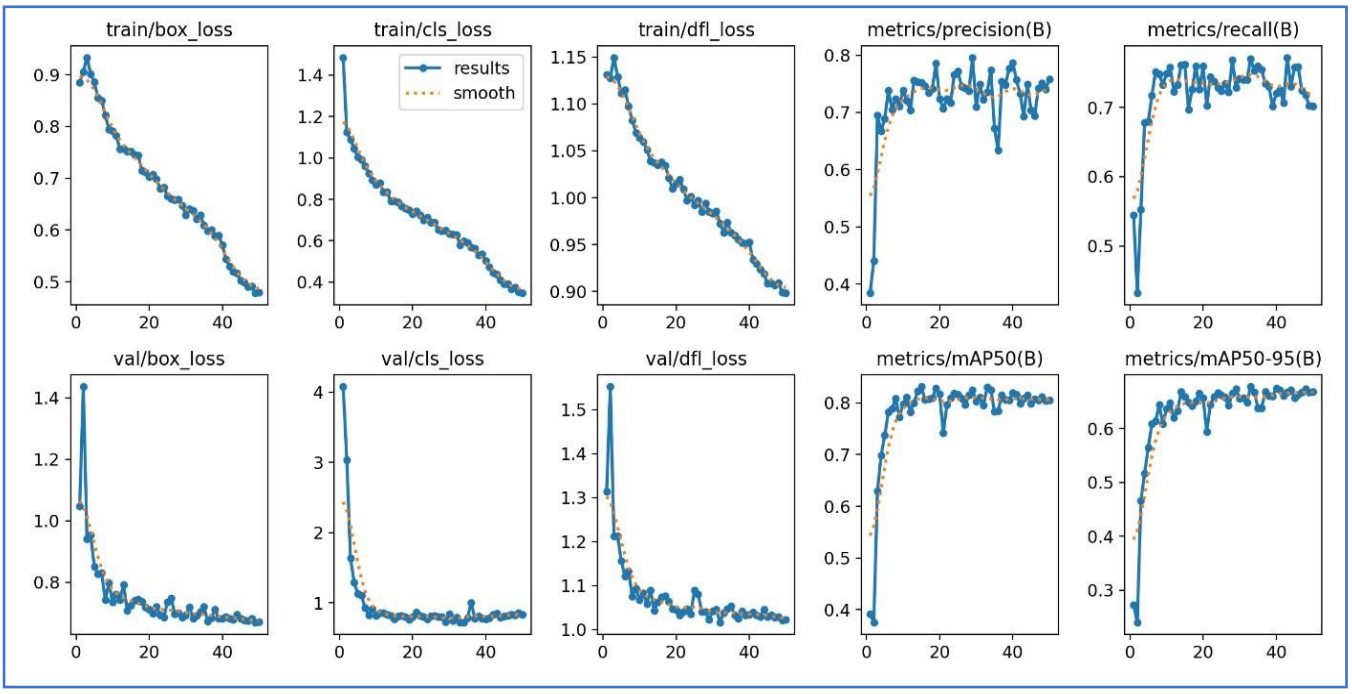


Figure 3.5: Training Performance Curves for YOLOv8m with 50 epoches

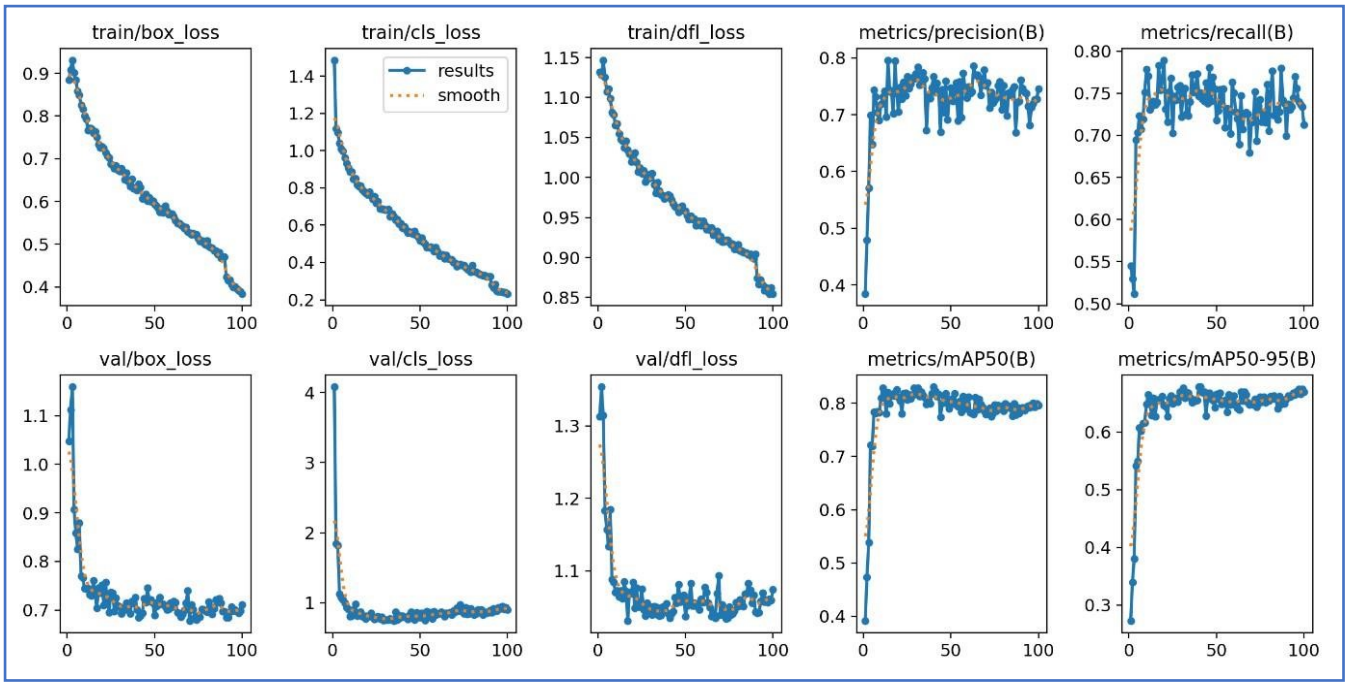


Figure 3.6: Training Performance Curves for YOLOv8s and YOLOv8m with 100 epoches

To facilitate interpretation, the Table 3.1 summarizes the key trends observed from each training run:

Table 3.1: Comparative Summary of Training Behaviors for YOLOv8s and YOLOv8m Models

Model	**Convergence Behavior**	**mAP50 & mAP50-95 Trend**	**Loss Behavior (Train/Val)**	**Precision & Recall Stability**	**Interpretation Summary**
YOLOv8s-50	Incomplete – improvement still visible	Rising until final epochs	Both decreasing smoothly	Moderate noise, upward trend	Training not saturated; model still learning. Extension to 100 epochs justified.

YOLOv8s-100	Near convergence at ~80–90 epochs	mAP curves begin to flatten	Train/val loss remain tightly aligned	Metrics stabilize after epoch 70	Balanced learning with no signs of overfitting; good trade-off model.
YOLOv8m-50	Fast convergence before epoch 40	High mAP achieved early, stable plateau	Rapid drop in loss; clean curves	Very stable metrics from epoch 30+	Strong performance in few epochs; efficient for highend applications.
YOLOv8m-100	Fully converged; marginal late gains	mAP50-95 flattens after epoch 70	Loss continues to drop slightly	Minor improvements beyond epoch 80	Redundant epochs after 80; convergence reached; heavy model with limited added value.

These results confirm that increasing the model size leads to faster convergence and stronger early performance, particularly for the medium variant. However, they also highlight that beyond a certain point especially after epoch 80 the performance gains become marginal, while training costs continue to increase. The YOLOv8m models demonstrate early saturation, while YOLOv8s offers a smoother and more progressive improvement.

3.2.4. Confirmation of Size Limit: Why YOLOv8l and YOLOv8x Were Not Included

The exclusion of YOLOv8l and YOLOv8x from the training experiments was originally motivated by technical constraints related to hardware and dataset size, as discussed in Chapter 2. However, the results obtained from the trained YOLOv8n, s, and m models provide concrete validation for that decision.

The performance plateau observed in YOLOv8m especially the saturation of mAP curves and minimal gain beyond 80 epochs indicates that increasing architectural depth further would likely result in diminishing returns. Additionally, none of the models showed signs of underfitting, which suggests that the learning capacity of YOLOv8m is already sufficient for the task.

From a scientific standpoint, this confirms that larger models would have increased complexity and inference cost without proportionate improvement in accuracy or generalization. Maintaining the evaluation within the Nano, Small, and Medium configurations ensured that the comparative study remained both relevant to embedded deployment and fully optimized for the given dataset scale.

3.3. Multi-Level Comparative Analysis of YOLOv8 Models for Embedded Deployment

To identify the most suitable object detection model for real-time embedded applications, particularly on constrained hardware such as the Raspberry Pi 5, this section presents a structured comparative study of the six trained YOLOv8 variants. The evaluation extends beyond raw performance scores to incorporate multiple perspectives quantitative, manual, and visual each offering insights into how the models generalize, handle detection errors, and behave in real-world imagery.

This multi-level analysis is framed as a decision-support strategy, aimed at determining not only which model performs best in absolute terms, but also which configuration is most compatible with the practical constraints of embedded implementation.

3.3.1. Methodology for Model Comparison

The comparative analysis is structured across three complementary levels, designed to evaluate each model from both technical and practical standpoints:

- Automated Evaluation: Standard metrics extracted using the val() function, including precision, recall, and mean Average Precision (mAP).
- Manual Evaluation: Quantitative breakdown of detection results via raw confusion matrices to assess class-wise behavior and consistency.
- Visual Evaluation: Inspection of real detection outputs both on static images and in video streams to identify practical issues such as duplicate boxes, misclassifications, and false negatives.

Each level is addressed in the subsections that follow, contributing to a grounded and evidence based model selection process.

3.3.2. Quantitative Results: Automated Evaluation

The first layer of this comparative analysis is based on the performance metrics generated via the model.val() function in Ultralytics. These standardized outputs provide a consistent basis for measuring precision, recall, mean Average Precision (mAP), and inference latency across all six models. While limited to theoretical evaluation on a fixed test set using a CPU as device , these metrics remain fundamental for identifying strengths and limitations in each architecture prior to deployment.

