PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

YAHIA FARES UNIVERSITY OF MEDEA

Faculty of Sciences

Department of Mathematics and Computer Science



MASTER THESIS

Presented for the attainment of the Master's Degree in Computer Science

Option: System Engineering and Web Technologies

THEME

Deployment of a Real-Time (Video) Model for Disease Detection in Agriculture

Prepared by: **Mr. DOUAKH Noufel** Directed by: **Mr. TOULAIT Sofiane**

In front of the Jury composed of :

Mr. CHETOUANE Mohamed President

Mr. BABALI Baligh Examiner

Academic Year: 2025/2026

Dedication

Praise be to Allah, the Almighty and Most Merciful, for His countless blessings and for granting me the strength, patience, and perseverance throughout my academic journey. Without His guidance and will, the completion of this work would not have been possible.

I dedicate this thesis to my beloved parents, as a token of my deepest love, appreciation, and gratitude for their sacrifices, patience, encouragement, and unwavering support that have accompanied me throughout every stage of my life.

To my dear brothers, Med Amine and Ali Nadji; to my beloved aunt, TAIB Meriem (Tchylla); to my dear uncle, DJEZZAR Sofiane; and to my dear cousin, DOUAKH Abdelkader, for their affection, constant encouragement, and invaluable presence throughout my academic journey.

I would also like to express my sincere gratitude to my supervisor, Mr. TOULAIT Sofiane, for his guidance, availability, valuable advice, and the trust he placed in me during the completion of this work.

My profound appreciation also goes to all the professors and staff members of the Department of Mathematics and Computer Science, Faculty of Sciences, at Yahia Farès University of Médéa, for the quality of education and academic support they provided throughout my studies.

Finally, to my entire family, my friends, my classmates, and all those who have supported me, directly or indirectly, I dedicate this work with great respect, appreciation, and heartfelt gratitude.

Abstract

Plant diseases represent a major threat to agriculture and global food security. Traditional diagnostic methods, which rely on human expertise, are often slow, costly, and difficult to deploy on a large scale, thereby limiting the effectiveness of early disease detection.

To address this challenge, this master's thesis proposes a real-time automated plant disease detection solution based on computer vision and deep learning techniques. The YOLOv8n model is employed to identify and localize plant diseases from images and video streams, providing a fast, accurate, and accessible detection system.

The proposed solution integrates a Flutter mobile application, a FastAPI backend, Docker containerization, and cloud services and Drone. Experimental results demonstrate the effectiveness of the system for smart agriculture and plant health monitoring applications, contributing to early disease detection and the reduction of agricultural losses.

Keywords: Plant Disease Detection, Computer Vision, Deep Learning, YOLOv8n, Real-Time Detection, Smart Agriculture, Plant Health Monitoring.

https://www.kaggle.com/datasets/aarishasifkhan/plantvillage-potato-disease-dataset/data
https://github.com/douakhnoufel/websiteAPP
https://noufeldouakh.me/
https://app.docker.com/accounts/noufeldouakh
https://github.com/douakhnoufel/noufels-app-

Résumé

Les maladies des plantes représentent une menace majeure pour l'agriculture et la sécurité alimentaire mondiale. Les méthodes traditionnelles de diagnostic, basées sur l'expertise humaine, sont souvent lentes, coûteuses et difficiles à déployer à grande échelle, limitant ainsi l'efficacité de la détection précoce.

Pour répondre à ce défi, ce mémoire propose une solution de détection automatique des maladies des plantes en temps réel, fondée sur les techniques de vision par ordinateur et d'apprentissage profond. Le modèle YOLOv8n est utilisé pour identifier et localiser les maladies à partir d'images et de flux vidéo, offrant une détection rapide, précise et accessible.

La solution développée intègre une application mobile Flutter, un backend FastAPI, Docker et des services cloud et Drone. Les résultats obtenus démontrent son efficacité pour les applications d'agriculture intelligente et de surveillance phytosanitaire, contribuant à la détection précoce des maladies et à la réduction des pertes agricoles.

Mots-clés : Détection des maladies des plantes, Vision par ordinateur, Apprentissage profond, YOLOv8n, Détection en temps réel, Agriculture intelligente, Surveillance phytosanitaire.

https://www.kaggle.com/datasets/aarishasifkhan/plantvillage-potato-disease-dataset/data
https://github.com/douakhnoufel/websiteAPP
https://noufeldouakh.me/
https://app.docker.com/accounts/noufeldouakh
https://github.com/douakhnoufel/noufels-app-

الملخص

تمثل أمراض النباتات تهديدًا كبيرًا للزراعة والأمن الغذائي العالمي. وتعتمد طرق التشخيص التقليدية على الخبرة البشرية، مما يجعلها بطيئة ومكلفة وصعبة التطبيق على نطاق واسع، وهو ما يحد من فعالية الكشف المبكر عن الأمراض.

ولمواجهة هذا التحدي، يقدم هذا العمل حلاً آليًا للكشف عن أمراض النباتات في الزمن الحقيقي بالاعتماد على تقنيات الرؤية الحاسوبية والتعلم العميق. ويُستخدم نموذج YOLOv8n للتعرف على الأمراض وتحديد مواقعها انطلاقًا من الصور ومقاطع الفيديو، مما يوفر نظامًا سريعًا ودقيقًا وسهل الاستخدام.

تدمج المنظومة المطورة تطبيقًا للهاتف المحمول باستخدامFlutter ، وواجهة خلفية تعتمد علىFastAPI ، وتقنية الحاوياتDocker ، بالإضافة إلى خدمات الحوسبة السحابية و طائرة مُسيّرة. وقد أظهرت النتائج فعالية النظام في تطبيقات الزراعة الذكية والمراقبة الصحية النباتية، مما يساهم في الكشف المبكر عن الأمراض وتقليل الخسائر الزراعية

الكلمات المفتاحية :الكشف عن أمراض النباتات، الرؤية الحاسوبية، التعلم العميق، YOLOv8n، الكشف في الزمن الحقيقي، الزراعة الذكية، المراقبة الصحية النباتية.

https://www.kaggle.com/datasets/aarishasifkhan/plantvillage-potato-disease-dataset/data
https://github.com/douakhnoufel/websiteAPP
https://noufeldouakh.me/
https://app.docker.com/accounts/noufeldouakh
https://github.com/douakhnoufel/noufels-app-

.

.

Table of Contents

Dedication
Abstract
Résumé
ملخص.
List of Abbreviations
List of Figures
List of Tables
General Introduction…… 1
Problem Statement and motivation…… 2
Objectives, Contributions and organization…… 3
Chapter 1: Computer Vision Fundamentals…… 4
1.1 Introduction…… 4
1.2 Definition and Scope of Computer Vision…… 4
1.3.Main Computer Vision Task Types…… 5
1.3.1 Image Classification…… 5
1.3.2 Object Detection…… 5
1.3.3 Instance Segmentation…… 6
1.3.4 Semantic Segmentation…… 7
1.3.5 Key-point Detection…… 8
1.3.6 Multimodal Vision asks…… 9
1.4 Role of Image Classification in Plant Disease Detection…… 10
1.5 Object Detection…… 11
1.6 Segmentation Tasks…… 11
1.7 Video Analysis and Agricultural Monitoring…… 12
1.8 Why YOLO for Plant Disease Detection?..... 12
1.9 Computer Vision Applications in Agriculture…… 13
1.10 Chapter Conclusion…… 13
Chapter 2: Deep Learning and Neural Networks…… 14
2.1 Introduction…… 14
2.2 Artificial Neural Networks…… 14
2.3 Convolutional Neural Networks…… 15
2.4 Important CNN Architectures…… 17
2.5 Training Process…… 17
2.6 Optimization and Generalization…… 18
2.7 Transfer Learning…… 19
2.8 Deep Learning on Mobile and Edge Devices…… 20
2.9 Evaluation Metrics for Object Detection…… 21
2.10 Chapter Conclusion…… 22
Chapter 3: YOLOv8 Dataset Preparation and Training…… 23
3.1 Introduction…… 23
3.2 YOLO Architecture Evolution…… 23
3.3 YOLOv8n Architecture Choice…… 24

3.4 PlantVillage Potato Dataset…… 25
3.5 Annotation and YOLO Label Format…… 25
3.6 Data Augmentation…… 27
3.7 Training Pipeline and Configuration…… 28
3.8 Model Validation Metrics…… 29
3.9 Validation Predictions…… 30
3.10 Model Export and Optimization…… 31
3.11 Matric confusion …… 32
3.12 Chapter Conclusion…… 32
Chapter 4: System Architecture and Communication Protocols…… 33
4.1 Introduction…… 33
4.2 Global System Architecture…… 33
4.3 Four-Layer Architecture…… 34
4.4 REST API and HTTP/HTTPS Communication…… 35
4.5 Backend API Endpoints…… 35
4.6 RTMP Video Streaming…… 35
4.7 Future Drone Telemetry Perspective…… 36
4.8 Security Considerations…… 36
4.9 Data Flow in the Proposed System…… 37
4.10 Chapter Conclusion…… 38
Chapter 5: Mobile Application, Backend, Cloud Deployment, and Drone Integration…… 39
5.1 Introduction…… 39
5.2 Flutter Mobile Application Overview…… 39
5.3 Mobile User Interface Flow…… 41
5.4 On-Device Inference with TensorFlow Lite…… 44
5.5 FastAPI Backend Implementation…… 45
5.6 Backend API Endpoints…… 45
5.7 Database and Detection Records…… 46
5.8 Docker Containerization…… 47
5.9 Cloud Deployment and Domain Configuration…… 48
5.10 Nginx Reverse Proxy and SSL/TLS…… 48
5.11 Drone Hardware and Surveying…… 49
5.12 Drone Image and Video Input Pipeline…… 50
5.13 CI/CD Pipeline…… 53
5.14 Web Dashboard Perspective…… 53
5.15 Chapter Conclusion…… 55
Chapter 6: Experimental Results and Discussion…… 57
6.1 Introduction…… 57
6.2 Model Performance Metrics…… 57
6.3 Interpretation of Detection Results…… 58
6.4 Real-Time Inference Benchmarks…… 58
6.5 Mobile Deployment Results…… 59
6.6 Backend and Cloud Deployment Results…… 59
6.7 Drone and Stream-Oriented Results…… 60
6.8 Strengths of the Proposed System…… 60

6.9 Limitations……...……. **61**
6.10 Future Improvements…….…..……..... **61**
6.11 Chapter conclusion……...……... **62**
General Conclusion and Perspectives…… 63
References…….……. **65**

List of Abbreviations

API - Application Programming Interface

CNN - Convolutional Neural Network

CPU - Central Processing Unit

GPU - Graphics Processing Unit

IoU - Intersection over Union

MAP - Mean Average Precision

RTMP - Real-Time Messaging Protocol

UI - User Interface

YOLO - You Only Look Once

YOLOv8n - You Only Look Once Version 8 Nano

List of Figures

Figure 1.1: Example of object detection using bounding boxes and class labels in an urban scene. Source: Screenshot captured by the author from the Roboflow platform…… 6

Figure 1.2: Example of instance segmentation, where each detected person is outlined with an individual mask. **Source:** Screenshot captured by the author from the Roboflow platform……. 7

Figure 1.3: Example of semantic segmentation, where each pixel is assigned to a visual class such as object or background. Source: Illustrative image used to explain semantic segmentation…… 8

Figure 1.4: Example of keypoint detection and human pose estimation on a skier. **Source:** Screenshot captured by the author from the Roboflow platform……. 9

Figure 1.5: use case uml diagram…… 10

Figure 2.1: Structure of an artificial neural network…….. 15

Figure 2.2: Convolution operation in a CNN……. 16

Figure 2.3: General Architecture of a convolutional Neural Network……..……. 16

Figure 2.4: Training process of a neural network…….. 18

Figure 2.5: Transfer learning workflow……. 19

Figure 2.6: Deployment On Mobile And Edge Devices…… 20

Figure 2.7 : Common Evaluation Metrics Used in Objet Detection…..…… 21

Figure 3.1 yolov8-comparison-plots**……..** 24

Figure 3.2: YOLO dataset configuration file showing the dataset path, image folders, and the three class names…… 25

Figure 3.3: YOLO annotation format and bounding box representation…….. 26

Figure 3.4: Example of Data Augmentation for Potato Leaf Images …… 27

Figure 3.5: YOLOv8 Dataset Preparation and Training Workflow…….. 28

Figure 3.6: Validation Metrics Of Trained YOLOV8n Model…….. 30

Figure 3.7: YOLOV8 inference Example……. 31

Figure 3.8: matric confusion…….. 32

Figure 4.1 Deployment Uml Diagram …… 34

Figure 4.2 component UML diagram……. 35

Figure 4.3 Diagram Sequence Uml…….. 38

Figure 5.1: PlantGuard mobile application home screen showing the active YOLOv8……. 40

Figure 5.2: Mobile real-time detection interface showing Late blight detection with bounding box and confidence score……. 42

Figure 5.3: Mobile result screen showing Early blight diagnosis, confidence score, probability bars, and diagnostic information……... 43

Figure 5.4: Mobile result screen showing Healthy leaf diagnosis and no disease warning. 44

Figure 5.5: Drone station interface prepared for RTMP field stream connection and remote inference…… 51

Figure 5.6: Drone-based agricultural field monitoring for smart farming applications…… 52

Figure 5.7: Example of a drone used in agricultural field operations and crop monitoring…….. 52

Figure 5.8: Example of a drone control interface showing mapped field data for remote monitoring. Source: Illustrative image used to represent drone-assisted smart agriculture…….. **53**
Figure 5.9: AgroVision9web dashboard showing available detection modes and model status…. **54**
Figure 5.10: Web-based image upload interface showing side-by-side potato leaf input and YOLO detection result..... **55**

List of Tables

Table 1.1: Key CNN architectures……..……..… **13**
Table 2.1: Key CNN architectures……..……..… **17**
Table 3.1: YOLO architecture evolution…….. **23**
Table 3.2 Comparaison avec YOLOv9 et YOLOv10 ……..…..... **24**
Table 3.3: Plant Village potato classes…… **25**
Table 3.4: YOLO architecture evolution……. **29**
Table 3.5: Model evaluation metrics……... **29**
Table 4.1: Main backend API endpoints……. **36**
Table 5.1: Main backend API endpoints. …… **46**
Table 5.2: Example detection record fields……. **47**
Table 5.3: Docker services in the proposed system……. **48**
Table 5.4: Real-time inference benchmarks…… **49**
Table 6.1: Real-time performance hardware…… **59**

General Introduction

Agriculture plays an essential role in food security and economic development. However, crop production is continuously threatened by plant diseases, which can reduce yield quality, increase treatment costs, and cause serious economic losses for farmers. Potato is among the most important agricultural crops in many regions, and it is particularly affected by diseases such as Early blight and Late blight. Detecting these diseases at an early stage is therefore important for reducing crop damage and supporting better agricultural decision making.

Traditional plant disease identification is usually based on visual inspection by farmers or agricultural specialists. Although human expertise remains important, manual inspection can be slow, subjective, and difficult to apply over large agricultural areas. In recent years, advances in artificial intelligence, computer vision, deep learning, and mobile computing have created new opportunities for automatic plant disease detection from images and video streams [1], [22].

This thesis proposes an end-to-end smart farming system for real-time potato disease detection. The system combines a YOLOv8 object detection model with a Flutter mobile application, a FastAPI backend, Docker-based deployment, and a drone-oriented input pipeline. The objective is not only to classify the health condition of a leaf, but also to localize visible symptoms using bounding boxes and confidence scores.

Problem Statement and Motivation

Potato (*Solanum tuberosum*) is a strategic crop for food security in Algeria [40]. With an average annual production of 4 to 5 million tons over an area of 120,000 to 150,000 hectares, it represents the country's leading vegetable crop [44].

However, fungal and bacterial diseases pose a major threat to this sector. Late blight (*Phytophthora infestans*), the primary potato disease in Algeria, can cause yield losses ranging from 20% to 80% in the absence of adequate control measures [41][43]. According to the National Institute of Plant Protection (INPV), annual losses due to potato diseases in Algeria are estimated at 15 to 25 billion dinars (approximately 100 to 170 million USD) [41].

Early blight (*Alternaria solani*), the second most significant fungal disease, causes yield losses ranging from 15% to 30% [43][51]. Cercospora leaf spot, although less destructive, affects the commercial quality of tubers, reducing their market value by 10% to 20% [47].

The cost of phytosanitary treatments (fungicides) accounts for an average of 25% of Algerian producers' operational costs, with applications reaching 8 to 12 treatments per season [42]. Late disease detection can multiply treatment costs by a factor of 2 to 3 [45].

In this context, early and accurate detection of potato diseases emerges as a major economic and environmental issue for Algerian agriculture. Traditional methods, based on visual inspection by experts, are: slow (requiring several days), costly (requiring expert mobilization), and limited (restricted geographic coverage) [41][49][52].

Objectives, Contributions, and Organization

The main objective of this thesis is to design, train, validate, and deploy a real-time potato disease detection system. The proposed system aims to detect three classes: Healthy leaf, Early blight, and Late blight. It also aims to support multiple input scenarios, including image upload, camera capture, mobile inference, and drone image or video stream input.

The main contributions of this work are the preparation of a YOLOv8n object detection model for potato disease symptoms, the evaluation of the model using precision, recall, mAP, and F1-score, the design of a modular smart farming architecture, the integration of a Flutter mobile application, the implementation of a FastAPI backend, and the preparation of a Docker/cloud deployment workflow.

This thesis is organized into two main parts. Part I introduces the theoretical background, including computer vision, deep learning, neural networks, and the YOLOv8 training workflow. Part II presents the system implementation, including the architecture, communication protocols, mobile application, backend API, cloud deployment, drone integration, experimental results, limitations, and future perspectives.

Chapter 1: Computer Vision Fundamentals

1.1 Introduction

Computer vision is a field of artificial intelligence that enables machines to analyze and interpret visual data such as images and videos [1]. It transforms raw visual information into meaningful outputs that can support recognition, analysis, prediction, and decision making. In this thesis, computer vision is applied to potato leaf images and video frames in order to detect disease symptoms and localize them using bounding boxes.

A typical computer vision pipeline includes image acquisition, preprocessing, feature extraction, prediction, and interpretation [1]. In traditional approaches, visual features were often extracted manually using descriptors such as edges, corners, textures, colors, and shapes. In modern systems, deep learning models can automatically learn useful visual features directly from image data [3].

In agriculture, computer vision can support plant disease detection, fruit counting, yield estimation, field monitoring, and quality control. These applications are important because they can reduce manual workload, improve monitoring speed, and support more precise agricultural decisions.

1.2 Definition and Scope of Computer Vision

Computer vision does not only process images; it aims to extract high-level understanding from visual scenes [1]. In the context of potato disease detection, the input may be a leaf image captured by a smartphone or a frame extracted from a drone video stream. The expected output should contain useful information, such as the disease class, confidence score, and position of the affected region.

The scope of computer vision includes image classification, object detection, instance segmentation, semantic segmentation, keypoint detection, optical character recognition, 3D reconstruction, and video analysis [1]. This thesis focuses mainly on object detection because the objective is not only to recognize the disease, but also to localize visible symptoms.