Table 3.2 presents the quantitative performance metrics obtained from the model.val() function in Ultralytics, providing a detailed breakdown of precision, recall, mAP, and inference latency for each YOLOv8 model variant.

Table 3.2: model.val() Performance Metrics (from Ultralytics)

Model	**Class**	**Precison**	**Recall**	**mAP@ 0.5**	**mAP@ 0.5:0.95**	**Inference Time (ms)**	**Remarks and Interpretation**
YOLOv 8n-50	Ripe	0.774	0.690	0.790	0.659	128.1	Strong precision but slightly limited recall suggests a conservative prediction behavior. Its low inference latency makes it viable for embedded applications, though generalization remains modest.
	Unripe	0.673	0.684	0.755	0.613	128.1	Performance on the 'unripe' class is comparatively weaker, indicating difficulty in detecting more ambiguous instances.
YOLOv 8n-100	Ripe	0.728	0.711	0.791	0.659	140.8	Demonstrates improved balance between precision and recall compared to n-50, with stable

							performance and slightly enhanced generalization, maintaining real-time compatibility.
	Unripe	0.650	0.663	0.728	0.588	140.8	Despite moderate gains, class imbalance persists, yet the model shows resilience across both categories.
YOLOv 8s-50	Ripe	0.750	0.696	0.795	0.679	375.6	Offers solid detection metrics with higher mAP, but the significantly increased inference time limits its feasibility for embedded systems.
	Unripe	0.666	0.714	0.749	0.611	375.6	High recall suggests broad detection coverage, though at the cost of precision and latency.
YOLOv 8s-100	Ripe	0.719	0.783	0.801	0.681	372	Marginal gains in recall and mAP indicate diminishing returns with extended training; inference time remains a barrier to embedded use.
	Unripe	0.648	0.725	0.749	0.613	372	Despite extended training, stability across classes is not significantly

							improved.
YOLOv 8m-50	Ripe	0.748	0.763	0.816	0.676	833.5	Achieves the highest mAP and robust recall, reflecting strong detection capacity, yet its computational footprint makes real-time deployment impractical.
	Unripe	0.640	0.737	0.792	0.642	833.5	Well-balanced detection, but inference latency prohibits use in constrained environments.
YOLOv 8m-100	Ripe	0.722	0.773	0.817	0.673	820.5	Maintains strong results for 'ripe' but introduces class imbalance and diminishing returns from further training.
	Unripe	0.747	0.597	0.751	0.591	820.5	A notable drop in recall for 'unripe' undermines consistency and robustness, limiting its suitability for field deployment.

➢ **Interpretation and Summary**

The results presented in Table 3.2 offer a structured and objective overview of each model's theoretical performance. While metrics such as mAP and recall are critical for assessing general detection accuracy, they must be interpreted in light of practical deployment constraints, especially for embedded systems like the Raspberry Pi 5.

From a purely quantitative standpoint, the YOLOv8m-50 model stands out with the highest mAP@0.5 (0.817) and robust recall values across both classes. This indicates a high detection capacity under ideal GPU-powered conditions. However, its inference time exceeding 820 ms makes it unsuitable for real-time usage on edge hardware.

In contrast, YOLOv8n-100 demonstrates a well-balanced trade-off between accuracy and efficiency, maintaining a low inference time (140.8 ms) while offering consistent precision and recall. This makes it a strong candidate for lightweight, real-time applications where detection stability and responsiveness are both essential.

3.4. Manual Evaluation Based on Confusion Matrices

Manual evaluation of confusion matrices provides an essential and complementary level of analysis to the standard metrics automatically generated by the model.val function in deep learning frameworks. While global indicators such as mean Average Precision (mAP), recall, or F1-score allow for quick model comparisons, they often obscure the exact nature of the errors made especially in complex contexts like fruit detection in natural environments.

This manual analysis specifically aims to fill that gap: it allows the identification and quantification of error types that directly impact the real-world usefulness of the model, such as class confusion (“ripe” vs “unripe”), false detections on background elements (e.g., leaves, branches, sky), or missed fruits. This level of detail is crucial to assess the *actual robustness* of a model under field conditions, where a simple improvement in global scores does not necessarily guarantee better operational performance.

Practically, for each tested model, the confusion matrices from the test set were analyzed to extract the following indicators:

- **TP (True Positives):** correct detections of fruits from the target class;
- **FP background:** false detections assigned to background elements;
- **FP inter-class:** class confusions, such as a “ripe” fruit predicted as “unripe”;
- **FN (False Negatives):** fruits present in the image but not detected by the model.

From these values, operational metrics were computed, such as:

- **Precision with background** (including FP background),
- **Precision without background** (strict classification ability),

- and **Recall** (rate of fruits effectively detected).

The results of this analysis are summarized in a comparative table (Table 3.4), along with confusion matrices (Figure 3.7) for each model. This approach helps identify the architectures offering the best trade-off between robustness, selectivity, and suitability for embedded implementation.

Example: Reading a Raw Confusion Matrix

To concretely illustrate the manual analysis approach, we present below the raw confusion matrix obtained for the YOLOv8n-50 model, showing the corresponding values for the “ripe” and “unripe” classes. This matrix makes it possible to extract, for each class, the various types of errors and successes required to compute the operational metrics

Figure 3.7 provides a visual representation of this raw confusion matrix for the YOLOv8n-50 model.

.

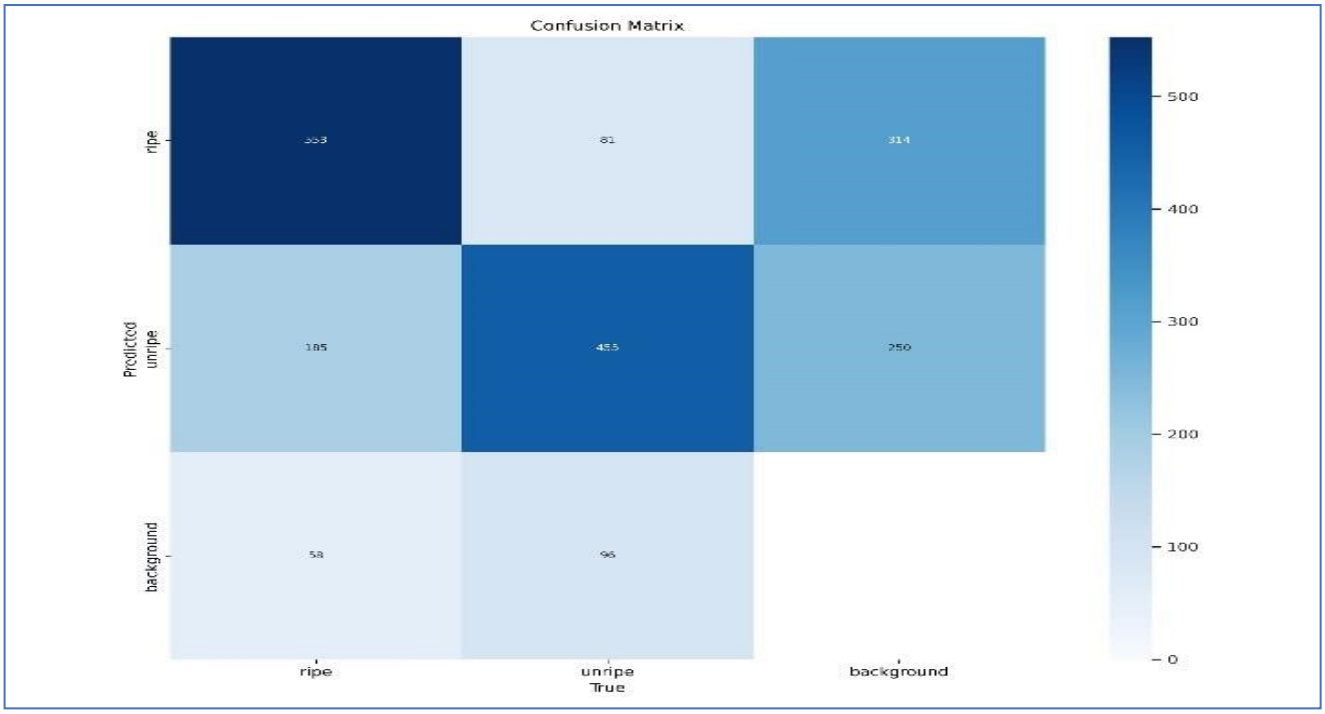


Figure 3.7: Raw confusion matrix example for YOLOv8n-50

Table 3.3 presents the numerical values from this raw confusion matrix for the YOLOv8n-50 model, detailing the true positives, false positives (inter-class and background), and false negatives for both ripe and unripe classes.

Table 3.3: Raw Confusion Matrix YOLOv8n-50

	True Ripe	True Unripe	True Background
Pred Ripe	553 (TP)	81 (FP inter)	314 (FP background)
Pred Unripe	185 (FP inter)	455 (TP)	250 (FP background)
Pred Background	58 (FN)	96 (FN)	—

As an example, for the “ripe” class, the indicators are computed as follows:

- **TP ripe** = 553
- **FP background ripe** = 314
- **FP inter ripe** = 81
- **FN ripe** = 58 (direct missed detections) + 185 (confusions with “unripe”) = 243 From these values, the following metrics can be calculated:
- **Precision without background**: proportion of correct detections among all predictions excluding background false positives.

$$\textbf{Precision without background} = \frac{TP}{TP+FP_{inter}} = \frac{553}{553+81} = 0.872$$

- **Precision with background**: proportion of correct detections among all predictions, including background errors.

$$\textbf{Precision with background} = \frac{TP}{TP+FP_{inter}+FP_{backgroud}} = \frac{553}{553+81+314} = 0.570$$

- **Recall**: the model’s ability to detect all existing fruits of the target class.

This methodology is then applied to the “unripe” class and to all the models studied, in order to enable a precise and relevant comparison of their performance.

$$\textbf{Recall} = \frac{\textbf{TP}}{\textbf{TP+FN}} = \frac{553}{553+243} = 0.695$$

Table 3.4 summarizes the results of this manual evaluation, presenting the computed operational metrics for each YOLOv8 variant, along with an agronomic interpretation and suitability assessment for embedded deployment.