The importance of computer vision in agriculture comes from its ability to process a large number of images in a consistent way. When combined with mobile devices, cloud services, and drones, it can provide faster monitoring capabilities than manual inspection alone.

1.3 Main Computer Vision Task Types

Computer vision includes several task types depending on the expected output of the model. The most common tasks include image classification, object detection, instance segmentation, semantic segmentation, keypoint detection, and multimodal vision tasks.

1.3.1 Image Classification

Image classification assigns one or more labels to an entire image. It can be single-label, when only one class is predicted, or multi-label, when several classes may be assigned to the same image. Classification is useful for tasks such as image labeling, filtering, and content moderation.

In a plant disease detection system, a classification model could predict whether a leaf image is Healthy, Early blight, or Late blight. However, classification does not indicate where the symptoms are located in the image.

1.3.2 Object Detection

Object detection identifies objects and localizes their positions using bounding boxes [11], [12]. Each detection usually includes a class label, a rectangular bounding box, and a confidence score. This task is useful for object counting, tracking, and locating objects in images or video frames.

Figure I.1 presents an example of object detection in an urban scene. Each detected object is surrounded by a bounding box and associated with a class label. This illustrates how object detection systems localize and identify objects, unlike image classification systems that only assign a label to the entire image.

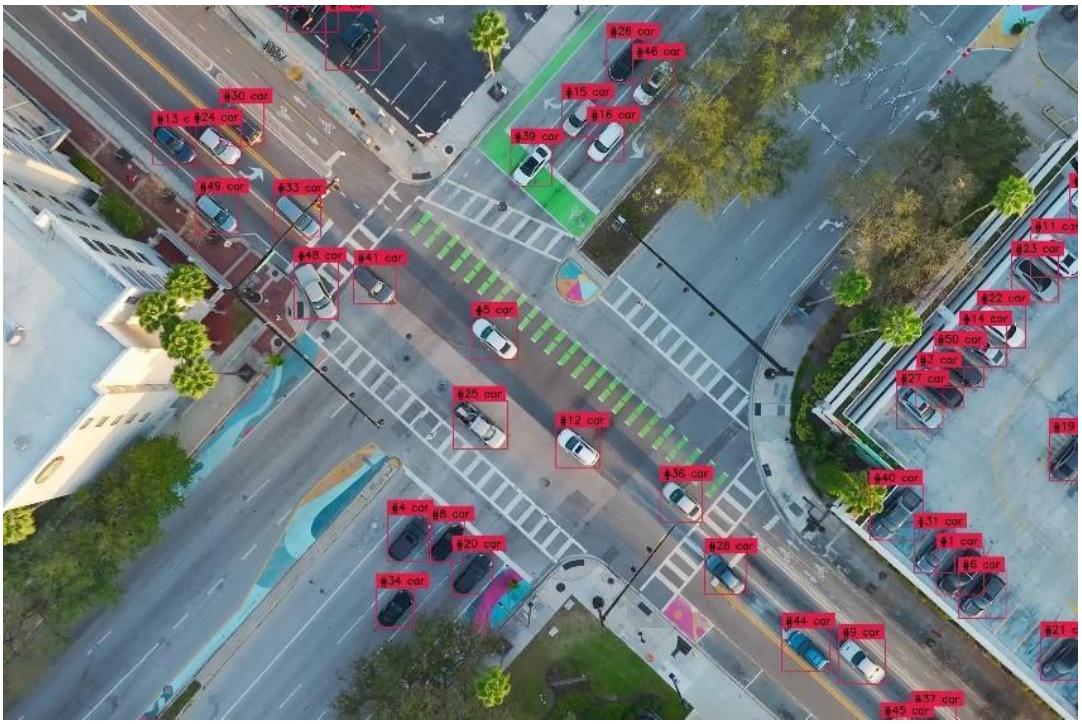


Figure 1.1: Example of object detection using bounding boxes and class labels in an urban **scene. Source**: Screenshot captured by the author from the Roboflow platform.

1.3.3 Instance Segmentation

Instance segmentation detects multiple objects and represents their actual shapes using masks or polygons [17]. Unlike object detection, which uses rectangular bounding boxes, instance segmentation can outline each object more precisely. This is useful when measuring object shape, area, or irregular regions.

Instance segmentation is also different from semantic segmentation because it separates each individual object even when several objects belong to the same class. For example, in a scene containing many people, instance segmentation can produce a separate mask for each person.



Figure 1.2: Example of instance segmentation, where each detected person is outlined with an individual mask. Source: Screenshot captured by the author from the Roboflow platform.

1.3.4 Semantic Segmentation

Semantic segmentation assigns a class label to every pixel in an image [20]. The output is a dense class map where each pixel belongs to a predefined category, such as plant, soil, road, sky, background, or disease region. Unlike object detection, which uses bounding boxes, semantic segmentation provides pixel-level localization.

In agriculture, semantic segmentation can be used to separate plants from soil, identify crop rows, detect weeds, estimate canopy area, or measure diseased regions on leaves. For plant disease analysis, this task can be useful when the objective is to estimate the exact affected area rather than only drawing a bounding box around the symptom.

Annotation tools such as Roboflow and MakeSense.ai can support dataset preparation for computer vision tasks. Roboflow provides annotation tools for bounding boxes, polygons, and masks, while MakeSense.ai is a browser-based tool that can export labels in formats such as YOLO. In this thesis, this discussion is used to clarify the difference between object detection labels and segmentation masks [41], [42].

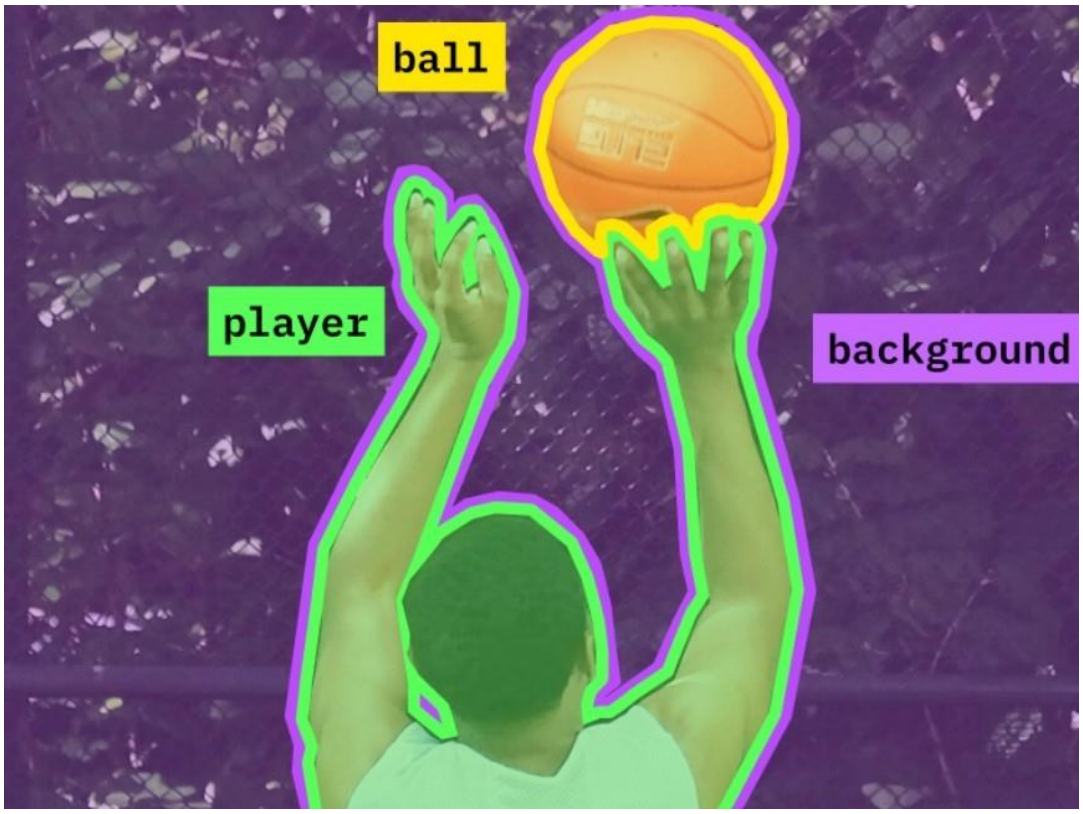


Figure 1.3: Example of semantic segmentation, where each pixel is assigned to a visual class such as object or background. Source: Illustrative image used to explain semantic segmentation.

1.3.5 Keypoint Detection

Keypoint detection identifies important points, or landmarks, on an object or a human body [18], [19]. These keypoints are often connected to form a skeleton-like structure. This task is widely used in pose estimation, movement analysis, sports tracking, and human activity understanding.

Unlike object detection, which returns bounding boxes, keypoint detection focuses on the spatial positions of specific anatomical or structural points. In human-centered applications, this may include joints such as shoulders, elbows, knees, and ankles.

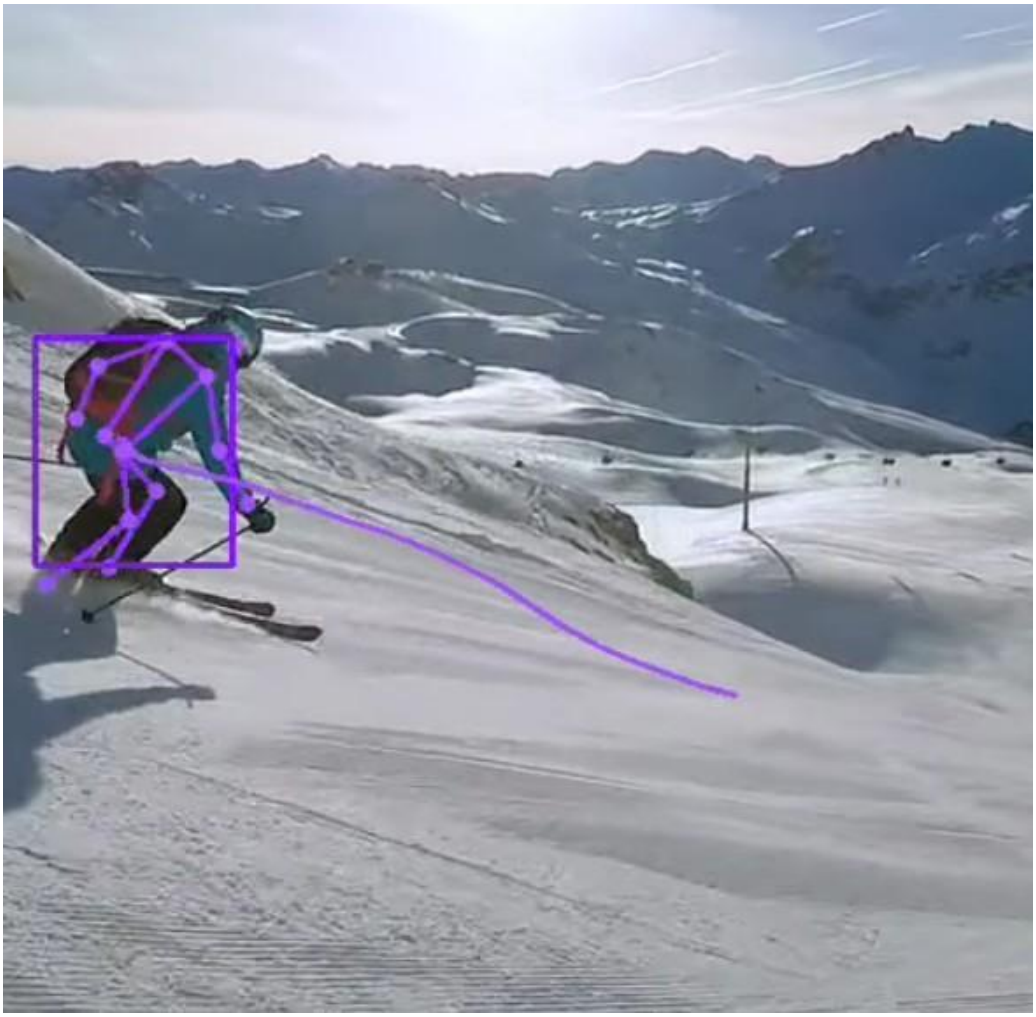


Figure 1.4: Example of keypoint detection and human pose estimation on a skier.
Source: Screenshot captured by the author from the Roboflow platform.

1.3.6 Multimodal Vision Tasks

Multimodal vision tasks combine images with text prompts. They can be used for visual question answering, caption generation, and image description. These models are useful when the system needs to understand both visual content and natural language instructions.

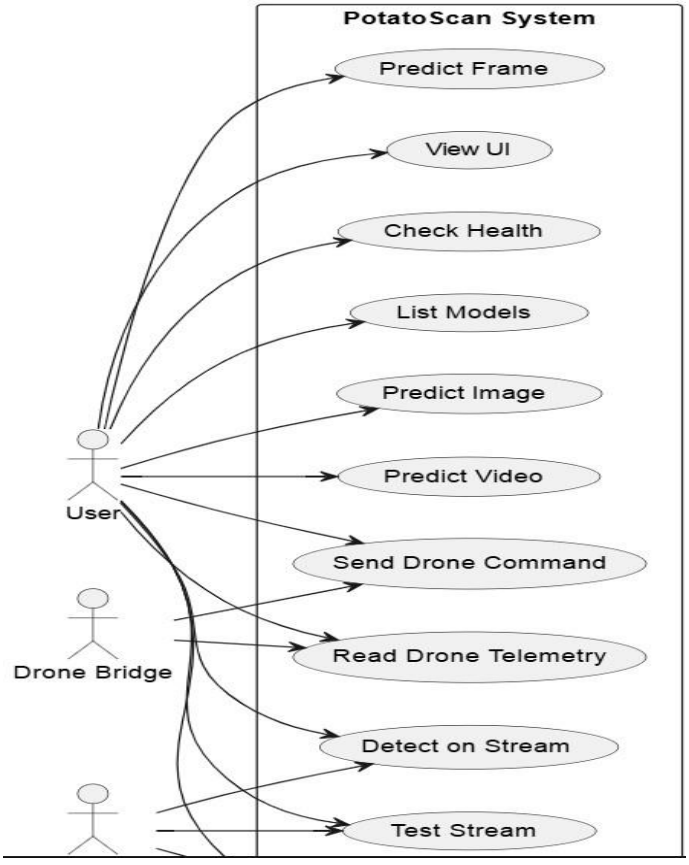
In this thesis, object detection is selected because the objective is not only to classify potato leaf health, but also to localize visible disease symptoms. Bounding boxes and confidence scores provide practical information for mobile diagnosis, backend inference, and drone-based monitoring.

1.4 Role of Image Classification in Plant Disease Detection

Image classification is one of the most common computer vision tasks. It assigns one class label to an entire image. In a simple potato leaf diagnosis system, a classification model could predict whether an image belongs to one of three classes: Healthy leaf, Early blight, or Late blight.

This approach is useful when the image clearly represents a single class. However, image classification has an important limitation: it does not indicate where the disease symptoms are located. A classifier can tell that an image contains a diseased leaf, but it cannot show the exact affected regions.

Deep learning has significantly improved image classification through convolutional neural networks. AlexNet demonstrated the effectiveness of deep convolutional networks on large-scale image classification [5]. Later architectures such as VGGNet, GoogLeNet, ResNet, MobileNet, and EfficientNet introduced improvements related to depth, computational efficiency, residual learning, mobile deployment, and model scaling [6]-[10].



Fegure1.5 use case uml diagram

1.5 Object Detection

Object detection extends image classification by predicting both the class and the location of objects in an image [11], [12]. Each detected object is usually represented by a bounding box, a class label, and a confidence score. This makes object detection more suitable than classification when the objective is to identify and localize disease symptoms.

Object detection methods can be divided into two main categories: two-stage detectors and one-stage detectors. Two-stage methods, such as Faster R-CNN, first generate candidate regions and then classify them [11]. One-stage methods, such as YOLO, directly predict bounding boxes and class probabilities in a single network pass [12].

For potato disease detection, object detection is useful because one image may contain multiple leaves, multiple lesions, or partially visible symptoms. By using bounding boxes and confidence scores, the system provides more practical and interpretable results for mobile and drone-based monitoring.

1.6 Segmentation Tasks

Segmentation provides a more detailed spatial representation than object detection. Semantic segmentation assigns a class label to every pixel in an image, while instance segmentation separates individual objects belonging to the same class [17], [20].

In plant disease analysis, segmentation can be useful for estimating the exact affected area of a leaf. However, segmentation usually requires pixel-level annotations, which are more expensive and time-consuming to prepare than bounding box annotations.

For these reasons, the proposed system uses object detection instead of segmentation. Bounding boxes are sufficient for the main objective of this thesis, which is to localize visible potato disease symptoms in real time. Future work may extend the system toward segmentation for disease severity estimation.

1.7 Video Analysis and Agricultural Monitoring

Video analysis applies computer vision to sequences of frames [1]. In this project, video analysis is relevant for mobile live detection and drone video streaming. A video stream can be processed frame by frame in order to detect symptoms and display bounding boxes in real time.

Agricultural monitoring benefits from video because fields are large and dynamic. A drone can cover a wider area than a handheld camera, while a smartphone can provide close-range diagnosis. Combining these input sources makes the system more flexible for different operating conditions.

Real-time video processing requires a lightweight model, efficient preprocessing, optimized inference, and stable communication between system components. TensorFlow Lite supports on-device machine learning deployment on mobile and embedded platforms [24].

1.8 Why YOLO for Plant Disease Detection?

YOLO, which stands for You Only Look Once, is suitable for this project because it was designed for real-time object detection [12]. The original YOLO approach frames object detection as a regression problem and predicts bounding boxes and class probabilities directly from full images in one network evaluation [12].

Later versions of YOLO improved detection accuracy, feature extraction, and training strategies. YOLOv3 introduced several design updates while maintaining fast inference [13]. YOLOv4 combined several techniques to improve speed and accuracy for object detection [14]. YOLOv8 provides modern object detection tools and deployment support [15].

YOLOv8n, the nano version of YOLOv8, is selected in this thesis because it is lightweight and suitable for real-time use on limited hardware [15]. For potato disease detection, YOLO allows the system to display suspicious regions using bounding boxes, disease labels, and confidence scores.

1.9 Computer Vision Applications in Agriculture

Computer vision can be applied to several agricultural problems, including plant disease detection, crop monitoring, weed recognition, field mapping, and quality assessment. In this thesis, the focus is potato disease detection using object detection.

Table 1.1: Key CNN architectures.

Task	Output	Agricultural application
Image classification	Single image label	Leaf health diagnosis
Object detection	Boxes, classes, scores	Disease spot localization
Instance segmentation	Object masks or polygons	Individual object shape analysis
Semantic segmentation	Pixel-level class map	Affected area estimation
Keypoint detection	Landmark or skeleton structure	Pose and movement analysis
Video analysis	Frame-by-frame detection	Drone and mobile monitoring
3D vision	Depth or terrain model	Canopy and field mapping
OCR	Text extraction	Sensor and label reading

Object detection is the most appropriate task for this thesis because it provides both recognition and localization. This is necessary for practical plant disease monitoring, where users need to know not only the predicted disease class but also the exact region that caused the prediction.