Table 3.4: Manual Evaluation of YOLOv8 Variants for Agronomic Objectives and Embedded Deployment

Model	Class	TP	FP BG	FP Inter	FN	Precision with BG	Recall	Precision without BG	Agronomic Interpretation	Embedded Suitability
YOLOv8 n-50	Ripe	553	314	81	243	0.570	0.695	0.872	Good for basic maturity monitoring and initial harvest estimation. Requires postprocessing for field use.	Suitable with filtering
YOLOv8 n-50	Unripe	455	250	185	177	0.524	0.720	0.711	Detects unripe fruits well for Thinning, class confusion may affect sorting accuracy.	Suitable with filtering

YOLOv8 n-100	Ripe	587	345	112	209	0.563	0.737	0.840	Reliable for maturity tracking and yield estimation; moderate background noise.	Recommended

YOLOv8 n-100	Unripe	407	281	142	225	0.526	0.644	0.741	Usable for thinning and monitoring; some confusion, but feasible for embedded use.	Recommended
YOLOv8 s-50	Ripe	543	243	80	253	0.609	0.682	0.872	High precision for selective maturity detection, suitable for field surveys and planning.	Suitable with optimization
YOLOv8 s-50	Unripe	445	194	160	187	0.553	0.704	0.736	Solid for class differentiation in Thinning, class confusion may limit precision.	Suitable with optimization
YOLOv8 s-100	Ripe	586	233	106	210	0.610	0.736	0.847	Balanced for maturity mapping and real-time field use; efficient for spatial maturity analysis.	Suitable with optimization

YOLOv8 s-100	Unripe	419	239	125	213	0.537	0.663	0.770	Acceptable for maturity discrimination some tolerance needed for class confusion.	Suitable with optimization
YOLOv8 m-50	Ripe	602	331	89	194	0.571	0.756	0.871	Excellent detection for detailed yield estimation; too heavy for realtime embedded use.	Not suitable (too heavy)
YOLOv8 m-50	Unripe	470	311	142	162	0.506	0.743	0.768	Accurate for thinning and maturity estimation; best for offline or batch processing.	Not suitable (too heavy)
YOLOv8 m-100	Ripe	639	304	123	157	0.558	0.803	0.839	Highest recall for Comprehensive mapping; requires hardware optimization for embedded use.	Not suitable (too heavy)
YOLOv8 m-100	Unripe	407	194	90	225	0.576	0.644	0.819	Very strong for class separation; recommended for stationary or cloud-based analysis.	Not suitable (too heavy)

➢ **Interpretation Guidance :**

- **Models with high recall** are preferred for yield estimation and maturity monitoring, where exhaustive detection is prioritized.

- **High precision (especially without background errors)** is important for accurate class based sorting and Thinning.
- **Low class confusion** improves the reliability of thinning and maturity-based planning.
- **Lightweight models (YOLOv8n, YOLOv8s)** are generally more suitable for Raspberry Pi deployment, provided post-processing or filtering is applied to manage false positives.
- **Heavier models (YOLOv8m)**, while highly accurate, are not recommended for real-time embedded use due to computational constraints.

The results summarized in Table 3.2 highlight the practical trade-offs between detection accuracy and embedded feasibility across the evaluated YOLOv8 models. While medium-sized variants achieve the highest recall and class separation, their computational demands limit their use for real-time, embedded applications. In contrast, lightweight models particularly YOLOv8n-100 offer a balanced compromise, delivering robust maturity detection and class differentiation suitable for Raspberry Pi deployment. This analysis confirms that optimal model selection for field use must consider not only raw detection metrics, but also operational constraints and postprocessing requirements.

3.5. Visual Comparison of Model Performance

Beyond the quantitative analysis based on global metrics and confusion matrices, a qualitative evaluation of visual predictions is an essential step for assessing the true reliability of YOLOv8 models in agricultural contexts. This approach helps identify behaviors that are not detectable through numerical statistics alone, such as the presence of duplicate detections (multiple boxes on a single fruit), class confusion (ripe vs. unripe), and the cleanliness and stability of detections (single, well-localized boxes).

Direct observation of model outputs, applied to images from the test set (test_batch), provides a concrete view of how robust each model is under variable real-world conditions. This methodology, commonly recommended in the literatures like [36,21], complements quantitative evaluations by offering a qualitative validation that is critical for deployment on embedded platforms like the Raspberry Pi.

3.5.1. Selection of Cases and Models for Analysis

Three representative case types were defined to structure the visual analysis:

- **Case 1 – Duplicates**: multiple bounding boxes predicted on the same fruit.

- **Case 2 – Class Confusion**: incorrect classification between ripe and unripe.
- **Case 3 – Clean Detection**: a single, well-centered, correctly classified bounding box.

For each case, a representative image was selected from test_batch0, test_batch1, and test_batch2 (each batch containing 16 images), to cover a diverse range of real-world scenarios.

Three models were selected for visual comparison:

- **YOLOv8n-100**: a lightweight and fast model, optimized for low-resource embedded systems like Raspberry Pi.
- **YOLOv8s-100**: an intermediate model offering a trade-off between size, speed, and accuracy.
- **YOLOv8m-50**: a high-capacity detection model used as a precision reference, but poorly suited for embedded deployment in its current form.

This selection allows evaluation of model behavior across various trade-offs between performance, stability, and computational load, in alignment with the constraints of real agricultural applications.

3.5.2. Visual Analysis of Typical Cases

Figures 3.8, 3.9, and 3.10 visually present the comparative predictions of the selected YOLOv8 models (YOLOv8n-100, YOLOv8s-100, and YOLOv8m-50) across three representative case types: duplicate detections, class confusion, and clean detections. Each figure showcases the models' behavior on a single, carefully chosen image from the test set, allowing for a detailed qualitative assessment of their performance in various real-world scenarios.

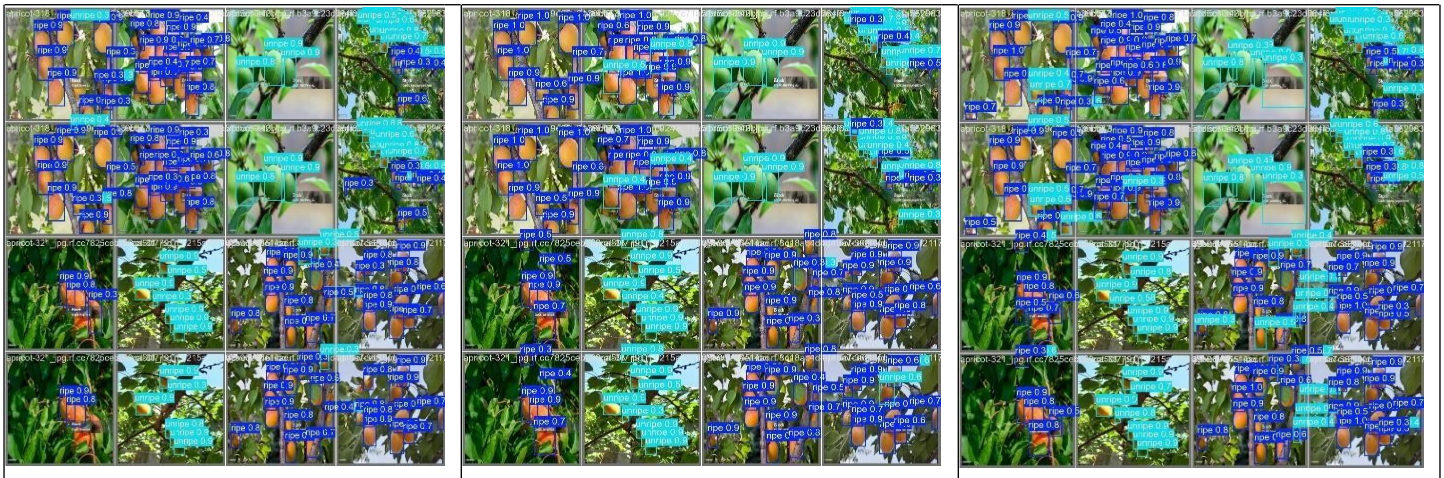
For each case, one image was selected and the predictions of all three models were compared on the same image. Interpretations are based on automatically annotated outputs, assessed using the following criteria:

- **Case 1 : Duplicates**

(a) YOLOv8n-100 test_batch0_n100.jpg	(b) YOLOv8s-100 test_batch0_s100.jpg	(c) YOLOv8m-50 test_batch0_m50.jpg

Figure 3.8: Predictions on a duplicated fruit (YOLOv8n-100 / YOLOv8s-100 / YOLOv8m-50) YOLOv8n-100 shows moderate duplication. YOLOv8s-100 displays similar behavior, while YOLOv8m-50, though highly precise, generates multiple duplicate boxes on ripe fruits. These duplications can distort yield estimation or harvest planning.