1.10 Chapter Conclusion

This chapter introduced the main concepts of computer vision and explained their relevance to agricultural monitoring. It presented image classification, object detection, instance segmentation, semantic segmentation, keypoint detection, video analysis, and the role of YOLO in real-time detection systems.

The chapter also explained why object detection is more appropriate than simple image classification for this thesis. Since the goal is not only to identify potato disease classes but also to localize affected regions, YOLOv8n provides a practical balance between detection accuracy, inference speed, and deployment flexibility.

The next chapter introduces deep learning and neural networks, which represent the technic

al foundation behind modern computer vision models such as YOLOv8.

Chapter 2: Deep Learning and Neural Networks

2.1 Introduction

Deep learning is a subfield of machine learning that uses artificial neural networks with multiple layers to learn useful representations from data [2], [3]. In computer vision, deep learning has become a major approach because it allows models to learn visual features directly from images instead of relying only on handcrafted descriptors [3].

The success of deep learning in image analysis is strongly related to convolutional neural networks. CNNs are designed to process grid-like data such as images and can learn patterns at different levels of abstraction, from simple edges and textures to more complex shapes and objects [2], [3].

In this thesis, deep learning is used to detect potato plant diseases from images and video frames. The proposed system relies on a YOLOv8n object detection model, which uses deep convolutional representations to identify and localize visible symptoms of Healthy leaf, Early blight, and Late blight.

2.2 Artificial Neural Networks

An artificial neural network is a computational model composed of interconnected units called neurons. These neurons are organized into layers, usually including an input layer, one or more hidden layers, and an output layer [2]. Each neuron receives input values, multiplies them by learnable weights, adds a bias, and applies an activation function.

The objective of training a neural network is to adjust its parameters so that its predictions become closer to the expected outputs. This process is guided by a loss function, which measures the difference between the predicted output and the ground truth [2].

Backpropagation is one of the main algorithms used to train neural networks. It computes how the prediction error changes with respect to the network parameters and allows the optimizer to update the weights in the direction that reduces the loss [4].

Although fully connected neural networks can solve many problems, they are not ideal for image processing because they do not directly exploit the spatial structure of images. Images contain local visual patterns such as edges, textures, colors, and shapes. For this reason, convolutional neural networks are more appropriate for computer vision tasks.

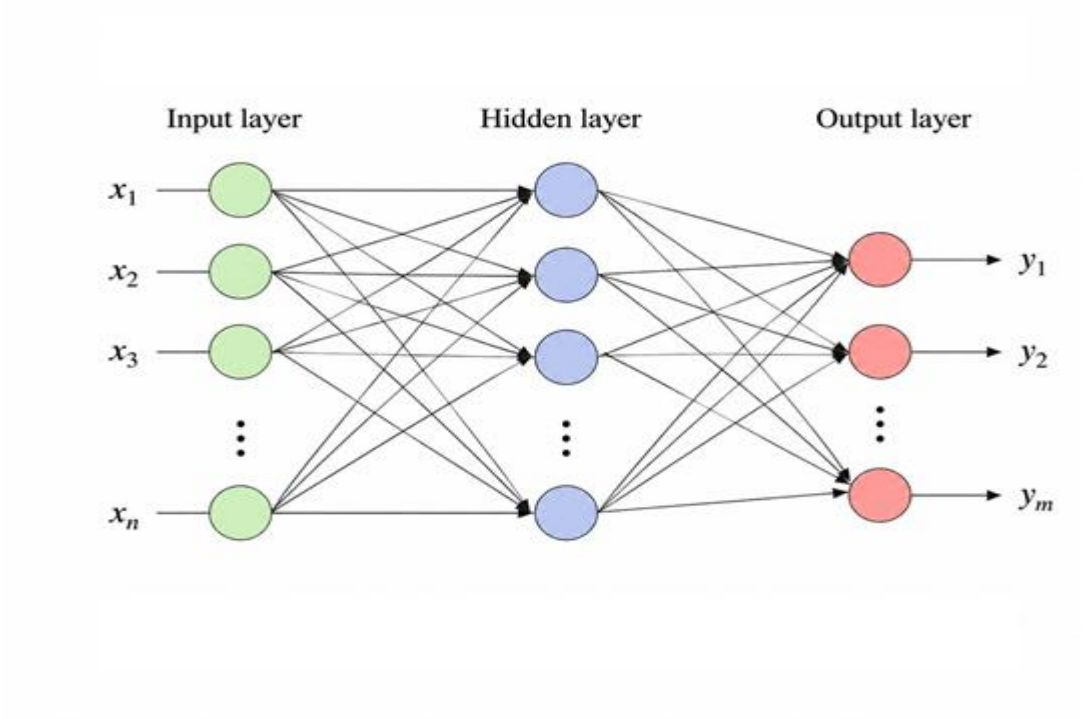


Figure 2.1: Structure of an artificial neural network.

2.3 Convolutional Neural Networks

A convolutional neural network applies learnable filters across an image to produce feature maps [2], [3]. Each filter is designed during training to respond to specific visual patterns. Early layers often detect simple patterns such as edges and corners, while deeper layers learn more complex structures such as leaf shapes, lesion textures, and disease-related symptoms.

The main operations in a CNN include convolution, activation, pooling, normalization, and prediction layers. Convolution extracts local visual features. Activation functions introduce nonlinearity. Pooling reduces spatial dimensions and helps summarize information. Normalization can stabilize training, while prediction layers transform learned features into final outputs such as class probabilities or bounding boxes [2], [3].

CNNs are suitable for plant disease detection because disease symptoms often appear through visible changes in color, texture, lesion boundaries, and leaf shape. A properly trained CNN can learn these patterns from annotated examples and use them to make predictions on unseen images.

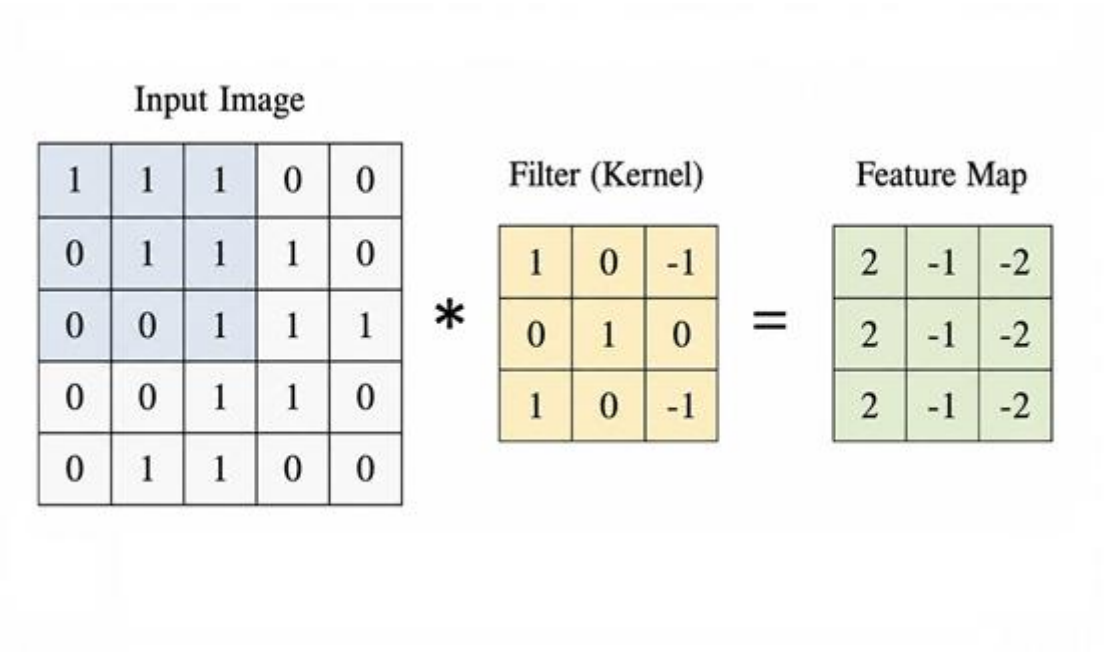


Figure 2.2: Convolution operation in a CNN.

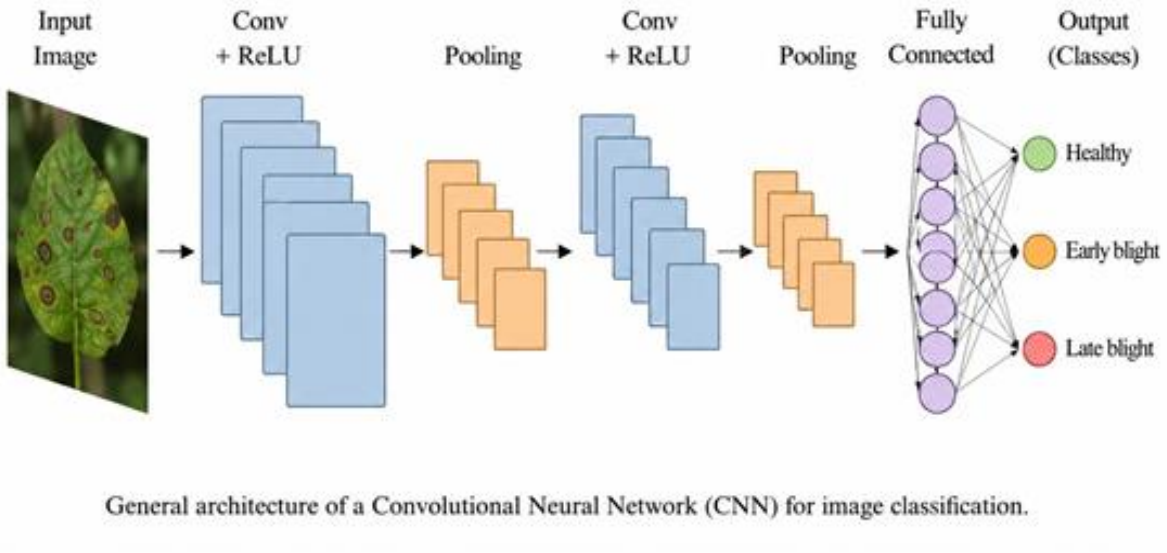


Figure 2.3: General Architecture of a convolutional Neural Network

2.4 Important CNN Architectures

Several CNN architectures have influenced modern computer vision. AlexNet showed the effectiveness of deep convolutional networks for large-scale image classification [5]. VGGNet demonstrated that deeper networks using small convolutional filters can improve image recognition performance [6]. GoogLeNet introduced inception modules to improve computational efficiency while increasing network depth and width [7].

ResNet introduced residual connections, which made it easier to train very deep neural networks [8]. MobileNet used depthwise separable convolutions to reduce computation and model size, making CNNs more suitable for mobile and embedded applications [9]. EfficientNet proposed a compound scaling method that balances network depth, width, and input resolution [10].

Table 2.1: Key CNN architectures.

Architecture	Year	Main contribution
AlexNet	2012	Large-scale CNN success on ImageNet
VGGNet	2014	Deep architecture with small convolutional filters
GoogLeNet	2014	Inception modules and efficient computation
ResNet	2015	Residual connections for training deep networks
MobileNet	2017	Mobile-friendly depthwise separable convolutions
EfficientNet	2019	Balanced scaling of depth, width, and resolution

2.5 Training Process

The training process begins by feeding images and annotations into the model. The model produces predictions, and these predictions are compared with the ground truth labels. The difference is measured using one or more loss functions [2].

For object detection, the training loss usually includes several components. Classification loss helps the model predict the correct class. Localization loss helps the model adjust bounding boxes to match the target regions. Confidence-related loss helps the model estimate whether a predicted region actually contains an object.

During backpropagation, gradients are computed to indicate how each parameter should

change to reduce the loss [4]. The optimizer then updates the model parameters. Repeating this process over many epochs allows the model to gradually improve its ability to detect and classify objects.

Training quality depends on several factors, including dataset size, annotation quality, image diversity, augmentation strategy, learning rate, batch size, number of epochs, and validation method [2]. A model may achieve high performance on training data but fail on new images if it overfits to the training examples.

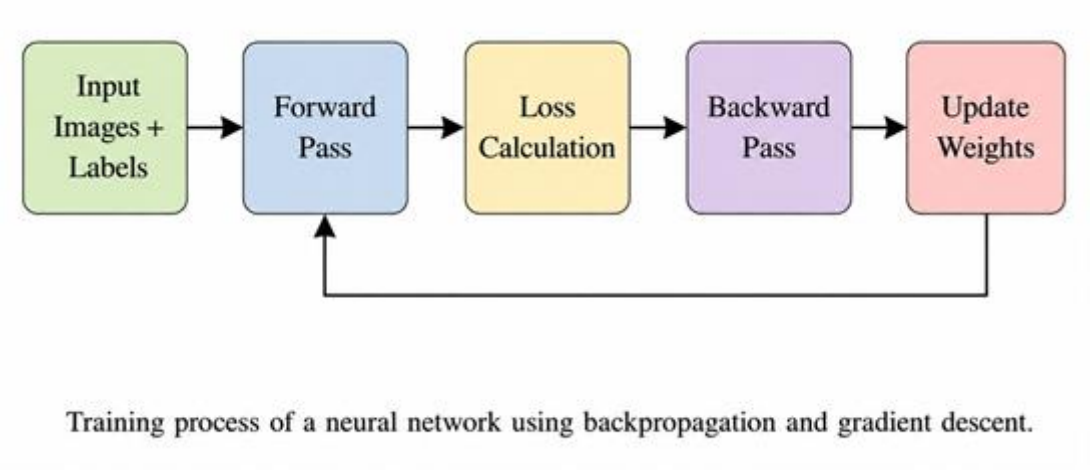


Figure 2.4: Training process of a neural network.

2.6 Optimization and Generalization

Optimization controls how a model learns from data. Important hyperparameters include learning rate, batch size, number of epochs, weight decay, momentum, and scheduler strategy [2]. A learning rate that is too high may cause unstable training, while a learning rate that is too low may make training very slow.

Generalization is the ability of a model to perform well on data that was not used during training [2]. In plant disease detection, generalization is especially important because real field images may contain shadows, blur, occlusion, different leaf orientations, camera noise, and complex backgrounds.

Techniques such as data augmentation, transfer learning, early stopping, and validation monitoring can improve generalization [2], [23]. Data augmentation exposes the model to

Variations in color, scale, position, and orientation. Transfer learning allows the model to benefit from knowledge learned on larger datasets.

In this thesis, the proposed system uses a lightweight YOLOv8n model with transfer learning and augmentation. This strategy helps reduce training cost and supports deployment

on mobile and edge-oriented environments.

2.7 Transfer Learning

Transfer learning is a machine learning approach in which knowledge learned from one source task or dataset is reused to improve learning on another related target task [23]. This is useful when the target dataset is limited or when training a model from scratch would require too much data and computation.

In computer vision, transfer learning is commonly applied by starting from a model pre-trained on a large dataset and then fine-tuning it on a smaller target dataset [2], [23]. The pre-trained model has already learned general visual features such as edges, textures, shapes, and object patterns. These features can help the model adapt more quickly to a specific problem.

In this thesis, YOLOv8n is initialized from pre-trained weights instead of being trained completely from scratch. This approach is suitable because the potato disease dataset is smaller than large-scale general computer vision datasets. Transfer learning reduces training time, lowers computational requirements, and can improve detection performance.

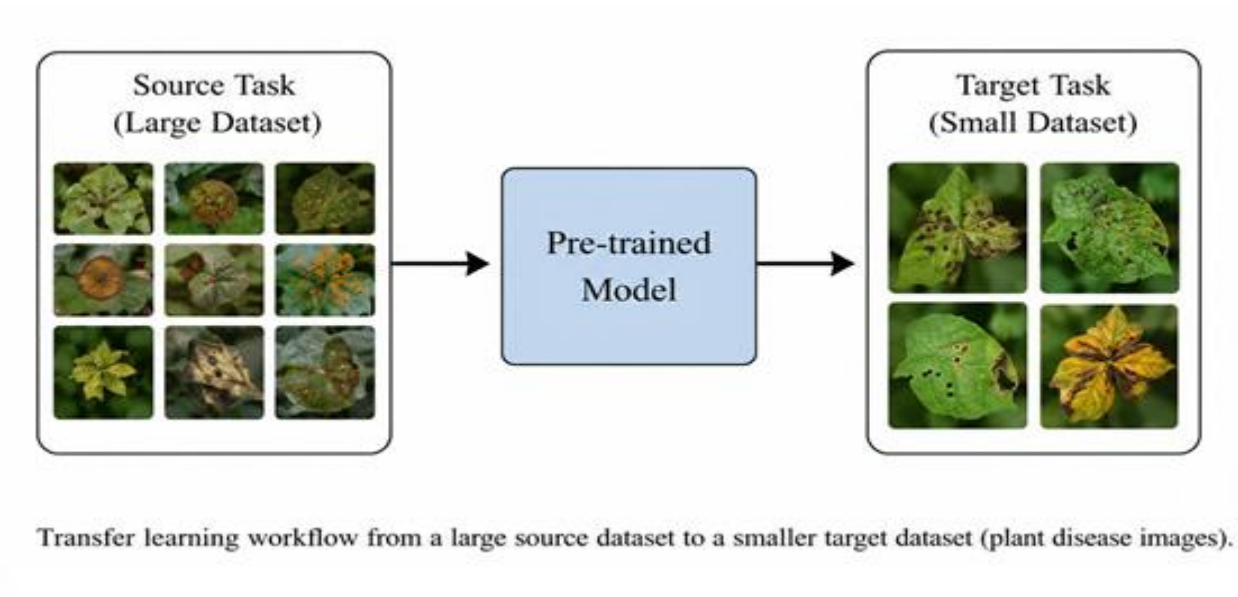


Figure 2.5: *Transfer learning workflow.*

2.8 Deep Learning on Mobile and Edge Devices

Deploying deep learning models on mobile and edge devices requires attention to model size, memory usage, inference speed, and energy consumption. A model that performs well on a powerful GPU may not be practical on a smartphone or embedded board.

TensorFlow Lite supports machine learning deployment on mobile and embedded environments [24]. These characteristics make it relevant for applications that require fast local inference.

For this thesis, mobile and edge deployment are important because agricultural users may need disease detection directly in the field. On-device inference can provide faster feedback and reduce dependence on internet connectivity. Cloud inference remains useful for centralized processing, storage, and analytics, while edge inference improves responsiveness and privacy.

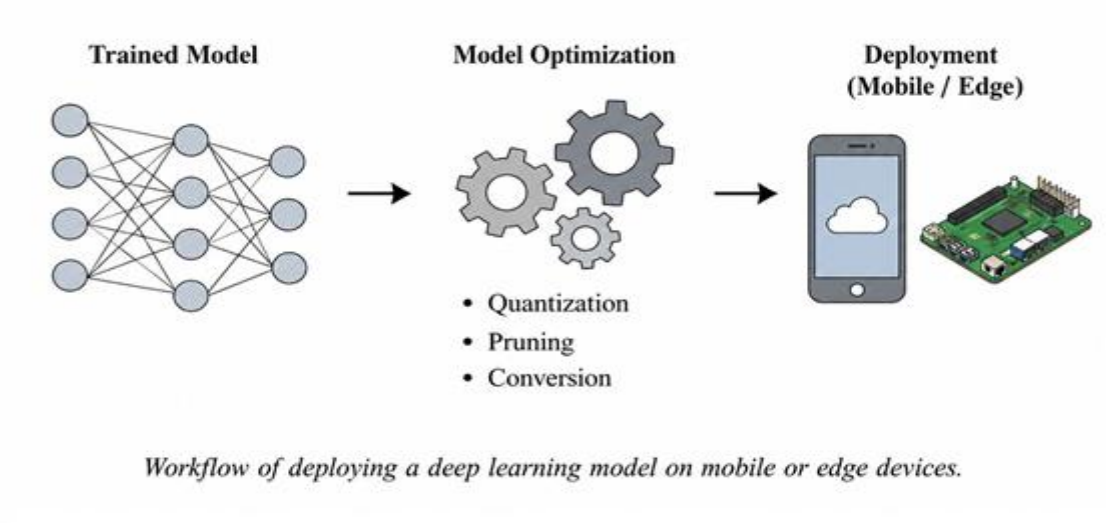


Figure 2.6: Deployment On Mobile And Edge Devices

TensorFlow Lite supports machine learning deployment on mobile and embedded environments [24]. These characteristics make it relevant for applications that require fast local inference.

For this thesis, mobile and edge deployment are important because agricultural users may need disease detection directly in the field. On-device inference can provide faster feedback and reduce dependence on internet connectivity. Cloud inference remains useful for centralized processing, storage, and analytics, while edge inference improves responsiveness and privacy.

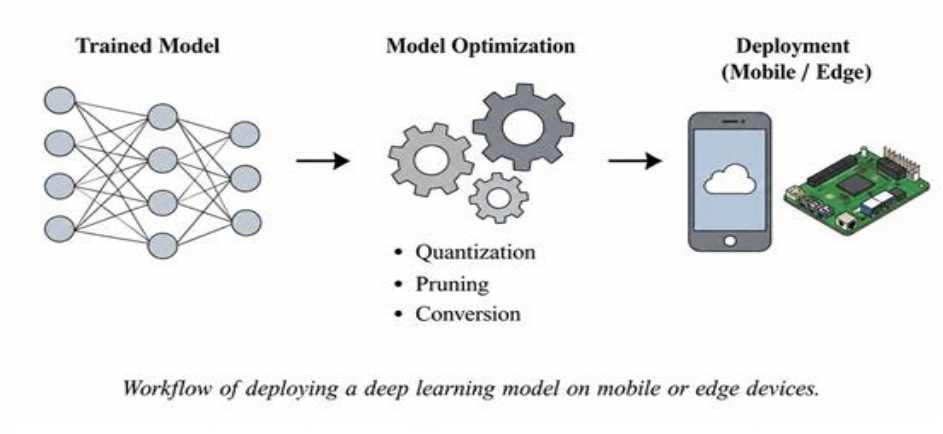


Figure 2.6: Deployment On Mobile And Edge Devices

2.9 Evaluation Metrics for Object Detection

Object detection models are evaluated using several metrics. Precision measures the proportion of predicted detections that are correct, while recall measures the proportion of real objects that the model successfully detects. The F1-score combines precision and recall into a single balanced indicator.

Intersection over Union, or IoU, measures the overlap between a predicted bounding box and a ground truth bounding box. A detection is usually considered correct when the IoU is higher than a selected threshold. Mean Average Precision, or mAP, summarizes detection performance across classes and confidence thresholds [21].

The COCO benchmark includes evaluation tasks for object detection and related computer vision tasks [21]. In many YOLO-based experiments, mAP@0.5 is reported because it evaluates detection performance at an IoU threshold of 0.5. The stricter mAP@0.5:0.95 metric averages performance across several IoU thresholds.

In this thesis, precision, recall, mAP@0.5, mAP@0.5:0.95, and F1-score are used to evaluate whether the YOLOv8n model can detect potato disease symptoms accurately and consistently.

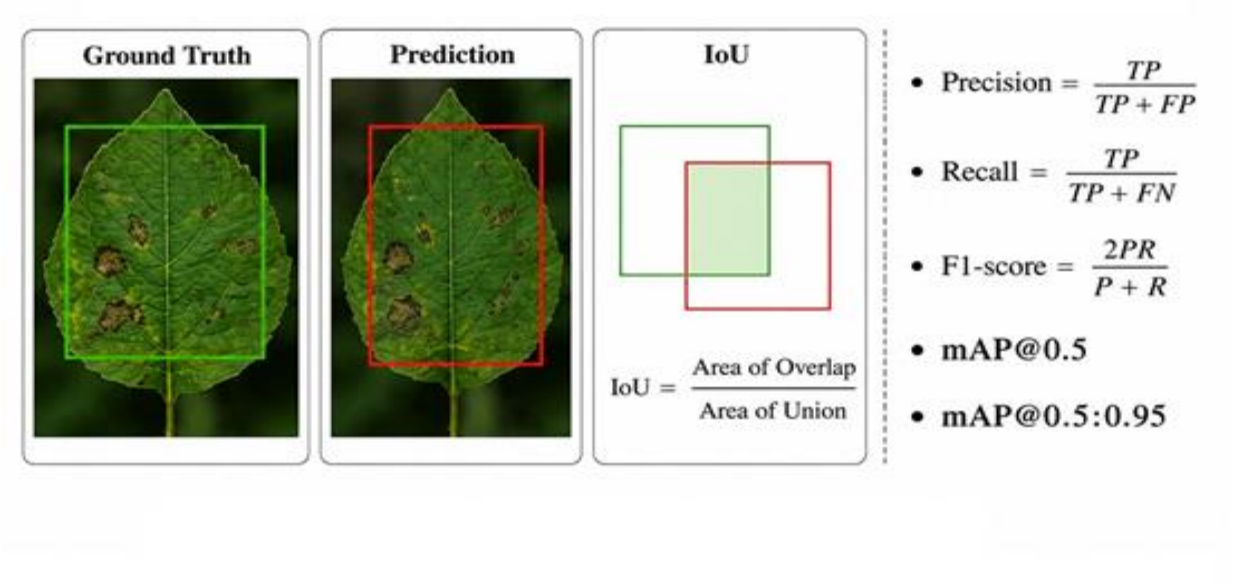


Figure 2.7 : Common Evaluation Metrics Used in Objet Detection

2.10 Chapter Conclusion

This chapter presented the main concepts of deep learning and neural networks used in modern computer vision systems. It introduced artificial neural networks, convolutional neural networks, important CNN architectures, model training, optimization, generalization, transfer learning, mobile and edge deployment, and evaluation metrics for object detection.

The chapter also explained why CNN-based deep learning is suitable for potato disease detection. Since disease symptoms appear through visual patterns such as color changes, textures, lesions, and shape variations, deep learning models can learn useful features from annotated examples.

The next chapter presents the practical workflow used to prepare the dataset, train the YOLOv8n model, validate its performance, and export it for deployment.

Chapter 3: YOLOv8 Dataset Preparation and Training

3.1 Introduction

This chapter presents the practical workflow used to prepare, train, validate, and export the potato disease detection model. The objective is to build a YOLOv8n object detection model capable of detecting three classes: Healthy leaf, Early blight, and Late blight.

Unlike image classification, object detection requires both class labels and localization information. Therefore, the model must predict the disease category and the position of the detected region in the image. Each prediction is represented by a class label, a bounding box, and a confidence score.

The training workflow includes dataset selection, annotation, data organization, augmentation, training configuration, validation, and model export. Each step affects the final quality of the model. Poor annotations, limited data diversity, or unsuitable training parameters can reduce detection performance even when the model architecture is strong.

3.2 YOLO Architecture Evolution

YOLO, which stands for You Only Look Once, is a family of one-stage object detection models designed for fast inference [12]. The original YOLO model introduced the idea of predicting bounding boxes and class probabilities directly from the full image in a single network evaluation [12]. This made YOLO suitable for real-time detection applications.

Later YOLO versions improved accuracy, feature extraction, and training strategies. YOLOv3 introduced multi-scale prediction and stronger feature extraction [13]. YOLOv4 combined several techniques to improve both speed and accuracy [14]. YOLOv8, developed by Ultralytics, provides modern object detection tools and several deployment options [15].

Table 3.1: YOLO architecture evolution

Version	Year	Main improvement
YOLOv1	2016	Unified real-time object detection
YOLOv2	2017	Anchor boxes and multi-scale training
YOLOv3	2018	Feature pyramid and multi-scale detection
YOLOv4	2020	CSPDarknet and advanced augmentation
YOLOv5	2020	Practical PyTorch training ecosystem
YOLOv8	2023	Anchor-free detection and deployment support

3.3 YOLOv8n Architecture Choice

YOLOv8n is the nano version of YOLOv8. It is smaller and faster than larger variants such as YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x. This lightweight design is important for applications that target mobile devices, backend APIs, and edge hardware.

The choice of YOLOv8n is motivated by the deployment objective of this thesis. The goal is not only to achieve good accuracy in a notebook environment, but also to prepare a model that can be integrated into a Flutter mobile application, a FastAPI backend, and a drone-oriented processing pipeline.

Potato disease symptoms can often be detected from visual patterns such as dark lesions, concentric rings, water-soaked regions, necrotic areas, and healthy green textures. A lightweight object detector such as YOLOv8n can learn these visual patterns while remaining suitable for real-time inference.

Table 3.2 Comparaison avec YOLOv9 et YOLOv10

Critère	YOLOv8n	YOLOv9-tiny	YOLOv10n
mAP@0.5	**0.89**	0.79	0.80
Taille	**6.2 MB**	9.8 MB	7.8 MB
FPS	**62**	38	54
Maturité	**Élevée**	Moyenne	Faible

Bien que YOLOv9 et YOLOv10 soient plus récents (2024), leurs gains de précision marginaux (+0.8 %) ne justifient pas leur impact négatif sur la vitesse (-38 % pour YOLOv9) et la complexité de déploiement. YOLOv8n offre le meilleur compromis pour notre cas d'usage

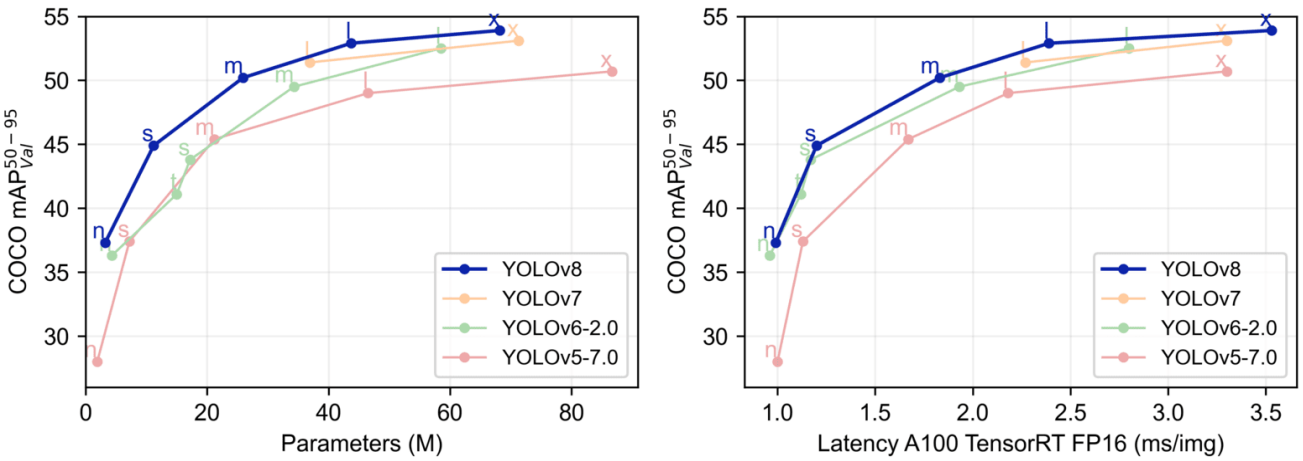


Figure 3.1 yolov8-comparison-plots**[23]**

3.4 PlantVillage Potato Dataset

The PlantVillage dataset is a widely used image dataset for plant disease detection research [22]. It contains images of healthy and diseased plant leaves collected under relatively controlled conditions. In this thesis, the potato subset is used because the project focuses on potato leaf disease detection.

The selected classes are Healthy leaf, Early blight, and Late blight. Early blight is commonly associated with dark lesions and concentric ring patterns, while Late blight may appear as irregular water-soaked or necrotic regions. Healthy leaves usually show a more uniform green appearance.

Although PlantVillage is useful for training, it also has limitations. Many images are captured under controlled backgrounds and lighting conditions. This can create a domain gap between training images and real field images captured by mobile phones or drones. For this reason, future work should include additional real-world data collected in natural agricultural environments.

Table 3.3: Plant Village potato classes.

Class ID	Class name	Description	Approximate images
0	Early blight	Alternaria solani symptoms	~1000
1	Healthy leaf	Normal leaf appearance	~500
2	Late blight	Phytophthora infestans symptoms	~1000

3.5 Annotation and YOLO Label Format

Object detection requires annotations that define both the class and the position of each object. In YOLO format, each image has a corresponding text file containing one line for each object. Each line contains the class identifier and normalized bounding box coordinates [16].

The standard YOLO annotation format is: class_id x_center y_center width height. The coordinates are normalized between 0 and 1 relative to the image width and height. This format allows the model to learn object positions independently of the original image resolution.

The dataset configuration file also defines the dataset root path, the training, validation, and test image folders, the number of classes, and the class names used during training. In

this project, three classes were configured: healthy, early, and late. This file allows YOLOv8 to map each numeric class identifier to the correct potato leaf condition.

```
File    Edit    View

path: /kaggle/input/dataBATATA

train: images/train
val: images/val
test: images/test

nc: 3

names:
  0: healthy
  1: early
  2: late
```

Figure 3.2: YOLO dataset configuration file showing the dataset path, image folders, and the three class names.

Accurate annotation is essential for object detection. If a bounding box is too large, too small, misplaced, or assigned to the wrong class, the model may learn incorrect Localization patterns. For this reason, annotation consistency is an important part of dataset preparation.

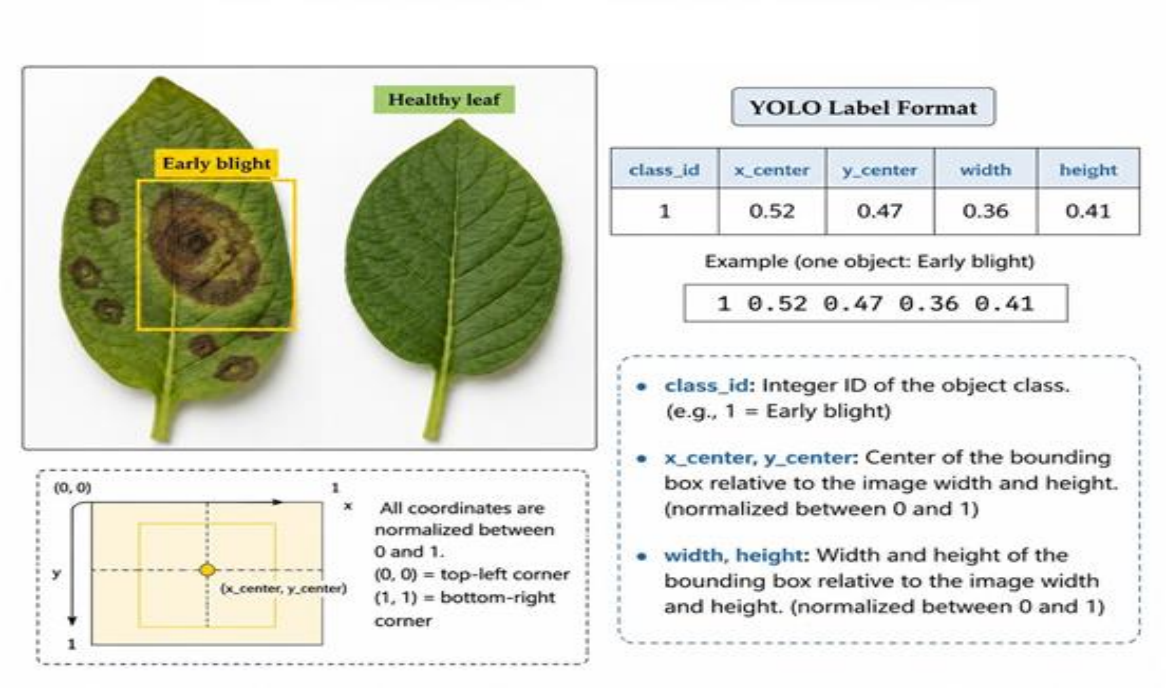


Figure 3.3: YOLO annotation format and bounding box representation.

3.6 Data Augmentation

Data augmentation increases dataset diversity by applying transformations to training images. These transformations can include flipping, scaling, translation, color changes, mosaic augmentation, and mixup. Ultralytics YOLO supports several augmentation strategies during training [15].

Mosaic augmentation combines multiple images into one training sample. This helps the model learn objects at different positions and scales. Color augmentation can simulate lighting variations, while scaling and translation improve robustness to changes in object size and position.

For potato disease detection, augmentation is important because the dataset is limited. It helps reduce overfitting and prepares the model for more varied images captured by smartphones or drones. However, augmentation must remain realistic so that it does not create unnatural visual patterns that could confuse the model.