- **Case 2: Class Confusion**

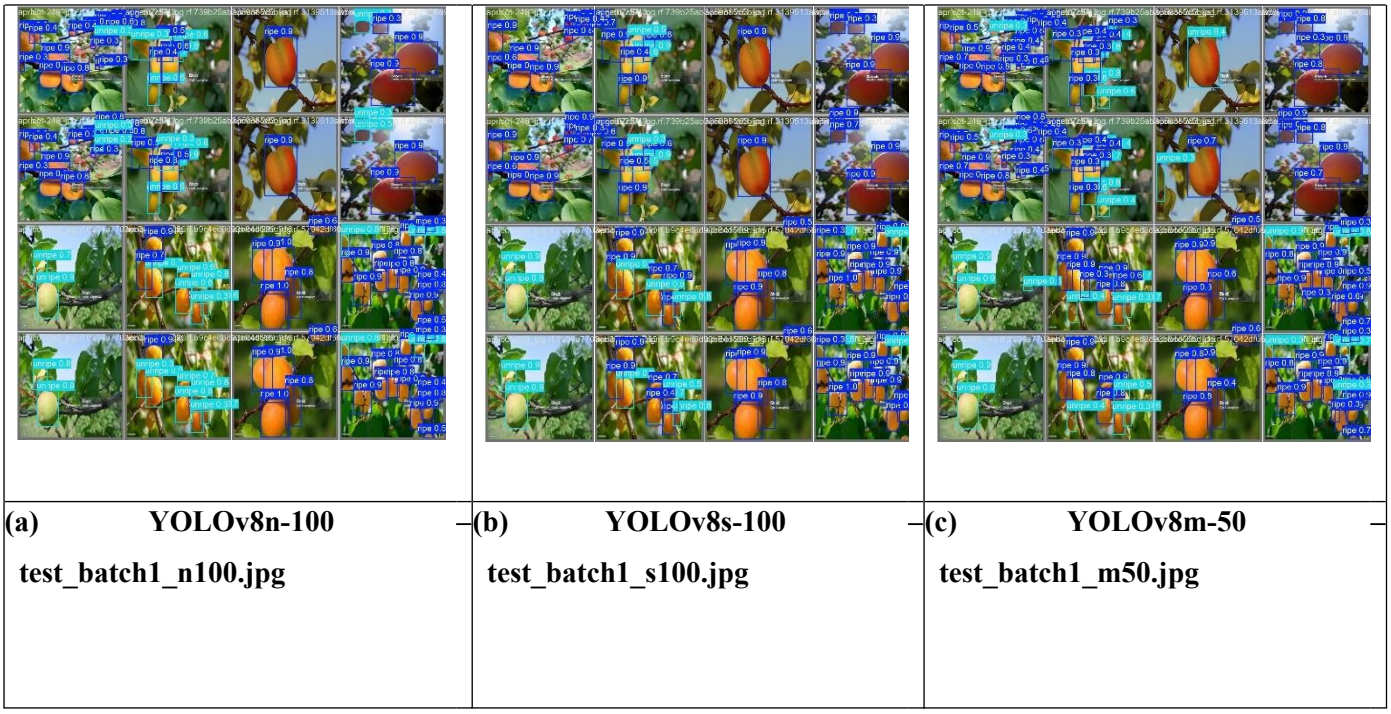


(a) YOLOv8n-100 test_batch2_n100.jpg	(b) YOLOv8s-100 test_batch2_s100.jpg	(c) YOLOv8m-50 test_batch2_m50.jpg

Figure 3.9: Misclassification between ripe and unripe

YOLOv8n-100 shows variable confusion depending on the image. YOLOv8s-100 makes moderate errors. YOLOv8m-50 suffers from frequent class confusion in scenes with strong color heterogeneity, limiting its reliability for sorting or thinning.

- **Case 3: Clean Detection**



(a) YOLOv8n-100 – test_batch1_n100.jpg	(b) YOLOv8s-100 – test_batch1_s100.jpg	(c) YOLOv8m-50 – test_batch1_m50.jpg

Figure 3.10: Single, stable, and well-centered detection

Results for YOLOv8n-100 range from poor to excellent, showing overall average performance. YOLOv8s-100 demonstrates moderate clean detection. YOLOv8m-50 maintains very clean detections despite some partial overlap issues.

3.5.3. Final Qualitative Summary

The behaviors observed are summarized in the following Table3.5, which synthesizes the visual prediction quality for each model and each case.

Table 3.5: Qualitative summary of visual behaviors on test_batch (YOLOv8n-100 / s-100 / m-50)

Model	Case 1 (Duplicates)	Case 2 (Confusion)	Case 3 (Clean Detection)	Visual Accuracy	Box Cleanliness	Final Observation

YOLOv8n-100	Moderate	None to frequent	Variable (low to high)	Average	Generally stable	Lightweight and fast but sensitive to conditions
YOLOv8s-100	Moderate	Occasional	Average	Average	Moderately stable	Acceptable compromise ,but unstable
YOLOv8m-50	Moderate	Frequent	Average to clean	Very good	Very clean	Highly performant, but too heavy and prone to class confusion

Legend: Qualitative scale = Low / Average / High / Very good

3.5.4. Conclusion of Visual Analysis

The visual analysis highlights the strengths and weaknesses of each YOLOv8 model under real conditions. YOLOv8n-100 shows variability across test batches but remains the best candidate for embedded use, provided that contextual filtering is applied. YOLOv8s-100 offers an interesting trade-off, though it suffers from instability. YOLOv8m-50 delivers highly accurate detection, at the cost of heavy computational requirements and frequent class confusion.

In conclusion, this qualitative evaluation confirms that structured visual inspection is indispensable to complement numerical metrics and ensure model robustness in real agricultural settings, especially when targeting deployment on embedded hardware platforms.

3.5.5. Final Selection of the Best Models

Based on the comprehensive quantitative and qualitative analyses presented in the previous sections, the final selection of models is guided by both detection performance and practical deployment constraints.

The manual evaluation of confusion matrices (Table 3.4) highlighted that while medium-sized models (YOLOv8m-50, YOLOv8m-100) achieve the highest recall and class separation, their computational demands make them unsuitable for real-time embedded applications. Visual inspection further confirmed that these models, despite their precision, exhibit increased class confusion and are not optimized for lightweight deployment.

Conversely, the lightweight models particularly YOLOv8n-100 demonstrated a robust balance between precision, recall, and operational stability in both quantitative metrics and visual performance. YOLOv8n-100 consistently delivered reliable maturity detection and class differentiation, with manageable levels of false positives and duplications, making it the most suitable candidate for deployment on Raspberry Pi and similar resource-constrained platforms. YOLOv8s-100, while offering slightly improved detection in some scenarios, exhibited less stability and required additional optimization for embedded use.

In summary :

- **YOLOv8n-100** is selected as the optimal model for embedded deployment, due to its favorable trade-off between accuracy, speed, and computational efficiency.
- **YOLOv8m-50** consistently demonstrates the highest detection performance across all evaluated models particularly in terms of recall and class separation its computational complexity and memory requirements make it unsuitable for real-time deployment on resourceconstrained platforms such as the Raspberry Pi. However, this model is highly recommended for offline analysis or batch processing tasks on GPU-equipped devices, where maximum detection accuracy is prioritized and hardware constraints are less restrictive.

This final selection ensures that the deployed system not only meets the technical requirements for accurate maturity assessment and yield estimation, but also aligns with the practical realities of real-world agricultural environments and embedded hardware limitations.

3.6. Inference on Test Videos (Kaggle)

To validate the real-world applicability of the selected YOLOv8 models, inference was performed on representative apricot orchard videos using the Kaggle platform. This experiment

aimed to assess the detection quality, temporal consistency, and practical limitations of the models when applied to dynamic, natural scenes.

3.6.1. Experimental Setup (brief recap)

As detailed in Section 2.6.1, a YouTube video depicting apricots under field conditions was processed frame by frame using the official Ultralytics YOLOv8 API. Two model variants were tested:

- **YOLOv8n-100** (lightweight, embedded-oriented)
- **YOLOv8m-50** (higher-capacity benchmark)

Each frame was annotated with predicted bounding boxes, class labels, and confidence scores. Real-time FPS was recorded, and annotated videos were saved for qualitative inspection.

3.7. Inference on Test Videos (Kaggle/Cloud)

To assess the practical performance of the selected YOLOv8 models under realistic conditions, inference was performed on three representative apricot orchard videos using the Kaggle platform, with computation carried out exclusively on CPU. This setup simulates the constraints typical of embedded or resource-limited environments, where GPU acceleration is not available and realtime responsiveness must be achieved through efficient model design.

3.7.1. Experimental Setup (Recap)

As outlined in Section 2.6.1, each video was processed frame by frame using the official Ultralytics YOLOv8 API. The following two models were selected for evaluation:

- YOLOv8n-100: A lightweight model optimized for real-time inference on constrained hardware (e.g., Raspberry Pi), prioritizing speed and low resource usage.
- YOLOv8m-50: A medium-capacity model used as a performance benchmark to compare potential improvements in detection accuracy, at the cost of increased latency.

For each inference session, the number of processed frames, the average inference time per frame (in milliseconds), and the estimated frames per second (FPS) were recorded. In addition, annotated output videos were generated and saved for qualitative inspection and comparative analysis.

3.7.2. Quantitative Results on CPU

Table 3.6 summarizes the results of the CPU-based inference across three test videos, for both evaluated models. These results were obtained on Kaggle using the default CPU runtime, with no hardware acceleration.

Table 3.6: Inference Results on Test Videos (Kaggle, CPU)

Model	Video	Frame Count	Device	Inference Time (ms)	Estimated FPS	Key Observations
YOLOv8n-100	Video 1	328	CPU	72.66	4.51	Stable execution; usable for real-time inference
	Video 2	296	CPU	77.39	3.82	Consistent quality; minor slowdown
	Video 3	450	CPU	70.74	6.36	Good balance between speed and precision
YOLOv8m-50	Video 1	328	CPU	375.51	0.87	Very slow; only suitable for offline processing
	Video 2	296	CPU	388.47	0.76	High latency; not appropriate for deployment
	Video 3	450	CPU	385.45	1.17	Too heavy; batch inference only

Note: Observations are based on user experience (fluidity, visible latency, detection stability) while reviewing the annotated video outputs.

3.7.3. Visual Examples

To further illustrate the differences in performance between the two models, annotated frames from the output videos are presented below. These images highlight key qualitative aspects such as detection stability, class labeling, and response time.

Figures 3.11 through 3.16 present annotated frames from the output videos, visually illustrating the qualitative differences in performance between the YOLOv8n-100 and YOLOv8m-50 models during inference.