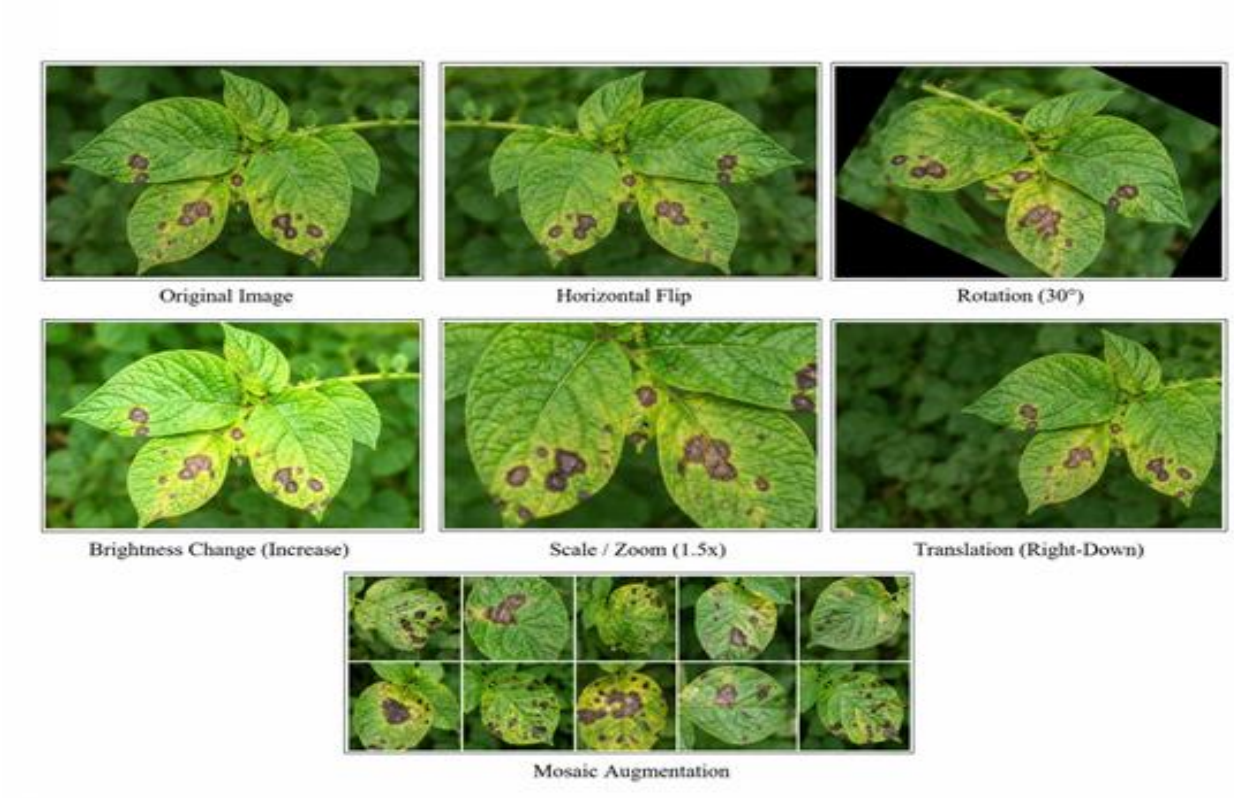


Figure 3.4: Example of Data Augmentation for Potato Leaf Images

3.7 Training Pipeline and Configuration

The training pipeline is implemented using the Ultralytics YOLO framework. The model is trained in a GPU environment such as Kaggle Notebook or Google Colab. GPU acceleration reduces training time and allows the model to process images more efficiently.

The input image size is set to 640 x 640 pixels, which is commonly used in YOLO training. The model is trained for 100 epochs with a batch size of 32. The optimizer uses stochastic gradient descent, momentum, and weight decay. The model starts from pre-trained YOLOv8n weights in order to benefit from transfer learning.

These settings are selected to balance training cost, accuracy, and deployment requirements. Since the final model is intended for real-time use, a lightweight architecture and standard input size are preferred.



Figure 3.5: YOLOv8 Dataset Preparation and Training Workflow

Table 3.4: YOLO architecture evolution

Parameter	Value
Platform	Kaggle / Google Colab
GPU	NVIDIA T4, 16 GB
Model	YOLOv8n
Image size	640 x 640
Batch size	32
Epochs	100
Optimizer	SGD
Initial learning rate	0.01
Weights	yolov8n.pt

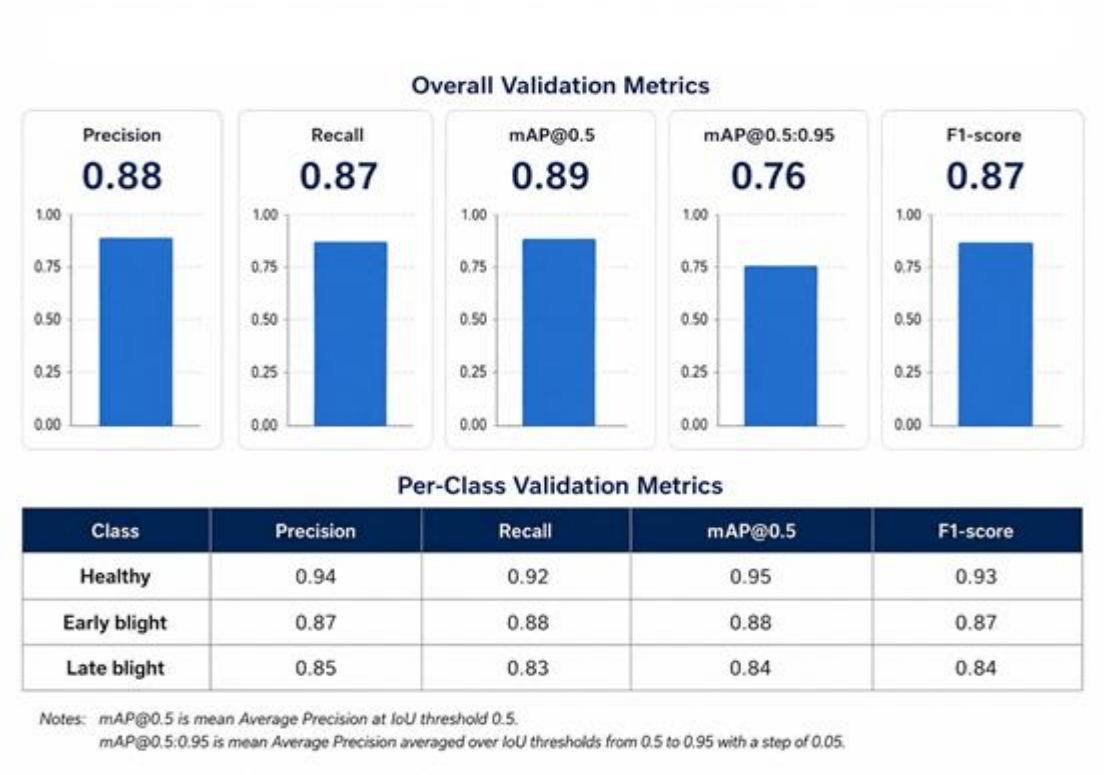
3.8 Model Validation Metricsx

Model validation is performed using standard object detection metrics. The overall precision is 0.88, and the overall recall is 0.87. The model achieves an mAP@0.5 of 0.89 and an mAP@0.5:0.95 of 0.76. The F1-score is 0.87.

These results indicate that the detector provides a balanced performance between correct detections and missed objects. The Healthy class obtains the strongest performance, while Late blight is more challenging because its visual appearance can vary and may overlap with other types of leaf damage.

Tableau 3.5: Model evaluation metrics

Metric	Overall	Healthy	Early blight	Late blight
Precision	0.88	0.94	0.87	0.85
Recall	0.87	0.92	0.88	0.83
mAP@0.5	0.89	0.95	0.88	0.84
mAP@0.5:0.95	0.76	0.82	0.74	0.71
F1-score	0.87	0.93	0.87	0.84



Per-Class Validation Metrics

Class	Precision	Recall	mAP@0.5	F1-score
Healthy	0.94	0.92	0.95	0.93
Early blight	0.87	0.88	0.88	0.87
Late blight	0.85	0.83	0.84	0.84

Notes: mAP@0.5 is mean Average Precision at IoU threshold 0.5.
mAP@0.5:0.95 is mean Average Precision averaged over IoU thresholds from 0.5 to 0.95 with a step of 0.05.

Figure 3.6: Validation Metrics Of Trained YOLOV8n Model.

3.9 Validation Predictions

Validation predictions provide a visual way to evaluate model behavior. The displayed images show class labels, confidence scores, and bounding boxes. This is important because numerical metrics do not always reveal whether the model draws boxes in reasonable positions.

A correct detection should place the bounding box around the relevant leaf or disease symptom. The confidence score indicates how certain the model is about the prediction. Low-confidence detections can be filtered depending on the requirements of the application.

Visual validation is especially important before deployment. It allows the developer to verify that the model output can be displayed correctly in the Flutter application, returned by the FastAPI backend, and stored as useful detection records.

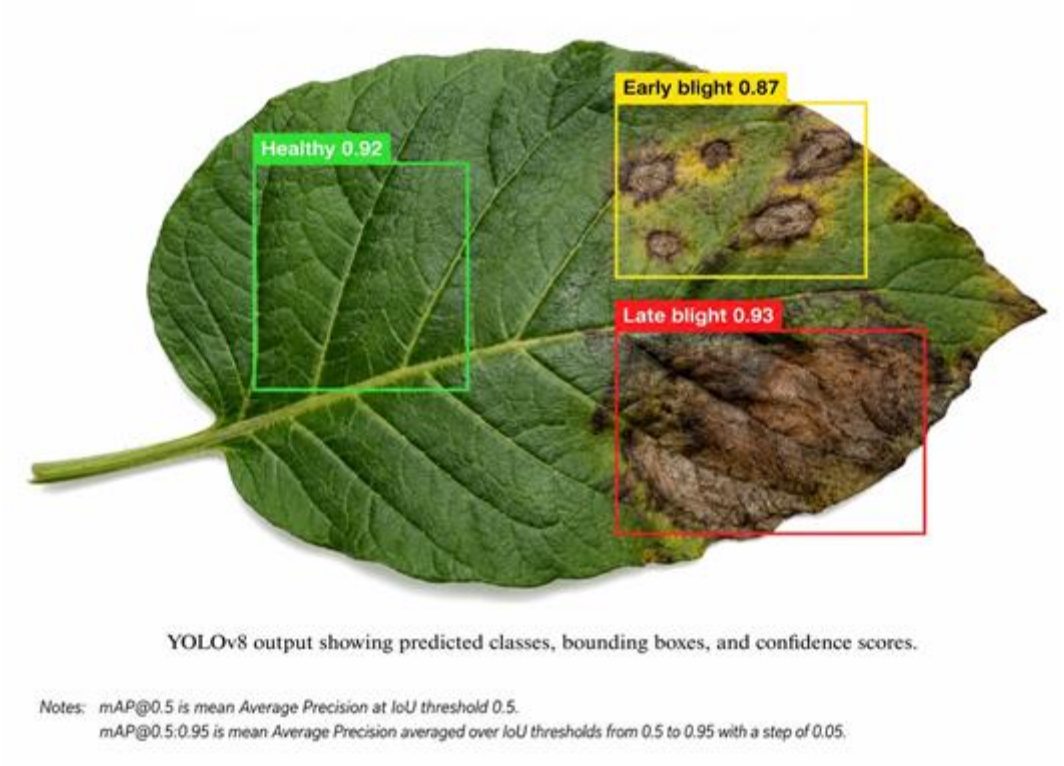


Figure 3.7: YOLOV8 inference Example

3.10 Model Export and Optimization

After training, the model must be exported to formats suitable for deployment. The PyTorch model can be used directly in the FastAPI backend through the Ultralytics package. For mobile inference, conversion to a mobile-compatible format such as TensorFlow Lite may be considered. For NVIDIA edge devices, TensorRT can be used to optimize inference on GPU hardware [25].

Model optimization may include quantization, pruning, or runtime-specific acceleration. Quantization can reduce model size and improve inference speed, but it may slightly affect accuracy. The selected export format depends on the target platform and performance requirements.

In the proposed architecture, the backend can use the original YOLOv8 model, the mobile application can use an optimized mobile format, and the drone-edge component can use a GPU-accelerated runtime. This hybrid deployment strategy supports flexibility across mobile, cloud, and drone-oriented environments.

3.11 Matric confusion

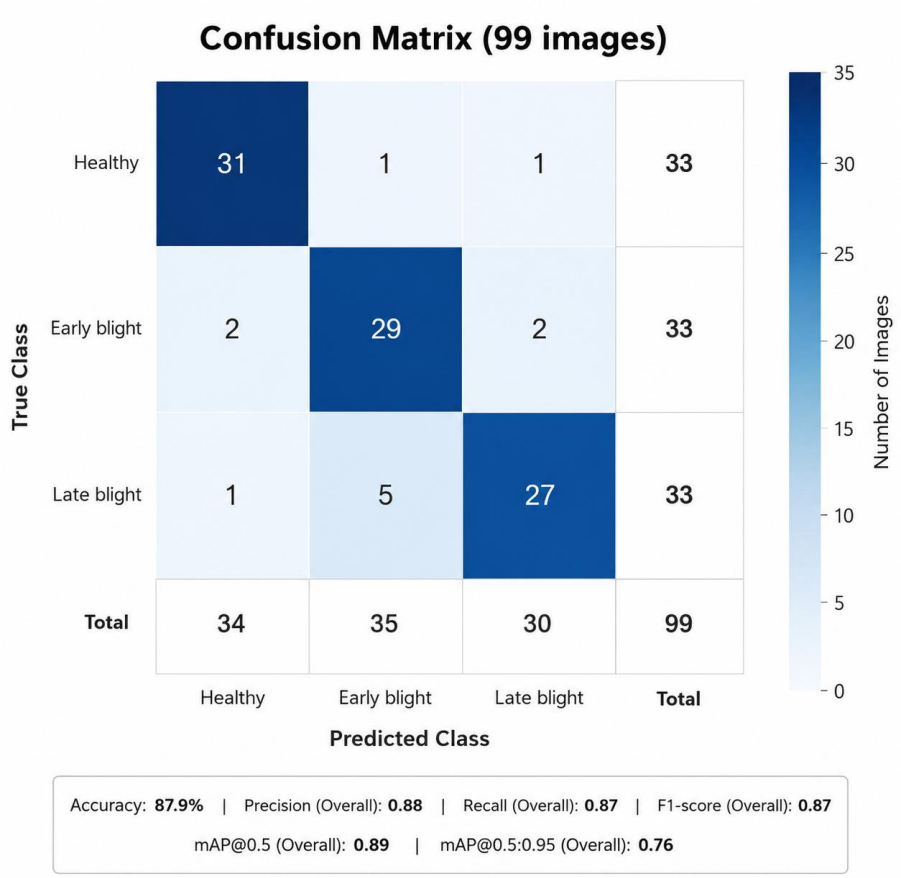


Figure 3.8: matric confusion

3.12 Chapter Conclusion

This chapter presented the complete dataset preparation and training workflow for the YOLOv8n potato disease detection model. The selected dataset contains three classes: Healthy leaf, Early blight, and Late blight. The annotation process converts leaf images into an object detection dataset with bounding boxes.

The model is trained using YOLOv8n with transfer learning, data augmentation, and GPU acceleration. The resulting model achieves an mAP@0.5 of 0.89, which is suitable for integration into the proposed smart farming system.

The chapter also emphasized the importance of annotation quality, visual inspection, validation metrics, model export, and deployment optimization. The next part of the thesis moves from model training to system implementation, including the mobile application, backend API, cloud deployment, and drone-oriented input pipeline.

Chapter 4: System Architecture and Communication Protocols

4.1 Introduction

The proposed system is not limited to a trained artificial intelligence model. It is designed as a complete smart farming architecture that connects mobile interaction, backend inference, cloud deployment, and drone-based monitoring. This chapter presents the main components of the system and explains the communication protocols used between them.

A complete deployment architecture is necessary because a model trained in a notebook cannot directly serve real users. Farmers, students, technicians, or agricultural specialists need an application that can capture images, send data to a backend, receive predictions, store results, and possibly process images or video streams from drones.

The proposed architecture follows a modular design. Each layer has a specific responsibility, which makes the system easier to develop, test, deploy, and extend. The mobile layer handles user interaction, the backend layer handles prediction services, the infrastructure layer supports deployment, and the drone/streaming layer supports aerial monitoring.

4.2 Global System Architecture

The system is organized into four main layers: the edge layer, the API layer, the infrastructure layer, and the streaming/drone layer.

The edge layer includes the Flutter mobile application and the drone-edge component. The mobile application allows users to capture images, select gallery images, and visualize detection results. The drone-edge component can capture aerial images or video frames and either run inference locally or forward the data to the backend.

The API layer is implemented using FastAPI. It receives image inputs, calls the YOLOv8n model, returns prediction results, and manages detection records. FastAPI is suitable for this type of backend because it supports modern Python development, OpenAPI standards, automatic interactive documentation, and request validation [27].

The infrastructure layer includes Docker containers, a reverse proxy, database storage, HTTPS configuration, domain management, and cloud hosting. Docker containers package application code and dependencies into standardized units, which helps the application run reliably across different environments [29].

The streaming/drone layer supports drone video or image input. RTMP can be used for video stream ingestion, while MAVLink can be used for drone telemetry and mission communication [34], [36].

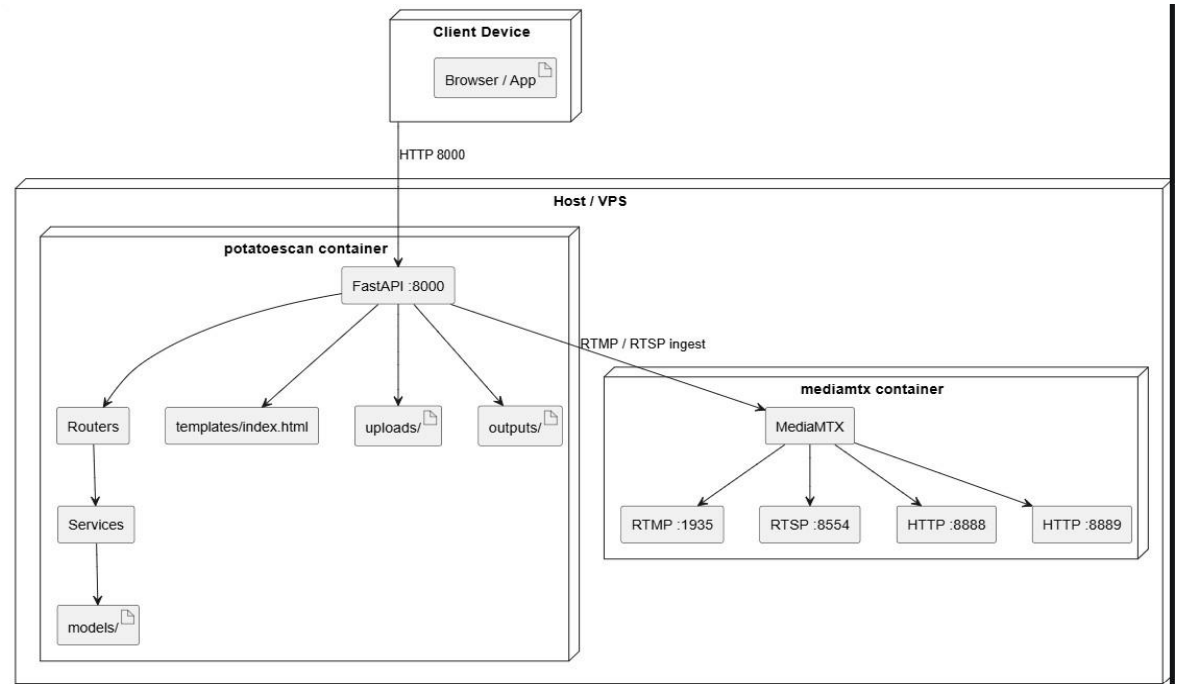


Figure 4.1 Deployment Uml Diagram

4.3 Four-Layer Architecture

The four-layer architecture provides a clear separation of responsibilities. The first layer is the edge layer, which includes the mobile device and optional drone-edge hardware. The mobile device is used for close-range diagnosis, while the drone component is used for larger field monitoring.

The second layer is the API layer. It exposes endpoints that allow external clients to send images and receive predictions. The API receives the input image, prepares it for inference, calls the YOLOv8n model, and returns structured detection results.

The third layer is the infrastructure layer. It supports the deployment and operation of the backend. Docker helps package the backend and its dependencies into containers, while Docker Compose can coordinate multiple services such as the API, database, and reverse proxy [30].

The fourth layer is the streaming and drone layer. It supports continuous image or video input from a drone. Telemetry information such as GPS position, altitude, speed, and mission

state can be exchanged using MAVLink [36], [37].

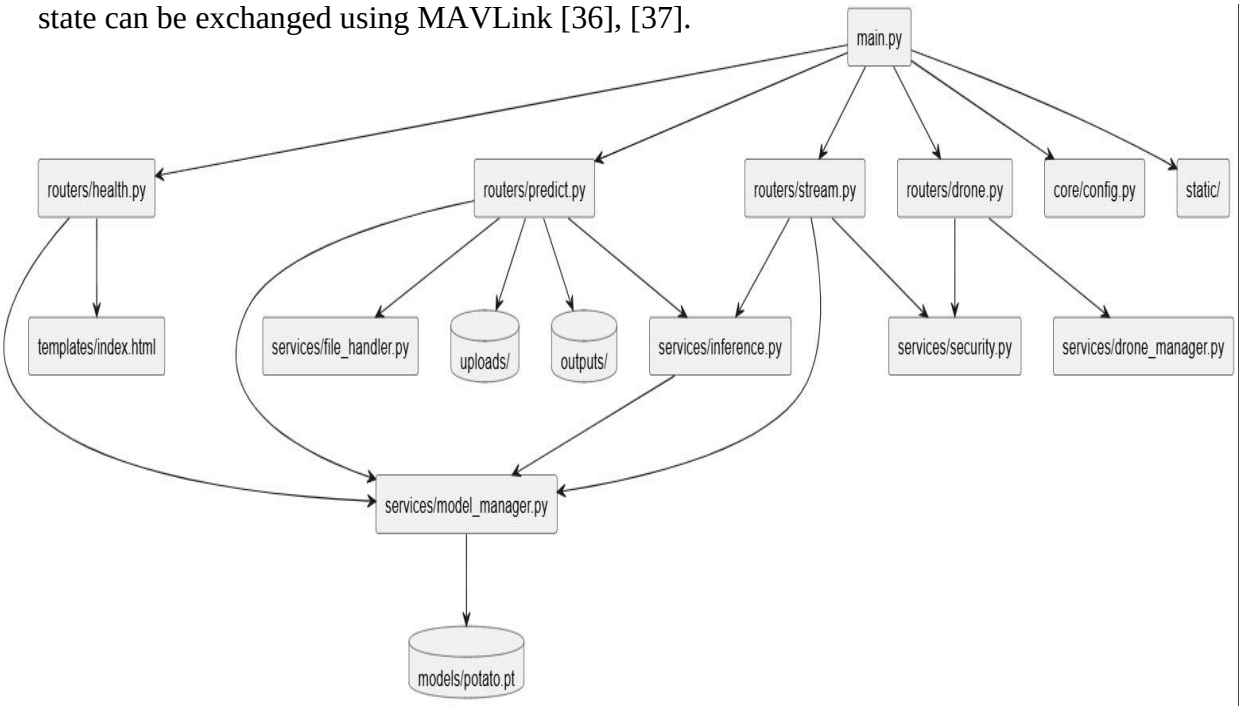


Figure 4.2 component UML diagram

4.4 REST API and HTTP/HTTPS Communication

A REST API provides a structured way for the mobile application, backend, and drone-related components to communicate. Each endpoint corresponds to a specific function, such as prediction, history retrieval, statistics, drone upload, or health checking.

HTTP is useful during local development and testing, but HTTPS is necessary for deployed systems. HTTPS protects requests and responses through encryption. This is important because the system may handle user information, uploaded images, detection results, timestamps, and optional geolocation data.

FastAPI is used to implement the backend API. It is based on OpenAPI standards and provides automatic interactive documentation through Swagger UI and ReDoc [27]. These tools are useful during development because they allow the developer to test endpoints directly from the browser.

4.5 Backend API Endpoints

The backend is designed around a small set of REST endpoints. These endpoints support prediction, history, drone uploads, statistics, and server monitoring.

Table 4.1: Main backend API endpoints

Endpoint	Method	Description
/api/v1/predict	POST	Send an image and receive detection results
/api/v1/history	GET	Retrieve previous detection records
/api/v1/drone/upload	POST	Upload GPS-tagged drone detections
/api/v1/stats	GET	Return disease statistics and summaries
/api/v1/health	GET	Check server health and availability

4.6 RTMP Video Streaming

Real-Time Messaging Protocol, or RTMP, is used in the system architecture as a possible protocol for drone video streaming. RTMP is an application-level protocol designed to multiplex and packetize multimedia streams such as audio, video, and data over transport protocols such as TCP [34].

In a drone-based monitoring scenario, a drone camera or companion computer can send a video stream to a backend service or to a media server. The backend can then extract frames from the stream and process them using the YOLOv8n model. This makes it possible to perform continuous monitoring rather than relying only on isolated image uploads.

RTMP is useful for live streaming scenarios because it was designed to support low-latency transmission of audio, video, and data streams [35]. However, video streaming requires careful management of bandwidth, frame rate, resolution, latency, and server load.

4.7 Future Drone Telemetry Perspective

The current prototype does not implement a complete MAVLink simulation. The drone part of this thesis is limited to a drone-oriented image and RTMP video input perspective, with future extension toward telemetry integration.

In future work, a telemetry protocol such as MAVLink could be connected to the drone pipeline to associate each detection with GPS position, altitude, and flight status [36].

4.8 Security Considerations

Security is important because the system can handle uploaded images, model predictions, API traffic, user records, and possible geolocation information.

Communication between the mobile application and the backend should use HTTPS to protect data during transmission.

The backend should validate uploaded files before processing them. This includes checking file type, image size, and request limits. Input validation helps prevent invalid files, oversized uploads, and abusive requests. API authentication can also be added if the system includes user accounts or private detection history.

Deployment security includes protecting environment variables, configuring firewalls, limiting exposed ports, updating packages, renewing SSL/TLS certificates, and monitoring server logs. Docker containers help isolate application components, but they should still be configured carefully and updated regularly [29].

4.9 Data Flow in the Proposed System

The data flow begins with image acquisition. A user can capture a potato leaf image using the mobile application, select an image from the gallery, or receive frames from a drone video stream. The image is then prepared for inference by resizing and normalizing it according to the model requirements.

The YOLOv8n model processes the image and predicts bounding boxes, class labels, and confidence scores. The prediction result can be returned as JSON by the backend or displayed directly in the mobile application if local inference is used.

If cloud synchronization is enabled, the detection result can be stored in a database with timestamp, predicted disease class, confidence score, bounding box coordinates, and optional GPS location. This stored data can later be used for history, statistics, and dashboard visualization.

The proposed data flow supports three operating modes. In offline mode, the mobile device can run local inference. In connected mode, the mobile application can send images to the backend for centralized inference. In drone mode, aerial images or video frames can be processed and associated with telemetry information.

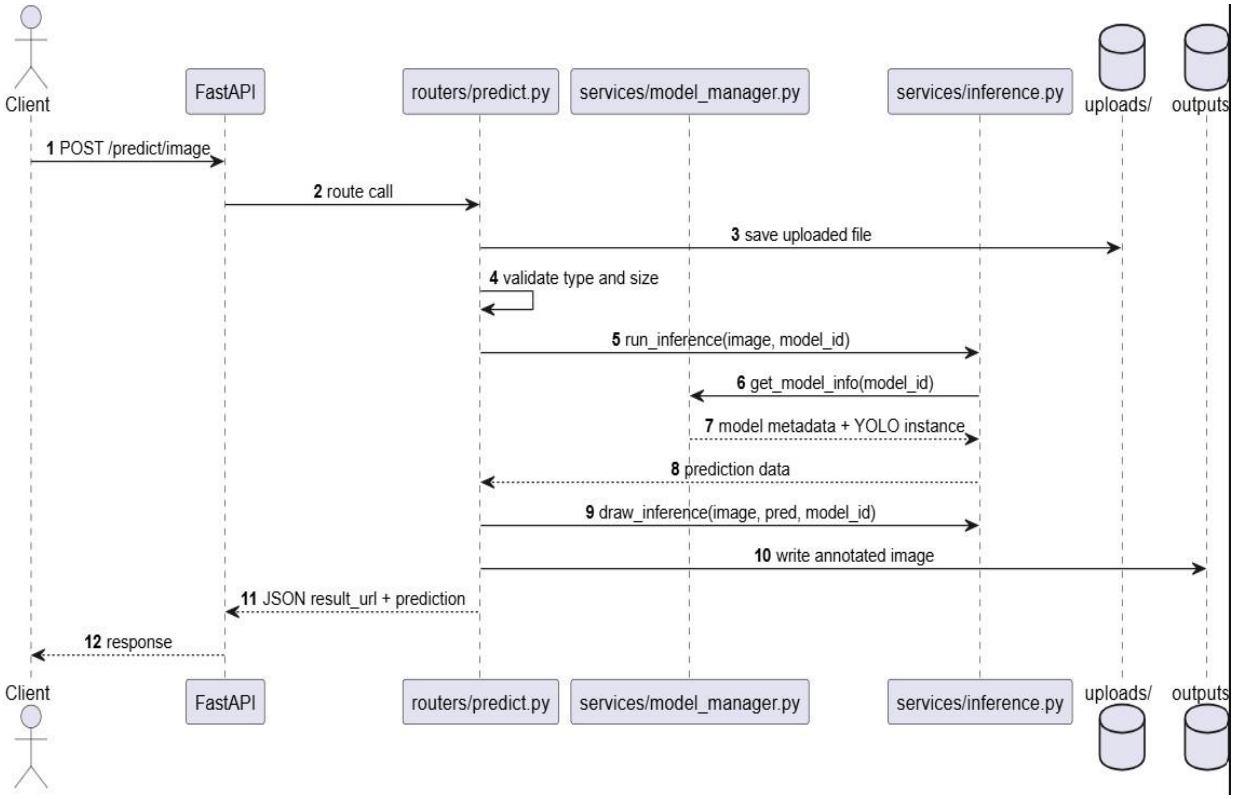


Figure 4.3 Diagram Sequance Uml

4.10 Chapter Conclusion

This chapter presented the proposed system architecture and communication protocols. The architecture is organized into four layers: edge, API, infrastructure, and streaming/drone. This modular design makes the system easier to develop, deploy, and extend.

The chapter also explained the role of REST APIs, HTTP/HTTPS, FastAPI, Docker, RTMP and optional future drone telemetry in the proposed smart farming system. FastAPI provides the backend prediction service, Docker supports reproducible deployment, RTMP supports video stream ingestion, and MAVLink provides a basis for drone telemetry integration.

The next chapter presents the implementation side of the system, including the Flutter mobile application, FastAPI backend, cloud deployment, database storage, and drone integration.

Chapter 5: Mobile Application, Backend, Cloud Deployment, and Drone Integration

5.1 Introduction

This chapter presents the implementation and deployment components of the proposed smart farming system. After training the YOLOv8n model, the next objective is to integrate it into a practical software architecture that can be used in real agricultural scenarios. A trained model alone is not sufficient for real use unless it is connected to a user interface, an API, a deployment environment, and a data management workflow.

The proposed system is composed of several connected components: a Flutter mobile application, a FastAPI backend, a PostgreSQL database, Docker-based deployment, cloud hosting, and a drone-oriented input pipeline. These components work together to support real-time potato disease detection from images and video frames.

The implementation follows a modular design. The mobile application handles user interaction and image acquisition. The backend receives image requests and performs inference using the trained YOLOv8n model. The database stores detection records. The cloud infrastructure makes the backend accessible remotely, while the drone pipeline extends the system toward larger-scale field monitoring.

5.2 Flutter Mobile Application Overview

Flutter is used to develop the mobile application of the proposed system. Flutter allows the creation of cross-platform applications from a single codebase, which is useful because agricultural users may use different mobile devices and operating systems [26].

The mobile application, named PlantGuard in this project, represents the main interface between the user and the disease detection system. It allows the user to capture a leaf image using the camera, select an image from the gallery, send the image for prediction, and display the detection results.

The result screen should present the output in a simple and understandable way. For each detection, the application displays the predicted class, confidence score, and bounding box around the affected region. The three target classes are Healthy leaf, Early blight, and Late

blight.

A mobile application is important for this project because farmers, technicians, and students need a practical tool that can be used directly in the field. A smartphone is portable, accessible, and suitable for close-range leaf analysis. For this reason, mobile integration is one of the main practical contributions of the proposed system.

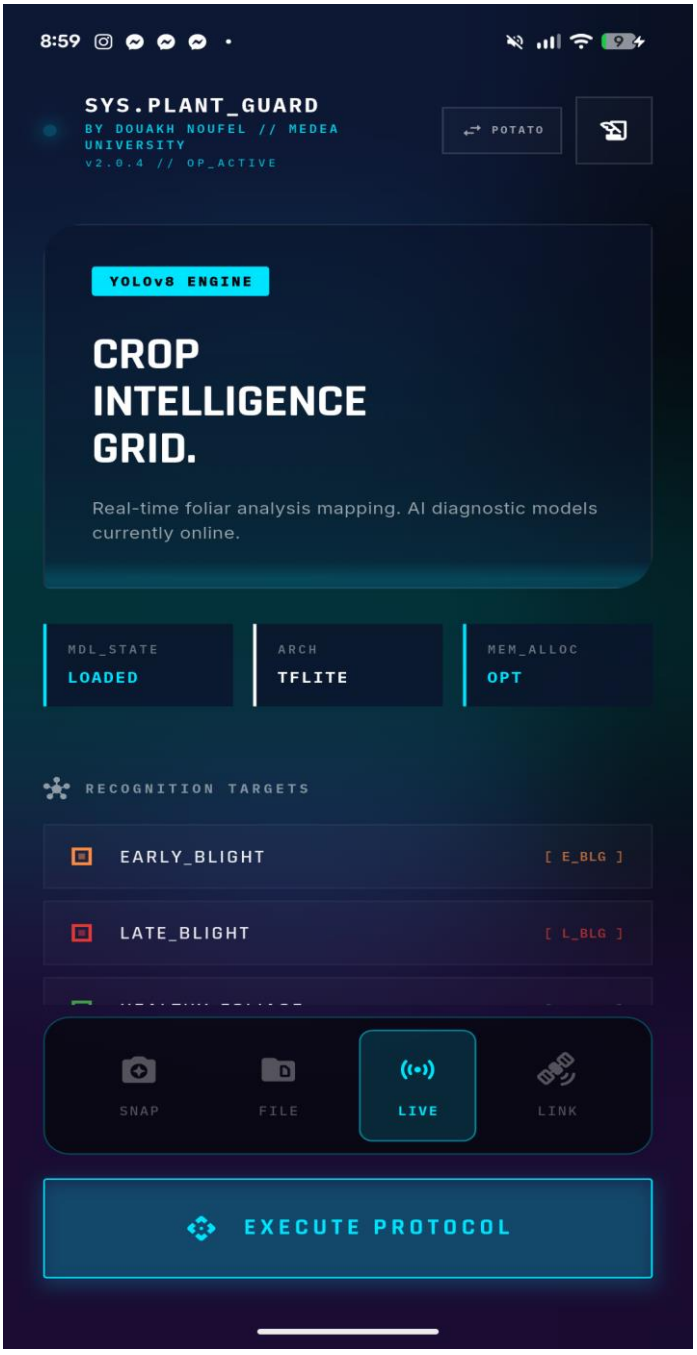


Figure 5.1: PlantGuard mobile application home screen showing the active YOLOv8.

5.3 Mobile User Interface Flow

The mobile application follows a simple user interaction flow. First, the user chooses the detection mode. The main input options include camera capture, gallery selection, and, in future versions, live detection from a video stream.

After selecting or capturing an image, the application sends it to the prediction system. The prediction can be performed either locally on the mobile device or remotely through the backend API. When the prediction is complete, the application displays the detected class, confidence score, and bounding box.

The user interface should be clear and practical. A user should immediately understand whether the detected leaf is healthy or affected by disease. The confidence score helps the user interpret the reliability of the result, while the bounding box shows the exact region that caused the prediction.

The application can also include a detection history screen. This screen stores previous results with timestamps, predicted classes, confidence scores, and optional GPS coordinates. This feature allows users to monitor repeated detections over time and compare disease observations from different location



Figure 5.2: Mobile real-time detection interface showing Late blight detection with bounding box and confidence score.



Figure 5.3: Mobile result screen showing Early blight diagnosis, confidence score, probability bars, and diagnostic information.

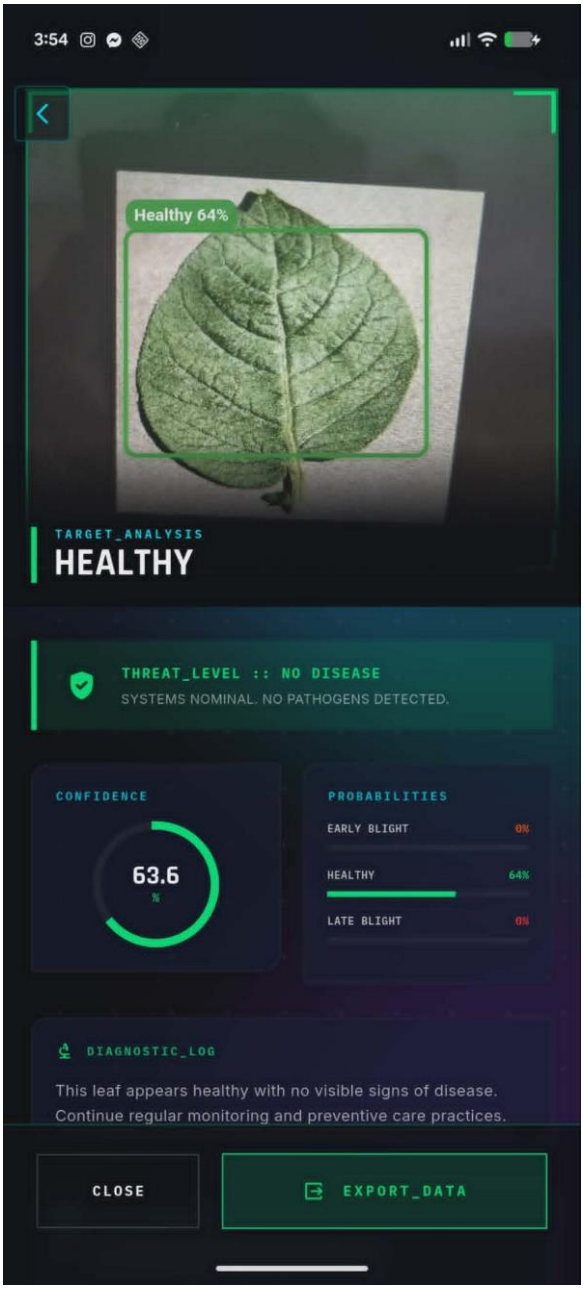


Figure 5.4: Mobile result screen showing Healthy leaf diagnosis and no disease warning.