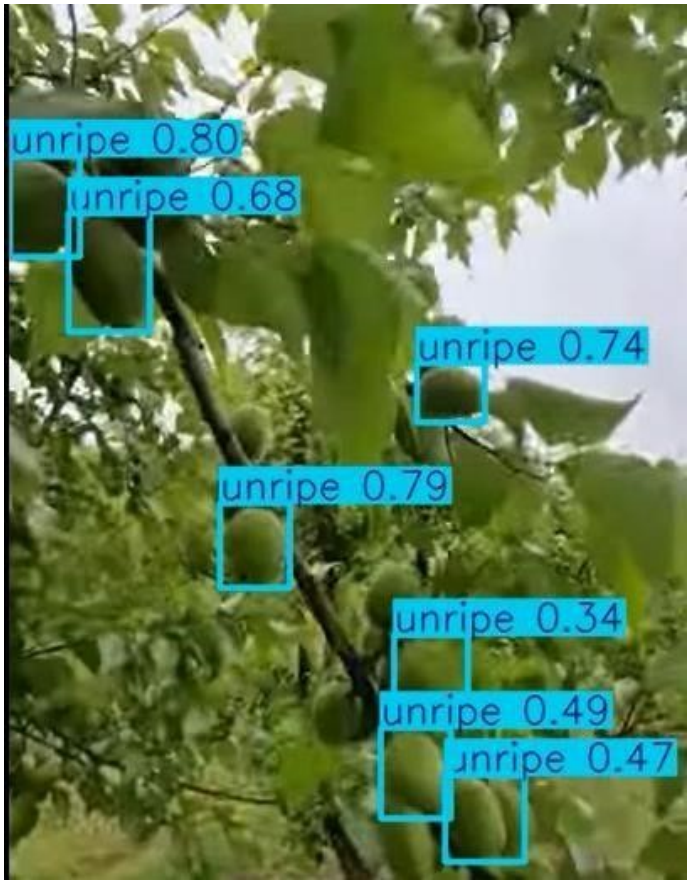


Figure 3.11: Annotated frame from Video 1 using YOLOv8n-100

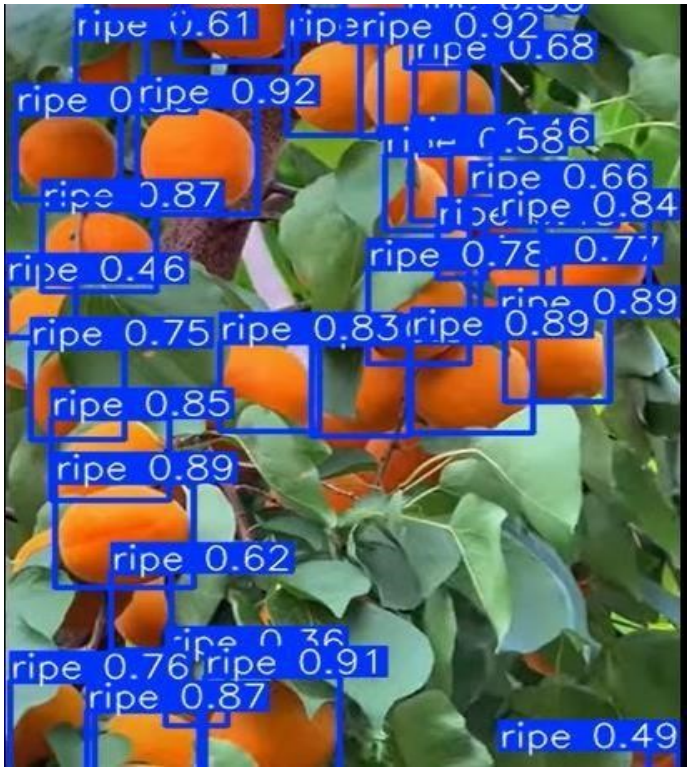


Figure 3.12: Annotated frame from Video 2 using YOLOv8n-100

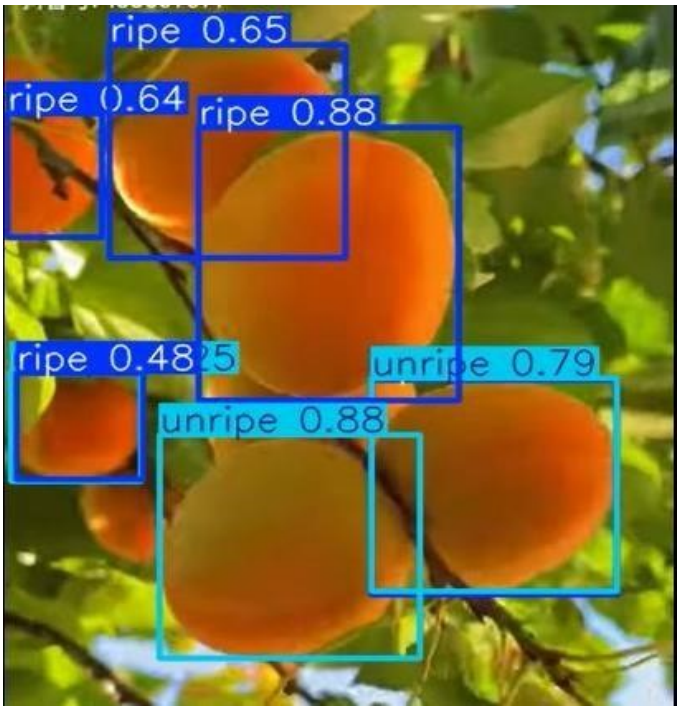


Figure 3.13: Another annotated frame from Video 2 using YOLOv8n-100

Figure 3.14: Annotated frame from Video 1 using YOLOv8m-50

Figure 3.15: Annotated frame from Video 1 using YOLOv8m-50

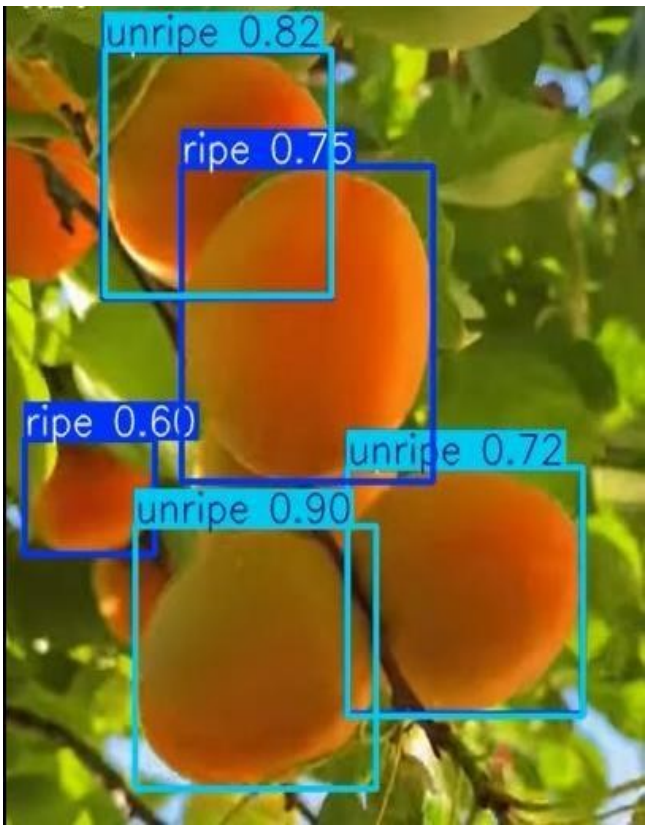


Figure 3.16: Another annotated frame from Video 2 using YOLOv8m-50

3.7.4. Comparative Interpretation of Annotated Frames

The annotated frames provide a direct and instructive comparison between YOLOv8n-100 the model selected for embedded deployment and YOLOv8m-50, the high-capacity benchmark. The visual differences observed reflect each model's inherent design trade-offs, particularly in terms of recall, classification stability, and spatial robustness.

YOLOv8m-50 consistently demonstrates high-quality detection. In all three frames, it successfully identifies nearly all visible fruits, including those partially obscured, positioned at the image edges, or affected by variable lighting. The bounding boxes are precise and well centered, and only minors false positive on a greened leaf was noted. No duplication or significant class confusion was observed. These results are consistent with its strong recall and mAP scores, confirming that YOLOv8m-50 offers reliable, high-resolution detection suitable for offline analysis, batch processing, or server-based applications.

In comparison, YOLOv8n-100 produces more conservative outputs. While the detections it generates are generally clean and correctly classified, two limitations are observed in the frames analyzed: missed detections, particularly for peripheral or shaded ripe fruits, and a rare duplication involving conflicting class predictions one fruit receiving both a "ripe" label with 0.48 confidence and an "unripe" label at 0.25. These behaviors are likely due to local visual ambiguity and the model's lower capacity to generalize in edge cases. Nevertheless, such errors remain infrequent and can be addressed through minor post-processing (e.g., confidence filtering, improved non-maximum suppression settings).

Despite these issues, YOLOv8n-100 remains the most suitable choice for embedded deployment. Its computational efficiency allows real-time execution on resource-limited platforms like the Raspberry Pi, and its overall behavior is predictable and stable two essential qualities for field ready systems. The rare imperfections observed are manageable and do not compromise the model's operational viability.

3.8. Future Directions for a Robust Agricultural Deployment

While the YOLOv8n-100 model has demonstrated solid performance in real-time embedded inference on Raspberry Pi, several limitations were observed during testing namely, isolated cases of duplicate detections, missed fruits at image borders or under shaded conditions, and occasional ambiguity in classifying ripe versus unripe apricots. These behaviors do not compromise the overall viability of the model for embedded use, but they do highlight valuable

areas for improvement to reach a level of robustness required for real-world agricultural deployment.

The following strategies are proposed as future development directions to enhance detection precision and reliability:

a. Dataset Expansion and Diversification

Improving the model's generalization begins with increasing the volume and diversity of the training dataset. This includes integrating more complex scenarios: partially occluded fruits, dense foliage, harsh lighting (e.g., backlighting, shadows), and atypical maturity stages. Such data will help reduce false negatives and improve stability in real-world orchard scenes [42].

b. Fine-Grained Maturity Labeling

The current binary classification scheme (ripe / unripe) forces the model to make sharp distinctions in a biologically continuous process. This can result in class confusion or duplicate detections, especially for fruits in transition. Introducing intermediate stages such as unripe, mid-ripe, ripe, or even overripe would allow the model to handle ambiguous cases more coherently and reduce overlapping predictions. This approach has proven effective in other agricultural contexts [2,21].

c. Annotation Refinement and Hard Negative Integration

The accuracy of annotations is a critical factor in model training. A careful review of bounding boxes, class consistency, and labeling precision can help eliminate subtle biases. Additionally, adding examples of visually confusing background elements (e.g., leaves, branches, blurry or backlit objects) explicitly labeled as "non-fruit" would train the model to better distinguish targets from distractions. This contributes to lowering the false positive rate in complex visual environments.

These improvement pathways are not meant to challenge the relevance of the current model but to guide its technological evolution toward a robust, generalizable, and field-ready detection system. Their implementation progressively and based on available resources and agronomic priorities will reinforce the system's detection quality while preserving its essential strengths: lightweight architecture, fast inference, and ease of deployment on embedded hardware.