5.4 On-Device Inference with TensorFlow Lite

On-device inference allows the mobile application to run the model locally without depending completely on an internet connection. This is valuable in rural environments where network access may be weak, unstable, or unavailable. TensorFlow Lite is designed for machine learning deployment on mobile and edge devices [24].

Using a lightweight model such as YOLOv8n helps reduce inference time, memory usage, and energy consumption. The trained model can be converted to a mobile-compatible format and integrated into the Flutter application. This allows the system to perform prediction directly from the smartphone camera or selected images.

On-device inference also improves privacy because images do not always need to be uploaded to a remote server. However, cloud-based inference remains useful for centralized processing, statistics, history synchronization, and future dashboard features.

For this reason, the proposed system can support a hybrid strategy. In offline mode, the mobile device can perform local inference. In connected mode, the mobile application can send images to the backend for centralized prediction and storage.

5.5 FastAPI Backend Implementation

The FastAPI backend provides server-side prediction services. FastAPI is a modern Python web framework that supports automatic API documentation, OpenAPI standards, and interactive testing through Swagger UI and ReDoc [27].

The backend receives an image through a POST request, loads the trained YOLOv8n model, performs inference, and returns the detection results as structured JSON. The response includes the predicted class, confidence score, and bounding box coordinates.

The backend is not limited to prediction only. It can also support additional services such as detection history, statistics, health checking, and drone upload. This transforms the backend from a simple inference script into a central service for the whole smart farming system.

The prediction endpoint is the core part of the backend. It allows the mobile application, web dashboard, or drone component to send an image and receive disease detection results. This makes the backend reusable across different clients.

5.6 Backend API Endpoints

The proposed backend exposes several REST API endpoints. Each endpoint has a specific role in the system. The main endpoints are shown in Tableau V.1.

The /api/v1/predict endpoint is the most important endpoint because it connects the user interface with the YOLOv8n model. The /api/v1/health endpoint is useful during

deployment because it allows the developer to verify that the backend service is running correctly.

An example deployed prediction endpoint can be written as: https://www.noufeldouakh.me/ . In this structure, the main domain noufeldouakh.me can be used as a portfolio or project website, while the subdomain api.noufeldouakh.me can be reserved for backend services.

Table 5.1: Main backend API endpoints.

Endpoint	Method	Description
/api/v1/predict	POST	Send an image and receive disease detection results
/api/v1/history	GET	Retrieve previous detection records
/api/v1/stats	GET	Return disease statistics and summaries
/api/v1/drone/upload	POST	Upload GPS-tagged drone detection data
/api/v1/health	GET	Check server status and availability

5.7 Database and Detection Records

A database is used to store detection history and related information. PostgreSQL is suitable for this role because it is an advanced open-source object-relational database system [28].

The database can store image metadata, prediction results, confidence scores, bounding box coordinates, timestamps, and optional GPS coordinates. This information makes it possible to review previous detections and generate statistics about disease occurrence.

Storing detection records transforms the system from a simple prediction tool into a decision-support platform. The system can calculate how many detections were made, which disease appears most frequently, and where suspicious symptoms were observed.

For a prototype, the database can remain simple. For a production system, additional features such as indexing, authentication, access control, backup, and data protection should be added.

Table 5.2: Example detection record fields.

Field	Description
id	Unique detection identifier
image_path	Path or URL of the processed image
predicted_class	Healthy leaf, Early blight, or Late blight
confidence	Confidence score of the prediction
bounding_box	Coordinates of the detected region
created_at	Date and time of detection
gps_latitude	Optional latitude value
gps_longitude	Optional longitude value
user_id	Optional user identifier

5.8 Docker Containerization

Docker is used to package the backend and its dependencies into a reproducible environment. A container includes the application code, required libraries, configuration, and runtime dependencies. This helps the application run more reliably across different environments [29].

In this project, the backend container can include Python, FastAPI, Uvicorn, the Ultralytics package, the YOLOv8n model weights, and the required machine learning libraries. This reduces compatibility problems when moving the project from local development to a cloud server.

Docker Compose can be used to define and run multiple services together. For example, the backend, PostgreSQL database, and Nginx reverse proxy can be described in one configuration file and started with a single command [30].

Containerization is useful for a thesis project because it demonstrates good engineering practice. It also makes the system easier to reproduce, test, and transfer to another server.

.

Table 5.3: Docker services in the proposed system.

Service	Role
FastAPI backend	Prediction API and business logic
PostgreSQL	Detection history and structured data storage
Nginx	Reverse proxy and HTTPS termination
YOLO model volume	Storage for trained model weights
Environment file	Configuration variables and secrets

5.9 Cloud Deployment and Domain Configuration

Cloud deployment allows the backend to be accessed remotely by the mobile application and other clients. In the proposed system, a virtual private server can host the Docker containers, FastAPI backend, PostgreSQL database, and Nginx reverse proxy.

DigitalOcean can be used as a cloud hosting provider through a Droplet, which is a Linux-based virtual machine [31]. The deployment process includes preparing the server, installing Docker, copying or pulling the project files, configuring environment variables, starting the containers, and testing the health endpoint.

A custom domain makes the system more professional and easier to access. The domain noufeldouakh.me can represent the project identity or personal portfolio. A subdomain such as api.noufeldouakh.me can be used for the backend API.

The DNS configuration connects the subdomain to the public IP address of the cloud server. After DNS propagation, the backend can be accessed through a readable URL instead of a raw IP address.

5.10 Nginx Reverse Proxy and SSL/TLS

Nginx can be used as a reverse proxy in front of the FastAPI backend. A reverse proxy receives client requests and forwards them to the appropriate backend service [32]. This is useful because it allows the backend to run internally while Nginx handles external traffic.

SSL/TLS certificates are necessary to enable HTTPS. HTTPS protects communication

between the mobile application and the backend. This is important when the system handles uploaded images, detection records, or optional location information.

Let's Encrypt can be used to obtain free SSL/TLS certificates for the domain [33]. With Nginx and SSL/TLS configured, HTTP requests can be redirected to HTTPS, making the deployment more secure and closer to a production-ready system.

A possible production URL for the prediction endpoint is: https://www.noufeldouakh.me . This structure separates the main website from the API and makes the project easier to maintain.

5.11 Drone Hardware and Surveying

The drone component extends the system from close-range mobile diagnosis to larger-scale agricultural monitoring. A drone can capture aerial images or video frames from a field and send them to the detection pipeline.

A flight controller such as Pixhawk can manage navigation and mission execution. A companion computer such as Jetson Orin Nano can process images, run inference, or communicate with the backend. future drone telemetry could provide information such as GPS position, altitude, speed, battery status, and mission state [36].

When a detection is made from a drone image or video frame, the result can be associated with GPS coordinates. This creates geo-referenced disease observations that can later be displayed on a map or used for field management decisions.

In the current thesis, the drone component is mainly presented as an architectural and prototype-oriented extension. Future work should include real field trials under different lighting conditions, flight altitudes, camera angles, and weather situations.

Table 5.4: Real-time inference benchmarks.

Component	Selection	Role
Flight controller	Pixhawk 4	Mission control and navigation
Companion computer	Jetson Orin Nano	AI inference and frame processing
Communication	MAVLink	Telemetry and mission data
Simulation	DroneKit-SITL	Testing without real flight
Camera	RGB camera	Image and video acquisition

5.12 Drone Image and Video Input Pipeline

The drone input pipeline can support two main modes: image upload and video streaming. In image upload mode, the drone or companion computer captures still images and sends them to the backend. In video streaming mode, a continuous stream is sent to a media server or backend service.

RTMP can be used as a possible protocol for video streaming. It supports the transmission of multimedia streams and can be used to send video from the drone pipeline to the processing server [34], [35].

For video-based monitoring, the system can extract selected frames from the stream and send them to the YOLOv8n model. Processing every frame may be expensive, so frame sampling can be used. For example, the system may process one frame every second depending on hardware capacity.

In the prototype configuration, the drone stream does not need to process every video frame. A practical setting is to sample the RTMP stream at about 1 to 5 predictions per second, depending on the server, GPU, network latency, and selected frame resolution. This keeps the system responsive while reducing unnecessary computation.

The detection output includes disease class, confidence score, bounding box coordinates, timestamp, and optional GPS information. This output can be stored in the database and later used for visualization, statistics, or mapping.

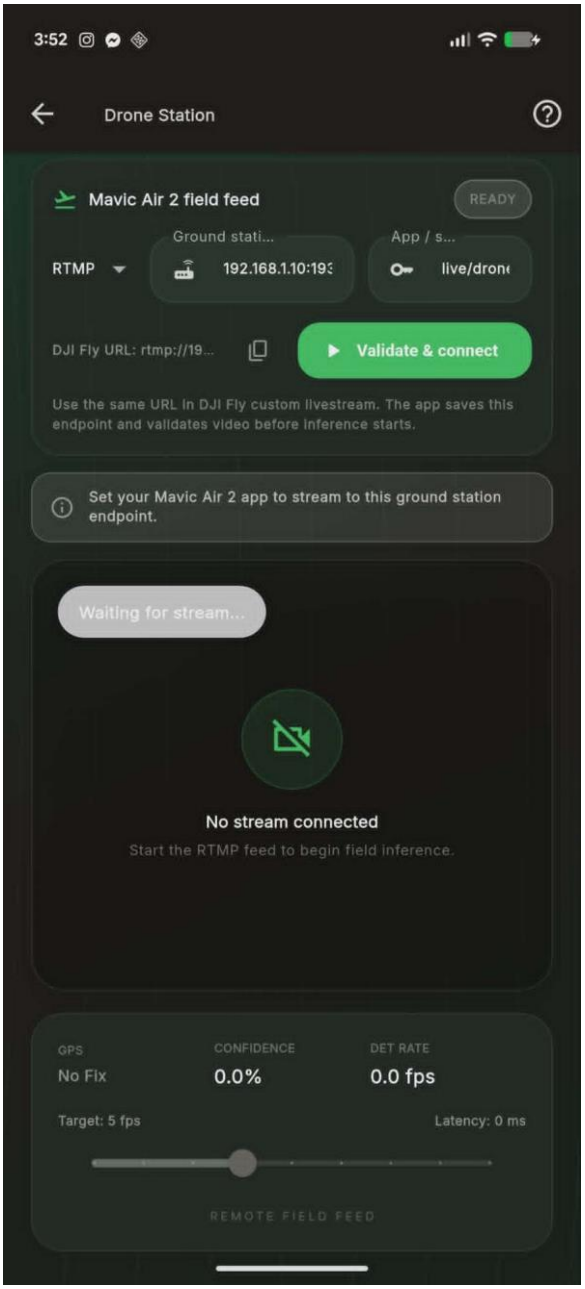


Figure 5.5: Drone station interface prepared for RTMP field stream connection and remote inference.

The following figures illustrate the drone-oriented perspective of the proposed system, including aerial monitoring, field operation, and mapped terrain visualization. These images are used as supporting illustrations for the drone integration layer.



Figure 5.6: Drone-based agricultural field monitoring for smart farming applications.



Figure 5.7: Example of a drone used in agricultural field operations and crop monitoring.



Figure 5.8: Example of a drone control interface showing mapped field data for remote monitoring. Source: Illustrative image used to represent drone-assisted smart agriculture.

5.13 CI/CD Pipeline

A CI/CD pipeline can automate parts of the development and deployment process. For the backend, the pipeline can build Docker images, run tests, push images to a registry, and deploy updates to the cloud server. This reduces manual errors and makes updates more reliable.

For the mobile application, automated build services can generate Android APK files and iOS packages. This is useful because mobile projects require platform-specific configuration, dependencies, and signing settings.

Version control with Git should be used for all source code. Branches, commits, tags, and release versions help organize development and make the project easier to maintain. A good CI/CD workflow also supports reproducibility during thesis evaluation.

5.14 Web Dashboard Perspective

A web dashboard can extend the system beyond mobile detection. It can display detection history, statistics, image results, user activity, and possibly a map of drone detections. Such a dashboard would be useful for supervisors, agronomists, or farm managers.

The dashboard can communicate with the same FastAPI backend used by the mobile application. This avoids duplicating prediction logic and keeps the backend as the central service. The API can provide endpoints for filtering detections by date, disease class, location, or confidence score.

Although the dashboard is not the central focus of the thesis, it is a valuable future extension. It can transform the project from a detection tool into a complete agricultural monitoring platform.

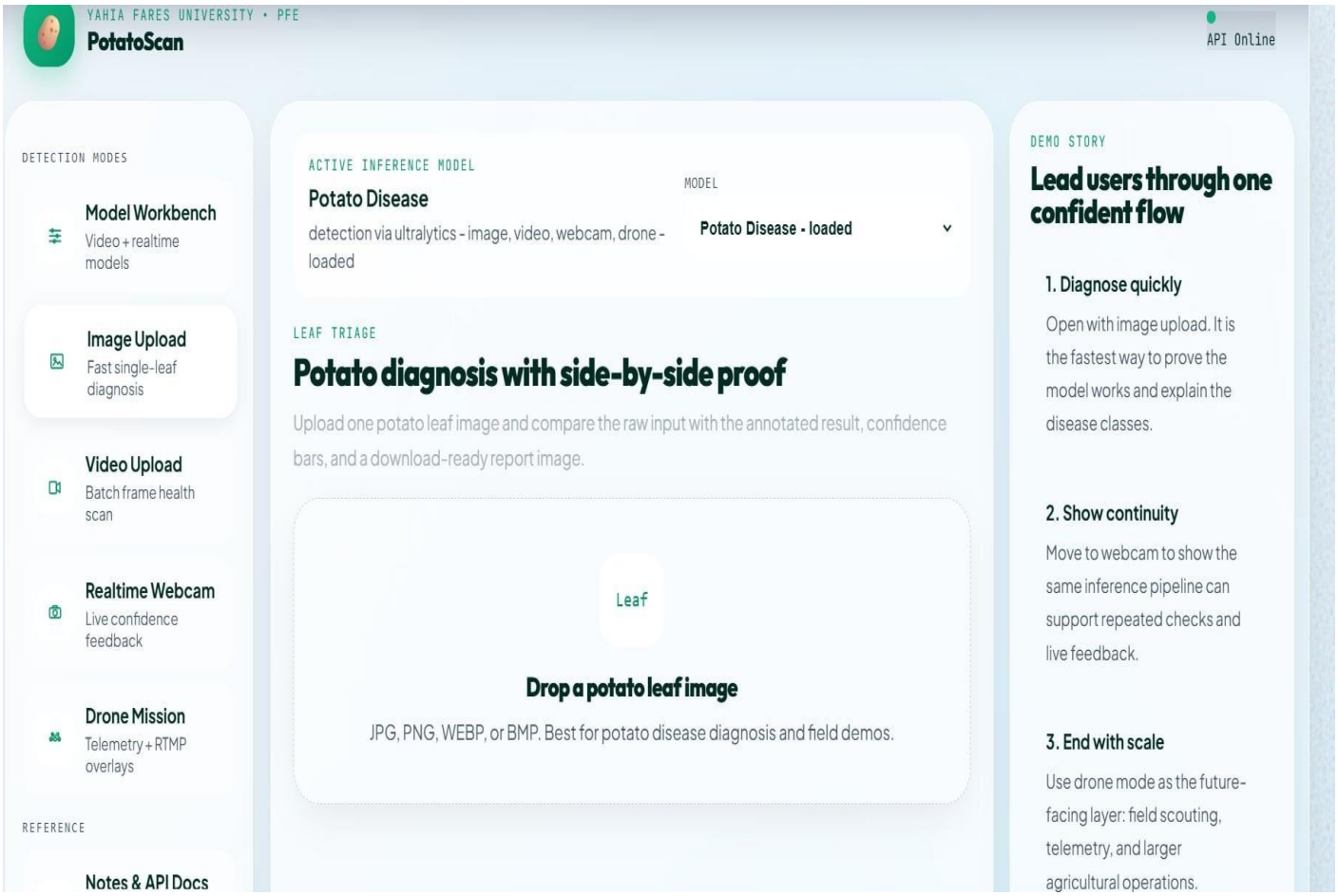


Figure 5.9: AgroVision9 web dashboard showing available detection modes and model status.

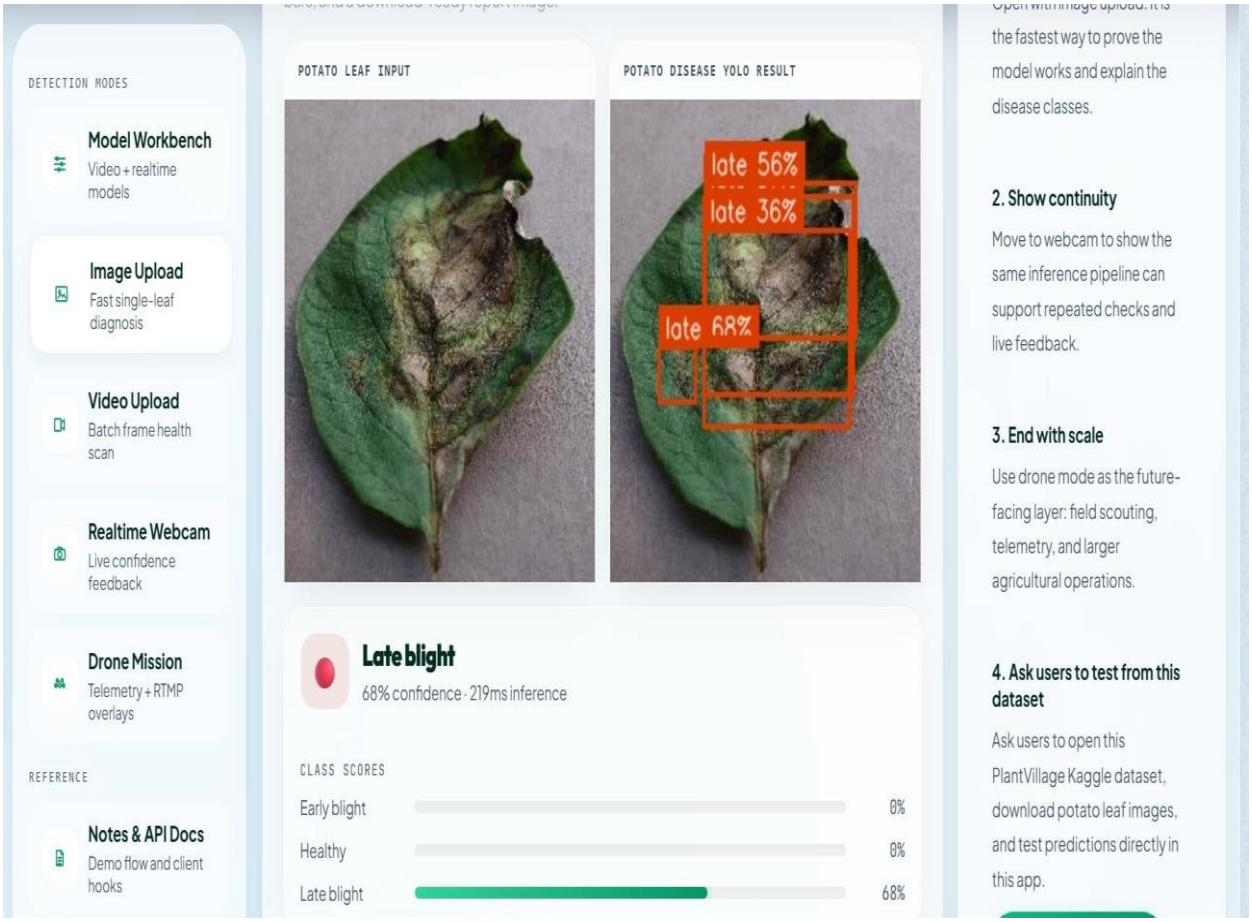


Figure 5.10: *Web-based image upload interface showing side-by-side potato leaf input and YOLO detection result...*

5.15 Chapter Conclusion

This chapter presented the implementation and deployment components of the proposed smart farming system. Flutter provides the mobile interface, TensorFlow Lite supports on-device inference, FastAPI exposes the backend API, PostgreSQL stores detection records, Docker supports reproducible deployment, and cloud hosting enables remote access.