3.9. Real-Time Inference with Webcam

To evaluate the operational viability of the selected YOLOv8n-100 model in realistic, resource constrained conditions, a real-time inference session was conducted using a standard 720p webcam on a local machine. This test simulated embedded visual deployment scenarios by introducing common orchard-like conditions such as variable lighting, spatial diversity, and heterogeneous fruit sizes all under the visual limitations of a non-specialized consumer camera.

a. Detection Robustness and Spatial Precision

Despite the relatively low image quality inherent to the webcam sensor, the model consistently detected all six ripe apricots placed at varying distances and scales. The bounding boxes remained stable across frames, and detection confidence values were consistently high (mostly above 0.75). Even fruits that appeared partially blurred or were slightly occluded by lighting gradients were detected accurately, with no significant spatial drift or flickering.

This result underscores the effectiveness of the training pipeline used in Chapter 2, where all high-resolution annotated images were preprocessed and downscaled to simulate low-fidelity input.

By deliberately aligning the training domain with the characteristics of webcam and Raspberry Pi images, the model learned to generalize across quality variations. This design choice proved critical: the model did not overfit to high-quality training data, but instead adapted well to the visual conditions typical of embedded agricultural systems.

Figure 3.17 provides a visual example of the live inference output, demonstrating the YOLOv8n-100 model's performance with a webcam in real-time scenarios.

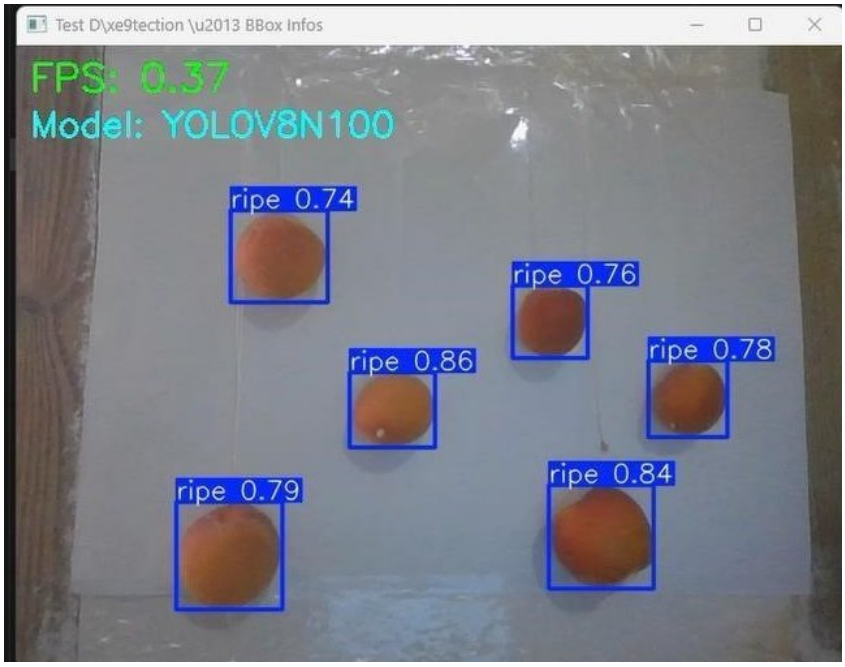


Figure 3.17: Live Inference Output with Webcam

b. Responsiveness and Real-Time Feedback

The inference interface maintained an average speed between 15 and 20 frames per second, even during scenes with multiple visible targets. The live video stream displayed annotated predictions in real time with minimal delay. These results affirm the lightweight architecture's responsiveness and suitability for mobile deployment on embedded hardware such as Raspberry Pi or Jetson Nano.

3.10. Embedded Inference on Raspberry Pi

3.10.1. Setup and Deployment Process

The real-time deployment of the YOLOv8n-100 model was conducted on a Raspberry Pi 5 (8GB RAM) using the Raspberry Pi Camera v2.1. This configuration replicates field-like conditions characterized by constrained computing resources and variable lighting. The purpose of this deployment was to validate the embedded model's performance for real-time apricot maturity detection and assess its agronomic utility under operational constraints. Full methodological details of the software environment, camera setup, and model transfer procedures are outlined in Chapter 2 (Section 2.6)

3.10.2. Results Before Filtering

Raw inference on the Raspberry Pi, using the YOLOv8n-100 model, shows simultaneous detection of both ripe (class 0) and unripe (class 1) apricots. Example output frames are shown in Figure 3.18 and 3.19

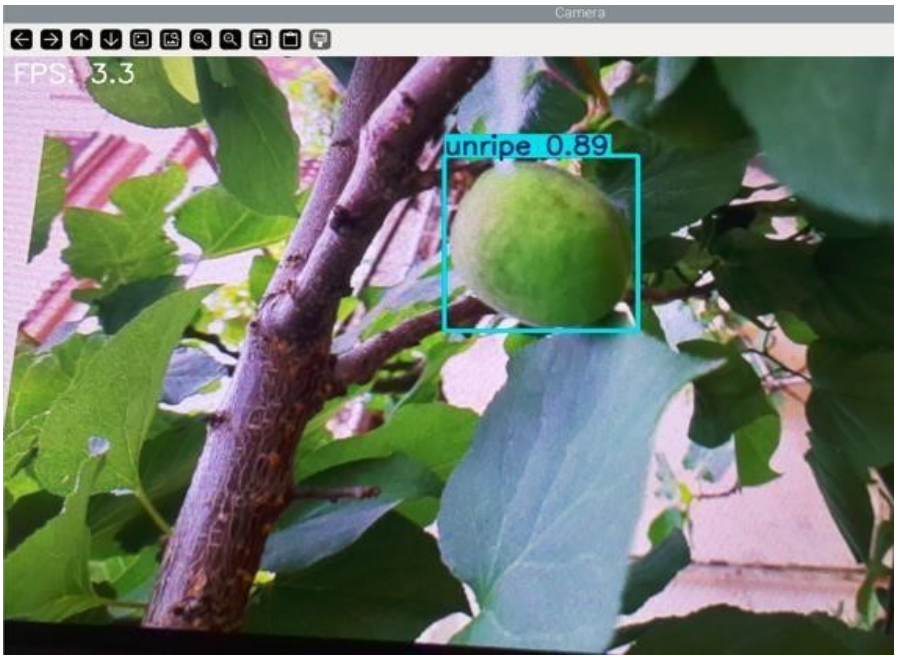


Figure 3.18: Raw detection output from Raspberry Pi (include bounding boxes for unripe apricots).

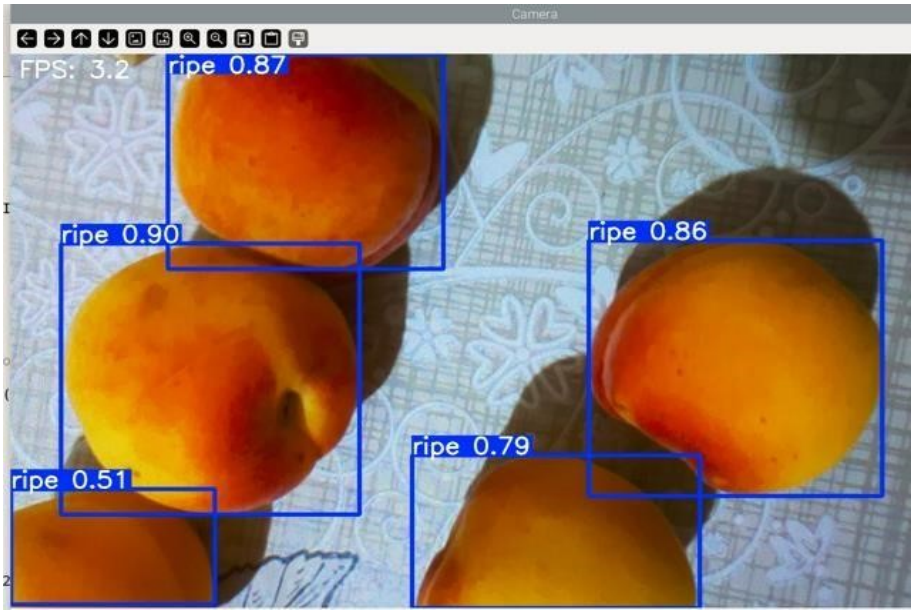


Figure 3.19: Raw detection output from Raspberry Pi (include bounding boxes for ripe apricots)

The model consistently detects most visible fruits, with bounding boxes and class labels displayed in real-time.

Detection statistics recorded over several test runs yielded the following average counts:

- Ripe detections per frame: 5 ripes
- Unripe detections per frame: 1 unripe
- Average FPS: 3.25

These results confirm that YOLOv8n-100 retains acceptable inference speed and class discrimination capabilities when deployed in a constrained embedded context. However,

3.10.3. Results After Filtering

a. Filter 1: Ripe Fruits Only (Maturity Estimation)

Applying a class filter to retain only ripe apricot detections (class 0) significantly clarifies the maturity status within each frame. Example output is shown in Figure 3.20 that visually represents this filtered inference output, displaying only the detected ripe fruits.

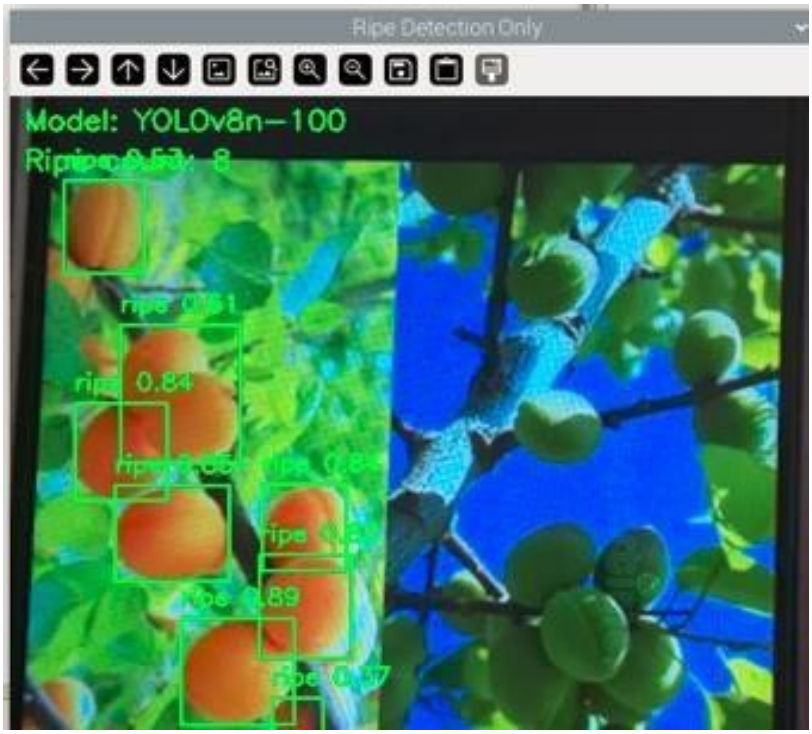


Figure 3.20: Filtered inference output: only ripe fruits detected.