The chapter also explained the role of the domain noufeldouakh.me and the API subdomain api.noufeldouakh.me in the deployment architecture. This domain configuration makes the system more professional and easier to access.

The drone integration extends the system toward larger-scale field monitoring. Image upload and video streaming can provide aerial input, while future telemetry integration could

provide GPS information. This creates the basis for geo-referenced disease detection and future mapping features.

The next chapter presents the experimental results and discussion, including model performance, inference speed, deployment behavior, limitations, and future improvements.

Chapter 6: Experimental Results and Discussion

6.1 Introduction

This chapter presents the experimental results of the proposed YOLOv8n potato disease detection model and discusses the behavior of the system in different deployment scenarios. The evaluation focuses on both detection accuracy and real-time performance, because the objective of the thesis is not only to train a model but also to prepare it for practical use.

The model is evaluated using standard object detection metrics, including precision, recall, mAP@0.5, mAP@0.5:0.95, and F1-score. These metrics help measure how well the system detects and localizes potato leaf conditions such as Healthy leaf, Early blight, and Late blight.

The chapter also discusses mobile deployment, backend inference, drone-oriented processing, strengths, limitations, and possible future improvements. This makes it possible to evaluate the proposed system as a complete smart farming solution rather than as an isolated artificial intelligence model.

In addition to the validation workflow, a set of 99 potato leaf images was used for manual testing and visual verification of the detection results. These images helped verify the behavior of the application output, including predicted classes, bounding boxes, confidence scores, and the visual consistency of the mobile and web interfaces.

6.2 Model Performance Metrics

The trained YOLOv8n model achieves an overall precision of 0.88 and an overall recall of 0.87. Precision indicates how many predicted detections are correct, while recall indicates how many real objects are successfully detected. A good balance between precision and recall is important because the system should avoid both false alarms and missed disease symptoms.

The model achieves an overall mAP@0.5 of 0.89. This result shows that the detector performs well when predictions are evaluated at an IoU threshold of 0.5. The stricter mAP@0.5:0.95 value is 0.76, which indicates that localization accuracy decreases when evaluation is performed over multiple IoU thresholds.

The overall F1-score is 0.87. Since the F1-score combines precision and recall, it confirms that the model provides a balanced detection performance across the selected classes.

The Healthy class obtains the highest performance, with an mAP@0.5 of 0.95 and an F1-score of 0.93. This can be explained by the more uniform visual appearance of healthy leaves. Early blight also performs well, with an mAP@0.5 of 0.88, because its lesion patterns can be visually distinctive. Late blight is the most challenging class, with an mAP@0.5 of 0.84, because its symptoms can vary and may resemble other forms of leaf damage.

See **Tableau III.**4: Model evaluation metrics [page 41]

6.3 Interpretation of Detection Results

The validation results show that YOLOv8n can successfully detect the three selected potato leaf classes. The model performs best when the target region is clear, well illuminated, and visually similar to the training samples. This is expected because deep learning models usually perform better when the validation images follow the same distribution as the training data.

The lower performance for Late blight suggests that this class may require more diverse training examples. Late blight symptoms can appear as irregular necrotic regions, water-soaked lesions, or damaged areas. These visual variations can make the model confuse Late blight with other symptoms or background artifacts.

The results also show the importance of localization. A classification model could only predict the general image label, but YOLOv8n provides bounding boxes that show where symptoms are detected. This makes the system more useful for farmers, technicians, and agricultural monitoring because the user can visually inspect the detected region.

6.4 Real-Time Inference Benchmarks

Real-time performance depends strongly on hardware. A CPU-only computer can run the model, but inference time is usually higher. GPU-based environments such as Kaggle

Or Google Colab with an NVIDIA T4 can process images much faster. Mobile devices and edge boards provide intermediate performance that may still be practical for field use.

The benchmark results support the selection of YOLOv8n. A larger YOLO model could

improve accuracy, but it would likely reduce inference speed on mobile and edge devices. Since the system targets real-time agricultural use, the balance between speed and accuracy is more important than maximizing accuracy alone.

The results also show that optimized deployment can significantly improve speed. GPU acceleration, TensorRT optimization, and mobile inference runtimes can reduce latency and make the system more suitable for real-time operation.

Table 6.1: Real-time performance hardware

Platform	Hardware	Inference time	FPS
Windows PC	Intel i5 CPU	~120 ms	~8
Kaggle / Colab	NVIDIA T4 GPU	~15 ms	~67
iPhone 13	Apple A15 + TFLite	~35 ms	~28
Jetson Orin Nano	CUDA / TensorRT	~25 ms	~40
Android mid-range	Snapdragon 680	~80 ms	~12

6.5 Mobile Deployment Results

Mobile deployment is one of the main practical contributions of the system. The mobile application allows users to capture or select an image and receive detection results directly. This reduces the delay between field observation and disease diagnosis.

The application can support several input modes, including camera capture, gallery image selection, and live detection. When on-device inference is available, the user can obtain results even without internet access. This is important for rural environments where connectivity may be limited.

When cloud synchronization is available, detection results can be stored for later analysis. This allows the user to review previous detections, monitor disease progression, and compare results over time. The mobile deployment therefore transforms the model into a practical field tool.

6.6 Backend and Cloud Deployment Results

The backend provides centralized prediction services through a FastAPI REST API. In connected mode, the mobile application sends images to the backend, and the backend returns predictions in JSON format. This approach allows the server to manage inference, history, statistics, and future dashboard features.

Cloud deployment makes the backend accessible from different devices. Docker containerization improves reproducibility by packaging the backend and its dependencies. A reverse proxy and HTTPS configuration make the deployment more professional and secure.

Backend inference is useful when the mobile device has limited hardware resources or when centralized storage is required. However, it depends on network availability and server performance. For this reason, the system can benefit from a hybrid strategy combining local inference and cloud inference.

6.7 Drone and Stream-Oriented Results

The drone-oriented part of the system is designed to support field-scale monitoring. Instead of analyzing only close-range smartphone images, the drone pipeline can process aerial images or video frames. This is useful for covering larger agricultural areas.

In the proposed architecture, the drone can send images or video frames to the backend. The YOLOv8n model then processes the frames and returns detections. If telemetry data is available, each detection can be associated with GPS coordinates, altitude, and timestamp.

The current drone integration is mainly presented as an architectural and simulation-oriented component. Real field testing remains necessary to evaluate performance under actual drone conditions, including altitude variations, motion blur, wind, changing illumination, and camera angle differences.

6.8 Strengths of the Proposed System

The first strength of the proposed system is its end-to-end design. The thesis does not stop at model training; it also considers mobile use, backend deployment, database storage, cloud hosting, and drone input.

The second strength is the use of object detection instead of simple image classification. Object detection provides both the predicted disease class and the localization of symptoms through bounding boxes. This makes the output more informative and easier to interpret.

The third strength is the choice of YOLOv8n. This model provides a good balance between speed and accuracy, making it suitable for real-time inference and deployment on different platforms.

The fourth strength is the modular architecture. Each component can be improved independently. For example, the dataset can be expanded, the model can be replaced, the backend can be redeployed, or the mobile interface can be upgraded without redesigning the entire system.

6.9 Limitations

Despite the promising results, the system has several limitations. The first limitation is the dataset. PlantVillage images are useful for training, but many of them are captured under controlled conditions. Real field images can contain complex backgrounds, shadows, blur, occlusion, and different lighting conditions.

The second limitation is the number of disease classes. The model detects only three classes: Healthy leaf, Early blight, and Late blight. In real agricultural environments, potato plants may be affected by other diseases, nutrient deficiencies, pests, or environmental stress.

The third limitation is the lack of extensive real-world field testing. The drone component is mainly described as a deployment and simulation-oriented extension. Real drone experiments are needed to evaluate detection performance under practical conditions.

The fourth limitation is the need for expert validation. Artificial intelligence predictions should support agricultural decision making, but they should not completely replace expert diagnosis. Misclassification can occur, especially when symptoms are visually similar or when image quality is poor.

6.10 Future Improvements

Future work should first focus on expanding the dataset with real field images captured using smartphones and drones. Adding more diverse images would reduce the domain gap between controlled datasets and real agricultural conditions.

Second, the system can be extended to detect more potato diseases, pest damage, nutrient deficiencies, and environmental stress symptoms. A larger class set would make the system more useful in practical agriculture.

Third, segmentation can be added to estimate disease severity. While object detection localizes symptoms using bounding boxes, segmentation can provide more precise affected-area measurement. This would help quantify disease progression.

Fourth, the drone component should be tested in real field conditions. GPS-tagged detections can be used to create disease distribution maps, helping farmers identify affected zones and plan interventions.

Finally, a web dashboard can be developed to display detection history, statistics, maps, user activity, and disease trends. This would transform the system from a detection application into a complete smart farming monitoring platform.

6.11 Chapter Conclusion

This chapter presented the experimental results and discussion of the proposed YOLOv8n potato disease detection system. The model achieved an overall mAP@0.5 of 0.89, precision of 0.88, recall of 0.87, and F1-score of 0.87. These results show that the model provides promising detection performance for a prototype smart farming system.

The discussion showed that YOLOv8n is suitable for real-time deployment because it balances accuracy and speed. The system can operate through mobile inference, backend inference, and drone-oriented input. However, real-world field validation, dataset expansion, and additional disease classes are necessary before operational deployment.

The next section presents the general conclusion and perspectives of the thesis.

General Conclusion and Perspectives

This master's thesis presented the design, training, and deployment of a real-time smart farming system for potato disease detection. The main objective was to develop a practical computer vision solution capable of detecting potato leaf conditions and localizing visible symptoms using object detection. The proposed system focused on three classes: Healthy leaf, Early blight, and Late blight.

The work began with the theoretical foundations of computer vision, deep learning, convolutional neural networks, and object detection. These concepts were necessary to understand why YOLO-based detection is more suitable than simple image classification for this project. Unlike classification models, which only assign one label to the entire image, object detection models provide bounding boxes, class labels, and confidence scores. This makes the output more practical for agricultural monitoring.

The practical part of the thesis focused on dataset preparation, YOLOv8n training, validation, and deployment planning. The PlantVillage potato dataset was used to train the model, and the images were prepared for object detection using YOLO-format annotations. The trained YOLOv8n model achieved promising prototype results, with an overall mAP@0.5 of 0.89, precision of 0.88, recall of 0.87, and F1-score of 0.87.

The system implementation was designed as a complete architecture rather than a simple model experiment. It includes a Flutter mobile application for user interaction, a FastAPI backend for prediction services, Docker for containerized deployment, PostgreSQL for detection records, cloud hosting for remote access, and a drone-oriented input pipeline for larger field monitoring.

The mobile application allows users to capture or select images and receive disease detection results. The backend API can perform centralized inference and return structured prediction data. The drone component extends the system toward aerial monitoring by allowing images or video frames to be processed and associated with telemetry information such as GPS coordinates.

Despite the promising results, the system still has limitations. The main limitation is the dataset, because PlantVillage images are mostly captured under controlled conditions. Real agricultural environments contain more variation, including shadows, blur, complex backgrounds, different lighting conditions, overlapping leaves, and camera movement. Another limitation is that the current model detects only three classes. Future work should first

focus on collecting more real-world images from fields using smartphones and drones. This would improve model robustness and reduce the gap between controlled datasets and practical agricultural conditions. The system should also be extended to include more disease classes and more diverse symptoms. Another important improvement is the integration of segmentation methods to estimate disease severity more precisely by measuring the affected area of leaves.

Further perspectives include improving mobile inference, optimizing the model for edge devices, testing the drone pipeline in real field conditions, and developing a web dashboard for visualization. A dashboard could display detection history, statistics, maps, disease trends, and user activity. This would transform the project from a detection prototype into a complete agricultural monitoring platform.

In conclusion, this thesis demonstrates that lightweight deep learning models such as YOLOv8n can be integrated into a practical smart farming architecture for real-time potato disease detection. The proposed system provides a foundation for future agricultural decision-support tools that combine artificial intelligence, mobile computing, cloud deployment, and drone-based monitoring.

REFERANCE

[1] R. Szeliski, Computer Vision: Algorithms and Applications, 2nd ed. Springer, 2022. Available: https://szeliski.org/Book/

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. Available: https://www.deeplearningbook.org/

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436-444, 2015. Available: https://www.nature.com/articles/nature14539

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, pp. 533-536, 1986. Available: https://www.nature.com/articles/323533a0

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," NeurIPS, 2012. Available: https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c4 5b-Abstract.html

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv, 2014. Available: https://arxiv.org/abs/1409.1556

[7] C. Szegedy et al., "Going Deeper with Convolutions," arXiv, 2014. Available: https://arxiv.org/abs/1409.4842

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv, 2015. Available: https://arxiv.org/abs/1512.03385

[9] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv, 2017. Available: https://arxiv.org/abs/1704.04861

[10] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," arXiv, 2019. Available: https://arxiv.org/abs/1905.11946

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," arXiv, 2015. Available: https://arxiv.org/abs/1506.01497

REFERANCE

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," arXiv, 2015. Available: https://arxiv.org/abs/1506.02640

[13] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv, 2018. Available: https://arxiv.org/abs/1804.02767

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv, 2020. Available: https://arxiv.org/abs/2004.10934

[15] Ultralytics, "YOLOv8 Documentation." Available: https://docs.ultralytics.com/models/yolov8/

[16] Ultralytics, "Object Detection Datasets Overview." Available: https://docs.ultralytics.com/datasets/detect/

[17] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li, "SOLO: A Simple Framework for Instance Segmentation," arXiv, 2021. Available: https://arxiv.org/abs/2106.15947

[18] Google AI Edge, "MediaPipe Solutions Guide." Available: https://developers.google.com/edge/mediapipe/solutions/guide

[19] CMU Perceptual Computing Lab, "OpenPose: Real-time Multi-Person Keypoint Detection Library." Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose

[20] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," arXiv, 2016. Available: https://arxiv.org/abs/1606.00915

[21] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," ECCV, 2014. Available: https://cocodataset.org/

[22] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," Frontiers in Plant Science, 2016. Available: https://arxiv.org/abs/1604.03169

[23]

https://learnopencv.com/ultralytics-yolov8/

https://cdn.learnopencv.com/wp-content/uploads/2023/01/04091552/yolov8-comparison-plots.png

[23] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, 2010. Available: https://ieeexplore.ieee.org/document/5288526

[24] Google AI Edge, "TensorFlow Lite Guide." Available:

https://www.tensorflow.org/lite/guide

[25] NVIDIA,"TensorRTDocumentation." Available: https://docs.nvidia.com/deeplearning/tensorrt/

[26] Flutter, "Build apps for any screen." Available: https://flutter.dev/

[27] FastAPI, "Features." Available: https://fastapi.tiangolo.com/features/

[28] PostgreSQL, "AboutPostgreSQL." Available:

https://www.postgresql.org/about/

[29] Docker, "What is a Container?" Available:

https://www.docker.com/resources/what-container/

[30] Docker, "DockerComposeDocumentation." Available:

https://docs.docker.com/compose/

[31] DigitalOcean, "Getting Started with Droplets." Available: https://docs.digitalocean.com/products/droplets/getting-started/

[32] NGINX, "NGINX ReverseProxy." Available:

https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/

[33] Let's Encrypt,"Let's Encrypt Documentation." Available:

https://letsencrypt.org/docs/

[34] Adobe, "Real Time Messaging Protocol Specification." Available: https://rtmp.veriskope.com/docs/spec/

[35] RTMP Documentation, "Real Time Messaging Protocol Overview." Available:

https://rtmp.veriskope.com/docs/overview/

[36] MAVLink, “MAVLink Developer Guide.” Available:

https://mavlink.io/

[38] NVIDIA, “Jetson Orin Nano Developer Kit.” Available:
https://developer.nvidia.com/embedded/jetson-orin-nano-developer-kit

[39] Pixhawk, “Pixhawk 4.” Available:
https://docs.px4.io/main/en/flight_controller/pixhawk4.html

[40] MADR (2023). *Statistiques agricoles 2022-2023*. Ministère de l'Agriculture et du Développement Rural, Alger.

[41] INPV (2022). Rapport annuel sur les maladies des cultures en Algérie. Institut National de la Protection des Végétaux, Alger.

[42] ITCMI (2021). Guide technique : Protection intégrée de la pomme de terre en Algérie. Institut Technique des Cultures Maraîchères et Industrielles, Tipaza.

[43] Benmoussa, A. & Khalfallah, H. (2022). Phytophthora infestans : Épidémiologie et stratégies de lutte en Algérie. Revue Phytopathologique, 15(2), 45-58.

[44] ONS (2023). Série statistique : Superficies et productions agricoles. Office National des Statistiques, Alger.

[45] DIEV (2022). État des lieux de la filière pomme de terre en Algérie. Direction des Industries de la Valorisation, Ministère du Commerce.

[46] Fezzai, K., Moussa, A., & Lekhouimes, A. (2021). Enquête des ravageurs et maladies de la culture de pomme de terre dans la région d'El Oued [Mémoire de master]. Université d'El Oued.

[47] Université d'Alger (2022). Études phytopathologiques sur les maladies de la pomme de terre. Faculté des Sciences Agronomiques.

[48] FAO (2024). Statistiques de production agricole - Algérie. Organisation des Nations Unies pour l'Alimentation et l'Agriculture.

[49] Hamiche, A. (2024, Décembre). Derrière la progression de la production de la pomme de terre, la moue du marché. Horizons.

[50] HortiTecNews (2024). Pomme de terre : Le potentiel algérien séduit des compagnies américaines. HortiTecNews.

[51] Profert Algérie (2026). Alternariose de la pomme de terre. Profert Algérie.

[52] Maniriho, A., Berti, F., Musabanganji, E., & Lebailly, P. (2021). La rentabilité de la production de pomme de terre dans la région des sols de laves au Rwanda. Tropicultura, 39(4).