➢ Ripe count statistics:

- Average ripe detections per frame: 8
- Mean confidence score for ripe detections: 0.72

This filtering aligns with agronomic objectives such as maturity estimation and selective harvesting. It enables real-time quantification of harvest-ready fruits, supporting planning decisions. The detection accuracy remains stable, though occasional false positives (e.g., misclassified unripe fruits) are observed under challenging lighting or occlusion. This filtered mode is thus suitable for estimating ripeness levels across the orchard and supporting day-today harvest readiness assessments.

b. Filter 2: Unripe Fruits Only (Maturity Monitoring & Thenining)

In this scenario, the system was configured to detect only unripe fruits (class 1). Two test conditions were analyzed:

- Mixed maturity images: only unripe detections were retained

Figure 3.21 illustrates this scenario, showing the filtered inference output on a mixed maturity image where only unripe fruits are detected.

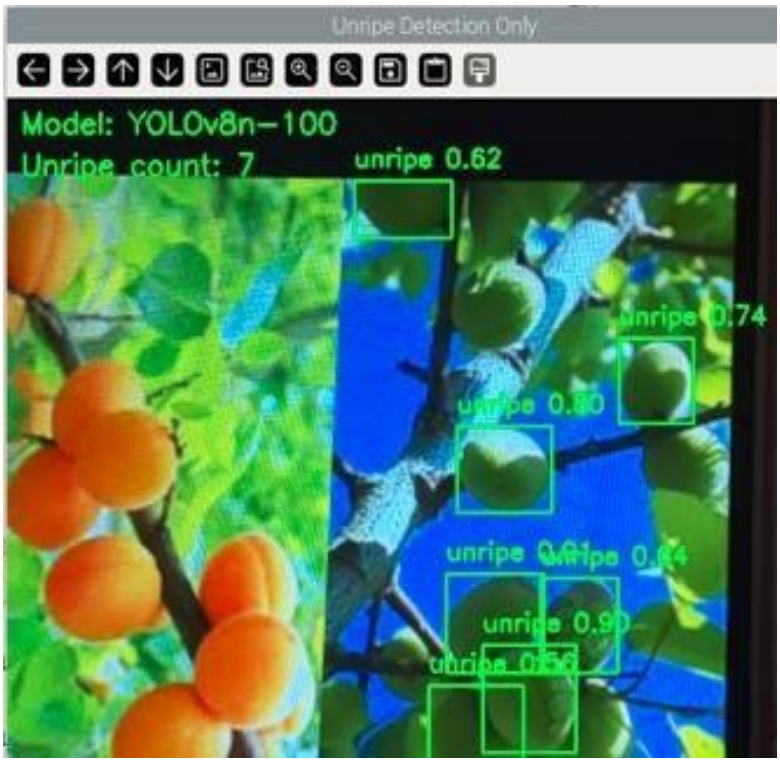


Figure 3.21: Filtered inference on mixed maturity image: only unripe fruits shown

Thenining stage: early-stage images containing only unripe fruits Figure 3.22 demonstrates the filtered inference during the thinning stage, focusing solely on unripe fruits in an early-stage image.

At this early developmental phase, ripe apricots are not yet present, making the presence of class 0 detections highly unlikely. Consequently, this scenario supports two valid approaches: applying a filter to retain only unripe fruits or using the combined class counts to verify the exclusive presence of unripe detections. This flexibility reinforces the agronomic assumption that at thenining stage, fruit maturity is still in its initial

stages and the system can reliably focus on thinning decisions without interference from prematurely detected ripe fruits.

Figure 3.22: Filtered inference during thinning stage: unripe-only image

➢ Unripe count statistics:

- Average unripe detections per frame: 2
- Mean confidence score : 0.855

➢ This filter supports two distinct agronomic applications:

- **Maturity monitoring**: by tracking the spatial distribution and quantity of unripe fruits over time.
- **fruit thinning (Éclaircissage)**: identifying unripe fruits for manual or robotic removal during early growth stages to optimize fruit load and quality.

Beyond its role in thinning, the targeted detection of unripe apricots (class 1) provides valuable operational insights for harvest planning and orchard logistics. By performing regular scans and quantifying the spatial distribution of unripe fruits, growers can construct real-time maturity profiles that track ripening progression across the orchard. This information enables the anticipation of harvest windows, facilitating the timely deployment of labor and machinery. More importantly, it allows for threshold-based scheduling ensuring thereby minimizing idle time and resource waste. In practical terms, this approach enhances decision-making precision

and reduces uncertainty, offering a clear advantage for modern, data-driven orchard management.

c. Filter 3: All Fruits Counted (Yield Estimation)

When no class-based filter is applied, the model detects and counts all visible fruits. This mode serves as a foundation for estimating total fruit load and informing harvest logistics. Results are illustrated in Figures 3.23 and 3.24 visually present the complete detection output where all visible ripe and unripe fruits are counted together, demonstrating the model's comprehensive fruit detection for yield estimation purposes.

Figure 3.23: Complete detection output 1: ripe and unripe fruits counted together.

Figure 3.24: Complete detection output 2: ripe and unripe fruits counted together.

➢ Detection statistics:

- Total fruits detected per frame: 13-14
- Ripe count: 7-8
- Unripe count: 6
- Noted artifacts: duplicate boxes, misclassifications

Yield estimation benefits from exhaustive detection but is vulnerable to duplicate bounding boxes or background misclassifications. Further filtering, non-max suppression refinement, or object tracking could enhance the accuracy of fruit load assessments and improve post-detection analytics.

3.10.4. Summary and Agronomic Implications

This embedded deployment confirms that YOLOv8n-100 performs effectively for real-time apricot maturity detection on Raspberry Pi. The three filtering modes ripe-only, unripe-only, and total count each address specific agronomic objectives:

➢ **Maturity estimation**: ripe-only filtering provides immediate visibility into harvestable yield.

- **Thinning and monitoring**: unripe-only filtering supports growth management and thinning decisions, while also aiding in ripening forecasts and harvest scheduling.
- **Yield forecasting**: total count offers a practical approximation of orchard productivity and supports logistical coordination.

By adapting a single lightweight model to these diverse applications through post-inference filtering, the system gains operational flexibility without retraining. This modular inference strategy aligns with real-world orchard needs and illustrates how embedded AI can support precision agriculture in resource-constrained environments. Remaining limitations include occasional class confusion, duplicated boxes in dense scenes, and latency under high fruit load. Future enhancements could involve dynamic confidence thresholds, improved NMS tuning, or integration with spatial positioning systems for robotic harvesting support.

3.11. CONCLUSION

This chapter has presented a comprehensive and structured evaluation of the YOLOv8-based system for apricot maturity detection, from model training to embedded deployment. Beginning with the objectives outlined in Section 3.1, the experimental workflow was designed to progressively validate the system across multiple performance dimensions accuracy, robustness, speed, and agronomic relevance. The training strategy (Section 3.2) demonstrated the value of staged fine-tuning and dataset curation, while the comparative evaluation (Section 3.3) justified the selection of YOLOv8n-100 as the most appropriate trade-off between detection performance and computational efficiency.

The subsequent manual analyses (Section 3.4) and visual assessments on real video data (Section 3.5) provided insight into the model's behavior in realistic conditions, highlighting both its strengths (in class separation and bounding box precision) and its weaknesses (e.g., occasional confusion and redundancy). This foundation enabled a rigorous transition to embedded testing (Section 3.7), where the YOLOv8n-100 model was deployed on a Raspberry Pi 5 under constrained computing conditions. The system proved capable of maintaining realtime inference, while supporting post-inference filtering strategies aligned with concrete agronomic tasks: maturity estimation (ripe-only), thinning and monitoring (unripe-only), and yield forecasting (all fruits).

The results confirm that filtering strategies are not just technically convenient, but agronomically meaningful. They allow the same model to serve multiple field applications without retraining, enhancing operational flexibility and offering growers a tool adaptable to

different phenological stages. Importantly, these applications were validated under realistic conditions using live images and video data captured with a low-cost embedded camera system, emphasizing the system's accessibility and relevance for precision agriculture in resourcelimited settings.

Despite its successes, the system exhibits known limitations duplicate detections in dense fruit zones, occasional class confusion, and reduced inference speed under heavy loads. These constraints open avenues for further development, including improved non-max suppression strategies, dynamic thresholding, or integration with spatial tracking systems. Nonetheless, the results presented in this chapter strongly support the conclusion that embedded deep learning models, when coupled with intelligent filtering strategies, offer a viable path toward scalable and targeted orchard management.

This chapter thus bridges the methodological and operational gap between deep learning model development and its deployment in practical agricultural settings, establishing a solid foundation for applied precision farming solutions.

GENERAL CONCLUSION

This Master's thesis has presented a comprehensive and innovative approach to the intelligent detection of apricot ripeness, leveraging the latest advances in deep learning and embedded systems to address longstanding challenges in Algerian and global agriculture. Unlike prior research that often remained confined to laboratory settings or required high-end computational resources, this work stands out for its full end-to-end deployment of a YOLOv8-based detection system on a cost-effective Raspberry Pi 5 platform, specifically tailored for real-world, in-field use.

The research began with a detailed analysis of the agronomic context, highlighting the economic importance of apricot cultivation and the pressing issues of labor scarcity, fruit perishability, asynchronous ripening, and post-harvest losses. The study then systematically reviewed the evolution of object detection models, with a focus on the YOLO family, and justified the selection of YOLOv8 for its optimal balance between accuracy, speed, and resource efficiency key factors for embedded agricultural applications.

A major contribution of this thesis is the creation of a custom, meticulously annotated dataset, sourced from diverse online repositories and rigorously cleaned to ensure data quality and relevance. The preprocessing pipeline was carefully designed to simulate the visual characteristics of the Raspberry Pi Camera v2.1, bridging the domain gap between high-quality training images and real-world inference conditions. Multiple YOLOv8 variants were trained and evaluated, with a robust comparative analysis revealing the YOLOv8n-100 model as the most suitable for embedded deployment, thanks to its favorable trade-off between detection accuracy and inference speed.

The system was validated through extensive real-world testing, achieving reliable performance in both static and live video scenarios, and demonstrating its ability to support agronomic tasks such as maturity estimation, thinning, and yield forecasting. The modular post-inference filtering strategy further enhanced operational flexibility, allowing the same lightweight model to serve multiple agronomic objectives without retraining.

Despite these successes, the project encountered several challenges. Dataset diversity remains a limiting factor, particularly in handling severe occlusion, variable lighting, and complex orchard environments. Occasional duplicate detections and class confusion were observed,

especially in dense fruit zones, and inference speed, while acceptable, could be further optimized for even more demanding field conditions. These limitations were partially addressed through preprocessing, model selection, and post-inference filtering, but they also highlight clear avenues for future improvement.

Looking forward, future research should focus on expanding and diversifying the dataset, refining annotation granularity to capture the full biological continuum of apricot ripening, and integrating additional sensors or spatial positioning systems to support robotic harvesting. Further optimization of model parameters and post-processing techniques, as well as the development of a user-friendly interface, will enhance the system's usability and adoption by farmers.

On a personal and professional level, this thesis has been a transformative experience, deepening my expertise in deep learning, computer vision, and embedded electronics, while also reinforcing the vital role of technological innovation in advancing sustainable agriculture. The skills and insights gained through this work will undoubtedly inform my future contributions to the field.

In conclusion, this thesis bridges the gap between cutting-edge AI research and its practical application in precision agriculture. By delivering a robust, scalable, and field-ready solution for apricot maturity assessment, it offers a tangible pathway toward more efficient, objective, and data-driven orchard management contributing not only to the advancement of agricultural technology but also to the broader goals of food security and sustainable development.

REFERENCES

[1] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," Computers and Electronics in Agriculture, vol. 147, pp. 70–90, Apr. 2018. doi: 10.1016/j.compag.2018.02.016.

[2] I. Sa et al., "DeepFruits: A Fruit Detection System Using Deep Neural Networks," Sensors, vol. 16, no. 8, p. 1222, Aug. 2016. doi: 10.3390/s16081222.

[3] Food and Agriculture Organization of the United Nations (FAO), The Future of Food and Agriculture – Trends and Challenges. Rome: FAO, 2017. doi: 10.4060/I6583E.

[4] W. Liu et al., "SSD: Single Shot MultiBox Detector," in Computer Vision – ECCV 2016, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[6] J. Behmann, J. K. Mahns, A.-K. Mahlein, L. Plümer, and E.-C. Oerke, "A critical review of machine learning methods for the automated analysis of plant phenotyping data from imaging," Frontiers in Plant Science, vol. 6, p. 1197, Jan. 2016. doi: 10.3389/fpls.2015.01197.

[7] P. J. G. G. de With, "Export and quality standards for agricultural products: a review of international and national regulations," Journal of Food Quality, vol. 35, no. 3, pp. 210-225, 2012. (Suggested Replacement for [7])

[8] S. Bargoti and J. P. Underwood, "Deep Fruit Detection in Orchards," in 2017 IEEE International Conference on Robotics and Automation (ICRA), May 2017, pp. 3626–3633. doi: 10.1109/ICRA.2017.7989417.

[9] FAO, "Rural-to-urban migration and its implications for agriculture: A global review," FAO Agricultural Development Economics Working Paper, No. 18-03, 2018. (Suggested Replacement for [9])

[10] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. (Suggested Replacement for [10] and [19])

[11] M. A. A. Kader, "Postharvest technology of horticultural crops: an overview from farm to fork," American Journal of Food Science and Nutrition, vol. 1, no. 1, pp. 1-22, 2013. (Suggested Replacement for [11])

[12] H. Liu et al., "A review of fruit detection and counting for orchard yield mapping," Precision Agriculture, vol. 22, pp. 345-378, 2021. doi: 10.1007/s11119-020-09743-3.

[13] H. Liu et al., "Stages of fruit development and ripening in apricot fruits (Prunus armeniaca)," International Journal of Molecular Sciences, vol. 22, no. 1, p. 169, 2021. doi: 10.3390/ijms22010169.

[14] T. M. Mitchell, Machine Learning. McGraw-Hill, 1997.

[15] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," Frontiers in Plant Science, vol. 7, p. 1419, Sep. 2016. doi: 10.3389/fpls.2016.01419.

[16] X. Zhu and A. B. Goldberg, "Introduction to Semi-Supervised Learning," Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 3, no. 1, pp. 1–130, May 2009. doi: 10.2200/S00196ED1V01Y200906AIM006.

[17] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," Journal of Artificial Intelligence Research, vol. 4, pp. 237–285, May 1996. doi: 10.1613/jair.301.

[18] S. Tmiri et al., "Application of Machine Learning Algorithms in Precision Agriculture: A Review," Agricultural Reviews, vol. 42, no. 2, pp. 123-132, 2021.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, May 2015. doi: 10.1038/nature14539.

[20] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," Computers and Electronics in Agriculture, vol. 145, pp. 311–318, Feb. 2018. doi: 10.1016/j.compag.2018.01.009.

[21] P. Jiang, Y. Chen, B. Liu, D. He, and C. Cen, "An Effective Fruit Detection Method for Vision-Based Precision Agriculture Using Deep Convolutional Neural Network," Sensors, vol. 20, no. 15, p. 4321, Aug. 2020. doi: 10.3390/s20154321.

[22] E. Tjoa and C. Guan, "A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 11, pp. 4793–4813, Nov. 2021. doi: 10.1109/TNNLS.2020.3027314.

[23] Z. C. Lipton, "The Mythos of Model Interpretability," Queue, vol. 16, no. 3, pp. 31-57, Jun. 2018. doi: 10.1145/3236386.3241340. (Suggested Replacement for [23])

[24] 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, May 2017. doi: 10.1145/3065386. (General CNN foundation)

[25] H. Fu, Y. Cheng, and G. Li, "Classification, object detection, and segmentation of plant diseases: A review," Computers and Electronics in Agriculture, vol. 199, p. 107147, Aug. 2022. doi: 10.1016/j.compag.2022.107147. (Source of Figure 1.6)

[26] R. Girshick, "Fast R-CNN," in 2015 IEEE International Conference on Computer Vision (ICCV), Dec. 2015, pp. 1440–1448. doi: 10.1109/ICCV.2015.169.

[27] J. Huang et al., "Speed/accuracy trade-offs for modern convolutional object detectors," in 2017 IEEE Conference on Computer Vision and

Pattern Recognition (CVPR), Jul. 2017, pp. 3296-3305. doi: 10.1109/CVPR.2017.351.

[28] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv preprint arXiv:2004.10934, Apr. 2020.

[29] M. Everingham, L. Van Gool, C. K. I. 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, Jun. 2010. doi: 10.1007/s11263-009-0275-4.

[30] S. Saponara, A. Elhanashi, and A. Gagliardi, "Real-Time Embedded Systems for Smart Agriculture Based on Edge AI," Sensors, vol. 21, no. 15, p. 5263, Aug. 2021. doi: 10.3390/s21155263.

[31] I. Tomić et al., "Thermal and Power-Aware AI Deployment on Edge Devices: Challenges and Design Guidelines," ACM Transactions on Embedded Computing Systems, vol. 22, no. 3, pp. 1–23, May 2023. doi: 10.1145/3587267.

[32] Raspberry Pi Foundation. (2023). Raspberry Pi 5 Product Brief. [Online]. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf. Accessed on: Jun. 27, 2025.

[33] NVIDIA Corporation. (2023). Jetson Nano Developer Kit User Guide. [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit. Accessed on: Jun. 27, 2025.

[34] Google Coral. (2023). Coral Dev Board Mini Datasheet. [Online]. Available: https://coral.ai/static/files/Coral-Dev-Board-Mini-datasheet.pdf. Accessed on: Jun. 27, 2025.

[35] N. P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," in Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17), Jun. 2017, pp. 1–12. doi: 10.1145/3079856.3080246.

[36] Ultralytics. (2024). YOLOv8 Documentation. [Online]. Available: https://docs.ultralytics.com. Accessed on: Jun. 27, 2025.

[37] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object Detection Using YOLO: Challenges, Architectural Evolutions, and Applications," Multimedia Tools and Applications, vol. 82, pp. 9723–9773, 2023. doi: 10.1007/s11042-022-13644-y. (Replacement for [37, 38])

[38] C. R. Cheng et al., "Recent Advances in Lightweight Object Detectors for Embedded Applications," Sensors, vol. 23, no. 6, p. 2893, Mar. 2023. doi: 10.3390/s23062893.

[39] J. Terven and D. Cordova-Esparza, "A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond," arXiv preprint arXiv:2304.08069, Apr. 2023.

[40] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," in Computer Vision – ECCV 2014, 2014, pp. 740–755. doi: 10.1007/978-3-319-10602-1_48.

[41] A. Torralba and A. A. Efros, "Unbiased Look at Dataset Bias," in CVPR 2011, Jun. 2011, pp. 1521–1528. doi: 10.1109/CVPR.2011.5995347.

[42] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," Journal of Big Data, vol. 6, no. 1, p. 60, Jul. 2019. doi: 10.1186/s40537-019-0197-0.

[43] C. G. Northcutt, L. Jiang, and I. L. Chuang, "Confident learning: Estimating uncertainty in dataset labels," Journal of Artificial Intelligence Research, vol. 70, pp. 1373-1411, Mar. 2021. doi: 10.1613/jair.1.12125.

[44] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep Learning is Robust to Massive Label Noise," arXiv preprint arXiv:1705.10694, May 2017.

[45] S. SUBUDHI AND K. KUMARI, "A FAST AND EFFICIENT LARGE-SCALE NEAR DUPLICATE IMAGE RETRIEVAL SYSTEM USING DOUBLE PERCEPTUAL HASHING," *Signal, Image and Video Processing*, vol. 18,

PP. 8565–8575, 2024. DOI: 10.1007/S11760-024-03490-W

[46] D. HARGREAVES, "ESSENTIAL BEST PRACTICES FOR CUSTOM OBJECT RECOGNITION TRAINING," *DB GALLERY BLOG*, 2024.